uvod_do_sage

October 8, 2017

1 Jemný úvod do Sage

Tomáš Kalvoda, 2014, 2017

1.1 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

```
sage --notebook=jupyter
```

lze však spustit interaktivnější rozhranní k Sage využívající Jupyter 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://cocalc.com (dříve SageMathCloud). Tato služba aktuálně nabízí bezplatný účet pro základní použití. 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 nebo IPython/Jupyter 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.

1.2 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/, nebo http://naucse.python.cz/).

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

```
In [1]: 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.

```
In [2]: 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* podmínky.

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

```
Ano!
```

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

```
In [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.

```
In [5]: 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.

```
In [6]: show(f(4))
        show(f(pi))
        show(f('Ahoj!'))
```

```
16
```

```
4*pi
```

```
'Ahoj!Ahoj!Ahoj!Ahoj!'
```

Pyhon je objektový jazyk. Všechna prostředí podporující Sage 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.

```
In [7]: # malé celé číslo
        a = 9
```

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

```
In [8]: 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 stisknoutí `TAB` klávesy za otevírací závorkou. Případně lze nápovědu vypsat pomocí otazníku za názvem funkce.

```
In [9]: factorial?
```

1.3 Číselné množiny

1.3.1 Celá čísla

```
In [10]: print ZZ
         show(ZZ)
```

```
Integer Ring


Integer Ring
```

```
In [11]: # ekvivalentní způsoby vytvoření obektu typy Sage Integer
         j = 1
         k = ZZ(1)
         l = Integer(1)
         j == k
         k == l
         type(j)
```

```
Out[11]: <type 'sage.rings.integer.Integer'>
```

1.3.2 Racionální čísla

```
In [12]: print QQ
         show(QQ)
```

```
Rational Field


Rational Field
```

```
In [13]: j = QQ(1.5)
         print j
```

```
3/2
```

```
In [14]: t = 55/12
         type(t)
```

```
Out[14]: <type 'sage.rings.rational.Rational'>
```

```
In [15]: j + k
```

```
Out[15]: 5/2
```

1.3.3 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 čísly pracuje zcela jiným způsobem.

```
In [16]: print RR
         show(RR)
```

```
Real Field with 53 bits of precision


Real Field with 53 bits of precision
```

```
In [17]: F = RealField(prec=10)
```

```
In [18]: x = RR(-1)
         type(x)
```

```
Out[18]: <type 'sage.rings.real_mpfr.RealNumber'>
```

```
In [19]: # odmocnina definována stejně jako v Mathematica
         y = x ^ (1/3)
         print y
         type(y)
```

```
0.500000000000000 + 0.866025403784439*I
```

```
Out[19]: <type 'sage.rings.complex_number.ComplexNumber'>
```

```
In [20]: RealField(100)
```

```
Out[20]: Real Field with 100 bits of precision
```

Eulerovo číslo.

```
In [21]: show(e)
         type(e)
```

```
e
```

```
Out[21]: <type 'sage.symbolic.constants_c.E'>
```

```
In [22]: print N(e,digits=10)
         print N(e,prec=32)
```

```
2.718281828
2.71828183
```

Ludolfovo číslo.

```
In [23]: show(pi)
         type(pi)
```

```
pi
```

```
Out[23]: <type 'sage.symbolic.expression.Expression'>
```

```
In [24]: print N(pi,digits=10)
         print N(pi,prec=32)
```

```
3.141592654
3.14159265
```

1.3.4 Komplexní čísla

Imaginární jednotka i.

```
In [25]: I
         type(I)
```

```
Out[25]: <type 'sage.symbolic.expression.Expression'>
```

```
In [26]: I^2
```

```
Out[26]: -1
```

1.3.5 Symbolický okruh

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

```
In [27]: SR
```

```
Out[27]: Symbolic Ring
```

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

```
In [28]: a = sqrt(2)
         b = log(pi)/sqrt(8)
         show(a*b)
```

```
1/2*log(pi)
```

1.4 Algebra a symbolické výrazy

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

```
In [29]: var('x')
```

```
Out[29]: x
```

Poté můžeme vytvořit výraz, který nás zajímá.

```
In [30]: expr = x^2 + sin(x) / (x^2 + 1)
         show(expr)
```

```
x^2 + sin(x)/(x^2 + 1)
```

```
In [31]: # jakého typu je tento objekt?
         type(expr)
```

```
Out[31]: <type 'sage.symbolic.expression.Expression'>
```

Dosazování.

```
In [33]: # pomocí rovnosti
         show(expr(x = pi/2))
         # pomocí slovniku (substituce)
         show(expr({x:pi/2}))
```

```
1/4*pi^2 + 4/(pi^2 + 4)
```

```
1/4*pi^2 + 4/(pi^2 + 4)
```

Algebraické úpravy.

```
In [34]: expr = (x+4)^5
         show(expr)
```

```
(x + 4)^5
```

Roznásobení.

```
In [35]: 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.

```
In [36]: expr = expr.factor()
         show(expr)
```

```
(x + 4)^5
```

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

```
In [37]: expr = sin(4*x)
         show(expr)
```

```
sin(4*x)
```

```
In [38]: expr = expr.trig_expand()
         show(expr)
```

```
4*cos(x)^3*sin(x) - 4*cos(x)*sin(x)^3
```

```
In [39]: expr = expr.trig_reduce()
         show(expr)
```

```
sin(4*x)
```

1.5 Sumace a Řady

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

```
In [40]: var('k,n,x')
```

```
Out[40]: (k, n, x)
```

Známý součet prvních n přirozených čísel.

```
In [41]: expr = sum(k, k, 1, n)
         show(expr)
```

```
1/2*n^2 + 1/2*n
```

Součet prvních n členů jisté geometrické posloupnosti.

```
In [42]: expr = sum(x^k, k, 0, n-1)
         show(expr)
```

```
(x^n - 1)/(x - 1)
```

Obskurnější součet.

```
In [43]: expr = sum(k^2, k, 1, n)
         show(expr)
```

```
1/3*n^3 + 1/2*n^2 + 1/6*n
```

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

```
In [44]: sum(x^k / factorial(k), k, 0, infinity)
```

```
Out[44]: e^x
```

```
In [45]: sum((-1)^k*x^(k+1)/(k+1),k, 0, infinity)
```

```
Out[45]: log(x + 1)
```

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

```
In [46]: taylor(sin(x), x, 0, 10)
```

```
Out[46]: 1/362880*x^9 - 1/5040*x^7 + 1/120*x^5 - 1/6*x^3 + x
```

```
In [47]: taylor(exp(x), x, 0, 10)
```

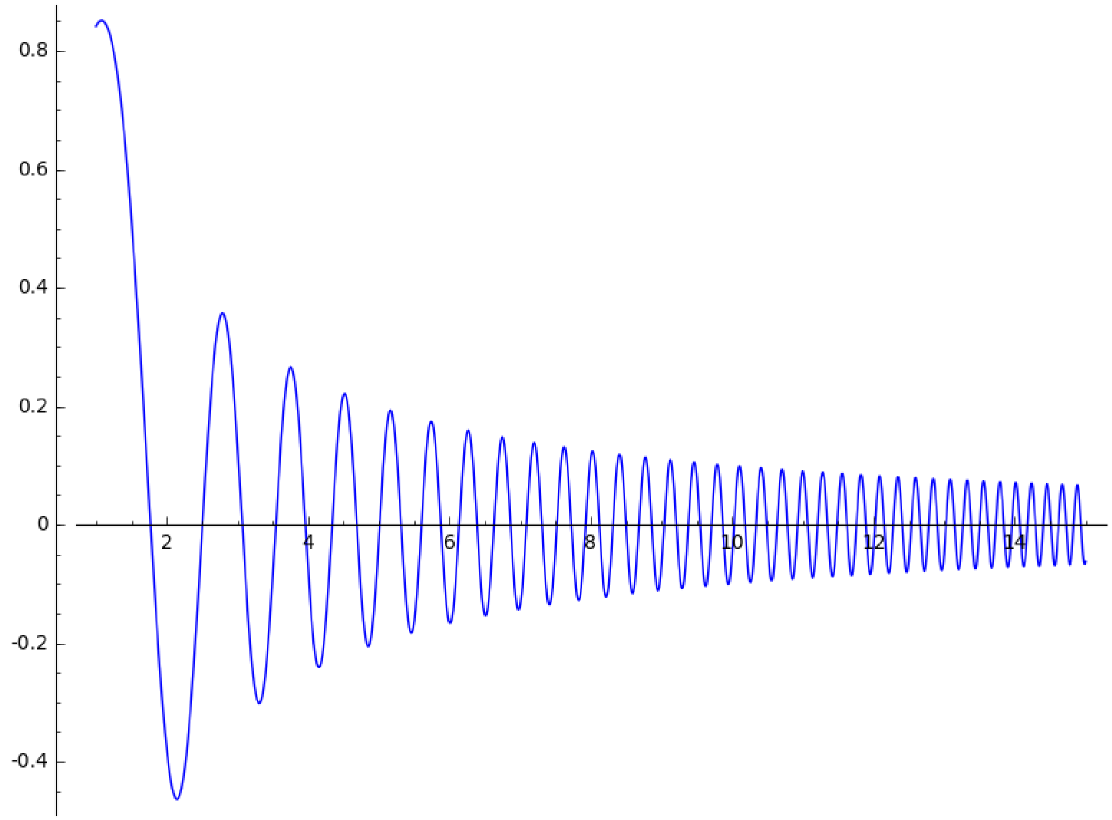
```
Out[47]: 1/3628800*x^10 + 1/362880*x^9 + 1/40320*x^8 + 1/5040*x^7 + 1/720*x^6 + 1/1
```

1.6 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.

```
In [48]: # 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))
```

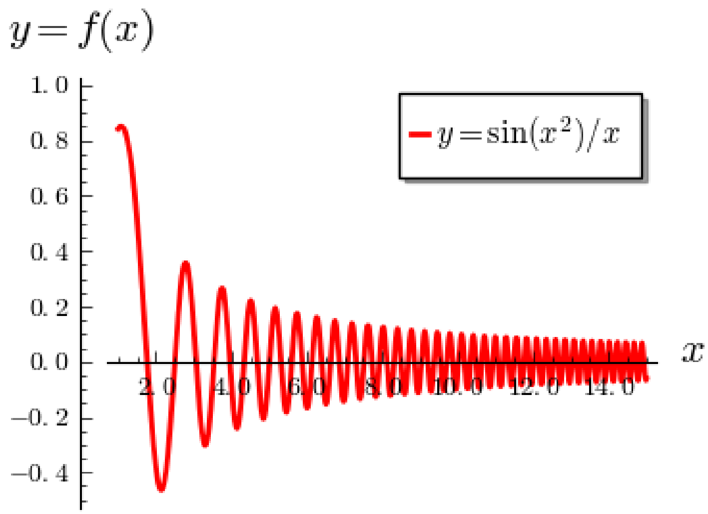
```
Out[48]:
```



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.

```
In [49]: plot(1/x*sin(x^2),(x,1,15),
               ymin=-0.5, ymax=1,                     # rozsah svislé osy
               thickness=2,                           # tloušt'ka křivky
               rgbcolor='red',                        # barva
               axes_labels=['$x$','$y=f(x)$'],        # popisky os, lze využívat LaTeX
               tick_formatter='latex',                # cejchování os stejným fontem jak
               legend_label='$y = \\sin(x^2)/x$',     # legenda (vhodné při kombinování
               figsize=4)                             # velikost výsledného obrázku
```

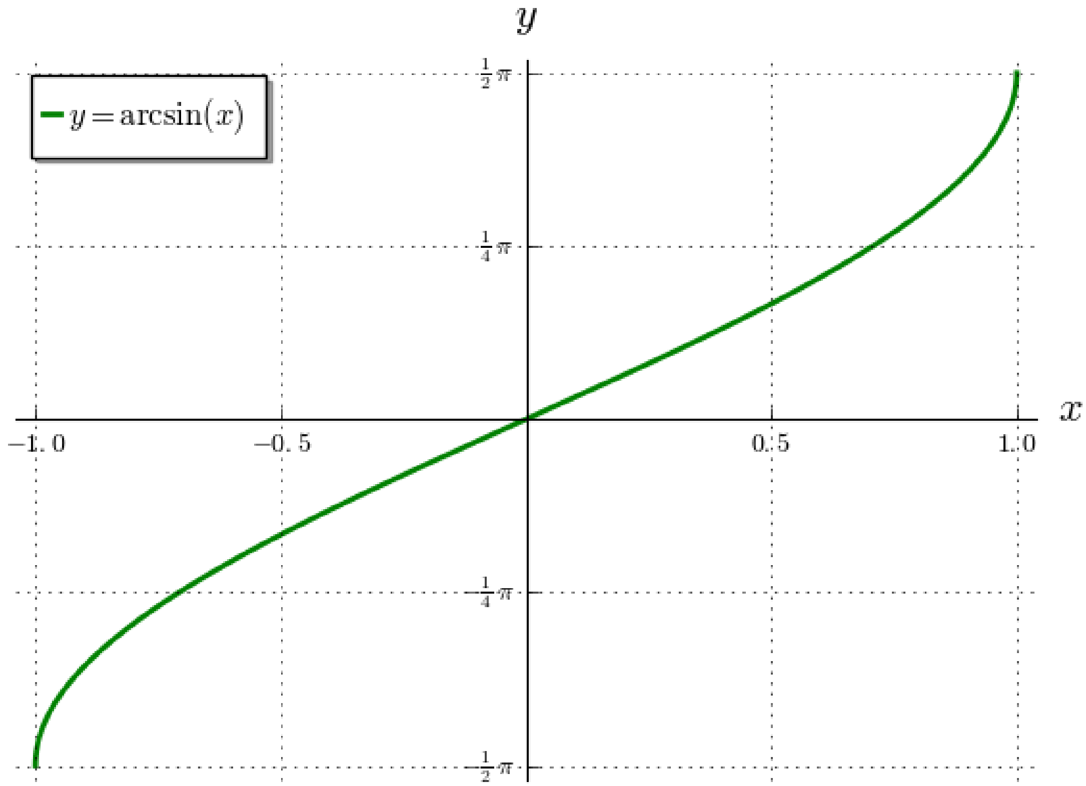
Out[49]:



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

```
In [50]: 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)
```

```
Out[50]:
```



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

```
In [51]: def lambert(z):
             """
                 Naivní implementace Lambertovy funkce, tedy inverze k g(w) = w*exp
             """

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

             # 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
```

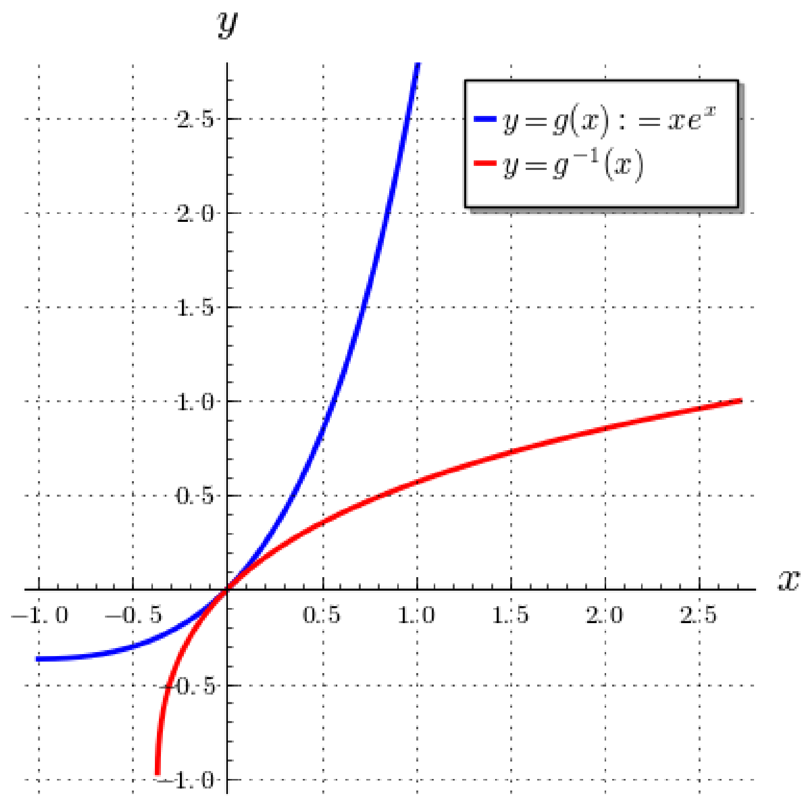
A nakonec graf s oběma funkcemi. Zde také ukazujeme, jak kombinovat více grafických objektů do jednoho. K tomu slouží operáotor "+". Různa nastavení grafiky (osy, velikost obrázku, atp.) stačí uvést jednou v prvním grafickém objektu.

```
In [52]: 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
```

```
Out[52]:
```



1.7 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.

```
In [54]: var('n,x')
         assume(n,'integer')
```

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

```
In [55]: limit(sin(pi*n),n=+infinity)

Out[55]: 0
```

Pro každé celoříselné n totiž platí $\sin(n\pi) = 0$. Sage nám proto dal dobrý výsledek pro limitu posloupnosti $\left(\sin(n\pi)\right)_{n=1}^{\infty}$.

```
In [57]: limit(sin(pi*x), x=+infinity)

Out[57]: 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.

```
In [59]: limit((1+1/n)^n, n=infinity)
```

```
Out[59]: e
```

```
In [61]: limit((e^x - 1)/x, x=0)
```

```
Out[61]: 1
```

```
In [62]: limit(ln(x+1)/x, x=0)
```

```
Out[62]: 1
```

```
In [63]: limit(ln(x+1)/x, x=0)
```

```
Out[63]: 1
```

Pomocí nepovinného argumentu `dir` (*direction*, směr) můžeme kontrolovat i to, zda-li počítáme limitu zleva či zprava.

```
In [64]: limit(1/x, x=0, dir='right')
```

```
Out[64]: +Infinity
```

```
In [66]: limit(1/x, x=0, dir='left')
```

```
Out[66]: -Infinity
```

```
In [67]: limit(sign(x), x=0, dir='right')
```

```
Out[67]: 1
```

```
In [68]: limit(sign(x), x=0, dir='left')
```

```
Out[68]: -1
```

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

```
In [69]: limit(1/x, x=0)
```

```
Out[69]: Infinity
```

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

1.8 Derivace

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

```
In [70]: var('x')
```

```
Out[70]: x
```

Dále definujeme funkci, kterou chceme derivovat.

```
In [71]: f(x) = sin(x)
```

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

```
In [72]: show(f)
         show(f(pi/2))
```

```
x |--> sin(x)


1
```

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

```
In [73]: g = f.derivative()
         show(g)
         show(g(x))
```

```
x |--> cos(x)


cos(x)
```

Druhou možností je použití funkce `diff`.

```
In [74]: g = diff(f)
         show(g)
         show(g(x))
```

```
x |--> cos(x)


cos(x)
```

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

```
In [75]: f(x) = arctan(x) + arctan(1/x)
         expr = diff(f(x))
         # po derivaci
         show(expr)
         # zjednoduseni
         show(expr.simplify_full())
```

```
1/(x^2 + 1) - 1/(x^2*(1/x^2 + 1))


0
```

1.9 Integrace

Opět definujme funkci f s nezávislou proměnnou x.

```
In [76]: var('x')
         f(x) = sin(x)
```

Primitivní funkci můžeme spočítat následujícím příkazem.

```
In [77]: F(x) = integrate(f(x),x)
         show(F(x))
```

```
-cos(x)
```

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

```
In [78]: (f(x) - diff(F(x))).simplify_full()
```

```
Out[78]: 0
```

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

```
In [79]: 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:

```
In [80]: f(x) = exp(-x^2)
         show(f(x))
```

```
e^(-x^2)
```

```
In [81]: F(x) = integrate(f(x),x)
         show(F(x))
```

```
1/2*sqrt(pi)*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.

```
In [82]: erf(2.0)
```

```
Out[82]: 0.995322265018953
```

```
In [83]: numerical_integral(2/sqrt(pi)*exp(-x^2),0,2.0)
```

```
Out[83]: (0.9953222650189529, 1.1050296955461036e-14)
```

```
In [ ]:
```