# Úvod do Sage

# Jemný úvod do Sage

*Tomáš Kalvoda, 2014*

## Povídání o Sage

Sage je open-source matematický software vyvýjený od roku 2005 zejména Williamem Steinem, profesorem matematiky na University of Washington. Cíle Sage nejsou nikterak malé, dle oficiální stránky:

Mission: *Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab*.

Všechny podstatné informace o Sage může zájemce nalézt na oficiální stránce https://www.sagemath.org. Zejména je zde k dispozici podrobná dokumentace a instalační balíčky ke stažení.

K Sage lze přistupovat několika způsoby. První možností je stáhnout si balíček a nainstalovat Sage na svém PC (platforma Windows v tento okamžik není podporována a je nutno přistoupit k virtualizaci). Sage lze pak spustit z příkazové řádky, což však není příliš pohodlné. Příkazem

```
notebook()
```

lze však spustit interaktivnější rozhranní zvané *Sage Notebook* běžící v prohlížeči. Uživatelské prostředí je velmi inspirovnáno programem *Mathematica*. Uživatel pracuje se sešitem rozděleným na buňky, které mohou obsahovat kód, výsledky výpočtů, text nebo i matematické výrazy vkládané pomocí LaTeXu.

Další možností jak vyzkoušet Sage je použít cloudovou službu https://cloud.sagemath.org. Tato služba je aktuálně zdarma. Po registraci uživatel získává možnost vytvářet projekty (plnohodnotné linuxové virtuální stroje) v nichž může provádět své výpočty. K Sage lze přistupovat buď pomocí *Sage Worksheet* prostředí (podobné *Sage Notebook*) nebo *IPython Notebook*. Cloudová služba umožňuje ale daleko širší spektrum možností. Lze například přehledně editovat LaTeX soubory, projekty sdílet a pracovat tak kooperativně, nebo využívat příkazovou řádku virtuálního linuxového stroje.

## Python

Sage je založen na rozšířeném programovacím jazyku Python. Díky tomu Sage obsahuje celou řadu zajímavých matematických balíčků (knihoven) napsaných v tomto jazyce (např. NumPy, SymPy a další).

V tomto odstavci shrneme jenom to nejnutnější minimum týkající se jazyka Python, aby případný čtenář neměl problém chápat a orientovat se v ukázkách. Čtenář prahnoucí po hlubším proniknutí do tohoto programovacího jazyka nebude mít problém nalézt zajímavé studijní zdroje na internetu (např. https://docs.python.org/2/tutorial/).

Není potřeba deklarovat typ proměnných, k inicializaci se používá standardní symbol přiřazení =.

```
a = 4
print(a)
a = 'Hello world!'
print(a)
```

```
4
Hello world!
```

Často používanou datovou strukturou je tzv. *list*, či pole. Pokud chceme *list* zadat explicitně používáme k tomu hranaté závorky a prvky oddělujeme čárkami. K prvkům pole pak přistupujeme pomocí indexu běžícího od nuly.

```
a = [1, 'B', pi ]
print(a[1])
print(len(a))
```

```
B
3
```

Nyní se podívejme na jednu specifickou a pro nováčka potenciálně matoucí vlastnost Pythonu. K oddělení bloku kódu se nepoužívá klíčových slov, ani závorek, ale **odsazení**. Demonstrujme tento jev na ukázce **if-else**.

```
if pi > 3:
    print('Ano!')
else:
    print('Ne!')
```

```
Ano!
```

Podobně se chová známá for konstrukce (pod **range(4)** je dobré představovat množinu indexů od 0 do 3, tedy délky 4)

```
for k in range(4):
    y = k^2
    print y
```

```
0
1
4
9
```

Pokud chceme definovat vlastní funkci, použijeme pro to klíčové slovo **def**. Všimněte si opět odsazení.

```
def f(x):
    y = 4*x
    return y
```

Dále je dobré zdůraznit, že u argumentů funkcí není potřeba udávat jejich typ. Naše funkce bude "fungovat" na všech objektech, pro které lze provést operace uvedené v těle funkce.

```
show(f(4))
```

```
show(f(pi))
show(f('Ahoj!'))
```

$$16$$

$$4\,\pi$$

`Ahoj!Ahoj!Ahoj!Ahoj!`

Pyhon je objektový jazyk. Prostředí *Sage Worksheet* i *Sage Notebook* nabízí kontextovou nápovědu snadno vyvolatelnou pomocí klávesy TAB. Zkuste nastavit kurzor za tečku a stisknout klávesu TAB. Například níže vytvoříme objekt reprezentující celé číslo 9 ne jako pythonovský **int**, ale jako prvek okruhu celých čísel.

```
# malé celé číslo
a = 9
```

K dispozici pak máme celou řadu metod, které obyčejný pythonovský **int** nemá. Například:

```
print 'Je prvočíslo?'
print a.is_prime()
print 'Faktorizace:'
show(a.factor())
```

```
Je prvočíslo?
False
Faktorizace:
```

$$3^2$$

Pokud si nevíme rady s jistou funkcí, můžeme vyvolat nápovědu po stisknutí TAB klávesy za otevírací závorkou. Případně lze nápovědu vypsat pomocí otazníku za názvem funkce.

```
factorial?
```

**File:** /opt/sage/local/lib/python2.7/site-packages/sage/functions/other.py

**Type:** <class 'sage.functions.other.Function_factorial'>

**Definition:** factorial(coerce=True, hold=False, dont_call_method_on_arg=False, *args)

**Docstring:**

Returns the factorial of $n$.

INPUT:

- n - any complex argument (except negative integers) or any symbolic expression

OUTPUT: an integer or symbolic expression

EXAMPLES:

```
sage: x = var('x')
sage: factorial(0)
1
sage: factorial(4)
24
sage: factorial(10)
3628800
sage: factorial(6) == 6*5*4*3*2
True
sage: f = factorial(x + factorial(x)); f
factorial(x + factorial(x))
sage: f(x=3)
362880
sage: factorial(x)^2
factorial(x)^2
```

To prevent automatic evaluation use the `hold` argument:

```
sage: factorial(5,hold=True)
factorial(5)
```

To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```
sage: factorial(5,hold=True).simplify()
120
```

We can also give input other than nonnegative integers. For other nonnegative numbers, the `gamma()` function is used:

```
sage: factorial(1/2)
1/2*sqrt(pi)
sage: factorial(3/4)
gamma(7/4)
sage: factorial(2.3)
2.68343738195577
```

But negative input always fails:

```
sage: factorial(-32)
Traceback (click to the left of this block for traceback)
...
```

# Číselné množiny a důležité konstanty

## Celá čísla

```
print ZZ
show(ZZ)
```

Integer Ring

$\mathbf{Z}$

```
# ekvivalentní způsoby vytvoření obektu typy Sage Integer
j = 1
k = ZZ(1)
l = Integer(1)
j == k
k == l
type(j)
```

```
<type 'sage.rings.integer.Integer'>
```

## Racionální čísla

```
print QQ
show(QQ)
```

```
Rational Field
```

$\mathbf{Q}$

```
j = QQ(1.5)
print j
```

```
3/2
```

```
t = 55/12
type(t)
```

```
<type 'sage.rings.rational.Rational'>
```

```
j + k
```

```
5/2
```

## Reálná čísla (v dané přesnosti)

Přesněji řečeno, nejde o reálná čísla ale o jejich numerickou aproximaci. Sage nám naštěstí umožnuje explicitně zadat přesnost, v jaké počítáme. V tom se poměrně podstatně liší od Mathematica, kde se s těmito tzv. strojovými číslay pracuje zcela jiným způsobem.

```
print RR
show(RR)
```

```
Real Field with 53 bits of precision
```

$\mathbf{R}$

```
F = RealField(prec=10)
```

```
x = RR(-1)
type(x)
```

```
<type 'sage.rings.real_mpfr.RealNumber'>
```

```
# odmocnina definována stejně jako v Mathematica
y = x ^ (1/3)
print y
```

```
type(y)
```

```
0.500000000000000 + 0.866025403784439*I
<type 'sage.rings.complex_number.ComplexNumber'>
```

```
RealField(100)
```

```
Real Field with 100 bits of precision
```

Eulerovo číslo.

```
show(e)
type(e)
```

$$e$$

```
<type 'sage.symbolic.constants_c.E'>
```

```
print N(e,digits=10)
print N(e,prec=32)
```

```
2.718281828
2.71828183
```

Ludolfovo číslo $\pi$

```
show(pi)
type(pi)
```

$$\pi$$

```
<type 'sage.symbolic.expression.Expression'>
```

```
print N(pi,digits=10)
print N(pi,prec=32)
```

```
3.141592654
3.14159265
```

## Komplexní čísla

Imaginární jednotka $i$.

```
I
type(I)
```

```
<type 'sage.symbolic.expression.Expression'>
```

```
I^2
```

```
-1
```

## Symbolický okruh

Často je potřeba pracovat s čísly v absolutní přesnosti, resp. pracovat se symbolickými výrazy. K tomu v Sage slouži symbolický okruh.

```
SR
```

```
Symbolic Ring
```

Pokud Sage dáme symbolický výraz (bez proměnné, o nich níže), automaticky ho interpretuje jako prvek **SR**.

```
a = sqrt(2)
b = log(pi)/sqrt(8)
show(a*b)
```

$$\frac{1}{2}\,\log\left(\pi\right)$$

## Algebra a symbolické výrazy

Sage umožňuje pracovat i se symbolickými proměnnými a výrazy. Nejprve je potřeba vytvořit proměnnou, s kterou budeme pracovat.

```
var('x')
```

```
x
```

Poté můžeme vytvořit výraz, který nás zajímá.

```
expr = x^2 + sin(x) / (x^2 + 1)
show(expr)
```

$$x^{2} + \frac{\sin\left(x\right)}{x^{2} + 1}$$

```
# jakého typu je tento objekt?
type(expr)
```

```
<type 'sage.symbolic.expression.Expression'>
```

Dosazování.

```
# pomocí rovnosti
show(expr(x = pi/2))
# pomocí slovniku (substituce)
show(expr({x:pi/2}))
```

$$\frac{1}{4}\,\pi^{2} + \frac{4}{\pi^{2} + 4}$$

$$\frac{1}{4}\,\pi^2 + \frac{4}{\pi^2 + 4}$$

Algebraické úpravy.

```
expr = (x+4)^5
show(expr)
```

$$\left(x + 4\right)^{5}$$

Roznásobení.

```
expr = expr.expand()
show(expr)
```

$$x^{5} + 20\,x^{4} + 160\,x^{3} + 640\,x^{2} + 1280\,x + 1024$$

A naopak faktorizace polynomu, tedy známá úprava na kořenové činitele.

```
expr = expr.factor()
show(expr)
```

$$\left(x + 4\right)^{5}$$

Sage umí pracovat nejen s polynomy. Můžeme provádět i úpravy trigonometrických výrazů.

```
expr = sin(4*x)
show(expr)
```

$$\sin\left(4\,x\right)$$

```
expr = expr.trig_expand()
show(expr)
```

$$4\,\cos\left(x\right)^{3}\sin\left(x\right) - 4\,\cos\left(x\right)\sin\left(x\right)^{3}$$

```
expr = expr.trig_reduce()
show(expr)
```

$$\sin\left(4\,x\right)$$

## Sumace a Řady

V BI-ZMA budeme často pracovat s částečnými součty a řadami. S některými součty nám může Sage pomoci.

```
var('k,n,x')
```

```
(k, n, x)
```

Známý součet prvních $n$ přirozených čísel.

```
expr = sum(k, k, 1, n)
show(expr)
```

$$\frac{1}{2}\, n^2 + \frac{1}{2}\, n$$

Součet prvních $n$ členů jisté geometrické posloupnosti.

```
expr = sum(x^k, k, 0, n-1)
show(expr)
```

$$\frac{x^n - 1}{x - 1}$$

Obskurnější součet.

```
expr = sum(k^2, k, 1, n)
show(expr)
```

$$\frac{1}{3}\, n^3 + \frac{1}{2}\, n^2 + \frac{1}{6}\, n$$

Nyní se pokusme sečíst některé mocninné řady. O nich se čtenář doví více později v semestru.

```
sum(x^k / factorial(k), k, 0, infinity)
```

```
e^x
```

```
sum((-1)^k*x^(k+1)/(k+1),k, 0, infinity)
```

```
log(x + 1)
```

Naopak, zadáme-li funkci, pak se ji můžeme pokoušet v mocninnou Taylorovu řadu rozvíjet.

```
taylor(sin(x), x, 0, 10)
```

```
1/362880*x^9 - 1/5040*x^7 + 1/120*x^5 - 1/6*x^3 + x
```

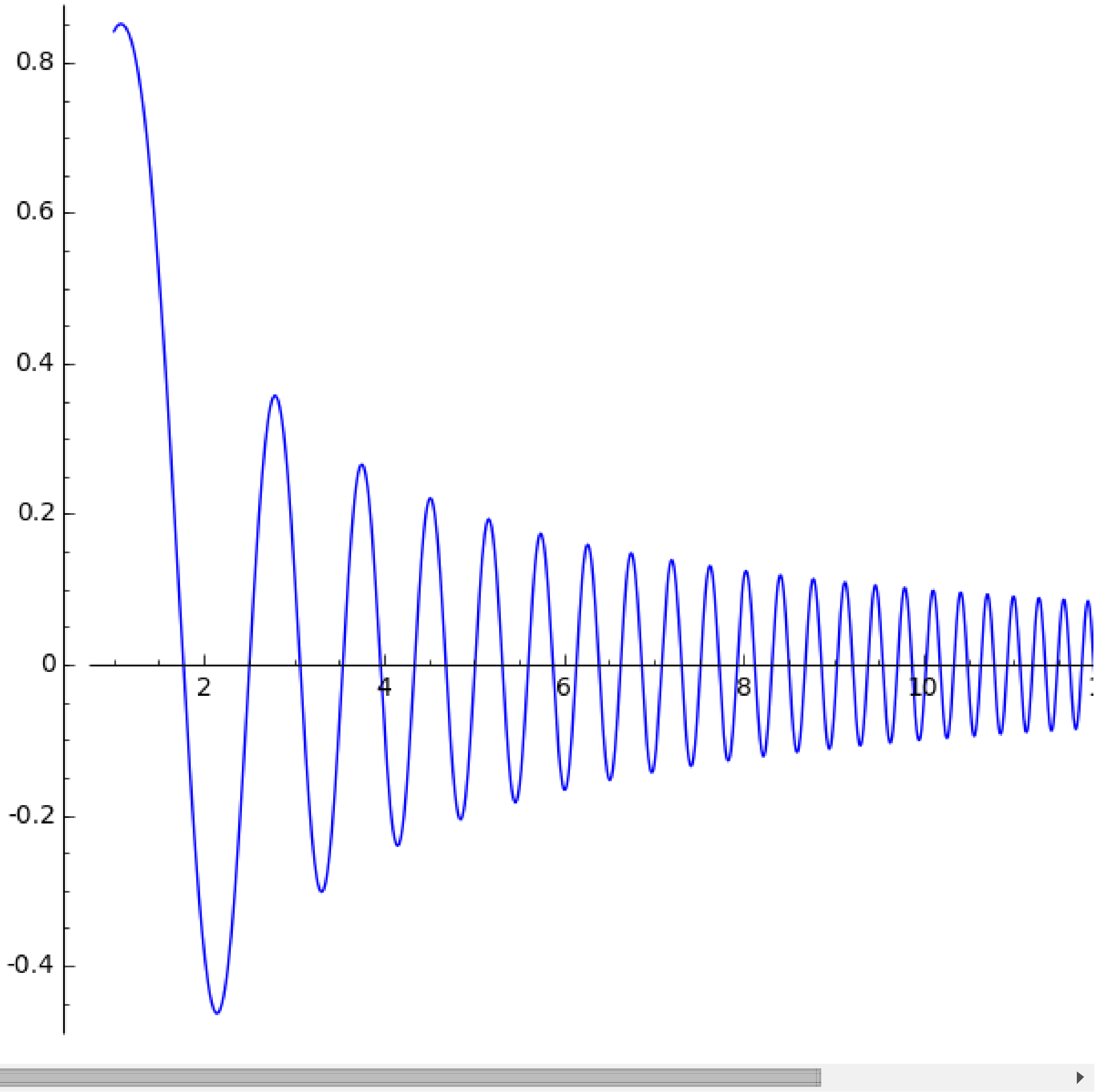
```
taylor(exp(x), x, 0, 10)
```

```
1/3628800*x^10 + 1/362880*x^9 + 1/40320*x^8 + 1/5040*x^7 + 1/72
+ 1/120*x^5 + 1/24*x^4 + 1/6*x^3 + 1/2*x^2 + x + 1
```

# Funkce a jejich grafy

Sage podporuje mnoho způsobů jak vytvářet všemožné typy grafů. Nejjednodušším způsobem je asi vytvoření symbolického výrazu s jednou symbolickou proměnnou a použití příkazu **plot**. Předveďme si tento postup na jednoduchém příkladě.

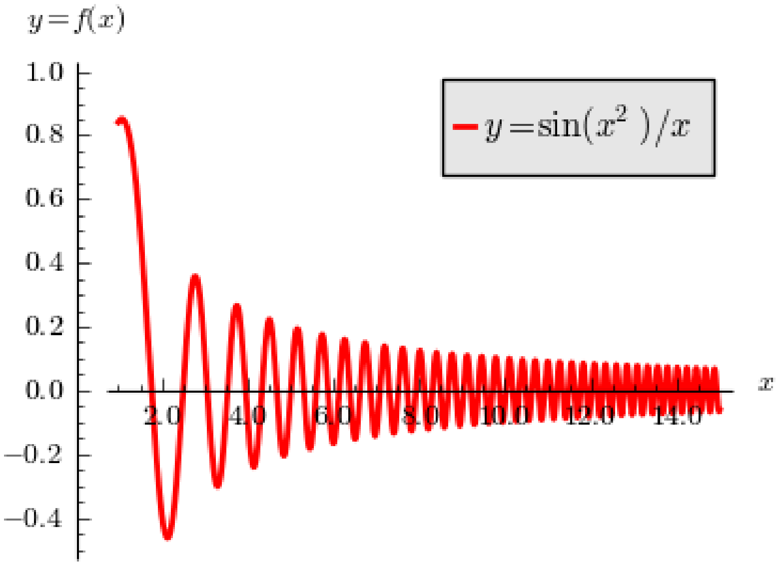
```
# Bud x symbolicka promenna.
var('x')
# Graf funkce 1/x*sin(x^2) pro x z intervalu <1,15>.
plot(1/x*sin(x^2),(x,1,15))
```




Na předchozím obrázku jsme jen specifikovali funkci a rozsah nezávisle proměnné. Sage nám umožňuje vyladit i ostatní parametry grafu. V následující ukázce si ukážeme několik užitečných parametrů. Interně Sage k tvorbě grafů využívá Pythonovskou knihovnu **matplotlib**.

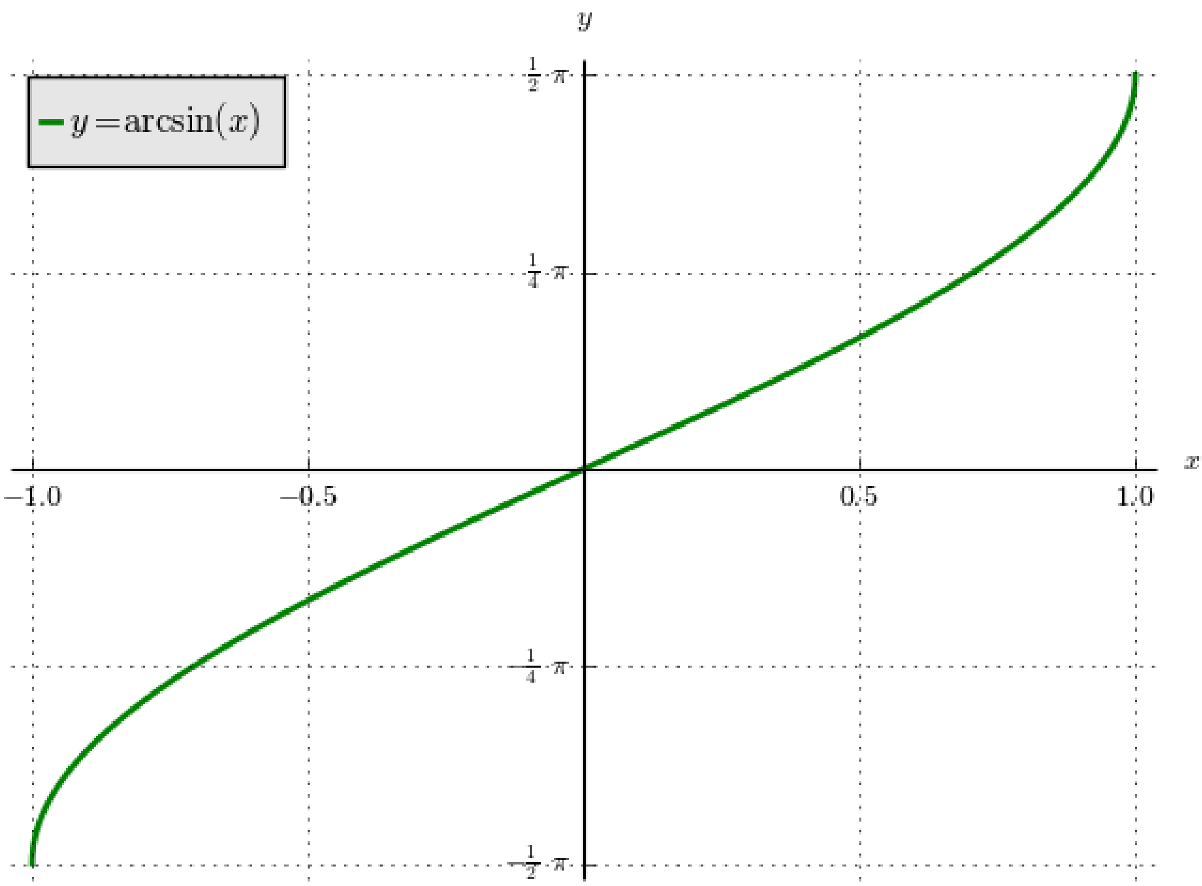
```
plot(1/x*sin(x^2),(x,1,15),
    ymin=-0.5, ymax=1,                    # rozsah svislé osy
    thickness=2,                          # tloušťka křivky
    rgbcolor='red',                       # barva
```

```
    axes_labels=['$x$','$y=f(x)$'],     # popisky os, lze
využívat LaTeX
    tick_formatter='latex',             # cejchování os stejným
fontem jako popisky os
    legend_label='$y = \\sin(x^2)/x$',  # legenda (vhodné při
kombinování více grafů)
    figsize=4)                          # velikost výsledného
obrázku
```




Občas je potřeba přesně specifikovat na kterých místech se mají osy cejchovat (typicky u goniometrických funkcí). V následující ukázce grafu funkce $\arcsin$ si uážeme jak na to.

```
plot(arcsin(x),(x,-1,1),
    ymin=-pi/2, ymax=pi/2,
    thickness=2, rgbcolor='green',
    axes_labels=['$x$','$y$'], tick_formatter='latex',
    legend_label='$y = \\arcsin(x)$',
    ticks=[[-1,-1/2,1/2,1],[-pi/2,-pi/4,pi/4,pi/2]], # cejchování
os
    gridlines=True,                                  # souřadná
mříž
    figsize=6)
```



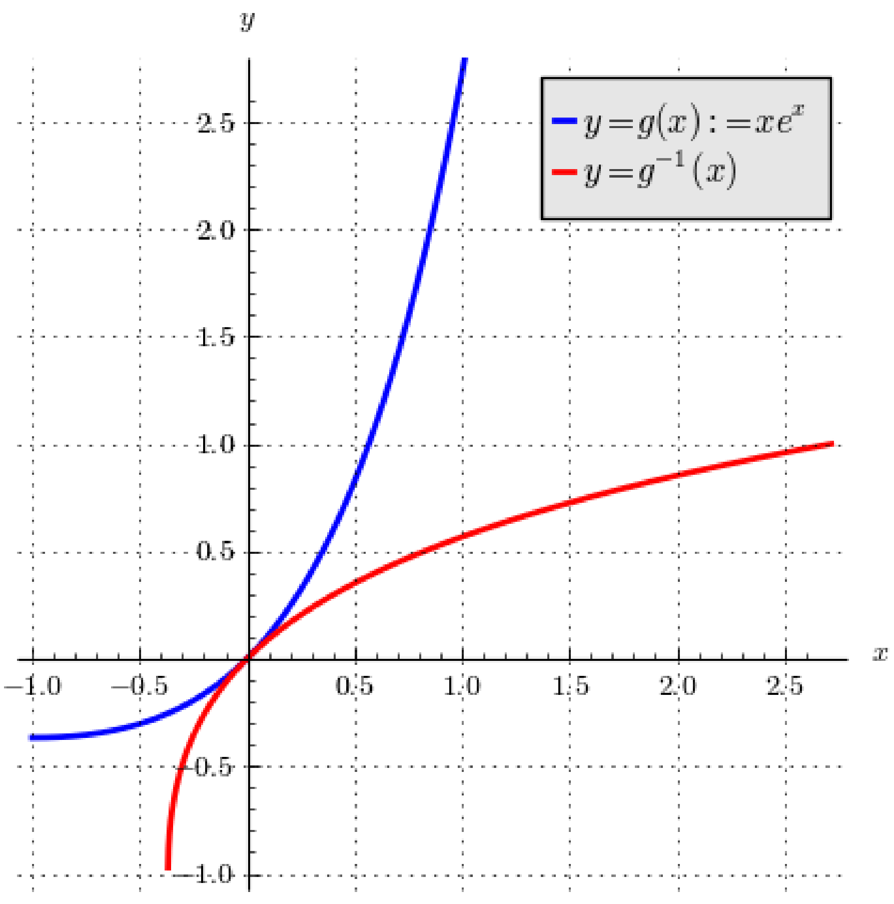

Funkce **plot** akceptuje i obyčejnou Pythonovskou funkci, která vrací číselné výsledky. Syntaxe je jen nepatrně odlišná (neuvádí se nezávisle proměnná).

```
def lambert(z):
    """
        Naivní implementace Lambertovy funkce, tedy inverze k
g(w) = w*exp(w), kde w > -1. Výpočet pomocí Newtonovy metody s
očekávanou přesností na 5 cifer za desetinnou tečkou.
    """

    # Je argument "z" z definičního oboru?
    if z <= -1/e:
        raise ValueError('Argument neni v definicnim oboru
Lambertovy funkce!')

    # Přesnost a iterátor rekurentní posloupnosti.
    eps = 1e-6
    newton = lambda w: w - (w*exp(w) - z) / (exp(w) + w*exp(w))

    # První nástřel.
    if z < 0:
        y1 = -0.5
    elif z > 0:
        y1 = z/2
    else:
        return 0
```

```
    # Iterativní výpočet.
    y2 = newton(z)
    while abs(y1 - y2) > eps:
        y1,y2 = y2,newton(y2)

    return y2
```

A nakonec graf s oběma funkcemi. Zde také ukazujeme, jak kombinovat více grafických objektů do jednoho. K tomu slouží operáotor "+". Různa nastavení grafiky (osy, velikost obrázku, atp.) stačí uvést jednou v prvním grafickém objektu.

```
fig1 = plot(x*exp(x),(x,-1,e),
    ymin=-1,ymax=e,
    thickness=2, rgbcolor='blue',
    axes_labels=['$x$','$y$'], tick_formatter='latex',
    legend_label='$y = g(x) := x e^x$', gridlines=True,
    figsize=6,aspect_ratio=1)

fig2 = plot(lambert,(-1/e,e),
    thickness=2, rgbcolor='red',
    legend_label='$y = g^{-1}(x)$')

fig1 + fig2
```




# Limity posloupností a funkcí

Pokud chceme pomocí Sage (ale i Mathematica) počítat limity posloupností je nutné dát CAS na vědomí, že počítáme s diskrétní celočíselnou proměnnou.

```
var('n,x')
assume(n,'integer')
```

O proměnné $x$ jsme žádný předpoklad neučinili. O $n$ předpokládáme, že je celočíselná.

```
limit(sin(pi*n),n=+infinity)
```

```
0
```

Pro každé celoříselné $n$ totiž platí $\sin(\pi n) = 0$. Sage nám proto dal dobrý výsledek pro limitu $(\sin(\pi n))$ jakožto číselné posloupnosti.

```
limit(sin(pi*x),x=+infinity)
```

```
ind
```

Pokud o proměnné neučiníme žádný předpoklad, Sage automaticky počítá s reálnou funkcí. Funkce $\sin(\pi x)$ je periodická nekonstatní a očividně nemá v nekonečnu limitu.

Ověřme z přednášky známé limity. Čili se také jedná o dobrý výsledek, ovšem z pohledu limity funkce.

```
limit((1+1/n)^n,n=infinity)
```

```
e
```

```
limit((e^x - 1)/x,x=0)
```

```
1
```

```
limit(ln(x+1)/x,x=0)
```

```
1
```

Pomocí nepovinného argumentu dir můžeme kontrolovat i to, zda-li počítáme limitu zleva či zprava.

```
limit(1/x,x=0,dir='right')
```

```
+Infinity
```

```
limit(1/x,x=0,dir='left')
```

```
-Infinity
```

```
limit(sign(x),x=0,dir='right')
```

```
1
```

```
limit(sign(x),x=0,dir='left')
```

```
-1
```

Povšimněte si, že Sage vrací i výsledek pro oboustranou limitu této funkce v $0$.

```
limit(1/x,x=0)
```

```
Infinity
```

Nekonečno je zde myšleno jako komplexní. Uvedená funkce je totiž chápána jako $\mathbb{C} \to \mathbb{C}$.

## Derivace

Ukažme si, jak Sage použít k výpočtu derivací funkcí. Prvním krokem je definovat symbolickou proměnnou $x$, která bude odpovídat naší nezávislé proměnné.

```
var('x')
```

x

Dále definujeme funkci, kterou chceme derivovat.

```
f(x) = sin(x)
```

Všimněte si, že Sage korektně rozlišuje mezi funkcí $f$ a její funkční hodnotou $f(a)$ v bodě $a$.

```
show(f)
show(f(pi/2))
```

$$x \mapsto \sin(x)$$

$$1$$

Derivaci funkce $f$ můžeme získat několika ekvivalentními způsoby. Prvním je zavolání metody derivative přímo na objektu odpovídajícímu funkci $f$.

```
g = f.derivative()
show(g)
show(g(x))
```

$$x \mapsto \cos(x)$$

$$\cos(x)$$

Druhou možností je použití funkce diff.

```
g = diff(f)
show(g)
show(g(x))
```

$$x \mapsto \cos(x)$$

$$\cos(x)$$

Často bývá potřeba výsledný symbolický výraz ještě zjednodušit. K tomu Sage poskytuje několik funkcí.

```
f(x) = arctan(x) + arctan(1/x)
expr = diff(f(x))
# po derivaci
show(expr)
# zjednoduseni
show(expr.simplify_full())
```

$$\frac{1}{x^2+1} - \frac{1}{x^2\left(\frac{1}{x^2}+1\right)}$$

$$0$$

# Integrace

Opět definujme funkci $f$ s nezávislou proměnnou $x$.

```
var('x')
f(x) = sin(x)
```

Primitivní funkci můžeme spočítat následujícím příkazem.

```
F(x) = integrate(f(x),x)
show(F(x))
```

$$-\cos(x)$$

Projistotu si tvrzení Sage ověřme. Musí platit $F' = f$.

```
(f(x) - diff(F(x))).simplify_full()
```

```
0
```

Určitý integrál vypočteme stejným příkazem a udáním integračního oboru (resp. mezí). Snadno si tento výsledek můžeme ověřit pomocí výše napočtené primitivní funkce.

```
print(integrate(f(x),(x,0,pi)))
print(F(pi) - F(0))
```

```
2
```

```
2
```

Ihned ale dodejme, že primitivní funkci k řadě funkcí nelze vyjádřit pomocí elementárních funkcí. Například:

```
f(x) = exp(-x^2)
show(f(x))
```

$$e^{\left(-x^2\right)}$$

```
F(x) = integrate(f(x),x)
show(F(x))
```

$$\frac{1}{2}\,\sqrt{\pi}\mathrm{erf}(x)$$

Sage vrací výsledek vyjádřený pomocí jisté speciální funkce (erf, viz BI-PST). Tato funkce je definována předpisem

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,\mathrm{d}t.$$

Její hodnotu můžeme počítat přímo i pomocí numerické integrace.

```
erf(2.0)
```

```
0.995322265018953
```

```
numerical_integral(2/sqrt(pi)*exp(-x^2),0,2.0)
```

```
(0.9953222650189529, 1.1050296955461036e-14)
```

Funkce numerical_integral vrací přibližnou hodnotu integrálu i odhad chyby výpočtu. Pro praktické výpočty je odhad chyby velmi důležitý.