

# Maxima

## *Algebra symboliczna na komputerze*

Zbigniew Koza

Uniwersytet Wrocławski  
Instytut Fizyki Teoretycznej

Wrocław, 2009

# Możliwości programu

- Algebra symboliczna
  - wielomiany i funkcje wymierne
  - funkcje trygonometryczne i specjalne
  - pochodne i całki
  - szeregi
  - układy równań (w tym nieliniowe)
  - równania różniczkowe
  - macierze (algebra liniowa), tensory
  - transformaty (Laplace'a, Fouriera)
  - upraszczanie, faktoryzacja wyrażeń

# Możliwości programu (c.d.)

- Obliczenia wartości numerycznych (z dowolną precyzją)
- Rysowanie wykresów (poprzez GNUPlot)
- Przetwarzanie skryptów (język proceduralny, instrukcje **if**, **for**, **do**,...)

# Możliwości programu (c.d.)

- Obliczenia wartości numerycznych (z dowolną precyzją)
- Rysowanie wykresów (poprzez GNUPlot)
- Przetwarzanie skryptów (język proceduralny, instrukcje **if**, **for**, **do**,...)

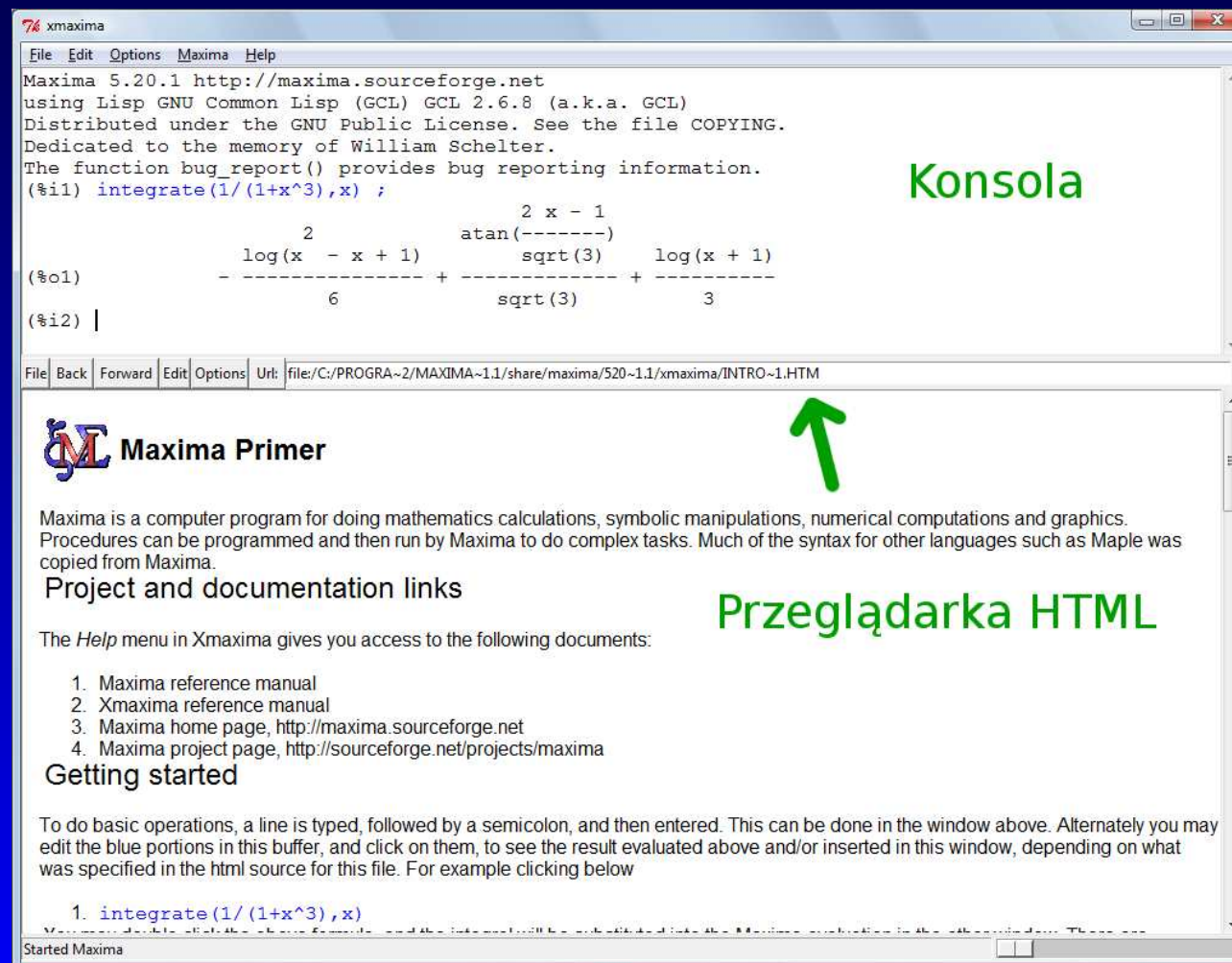
Maxima koncentruje się na operacjach **algebraicznych** – posiada bardzo mało metod stricte numerycznych.

# Uruchamianie programu

- W konsoli tekstowej:  
    > **maxima**
- Tradycyjny interfejs okienkowy:  
    > **xmaxima**
- Nowy interfejs okienkowy:  
    > **wxmaxima**

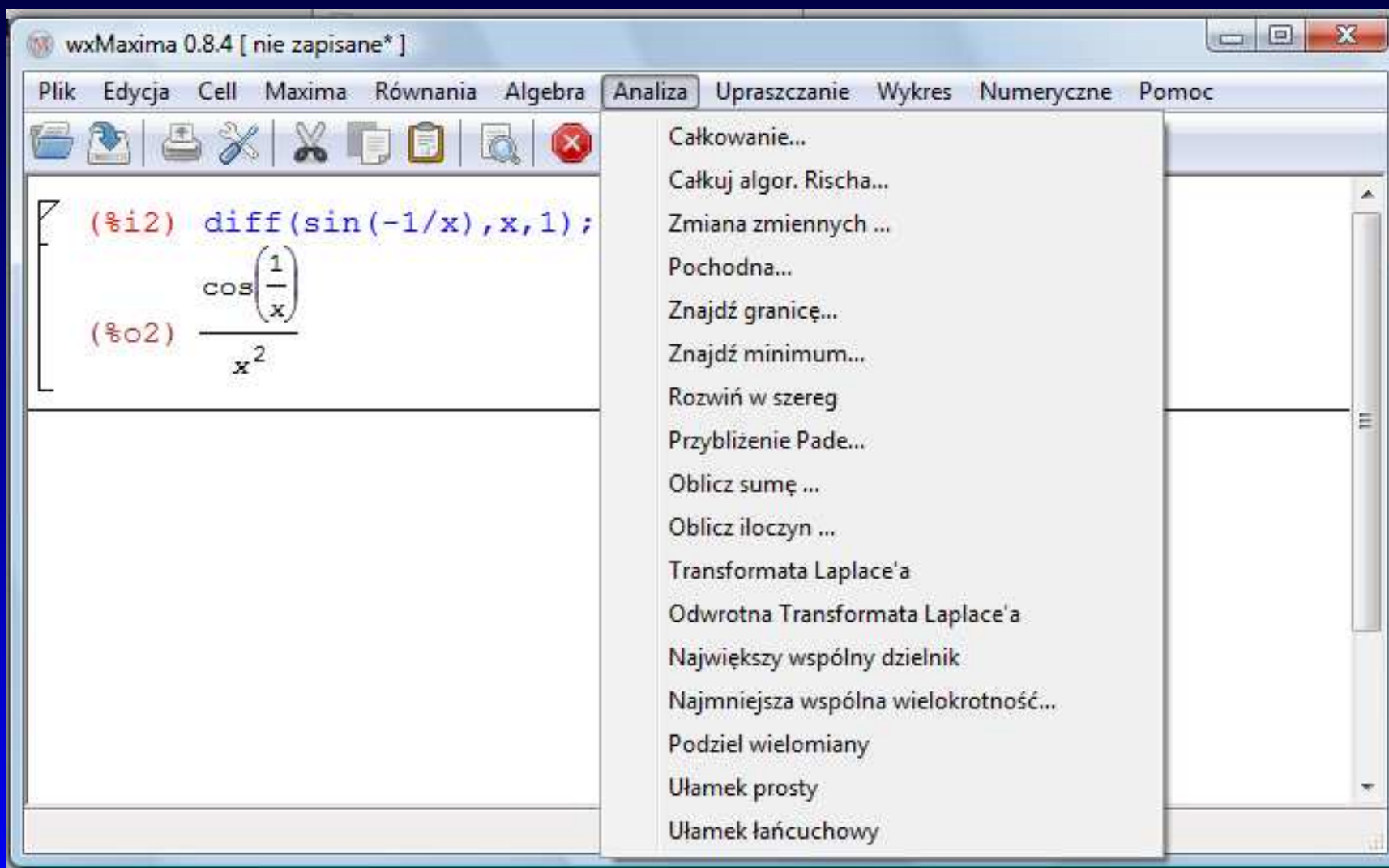
# Wersja xmaxima

- może wyświetlać okno przeglądarki HTML (system pomocy).



# Wersja wxmaxima

- jest najłatwiejsza w obsłudze (i spolszczona)



# xmaxima, wxmaxima

- Dłaska część prezentacji opisuje interfejs w wersji xmaxima
- Jednak początkującym zaleca się używać wersji wxmaxima
- Możliwości obu programów są takie same, różnią się interfejsem

# Źródła informacji

- W programie:
  - **describe(integrate);**
  - **? integrate**
  - **example(integrate);**
  - **apropos(integrate);**
  - Menu **help**
- Inne:
  - <http://maxima.sourceforge.net>
  - Linux+, nr 10/2002

# Wprowadzanie wyrażeń

- W nazwach stałych, zmiennych, funkcji i poleceń odróżnia się wielkość liter.
- Identyfikatory systemowe piszemy małymi literami (od wersji 5.9.2).
- Każdą instrukcję należy zakończyć:
  - średnikiem (;) – wyświetla „wynik”
  - lub dolarem (\$) – nie wyświetla wyniku (ale go oblicza)
- Każda instrukcja wejścia ma swój identyfikator, np. %i2
- Każdy wynik obliczeń ma swój identyfikator, np. %o2
- % oznacza poprzednią instrukcję

# Praca z programem

- **quit();** kończy program
- **reset();** przywraca ustawienia domyślne
- **Alt+P** (**Alt+N**) przywołuje poprzednie (następne) polecenie
- **Ctrl+G** przerywa bieżące obliczenia
- Menu: **File/Restart** odnawia sesję

# Wyrażenia arytmetyczne

- Operatory
  - arytmetyka: **+**, **-**, **\***, **/**
  - potęgowanie: **\*\*** lub **^**
  - silnie: **!**, **!!**
  - relacje: **=**, **<=**, **<**, **>=**, **>**, **#** (nie równe)
  - mnożenie nieprzemienne (np. macierzy): **.**
  - logiczne: **and**, **or**, **not**, **equal**.
- Indeksowanie tablic, list etc: **[]**

# Operatory przypisania

- Istnieją 4 operatory przypisania:

$:$ ,  $::$ ,  $:=$ ,  $::=$

- $x:y$  oznacza „niech wartością  $x$  będzie wartość  $y$ ”
- $F(x) := \sin(x)$  (definiowanie funkcji)
- $x::y$  oznacza „przypisz wartości  $x$  wartość  $y$ ” (używany w funkcjach)
- $::=$  to zapis używany do definiowania makr

Z powyższych operatorów największe zastosowanie mają  $:$  i  $:=$ .

# Przykład

(%i1) a : x;

(%o1)

x

(%i2) x : 1;

(%o2)

1

(%i3) a;

(%o3)

x

(%i4) ev(a);

(%o4)

1

# Predefiniowane stałe

- **%i** – jednostka urojona ( $\sqrt{-1}$ )
- **%e** – podstawa logarytmów naturalnych
- **%pi** –  $\pi$  (ludolfina)
- **%gamma** – stała Eulera ( $\lim_{n \rightarrow \infty} \sum_{i=1}^n (1/i) - \ln(n)$ )
- **%phi** =  $(\sqrt{5} + 1)/2$  (złoty podział)
- **inf** =  $\infty$
- **minf** =  $-\infty$
- **infinity** =  $\infty$  w dziedzinie liczb zespolonych

# Wybrane funkcje

- min, max
- sin, cos, tan, cot
- asin, acos, atan, acot
- sinh, cosh, tanh, coth
- asinh, acosh, atanh, acoth
- log, exp, sqrt, abs
- binomial
- airy, bessel, bessel\_j, erf, gamma, elliptic\_kc, zeta, hermite,...
- random, gauss
- primep

# Obliczanie wartości numerycznej

```
(%i7) erf(1);  
(%o7)          erf(1)  
(%i8) ev(erf(1),numer);  
(%o8)          0.84270079294971  
(%i9) erf(1), numer;  
(%o9)          0.84270079294971
```

Domyślnie Maxima stara się wykonywać obliczenia  
*dokładnie...*

# Operator ' /

Operator ' zakazuje obliczania wartości swojego argumentu

```
(%i1) x : 7;
```

```
(%o1) 7
```

```
(%i2) y : x;
```

```
(%o2) 7
```

```
(%i3) y : 'x;
```

```
(%o3) x
```

```
(%i4) diff(sin(x), x)
```

```
(%o4) cos(x)
```

```
(%i5) 'diff(sin(x), x);
```

```
d
```

```
(%o5) -- (sin(x))
```

```
dx
```

# Zmienne związane i swobodne

```
(%i1) x:1;      związane zmiennej x z 1
```

```
(%o1)          1
```

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

```
Non-variable 2nd argument to diff: 1
```

```
-- an error.  Quitting.  To debug this try
```

```
debugmode(true);
```

```
(%i3) kill(x);   uwolnienie zmiennej x
```

```
(%o3)           done
```

```
(%i4) diff(sin(x),x);
```

```
(%o4)           cos(x)
```

```
(%i5) diff(sin('x),'x);
```

```
(%o5)           cos(x)
```

# Usuwanie wartości zmiennej

- **x : 'x;** „resetuje” zmienną
- **kill(x);** usuwa zmienną z pamięci
- **kill(values);** usuwa zmienne użytkownika
- **kill(functions);** usuwa funkcje użytkownika
- **kill(all);** czyści pamięć
- **kill(allbut(x,y));**

# Listy

Maxima uwielbia operacje na listach.

```
(%i1) l1 : [1,2,3];  
(%o1)      [1, 2, 3]  
(%i2) makelist(i^2,i,0,5);  
(%o2)      [0, 1, 4, 9, 16, 25]  
(%i3) makelist(i^2,i,[0,10,100]);  
(%o3)      [0, 100, 10000]  
(%i4) %[2];  
(%o4)      100
```

- elementy list ujmujemy w nawiasy kwadratowe
- do elementu listy odwołujemy się poprzez operator []
- indeksem pierwszego elementu listy jest 1

# Operacje na listach

```
(%i1) append([1,2],[1,3,x]);
```

```
(%o1)      [1, 2, 1, 3, x]
```

```
(%i2) apply (min, %);
```

```
(%o2)      min(x, 1)
```

```
(%i3) concat(a,1);
```

```
(%o3)      a1
```

```
(%i4) delete (1, %o1);
```

```
(%o4)      [2, 3, x]
```

```
(%i1) sort([1,3,7,2]);
```

```
(%o1)      [1, 2, 3, 7]
```

# Środowisko lokalne - **ev**

Funkcja **ev** wprowadza lokalne środowisko obliczeniowe.

```
(%i1) ev(a*x+b, a=1, b=2);
```

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

```
(%i2) ev(f(x)*x, f(x)=sin(x));
```

```
(%o2)      x sin(x)
```

```
(%i3) ev('diff(a*sin(b*x), x), diff);
```

```
(%o3)      a b cos(b x)
```

```
(%i4) ev((x+y)^2, expand);
```

```
(%o4)      y^2 + 2xy + x^2
```

```
(%i5) ev(log(x) + log(y), logcontract);
```

```
(%o5)      log(x y)
```

```
(%i6) ev(sum(i, i, 0, n), simpsum);
```

```
(%o6)      (n^2 + n)/2
```

# Loklane środowisko - **ev**

Przykłady numeryczne

```
(%i1) ev(sin(x), x=1);
```

```
(%o1) sin(1)
```

```
(%i2) ev(sin(x), x=1, numer);
```

```
(%o2) 0.8414709848079
```

```
(%i3) ev(sin(x) + cos(x), x=1, sin);
```

```
(%o3) cos(1) + 0.8414709848079
```

```
(%i4) ev(sin(x)+cos(x), x=1, fpprec=50, bfloat);
```

```
(%o4) 1.38177329067603622405343892907327560335487348141  
6293293342848965373010799165711433466591599630235785B0
```

```
(%i5) ev(sin(x)+cos(x), x=1.0, fpprec=50, bfloat);
```

Warning: Float to bigfloat conversion of

```
1.3817732906760363
```

```
(%o5) 1.38177329067603629899340644296930997499514646367  
8238418627629897151250514070420421838102205493429226B0
```

# Argumenty funkcji **ev**

**EV(expr, arg1, arg2, ...);**

**expr** to obliczane wyrażenie

Argumentami (opcjonalnymi) mogą być:

- podstawienia zmiennych, np. **x = z**;
- definicje funkcji, np. **f(x) = z**;
- nazwa funkcji występującej w **expr**
- nazwa funkcji przekształcającej **expr**: factor, trigexpand, trigreduce, bfloat, ratsimp, ratexpand, radcan, logcontract, rectform, polarform.
- flaga: float, pred, simp, numer, detout, demoivre, keepfloat, exponentialize, listarith, trigexpand, simpsum, algebraic, ratalgdenom, factorflag, %emode, %enumber, logarc, lognumer, radexpand, ratsimpexpons, ratmx, ratfac, infeval, halfangles, programmode, lognegint, logabs, letrat, exptisolate, isolate\_wrt\_times, sumexpand, cauchysum, numer\_pbranch, m1pbranch, dotscrules, logexpand.

# Upraszczanie zapisu

Zamiast

```
ev(sin(x), x=1, fpprec=50, bfloat);
```

można napisać

```
sin(x), x=1, fpprec=50, bfloat;
```

lub

```
sin(x), x=1, fpprec:50, bfloat;
```

# Funkcje przekształcające wyrażenia

- **factor** – faktoryzuje wyrażenie
- **expand** – rozwija wyrażenie
- **trigexpand** – rozwija wyrażenie trygonometryczne lub hiperboliczne, np.  **$\sin(2*x+y)$** ,  **$\sinh(2*x+y)$** .
- **trigreduce** – operacja odwrotna do **trigexpand**
- **ratsimp**, **ratexpand** – upraszcza wyrażenia wymierne
- **radcan** – upraszcza funkcje logarytmiczne, wykładnicze i pierwiastki.

# Funkcje przekształcające wyrażenia

- **bfloat** – zamienia liczby na "Big Float"
- **logcontact** – upraszcza wyrażenia typu  $a \log x + b \log y$ .
- **rectform** – przedstawia wyrażenie w postaci  $A + Bi$ .
- **polarform** – przedstawia wyrażenie zespolone w postaci biegunowej,  $re^{i\theta}$ .

# Upraszczenie wyrażeń

- Maxima domyślnie nie stosuje uproszczeń typu:

- $\sqrt{a}\sqrt{b} = \sqrt{ab}$

- $(a^b)^c = a^{bc}$

- $\log x + \log y = \log xy$

gdyż nie są one zawsze prawdziwe np. dla liczb zespolonych.

- Maxima nie wykonuje wielu innych „uproszczeń”, bo
  - mogą być kosztowne
  - nie zawsze są prawdziwymi uproszczeniami
- Dlatego wiele uproszczeń wykonuje się „ręcznie” (**radcan**, **ratsimp**,...)

# assume i forget

- Poleceniem **assume** można poinformować Maksimę o dodatkowych właściwościach zmiennych, co może ułatwić upraszczanie wyrażeń
- **forget** odwołuje definicje **assume**

```
(%i1) sqrt(a*a);  
(%o1)      abs(a)  
(%i2) assume(a>0);  
(%o2)      [a > 0]  
(%i3) sqrt(a*a);  
(%o3)      a  
(%i4) forget(a>0);  
(%o4)      [a > 0]  
(%i5) sqrt(a*a);  
(%o5)      abs(a)
```

# subst i ratsubst

```
(%i1) subst(0, x, a*sin(x));
```

```
(%o1)      0
```

```
(%i2) subst(x=0, a+sin(x));
```

```
(%o2)      a
```

```
(%i3) subst([x=y+1, z=1], x*y*z);
```

```
(%o3)      y (y + 1)
```

```
(%i4) subst(sin(x)=z, sin(x)*a + sin(x));
```

```
(%o4)      a z + z
```

```
(%i5) ratsubst(z, x+y, x+z+y);
```

```
(%o5)      2 z
```

- `subst(co, za_co, gdzie)`
- `subst(za_co = co, gdzie)`
- `ratsubst(co, za_co, gdzie)`

# scsimp

- **scsimp** służy do upraszczania wyrażenia, jeśli znane są dodatkowe warunki (więzy).

```
(%i1) scsimp(x + y*x + x*y*z + 1,  
x*z=y, x+1 = y );  
(%o1)  $2y^2$ 
```

# Wielomiany

```
(%i1) gcd(x^100-1, x^15-1);
```

```
(%o1)      x5 - 1
```

```
(%i2) divide(x^2+1, x+1);
```

```
(%o2)      [x - 1, 2]
```

```
(%i3) quotient(x^2+1, x+1);
```

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

```
(%i4) remainder(x^100-1, x^3+1);
```

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

```
(%i5) factor(x^4-1);
```

```
(%o5)      (x - 1) (x + 1) (x2 + 1)
```

- **gcd** – największy wspólny dzielnik
- **quotient** – iloraz
- **remainder** – reszta z dzielenia
- **divide** – zwraca [iloraz, reszta] z dzielenia
- **factor** – rozkłada na czynniki

# Sumy

```
(%i1) sum(i^2,i,0,10);
```

```
(%o1)      385
```

```
(%i2) sum(i^2,i,0,n);
```

```
(%o2)      
$$\sum_{i=0}^n i^2$$

```

```
(%i3) sum(i^2,i,0,n), simpsum:true;
```

```
(%o3)      
$$(2n^3 + 3n^2 + n)/6$$

```

```
(%i4) nusum(i^2!,i,0,n);
```

```
(%o4)      
$$n(n+1)(2n+1)/6$$

```

- **sum** – oblicza proste sumy
- flaga **simpsum** kontroluje możliwość algebraicznego wyznaczenia sumy przez **sum**.
- **nusum** – oblicza sumy algebraiczne

# Iloczyny

```
(%i1) product(i/(i+1), i, 1, 10);  
(%o1)          1/11
```

- **product** – oblicza iloczyn

# Szeregi nieskończone

(%i1)

```
powerseries(log(sin(x)/x), x, 0);
```

(%o1)

$$\frac{1}{2} \sum_{i=1}^{\infty} \frac{(-1)^i 2^{2i} \text{Bern}(2i) x^{2i}}{i (2i)!}$$

# Szeregi Taylora (Laurenta)

```
(%i1) taylor(sin(x/exp(x)), x, 0, 4);
```

```
(%o1) /T/  $x - x^2 + x^3/3 + x^4/3 + \dots$ 
```

```
(%i2) taylor (1/(1+x),  
[x, 4, 4, asymp]);
```

```
(%o2) /T/
```

$$\frac{1}{x-4} - \frac{5}{(x-4)^2} + \frac{25}{(x-4)^3} - \frac{125}{(x-4)^4} + \dots$$

- `taylor(expr(var), var, gdzie, ile);`
- `taylor(expr(var), [var, gdzie, ile, asymp]);`

# Funkcje Pade

```
(%i1) taylor(sin(x/exp(x)), x, 0, 4);
```

```
(%o1) /T/  $x - x^2 + x^3/3 + x^4/3 + \dots$ 
```

```
(%i2) pade(%, 2, 2);
```

```
(%o2)
```

$$\frac{3x}{2x^2 + 3x + 3}$$

- `pade(expr(var), licz, mian);`
- `pade(expr(var), inf, inf);`
- Pierwszym argumentem **pade** musi być rozwinięcie Taylora

# Residua

```
(%i1) residue(1/sin(x), x, 0);  
(%o1)      1
```

- `residue(expr, zmienna, gdzie)`

# Ułamki łańcuchowe

```
(%i1) cf(2*sqrt(2)-1/sqrt(3));
```

```
(%o1) [2, 4, 11]
```

```
(%i2) cfdisrep(%);
```

```
(%o2)
```

$$2 + \frac{1}{4 + \frac{1}{11}}$$

# Granice

```
(%i1) limit(1/x, x, 0);
```

```
(%o1)          und
```

```
(%i2) limit(1/x, x, 0, plus);
```

```
(%o2)          inf
```

```
(%i3) limit(1/x, x, 0, minus);
```

```
(%o3)          minf
```

```
(%i4) limit(erf(x), x, inf);
```

```
(%o4)          1
```

- `limit(expr, zmienna, gdzie)`
- `limit(expr, zmienna, gdzie, plus)`
- `limit(expr, zmienna, gdzie, minus)`

# Pochodne

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

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

```
(%i2) diff(sin(x)*cos(x),x,2);
```

```
(%o2)      - 4 cos(x) sin(x)
```

```
(%i3) diff(sin(x)*cos(y),x,1,y,1);
```

```
(%o3)      - cos(x) sin(y)
```

- `diff(expr, zmienna)`
- `diff(expr, zmienna, ile_razy)`
- `diff(expr, var1,n1, var2,n2,...)`

# depends

```
(%i1) diff(f,x);
```

```
(%o1)      0
```

```
(%i2) depends(f,x);
```

```
(%o2)      [f(x)]
```

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

```
(%o3)       $\frac{df}{dx}$ 
```

```
(%i4) depends([x],[u,v,w]);
```

```
(%o4)      [x(u, v, w)]
```

```
(%i5) diff(f,u);
```

```
(%o5)       $\frac{df}{dx} \frac{dx}{du}$ 
```

```
(%i5) dependencies;
```

```
(%o5)      [f(x), x(u, v, w)]
```

- `depends(zmienna, zmienna)`
- `depends([zmienne], [zmienne])`

# Całkowanie

```
(%i1) integrate(sin(x*2),x);
```

```
(%o1)          -cos(2 x)/2
```

```
(%i2) integrate(exp(-x*x),x,0,inf);
```

```
(%o2)           $\frac{\sqrt{\pi}}{2}$ 
```

- `integrate(co, zmienna)`
- `integrate(co, zmienna, od, do)`

# changevar

(%i1) 'integrate(exp(a\*x), x, 0, 1);

(%o1)  $\int_0^1 \exp(ax) dx$

(%i2) changevar(%, a\*x-y, y, x);

Is a positive, negative, or zero?

positive;

(%o2)

$$\frac{1}{a} \int_0^a \exp(y) dy$$

- **changevar(expr, f(x,y), new, old)**
- w powyższym wywołaniu **f(x,y)= 0**
- **changevar** można też stosować do sum

# Transformata Laplace'a

```
(%i1) laplace(exp(t)*cos(t), t, s);
```

```
(%o1) (s - 1)/(s^2 - 2s + 2)
```

```
(%i2) laplace('diff(f(x), x, 2), x, s);
```

$$(\%o2) -\frac{d}{dx} f(x) \Big|_{x=0} + s^2 \mathcal{L}(f(x), x, s) - f(0) s$$

- `laplace(fun, old_var, new_var)`

# Odwrotna transformata Laplace'a

```
(%i2) ilt((s-1)/(s*s-2*s+2), s, t);  
(%o8)      exp(t)*cos(t)
```

- `ilt(fun, old_var, new_var)`

# atvalue

```
(%i1) atvalue(f(x), x=0, a);  
(%o1)      a  
(%i2) atvalue('diff(f(x), x), x=0, b);  
(%o2)      b  
(%i3) laplace('diff(f(x), x, 2), x, s);  
(%o3) s laplace(f(x), x, s) - a s - b  
(%i4) printprops(f, atvalue);
```

$$\frac{d}{dx}f|_{x=0} = b$$
$$f(0) = a$$

- `atvalue(fun, [gdzie], ile)`
- `printprops(fun, atvalue)`