Elementary maths

This is a collection of basic mathematical computations using `sympy`. The main purpose is to demonstrate the use of `\py` and `\py*`. Note that `sympy 1.1.1` appears unable to simplify $\tanh(\log(x))$ (compare `rhs.108` shown below against `ans.108` shown in the Mathematica examples). Note also the separate computaions for the left and right hand sides of results 108, 109 and 110.

```
from sympy import *
x, y, z, a, b, c = symbols('x y z a b c')
ans = expand((a+b)**3)                                  # py (ans.101,ans)
ans = factor(-2*x+2*x+a*x-x**2+a*x**2-x**3)              # py (ans.102,ans)
ans = solve(x**2-4, x)                                  # py (ans.103,ans)
ans = solve([2*a-b - 3, a+b+c - 1,-b+c - 6],[a,b,c])    # py (ans.104,ans)
ans = N(pi,50)                                          # py (ans.105,ans)
ans = apart(1/((1 + x)*(5 + x)))                        # py (ans.106,ans)
ans = together((1/(1 + x) - 1/(5 + x))/4)               # py (ans.107,ans)
ans = simplify(tanh(log(x)))                            # py (rhs.108,ans)
ans = simplify(tanh(I*x))                               # py (rhs.109,ans)
ans = simplify(sinh(3*x) - 3*sinh(x) - 4*(sinh(x))**3)  # py (rhs.110,ans)
ans = tanh(log(x))                                      # py (lhs.108,ans)
ans = tanh(UnevaluatedExpr(I*x))                        # py (lhs.109,ans)
ans = sinh(3*x) - 3*sinh(x) - 4*(sinh(x))**3            # py (lhs.110,ans)
```

```
\begin{align*}
   &\py*{ans.101}\\
   &\py*{ans.102}\\
   &\py*{ans.103}\\
   &\py*{ans.104}\\
   &\py*{ans.105}\\
   &\py*{ans.106}\\
   &\py*{ans.107}\\
   \py{lhs.108} &= \Py{rhs.108}\\
   \py{lhs.109} &= \Py{rhs.109}\\
   \py{lhs.110} &= \Py{rhs.110}
\end{align*}
```

$$\texttt{ans.101} := a^3 + 3a^2b + 3ab^2 + b^3$$

$$\texttt{ans.102} := -x\left(-a + x\right)\left(x + 1\right)$$

$$\texttt{ans.103} := \left[-2,\ 2\right]$$

$$\texttt{ans.104} := \left\{a : \frac{1}{5},\ b : -\frac{13}{5},\ c : \frac{17}{5}\right\}$$

$$\texttt{ans.105} := 3.14159265358979323846264338327950288419716939937510$$

$$\texttt{ans.106} := -\frac{1}{4\left(x + 5\right)} + \frac{1}{4\left(x + 1\right)}$$

$$\texttt{ans.107} := \frac{1}{\left(x + 1\right)\left(x + 5\right)}$$

$$\tanh\left(\log\left(x\right)\right) = \tanh\left(\log\left(x\right)\right) \qquad (\texttt{rhs.108})$$

$$\tanh\left(ix\right) = i\tan\left(x\right) \qquad (\texttt{rhs.109})$$

$$-4\sinh^3\left(x\right) - 3\sinh\left(x\right) + \sinh\left(3x\right) = 0 \qquad (\texttt{rhs.110})$$

Linear Algebra

```
from sympy import linsolve
lamda = Symbol('lamda')
mat   = Matrix([[2,3], [5,4]])                    # py (ans.201,mat)
eig1  = mat.eigenvects()[0][0]                    # 1st eigenvalue
eig2  = mat.eigenvects()[1][0]                    # 2nd eigenvalue
v1    = mat.eigenvects()[0][2][0]                 # 1st eigenvector
v2    = mat.eigenvects()[1][2][0]                 # 2nd eigenvector
eig   = simplify(Matrix([eig1,eig2]))             # py (ans.202,eig)
vec   = simplify(5*Matrix([]).col_insert(0,v1)
                             .col_insert(1,v2))   # py (ans.203,vec)
det   = expand((mat - lamda * eye(2)).det())      # py (ans.204,det)
rhs   = Matrix([[3],[7]])                         # py (ans.205,rhs)
ans   = list(linsolve((mat,rhs),x,y))[0]          # py (ans.206,ans)
```

```
\begin{align*}
   &\py*{ans.201}\\
   &\py*{ans.202}\\
   &\py*{ans.203}\\
   &\py*{ans.204}\\
   &\py*{ans.205}\\
   &\py*{ans.206}
\end{align*}
```

$$\begin{aligned}
\texttt{ans.201} &:= \begin{bmatrix} 2 & 3 \\ 5 & 4 \end{bmatrix} \\
\texttt{ans.202} &:= \begin{bmatrix} -1 \\ 7 \end{bmatrix} \\
\texttt{ans.203} &:= \begin{bmatrix} -5 & 3 \\ 5 & 5 \end{bmatrix} \\
\texttt{ans.204} &:= \lambda^2 - 6\lambda - 7 \\
\texttt{ans.205} &:= \begin{bmatrix} 3 \\ 7 \end{bmatrix} \\
\texttt{ans.206} &:= \left(\frac{9}{7}, \ \frac{1}{7} \right)
\end{aligned}$$

Limits

```
n, dx = symbols('n dx')
ans = limit(sin(4*x)/x,x,0)                    # py (ans.301,ans)
ans = limit(2**x/x,x,oo)                       # py (ans.302,ans)
ans = limit(((x+dx)**2 - x**2)/dx, dx,0)       # py (ans.303,ans)
ans = limit((4*n + 1)/(3*n - 1),n,oo)          # py (ans.304,ans)
ans = limit((1+(a/n))**n,n,oo)                 # py (ans.305,ans)
```

```
\begin{align*}
   &\py*{ans.301}\\
   &\py*{ans.302}\\
   &\py*{ans.303}\\
   &\py*{ans.304}\\
   &\py*{ans.305}
\end{align*}
```

$$\begin{aligned}
\texttt{ans.301} &:= 4 \\
\texttt{ans.302} &:= \infty \\
\texttt{ans.303} &:= 2x \\
\texttt{ans.304} &:= \frac{4}{3} \\
\texttt{ans.305} &:= e^{a}
\end{aligned}$$

Series

```
ans = series((1 + x)**(-2), x, 1, 6)           # py (ans.401,ans)
ans = series(exp(x), x, 0, 6)                  # py (ans.402,ans)
ans = Sum(1/n**2, (n,1,50)).doit()             # py (ans.403,ans)
ans = Sum(1/n**4, (n,1,oo)).doit()             # py (ans.404,ans)
```

```
\begin{align*}
   &\py*{ans.401}\\
   &\py*{ans.402}\\
   &\py*{ans.403}\\
   &\py*{ans.404}
\end{align*}
```

$$\begin{aligned}
\texttt{ans.401} &:= \frac{1}{2} + \frac{3}{16}(x-1)^2 - \frac{1}{8}(x-1)^3 + \frac{5}{64}(x-1)^4 - \frac{3}{64}(x-1)^5 - \frac{x}{4} + O\left((x-1)^6; x \to 1\right) \\
\texttt{ans.402} &:= 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + O\left(x^6\right) \\
\texttt{ans.403} &:= \frac{312157992955169267846963566083562620966 1709}{192081536785946309960051152615192956019 2000} \\
\texttt{ans.404} &:= \frac{\pi^4}{90}
\end{aligned}$$

Calculus

This example shows how `\Py` can be used to set the equation tag on the far right hand side.

```
ans = diff(x*sin(x),x)                                  # py (ans.501,ans)
ans = diff(x*sin(x),x).subs(x,pi/4)                     # py (ans.502,ans)
ans = integrate(2*sin(x)**2, (x,a,b))                   # py (ans.503,ans)
ans = Integral(2*exp(-x**2), (x,0,oo))                  # py (lhs.504,ans)
ans = ans.doit()                                        # py (ans.504,ans)
ans = Integral(Integral(x**2 + y**2, (y,0,x)), (x,0,1)) # py (lhs.505,ans)
ans = ans.doit()                                        # py (ans.505,ans)
```

```
\begin{align*}
   &\py*{ans.501}\\
   &\py*{ans.502}\\
   &\py*{ans.503}\\
    \py{lhs.504}&=\Py{ans.504}\\
    \py{lhs.505}&=\Py{ans.505}
\end{align*}
```

$$\texttt{ans.501} := x\cos\left(x\right) + \sin\left(x\right)$$

$$\texttt{ans.502} := \frac{\pi}{8}\sqrt{2} + \frac{\sqrt{2}}{2}$$

$$\texttt{ans.503} := -a + b + \sin\left(a\right)\cos\left(a\right) - \sin\left(b\right)\cos\left(b\right)$$

$$\int_{0}^{\infty} 2e^{-x^2}\,dx = \sqrt{\pi} \tag{ans.504}$$

$$\int_{0}^{1}\int_{0}^{x} \left(x^2 + y^2\right)\,dy\,dx = \frac{1}{3} \tag{ans.505}$$

Differential equations

```
y = Function('y')
C1, C2 = symbols('C1 C2')

ode = Eq(y(x).diff(x) + y(x), 2*a*sin(x))
sol = expand(dsolve(ode,y(x)).rhs)                                  # py (ans.601,sol)
cst = solve([sol.subs(x,0)],dict=True)
sol = sol.subs(cst[0])                                              # py (ans.602,sol)

ode = Eq(y(x).diff(x,2) + y(x), 0)
sol = expand(dsolve(ode,y(x)).rhs)                                  # py (ans.603,sol)
cst = solve([sol.subs(x,0),sol.diff(x).subs(x,0)-1],dict=True)
sol = sol.subs(cst[0])                                              # py (ans.604,sol)

ode = Eq(y(x).diff(x,2) + 5*y(x).diff(x) - 6*y(x), 0)
sol = expand(dsolve(ode,y(x)).rhs)                                  # py (ans.605,sol)
sol = sol.subs({C1:2,C2:3})                                         # py (ans.606,sol)
```

```
\begin{align*}
   &\py*{ans.601}\\
   &\py*{ans.602}\\
   &\py*{ans.603}\\
   &\py*{ans.604}\\
   &\py*{ans.605}\\
   &\py*{ans.606}
\end{align*}
```

$$\texttt{ans.601} := C_1 e^{-x} + a\sin(x) - a\cos(x)$$
$$\texttt{ans.602} := a\sin(x) - a\cos(x) + ae^{-x}$$
$$\texttt{ans.603} := C_1\sin(x) + C_2\cos(x)$$
$$\texttt{ans.604} := \sin(x)$$
$$\texttt{ans.605} := C_1 e^{-6x} + C_2 e^{x}$$
$$\texttt{ans.606} := 3e^{x} + 2e^{-6x}$$

A table of derivatives and anti-derivatives

This example is based upon a nice example in the Pythontex gallery, see https://github.com/gpoore/pythontex/. It uses a tagged block to capture the `Sympy` output for later use in the body of the LaTeX table.

```
from sympy import *

var('x')

# Create a list of functions to include in the table
funcs = [['sin(x)',r'\\'],       ['cos(x)',r'\\'],       ['tan(x)',r'\\'],
         ['asin(x)',r'\\[5pt]'], ['acos(x)',r'\\[5pt]'], ['atan(x)',r'\\[5pt]'],
         ['sinh(x)',r'\\'],      ['cosh(x)',r'\\'],      ['tanh(x)',r' ']]

# pyBeg (CalculusTable)
for func, eol in funcs:
    myddx = 'Derivative(' + func + ', x)'
    myint = 'Integral(' + func + ', x)'
    print(latex(eval(myddx)) + '&=' + latex(eval(myddx + '.doit()')) + r'\quad & \quad')
    print(latex(eval(myint)) + '&=' + latex(eval(myint + '.doit()')) + eol)
# pyEnd (CalculusTable)
```

```
\begin{align*}
   \py {CalculusTable}
\end{align*}
```

$$\frac{d}{dx}\sin(x) = \cos(x)$$
$$\frac{d}{dx}\cos(x) = -\sin(x)$$
$$\frac{d}{dx}\tan(x) = \tan^2(x) + 1$$
$$\frac{d}{dx}\operatorname{asin}(x) = \frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\operatorname{acos}(x) = -\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\operatorname{atan}(x) = \frac{1}{x^2+1}$$
$$\frac{d}{dx}\sinh(x) = \cosh(x)$$
$$\frac{d}{dx}\cosh(x) = \sinh(x)$$
$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$

$$\int \sin(x)\,dx = -\cos(x)$$
$$\int \cos(x)\,dx = \sin(x)$$
$$\int \tan(x)\,dx = -\log(\cos(x))$$
$$\int \operatorname{asin}(x)\,dx = x\operatorname{asin}(x) + \sqrt{1-x^2}$$
$$\int \operatorname{acos}(x)\,dx = x\operatorname{acos}(x) - \sqrt{1-x^2}$$
$$\int \operatorname{atan}(x)\,dx = x\operatorname{atan}(x) - \frac{\log(x^2+1)}{2}$$
$$\int \sinh(x)\,dx = \cosh(x)$$
$$\int \cosh(x)\,dx = \sinh(x)$$
$$\int \tanh(x)\,dx = x - \log(\tanh(x)+1)$$

Step-by-step integration

This is another nice example drawn from the Pythontex gallery, see https://github.com/gpoore/pythontex.

It shows the step-by-step computations of a simple triple integral.

```
from sympy import *

x, y, z = symbols('x,y,z')
f = Symbol('f(x,y,z)')

# Define limits of integration
x_max = 2;   y_max = 3;   z_max = 4;
x_min = 0;   y_min = 0;   z_min = 0;

lhs = Integral(f, (x, x_min, x_max),
                  (y, y_min, y_max),
                  (z, z_min, z_max))                    # py(lhs.01,lhs)

f = x*y + y*sin(z) + cos(x+y)

rhs = Integral(f, (x, x_min, x_max),
                  (y, y_min, y_max),
                  (z, z_min, z_max))                    # py(rhs.01,rhs)
rhs = Integral(Integral(f, (x, x_min, x_max)).doit(),
                           (y, y_min, y_max),
                           (z, z_min, z_max))           # py(rhs.02,rhs)
rhs = Integral(Integral(f, (x, x_min, x_max),
                           (y, y_min, y_max)).doit(),
                           (z, z_min, z_max))           # py(rhs.03,rhs)
rhs = Integral(f, (x, x_min, x_max),
                  (y, y_min, y_max),
                  (z, z_min, z_max)).doit()             # py(rhs.04,rhs)

# And now, a numerical approximation
rhs = N(rhs)                                            # py(rhs.05,rhs)
```

$$\begin{aligned}
\int_0^4 \int_0^3 \int_0^2 f(x,y,z)\,dx\,dy\,dz &= \int_0^4 \int_0^3 \int_0^2 \left(xy + y\sin\left(z\right) + \cos\left(x+y\right)\right)\,dx\,dy\,dz \\
&= \int_0^4 \int_0^3 \left(2y\sin\left(z\right) + 2y - \sin\left(y\right) + \sin\left(y+2\right)\right)\,dy\,dz \\
&= \int_0^4 \left(9\sin\left(z\right) + \cos\left(3\right) + \cos\left(2\right) - \cos\left(5\right) + 8\right)\,dz \\
&= 4\cos\left(3\right) + 4\cos\left(2\right) - 4\cos\left(5\right) - 9\cos\left(4\right) + 41 \\
&\approx 40.1235865133293
\end{aligned}$$

```
\begin{align*}
   \py{lhs.01} &= \py{rhs.01}\\
               &= \py{rhs.02}\\
               &= \py{rhs.03}\\
               &= \py{rhs.04}\\[8pt]
               &\approx \py{rhs.05}
\end{align*}
```

Plotting Bessel functions

This simple example uses `numpy`, `scipy` and `Matplotlib` to produce a plot of the first six Bessel functions. Two plots are shown, one created by `Matplotlib` and a second created by LaTeX using the plotting package `pgfplots` and the data exported from `Matplotlib`.

If you are using macOS, you may need to use the `-Ppythonw` option when running `pylatex.sh`. This is a known problem with macOS and `Matplotlib`, see https://matplotlib.org/faq/osx_framework.html.

```
import numpy as np
import scipy.special as sp
import matplotlib.pyplot as plt

plt.matplotlib.rc('text', usetex = True)
plt.matplotlib.rc('grid', linestyle = 'dotted')
plt.matplotlib.rc('figure', figsize = (6.4,4.8)) # (width,height) inches

x = np.linspace(0, 15, 500)

for v in range(0, 6):
    plt.plot(x, sp.jv(v, x))

plt.xlim((0, 15))
plt.ylim((-0.5, 1.1))
plt.legend(('${J}_0(x)$', '${J}_1(x)$', '${J}_2(x)$',
            '${J}_3(x)$', '${J}_4(x)$', '${J}_5(x)$'), loc = 0)
plt.xlabel('$x$')
plt.ylabel('${J}_n(x)$')
plt.grid(True)
plt.tight_layout(pad=0.5)

plt.savefig('example-04-fig.pdf')

# save the data for later use by pgfplots
np.savetxt('example-04.txt',list(zip(x,sp.jv(0,x),sp.jv(1,x),sp.jv(2,x),
                                     sp.jv(3,x),sp.jv(4,x),sp.jv(5,x))),
                            fmt="% .10e")
```

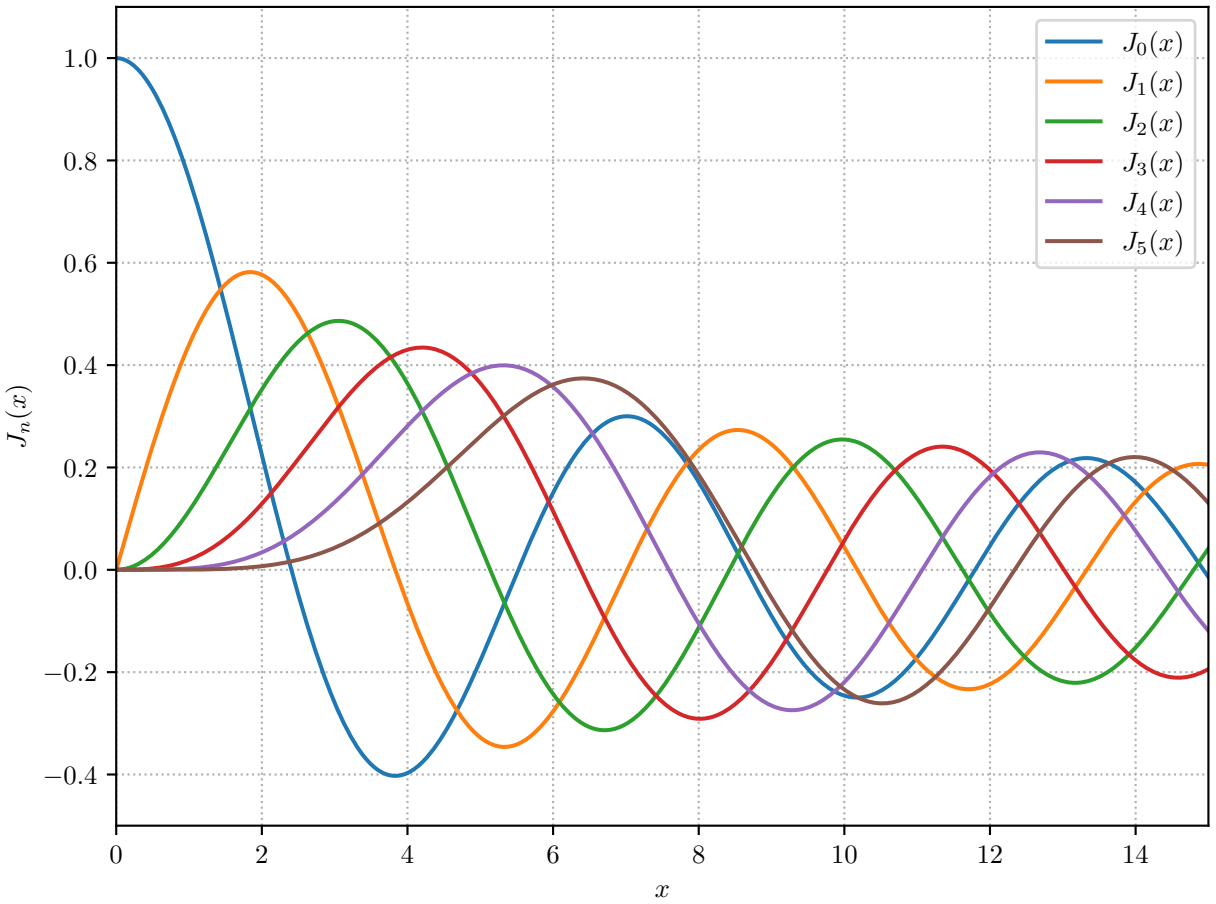

Figure 1: The first six Bessel functions.

```
\IfFileExists{example-04-fig.pdf}%
{\includegraphics[width=6.4in]
   {example-04-fig.pdf}}{Failed to create pdf plot.}
\captionof{figure}{The first six Bessel functions.}
```

Using pgfplots

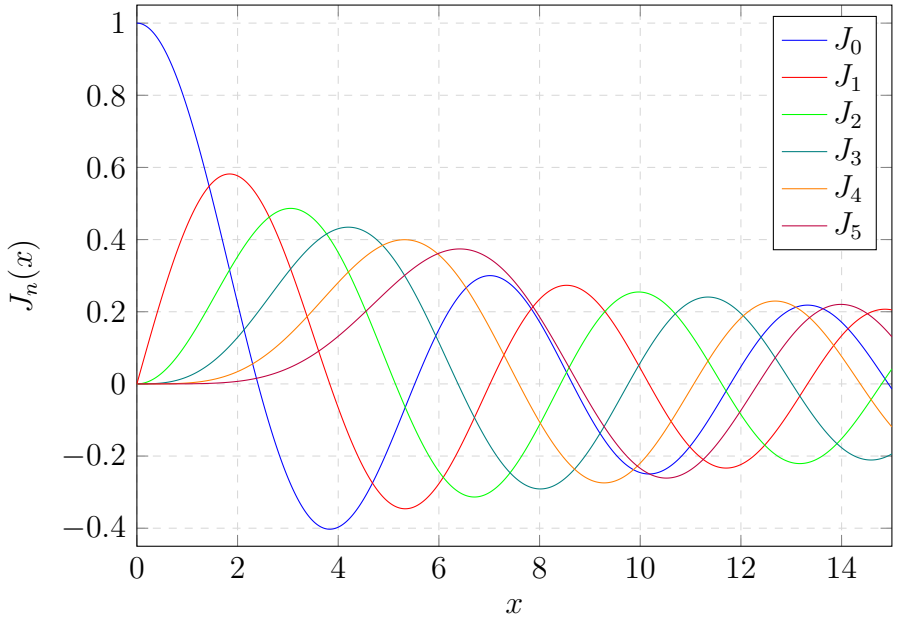

Figure 2: The first six Bessel functions.

```
\begin{tikzpicture} % requires \usepackage{pgfplots}
   \begin{axis}
      [xmin= 0.0,  xmax=15.0,
       ymin=-0.45, ymax=1.05,
       xlabel=$x$, ylabel=$J_n(x)$,
       grid=major, grid style={dashed,gray!30},
       legend entries = {$J_0$, $J_1$, $J_2$, $J_3$, $J_4$, $J_5$}]
       \addplot[blue]   table [x index=0, y index=1]{example-04.txt};
       \addplot[red]    table [x index=0, y index=2]{example-04.txt};
       \addplot[green]  table [x index=0, y index=3]{example-04.txt};
       \addplot[teal]   table [x index=0, y index=4]{example-04.txt};
       \addplot[orange] table [x index=0, y index=5]{example-04.txt};
       \addplot[purple] table [x index=0, y index=6]{example-04.txt};
   \end{axis}
\end{tikzpicture}
\captionof{figure}{The first six Bessel functions.} % requires \usepackage{caption}
```

Displaying long expressions

This example uses a simple (though contrived) example of a Taylor series expansion of $1/(1+x)$ to demonstrate the problems that can arise when displaying very long expressions.

```
from sympy import *
x = Symbol('x')
ans    = 1/(1+x)                  # py (ans.511,ans)
taylor = ans.series(x, 0, 10)     # py (ans.512,taylor)
taylor = ans.series(x, 0, 20)     # py (ans.513,taylor)
taylor = ans.series(x, 0, 23)     # py (ans.514,taylor)
```

```
\begin{dgroup*}[spread={5pt}]
   \begin{dmath*} f(x) = \Py*{ans.511} \end{dmath*}
   \begin{dmath*}    {}= \Py*{ans.512} \end{dmath*}
   \begin{dmath*}    {}= \Py*{ans.513} \end{dmath*}
   \begin{dmath*}    {}= \Py*{ans.514} \end{dmath*}
   \begin{dmath*}    {}= \Py*[\hskip 2cm]{ans.514} \end{dmath*}
\end{dgroup*}
```

The first four lines of the following output were set using `\Py*` while the final line used `\Py*[\hskip=2cm]`. The last pair of lines displays the output for the same tag `ans.514` and clearly the formatting of the second last line is not ideal as the text has overlapped the tag. This was corrected in the final line by using the optional argument `[\hskip=2cm]` in the call to `\Py*`.

$$f(x) = \frac{1}{x+1} \tag{ans.511}$$

$$= 1 - x + x^2 - x^3 + x^4 - x^5 + x^6 - x^7 + x^8 - x^9 + O\left(x^{10}\right) \tag{ans.512}$$

$$= 1 - x + x^2 - x^3 + x^4 - x^5 + x^6 - x^7 + x^8 - x^9 + x^{10} - x^{11} + x^{12} - x^{13} + x^{14} - x^{15} + x^{16} - x^{17} + x^{18} - x^{19} + O\left(x^{20}\right) \tag{ans.513}$$

$$= 1 - x + x^2 - x^3 + x^4 - x^5 + x^6 - x^7 + x^8 - x^9 + x^{10} - x^{11} + x^{12} - x^{13} + x^{14} - x^{15} + x^{16} - x^{17} + x^{18} - x^{19} + x^{20} - x^{21} + x^{22} + O\left(x^{23}\right) \tag{ans.514}$$

$$= 1 - x + x^2 - x^3 + x^4 - x^5 + x^6 - x^7 + x^8 - x^9 + x^{10} - x^{11} + x^{12} - x^{13} + x^{14} - x^{15} + x^{16} - x^{17} + x^{18} - x^{19} + x^{20} - x^{21} + x^{22} + O\left(x^{23}\right) \tag{ans.514}$$

Quadratic convergence of Newton-Raphson iterations

This is a simple example that uses Python and `sympy` to demonstrate the quadratic converegnce of Newton-Raphson iterations to the exact root of a non-linear equation.

```
from sympy import *

x = Symbol('x')

f    = Lambda (x,x-exp(-x))
df   = Lambda (x,diff(f(x),x))
Step = Lambda (x,x-f(x)/df(x))

Digits = 200  # use 200 decimal digits for all numerical computations

x_new = Float('0.5',Digits)
f_new = N (f(x_new),Digits)

# pyBeg (table)

print ('\RuleA {:2d} & {: .25f} & {: .10e} &\\\\'.format(0,x_new,f_new))

for n in range (1,7):
   x_old = x_new
   x_new = N (Step(x_new),Digits)
   f_old = N (f(x_old),Digits)
   f_new = N (f(x_new),Digits)
   ratio = N (f_new / f_old**2,Digits)
   print ('\RuleA {:2d} & {: .25f} & {: .10e} & {: .5f}\\\\'.format(n,x_new,f_new,ratio))

# pyEnd (table)
```

Note the clear quadratic convergence in the iterations – the last column settles to approximately -0.11546 independent of the number of iterations. This behaviour would not be seen using normal floating point computations as they are normally limited to no more than 18 decimal digits. This computation used 200 decimal digits.

Newton-Raphson iterations $x_{n+1} = x_n - f_n/f'_n\ , \quad f(x) = x - e^{-x}$

n	x_n	$\epsilon_n = x_n - e^{-x_n}$	$\epsilon_n/\epsilon_{n-1}^2$
0	0.50000000000000000000000000	-1.0653065971e-1	
1	0.56631100319721815304164922	-1.3045098060e-3	-0.11495
2	0.56714316503486221278651210	-1.9648047172e-7	-0.11546
3	0.56714329040978102869957660	-4.4574262753e-15	-0.11546
4	0.56714329040978387299996870	-2.2941072910e-30	-0.11546
5	0.56714329040978387299996870	-6.0767705445e-61	-0.11546
6	0.56714329040978387299996870	-4.2637434326e-122	-0.11546

```
\def\RuleA{\vrule depth0pt  width0pt height14pt}
\def\RuleB{\vrule depth8pt  width0pt height14pt}
\def\RuleC{\vrule depth10pt width0pt height16pt}

\setlength{\tabcolsep}{0.025\textwidth}%

\begin{center}
   \begin{tabular}{cccc}%
      \noalign{\hrule height 1pt}
      \multicolumn{4}{c}{\RuleC\rmfamily\bfseries%
      Newton-Raphson iterations \quad%
      $x_{n+1} = x_n - f_n/f'_n\ ,\quad f(x) = x-e^{-x}$}\\
      \noalign{\hrule height 1pt}
      \RuleB$ n$&$ x_n$&$ \epsilon_{n} =  x_{n} - e^{-x_{n}}$&$\epsilon_{n}/\epsilon_{n-1}^2$\\
      \noalign{\hrule height 0.5pt}
      \py{table}
      \noalign{\hrule height 1pt}
   \end{tabular}
\end{center}
```

Using tagged blocks

The following Python code block contains a matched `pyBeg/pyEnd` pair, with the tag name `info`, to capture the output from the formatted Python `print` statements.

```
import platform, datetime
# pyBeg(info)
print("date :      &"+'{:%a %d %b %Y %H:%M:%S}'.format(datetime.datetime.now())+"\\\\")
print("python :    &"+str(platform.python_version())+"\\\\")
print("system :    &"+str(platform.system())+"\\\\")
print("release :   &"+str(platform.release())+"\\\\")
print("machine :   &"+str(platform.machine())+"\\\\")
print("processor : &"+str(platform.processor())+"\\\\")
print("platform :  &"+str(platform.platform()))
# pyEnd(info)
```

```
\bgroup\tt
\begin{tabular}{rl}
   \py{info}
\end{tabular}
\egroup
```

Here is the output caught from the above block.

```
      date :  Sat 12 Apr 2025 10:41:45
    python :  3.9.19
    system :  Darwin
   release :  24.3.0
   machine :  arm64
 processor :  arm
  platform :  macOS-15.3.2-arm64-arm-64bit
```

Visible Python code

This is the first of two-part example. In this first part the Python code and tags are made visible using `\PySetup{action=show}` (this is the default). In the second part the Python code will be hidden using `\PySetup{action=hide}`. In both cases the Python output is identical (apart from the different tag names).

```
from sympy import *

x, y = symbols('x y')

ans = limit((sin(x)-x)/x**3, x, 0)              # py(ans.101,ans)
ans = (1/cos(x)).series(x, 0, 6)                # py(ans.102,ans)
ans = diff(cos(x**2)**2 / (1+x), x)             # py(ans.103,ans)
ans = integrate(x**2 * cos(x), x)               # py(ans.104,ans)
ans = integrate(x**2 * cos(x), (x, 0, pi/2))    # py(ans.105,ans)

f = Function('f')

eqn = Eq(Derivative(f(x),x,x) + 9*f(x), 1)      # py(ans.106,eqn)
sol = dsolve(eqn, f(x))                         # py(ans.107,sol)

eqn = Eq(f(x).diff(x, x) + 9*f(x), 1)           # py(ans.108,eqn)
sol = dsolve(eqn, f(x))                         # py(ans.109,sol)

my_int = Integral(cos(x),(x,0,2))               # py(ans.110,my_int)
my_ans = my_int.doit()                          # py(ans.111,my_ans)
```

Visible Python code output

Here is the output from the above Python code. This line of text is here for no other reason than to fill out the line so that we have two lines of text before the Python output. This makes it easier to compare the vertical spacing between this and the next example.

$$\texttt{ans.101} := -\frac{1}{6}$$

$$\texttt{ans.102} := 1 + \frac{x^2}{2} + \frac{5}{24}x^4 + O\left(x^6\right)$$

$$\texttt{ans.103} := -\frac{4x}{x+1}\sin\left(x^2\right)\cos\left(x^2\right) - \frac{\cos^2\left(x^2\right)}{\left(x+1\right)^2}$$

$$\texttt{ans.104} := x^2\sin\left(x\right) + 2x\cos\left(x\right) - 2\sin\left(x\right)$$

$$\texttt{ans.105} := -2 + \frac{\pi^2}{4}$$

$$\texttt{ans.106} := 9f(x) + \frac{d^2}{dx^2}f(x) = 1$$

$$\texttt{ans.107} := f(x) = C_1\sin\left(3x\right) + C_2\cos\left(3x\right) + \frac{1}{9}$$

$$\texttt{ans.108} := 9f(x) + \frac{d^2}{dx^2}f(x) = 1$$

$$\texttt{ans.109} := f(x) = C_1\sin\left(3x\right) + C_2\cos\left(3x\right) + \frac{1}{9}$$

$$\texttt{ans.110} := \int_0^2 \cos\left(x\right)dx$$

$$\texttt{ans.111} := \sin\left(2\right)$$

Hidden Python code and output

In this example the Python code is hidden and consumes no vertical space on the page. Compare the gap between this pair of lines and the following output text against that seen on the previous page.

$$\texttt{ans.201} := -\frac{1}{6}$$

$$\texttt{ans.202} := 1 + \frac{x^2}{2} + \frac{5}{24}x^4 + O\left(x^6\right)$$

$$\texttt{ans.203} := -\frac{4x}{x+1}\sin\left(x^2\right)\cos\left(x^2\right) - \frac{\cos^2\left(x^2\right)}{(x+1)^2}$$

$$\texttt{ans.204} := x^2\sin(x) + 2x\cos(x) - 2\sin(x)$$

$$\texttt{ans.205} := -2 + \frac{\pi^2}{4}$$

$$\texttt{ans.206} := 9f(x) + \frac{d^2}{dx^2}f(x) = 1$$

$$\texttt{ans.207} := f(x) = C_1\sin(3x) + C_2\cos(3x) + \frac{1}{9}$$

$$\texttt{ans.208} := 9f(x) + \frac{d^2}{dx^2}f(x) = 1$$

$$\texttt{ans.209} := f(x) = C_1\sin(3x) + C_2\cos(3x) + \frac{1}{9}$$

$$\texttt{ans.210} := \int_0^2 \cos(x)\,dx$$

$$\texttt{ans.211} := \sin(2)$$

Visible Python markup

There is nothing special in this particular example. Its purpose is to provide a contrast with the following example where the tags have been hidden from view.

```
from sympy import *

x = Symbol('x')

ans = diff(cos(x)**2, x)                    # py(ans.301,ans)
ans = integrate(2*sin(x)*exp(-x), x)        # py(ans.302,ans)
```

$$\texttt{ans.301} := -2\sin(x)\cos(x)$$
$$\texttt{ans.302} := -e^{-x}\sin(x) - e^{-x}\cos(x)$$

Hidden Python markup

This example creates the impression that there are no tags in the following block. But looks can be deceptive, the truth is revealed on the following page.

```
from sympy import *

x = Symbol('x')

ans = diff(cos(x)**2, x)
ans = integrate(sin(x)*exp(-x), x)
```

$$\texttt{ans.401} := -2\sin(x)\cos(x)$$
$$\texttt{ans.402} := -e^{-x}\sin(x) - e^{-x}\cos(x)$$

How to hide the tags

The easiest way to hide the tags from view is to include two blocks, one containing the tags but hidden from view using `\PySetup{action=hide}`, the other block is a copy of the first but with all tags stripped out. That second block is made visible using `\PySetup{action=verbatim}`. This is a bit cumbersome and prone to errors (the second block must be a faithful copy of the first block). Better solutions could be devised but they all appear to require significant changes to the current preprocessors – so much so that the this cloning trick was deemed a reasonable compromise.

Here is the LaTeX code from the previous example.

```
\PySetup{action=hide}

\begin{python}

   from sympy import *

   x = Symbol('x')

   ans = diff(cos(x)**2, x)                    # py(ans.401,ans)
   ans = integrate(2*sin(x)*exp(-x), x)        # py(ans.402,ans)
\end{python}

\PySetup{action=verbatim}

\begin{python}

   from sympy import *

   x = Symbol('x')

   ans = diff(cos(x)**2, x)
   ans = integrate(sin(x)*exp(-x), x)
\end{python}

\begin{align*}
   &\py*{ans.401}\\
   &\py*{ans.402}
\end{align*}
```

Passing LaTeX data to Python

There are occassions where the active Python code may require information from the LaTeX source. Since Python has no direct access to LaTeX some other means must be provided to build this bridge. The idea presented here follows a standard LaTeX pattern – run LaTeX twice, once before Python, leaving breadcrumbs for Python to pickup during its run, then a final LaTeX run to complete the job.

The example shown here is a simple proof of concept. It shows how the dimensions of plot can be specified in the LaTeX source and then accessed later by Python.

To compile this example, use

```
pdflatex      example-09
pylatex.sh -i example-09
```

The idea is to use LaTeX to create a Python dictionary saved as a `.json` file. The dictionary will contain just two entries, one for the height and one for the width of the plot. Here are the relevant lines of LaTeX.

```
\newdimen\mywidth\mywidth=17cm       % target width
\newdimen\myheight\myheight=13.5cm   % target height

\newwrite\breadcrumbs
\immediate\openout\breadcrumbs=\jobname.json                % create Json file
\immediate\write\breadcrumbs{\writebgroup}                  % {
\immediate\write\breadcrumbs{"height":\inches{\myheight},} %    "height":5.31496,
\immediate\write\breadcrumbs{"width":\inches{\mywidth}}    %    "width":6.69292
\immediate\write\breadcrumbs{\writeegroup}                  % }
\immediate\closeout\breadcrumbs                            % close the file
```

The target dimensions are: (width, height) = (17cm, 13.5cm).

The following Python code does the job of reading the dictionary and setting the values of `height` and `width`.

```
1  import io, os, json
2
3  # read the dictionary
4  try:
5     with io.open(os.getcwd() + '/' + 'example-09.json') as inp_file:
6        inp = json.load(inp_file)
7  except:
```

```
   inp = { "width":"6.4",
           "height":"4.8" }

# set figure dimensions in inches (yikes)
width  = inp['width']
height = inp['height']
```

This Python code does the job of plotting the Bessel functions (it is based on the code from example-04).

```
# plot the Bessel functions
import numpy as np
import scipy.special as sp
import matplotlib.pyplot as plt

plt.matplotlib.rc('text', usetex = True)
plt.matplotlib.rc('grid', linestyle = 'dotted')
plt.matplotlib.rc('figure', figsize = (width,height))

x = np.linspace(0, 15, 500)

for v in range(0, 6):
    plt.plot(x, sp.jv(v, x))

plt.xlim((0, 15))
plt.ylim((-0.5, 1.1))
plt.legend(('${J}_0(x)$', '${J}_1(x)$', '${J}_2(x)$',
            '${J}_3(x)$', '${J}_4(x)$', '${J}_5(x)$'), loc = 0)
plt.xlabel('$x$')
plt.ylabel('${J}_n(x)$')
plt.grid(True)

plt.tight_layout(pad=0.5)

plt.savefig('example-09-fig.pdf')

width_cm  = round (width  * 2.54, 2)   # py (width,width_cm)
height_cm = round (height * 2.54, 2)   # py (height,height_cm)
```

The following figure should have dimensions: (width, height) = (17.0 cm, 13.5 cm).

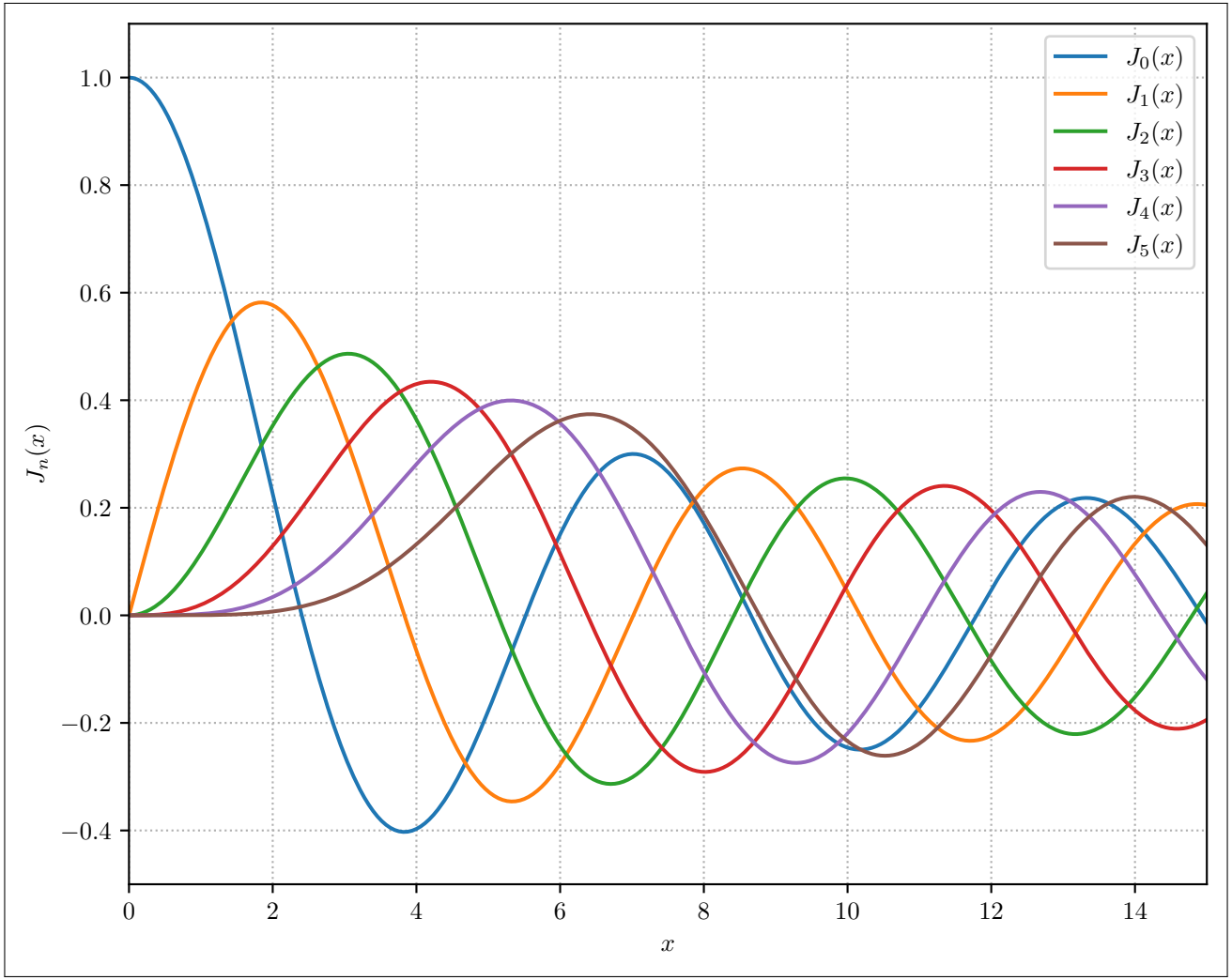

Figure 1: The first six Bessel functions.

A mixed Maple-Python example

This example demonstrates a cooperative effort where Maple is used to do the analytic computations while Python is used to plot the data.

The example chosen here is to find and plot the solution to the boundary value problem defined by

$$\frac{d^2y}{dx^2} + 2\frac{dy}{dx} + 10y = 0 \qquad \text{with } y(0) = 3,\ y'(0) = 0$$

This example requires two passes, once for Maple and once for Python (and in that order). This example can be run using

```
mpllatex.sh -x -i mixed
pylatex.sh  -x -i mixed
pdflatex         mixed
```

Note that the last pair of commands could also be combined as `pylatex.sh -i mixed`.

The Maple code

Here Maple is used to first find the general solution of th differential equation. The boundary conitions are then imposed and finally a uniform sampling of the solution is written to a file for later use by Python and Matplotlib.

```
# a second order ode
ode := diff(y(x), x, x) + 2*diff(y(x),x) + 10*y(x) = 0:  # mpl (ans.101,ode)

# find the general solution
ans := dsolve(ode):                      # mpl (ans.102,ans)

# set initial conditions
ics := y(0) = 3, (D(y))(0) = 0:
tmp := {ics}:                            # mpl (ans.103,tmp)

# find the particular solution
 f := rhs(dsolve([ics, ode])):           # mpl (ans.104,f)
df := diff(f,x):                         # mpl (ans.105,df)

 y := x -> f:
dy := x -> df:

# now sample y and dy at selected points
a,b,n := 0.0,2.0*Pi,300:         # domain and number of samples
dx := (b-a)/n:                   # uniform step

fd := fopen ("mixed.txt", WRITE):
for i from 0 to n by 1 do
   x := a + dx*i:
   fprintf(fd,"% .10e % .10e % .10e\n",x,evalf(y(x)),evalf(dy(x))):
end do:
fclose(fd):
```

The general solution of the differential equation is

$$y\left(x\right) = C_1\,\mathrm{e}^{-x}\sin\left(3\,x\right) + C_2\,\mathrm{e}^{-x}\cos\left(3\,x\right)$$

while the particular solution satifying the boundary conditions is given by

$$y(x) = \mathrm{e}^{-x}\sin\left(3\,x\right) + 3\,\mathrm{e}^{-x}\cos\left(3\,x\right)$$

The Python code

This is a straighforward use of Matplotlib to plot two functions. The code reads the datafile created previously by Maple and then calls Matplotlib to plot that data.

```
import numpy as np
import matplotlib.pyplot as plt

plt.matplotlib.rc('text', usetex = True)
plt.matplotlib.rc('grid', linestyle = 'dotted')
plt.matplotlib.rc('figure', figsize = (5.5,4.1)) # (width,height) inches

x, y, dy = np.loadtxt ('mixed.txt', unpack=True)

plt.plot (x,y)
plt.plot (x,dy)

plt.xlim (0.0,4.0)

plt.legend(('$y(x)$', '$dy(x)/dx$'), loc = 0)
plt.xlabel('$x$')
plt.ylabel('$y(x),\> dy/dx$')
plt.grid(True)
plt.tight_layout(0.5)

plt.savefig('mixed-fig.pdf')
```

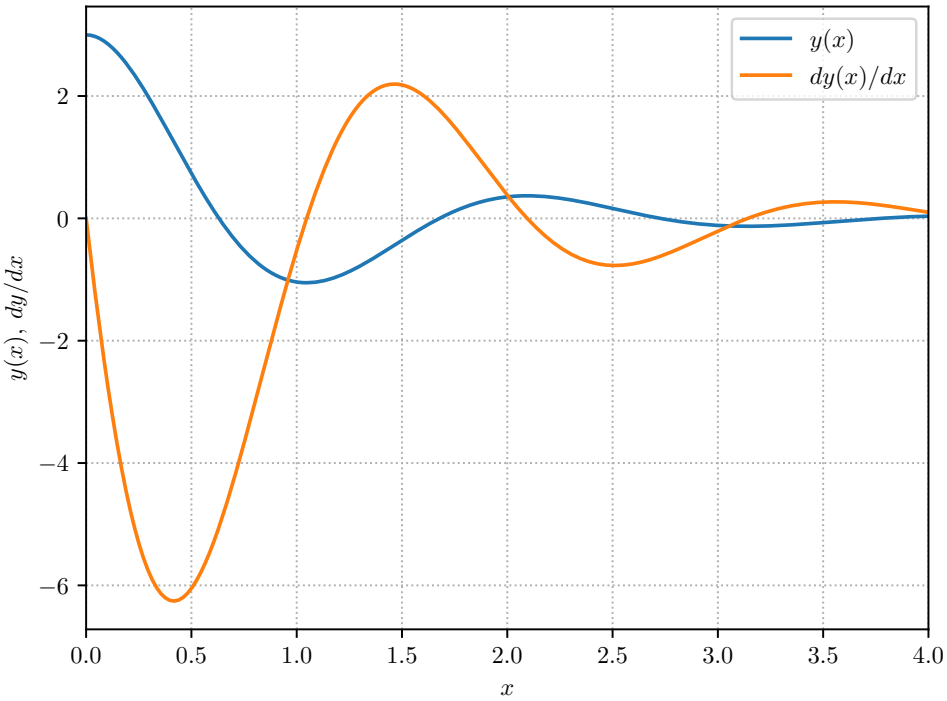

Figure 1: The function and its derivative.