SymPy_SymbolischeMathematik

February 2, 2023

1 Symbolische Mathematik mit SymPy

Das Modul *sympy* ermöglicht es, symbolische Mathematik anzuwenden.

Falls SymPy noch nicht installiert wurde, kann man dies unter Anaconda in einer Anaconda-Konsole nachholen:

conda install -c anaconda sympy

Was bedeutet dies? Dazu zuerst eine Gegenüberstellung mit dem *math*-Modul, dem Standardmodul von Python für mathematische Operationen. Im folgenden Code importieren wir sowohl *sympy* als auch *math* und berechnen jeweils die Wurzel aus der Zahl 9. Da wir keine konkreten Funktionen aus den Modulen importieren, müssen wir den Modulnamen als Präfix davorschreiben, was uns hier auch bei der Unterscheidung der beiden Modulen hilft.

[29]:
```
import sympy
import math

print(math.sqrt(9))
sympy.sqrt(9)
```

```
3.0
```

[29]:
3

Beide Ausgaben sehen ganz ähnlich aus: Es wird, wie erwartet, eine 3 ausgegeben. Allerdings sehen wir keine Kommazahl bei der Ausgabe von *sympy*. Um welche Objekte handelt es sich hier?

[30]:
```
print(type(math.sqrt(9)))
print(type(sympy.sqrt(9)))
```

```
<class 'float'>
<class 'sympy.core.numbers.Integer'>
```

Die erste Zahl ist vom Typ *float*, einem Standard-Datentyp von Python. Die zweite zahl ist allerdings ein Objekt einer Klasse aus der *sympy*-Bibliothek! Sehen wir uns ein zweites Beispiel an: Die Wurzel aus 8:

[31]:
```
print(math.sqrt(8))
sympy.sqrt(8)
```

```
2.8284271247461903
```

[31]: $2\sqrt{2}$

sympy gibt jetzt nicht das (ungenaue) Ergebnis aus, sondern das symbolische Ergebnis $2\sqrt{2}$ in der LaTeX-Schreibweise. Die Anzahl der Nachkommastellen ist beim Datentyp *float* begrenzt (IEEE754). *sympy* kann hier aber auch eine beliebige Anzahl Nachkommastellen korrekt ausgeben. Wir verwenden hierfür die Methode *evalf*, um 30 Stellen korrekt auszugeben:

[32]:
```
sympy.sqrt(8).evalf(40)
```

[32]: 2.828427124746190097603377448419396157139

Diese symbolische Berechnung führt auchd dazu, dass wir keine Fehler mehr erhalten, die auf den begrenzten Möglichkeiten der Speicherung von Fließkommazahlen beruhen. Berechnen wir die Wurzel aus 8 und quadrieren dieses Ergebnis, sollte eigentlich wieder 8 herauskommen:

[33]:
```
math.sqrt(8)**2
```

[33]: 8.000000000000002

Tut es aber nicht! Berechnen wir das Ergebnis mit *sympy*, kommt exakt wieder die Zahl 8 heraus:

[34]:
```
sympy.sqrt(8)**2
```

[34]: 8

Wollen wir das symbolische Ergebnis als Kommazahl ausgeben, so können wir wie schon gezeigt *evalf* verwenden, oder alternativ die Funktion N():

[35]:
```
result = sympy.sqrt(10)
result.evalf()
```

[35]: 3.16227766016838

[36]:
```
sympy.N(result)
```

[36]: 3.16227766016838

1.1 Beispiele mit sympy

sympy ist unglaublich mächtig! Es ist sozusagen die kostenlose Version der kommerziellen Tool *Mathematica* oder *Maple*. Einige Funktionen heissen sogar exakt gleich (wie das schon gezeigte *evalf*).

Wir wollen uns hier noch einige Beispiele ansehen, um die Mächtigkeit von *sympy* zu demonstrieren. Wer tiefer einsteigen möchte, dem bieten sich natürlich viele Tutorials im Internet bzw. die Dokumentation auf der Webseite von *sympy* an (https://www.sympy.org/).

Für die folgenden Beispiele importieren wir für die "schnelle" Berechnung ausnahmsweise alle Module mit einem *. Aber Vorsicht: Es werden ggf. andere Funktionen oder Klassen dadurch überschrieben!

[51]:
```
# Alles mit einem * importieren (eigentlich keine gute Idee!)
from sympy import *
```

Zuerst definieren wir symbolische Variablen. Das geht mit der Methode *symbols*, der wir komma- oder leerzeichengetrennt gleich mehrere Variablennamen übergeben können. Die Methode gibt ein Tupel zurück. Möchte man nur eine symbolische Variable erstellen, kann man auch den Konstruktor der Klasse *Symbol* verwenden.

[52]:
```
# Definiere symbolische Variablen
x = Symbol("x")

## auch möglich
x, y = symbols("x y")
type(x)
```

[52]:
```
sympy.core.symbol.Symbol
```

Wir definieren eine Funktion f mit $f(x) = 3x^2 - 2x - 5$

[39]:
```
f = 3*x**2 - 2*x - 5
f
```

[39]:
$$3x^2 - 2x - 5$$

Können wir dieses Polynom durch Faktoren darstellen, um die Nullstellen zu finden?

[40]:
```
f.factor()
```

[40]:
$$(x + 1)(3x - 5)$$

Häufig hat man die Wahl, ob man eine Methode auf einem Objekt ausführt, oder direkt eine Funktion der Klasse aufruft.

[54]:
```
# Statt Methode dem Objekt: Funktionsaufruf der Klasse und Übergabe des Objekts␣
↪als Argument
factor(f)
```

[54]:
$$(x + 1)(3x - 5)$$

Die Nullstellen wären hier also -1 und $\frac{5}{3}$. Wir können die Nullstellen natürlich auch von SymPy berechnen lassen. Dazu erstellen wir ein Objekt der Klasse *Equation*. Der Konstruktor erwartet als erstes Argument die linke, als zweite Argument die rechte Seite der Gleichung.

[41]:
```
eq1 = Eq(3*x**2 - 2*x - 5, 0)
eq1
```

[41]:
$$3x^2 - 2x - 5 = 0$$

Mit *solveset* können wir die Gleichung lösen:

[61]:
```
solveset(eq1)
```

[61]:
$$\left\{-1, \frac{5}{3}\right\}$$

Mit der Funktion *expand* können wir auch ausklammern, z.B. $f(x) = (x-3)(x+5)(x+\pi)$

[43]:
```
f2 = (x-3)*(x+5)*(x+pi)
f2
```

[43]:
$$(x-3)\,(x+5)\,(x+\pi)$$

[55]:
```
f.expand()
```

[55]:
$$3x^2 - 2x - 5$$

Wir können auch ein Gleichungssystem lösen! Hierzu verwenden wir die Methode *linsolve*. Zum Beispiel:

$$-3x - 2y + 2z = 1 \quad (1)$$
$$-2x + 5y - 6z = 4 \quad (2)$$
$$4x + 3y - 2z = 2 \quad (3)$$

[63]:
```
x,y,z = symbols("x y z")
f1 = Eq(-3*x-2*y+2*z,1)
f2 = Eq(-2*x+5*y-6*z,4)
f3 = Eq(4*x+3*y-2*z,2)

linsolve([f1,f2,f3], x,y,z)
```

[63]:
$$\{(-1,\ 4,\ 3)\}$$

Wie lautet die erste Ableitung der Funktion $f(x) = 3x^2 - 2x - 5$? Ermitteln wir diese mit der Funktion *diff*:

[64]:
```
f = 3*x**2 - 2*x - 5
diff(f)
```

[64]:
$$6x - 2$$

Wie lautet die zweite Ableitung von $f(x)$?

[65]:
```
diff(f,x,2)
```

[65]:
$$6$$

Welchen Wert hat die 1. Ableitung an der Stelle $x = 3$?

[66]:
```
diff(f).evalf(subs={x:3})
```

[66]:
$$16.0$$

Mit der Funktion *integrate* können wir das unbestimmte Integral unserer Funktion $f(x)$ ermitteln, also $\int x^3 - x^2 + 5x dx$. Das Ergebnis ist $x^3 - x^2 + 5x\,(+c)$

[67]:
```
integrate(f)
```

[67]: $x^3 - x^2 - 5x$

integrate kann als weiteres Argument ein Tupel entgegennehmen, das als erstes die Variable, nach der Integriert werden soll, sowie die untere und obere Grenze des bestimmten Integrals enthält. Wir wollen beispielsweise das Integral von $f(x)$ für den Bereich 0 bis 5 ausrechnen, also

$\int_0^5 3x^2 - 2x + 5\,dx$

so liefert uns *integrate* das Ergebnis 125:

[68]:
```
integrate(f, (x,0,5))
```

[68]: 75

Nehmen wir an, wir kennen unser Polynom $f(x)$ gar nicht, sondern nur empirisch ermittelte Messwerte und wollen ein Polynom erstellen, das sich möglichst exakt durch die gemessenen Punkte bewegt. Hier hilft die Funktion *interpolate*. Als erstes Beispiel geben wir für y die Werte 1, 4 und 9 an. Beginnen die x-Werte bei 1 und werden jeweils inkrementiert, braucht man diese nicht extra anzugeben:

[69]:
```
interpolate((1,4,9),x)
```

[69]: x^2

[70]:
```
interpolate((6,13,26),x)
```

[70]: $3x^2 - 2x + 5$

Wir wollen eine Gleichung lösen: Berechne die Nullstellen der Funktion

$f(x) = x^2 - 2x - 8$

[71]:
```
from sympy import *
solve(x**2-2*x-8)
```

[71]: [-2, 4]

Mit etwas mathmatischem Schulwissen, Stichwort Binomische Formeln, weiß man, dass folgende zwei Gleichungen mathematisch gleich sind:

$f_1(x) = (2x + 3)^2$ und $f_1(x) = 4x^2 + 12x + 9$

Vergleichen wir dies mit dem == Operator von Python ist das Ergebnis allerdings *False*. SymPy-Objekte bieten jedoch die Methode *equals* an (Java-Programmierer kennen das!). Diese liefert *True*:

[72]:
```
from sympy import *
f1 = (2*x+3)**2
```

```
f2 = 4*x**2 +12*x + 9
x = Symbol("x")
print(f1==f2)
print(f1.equals(f2))
```

```
False
True
```

1.2 Beispiel: Berechnen der Länge einer Kurve

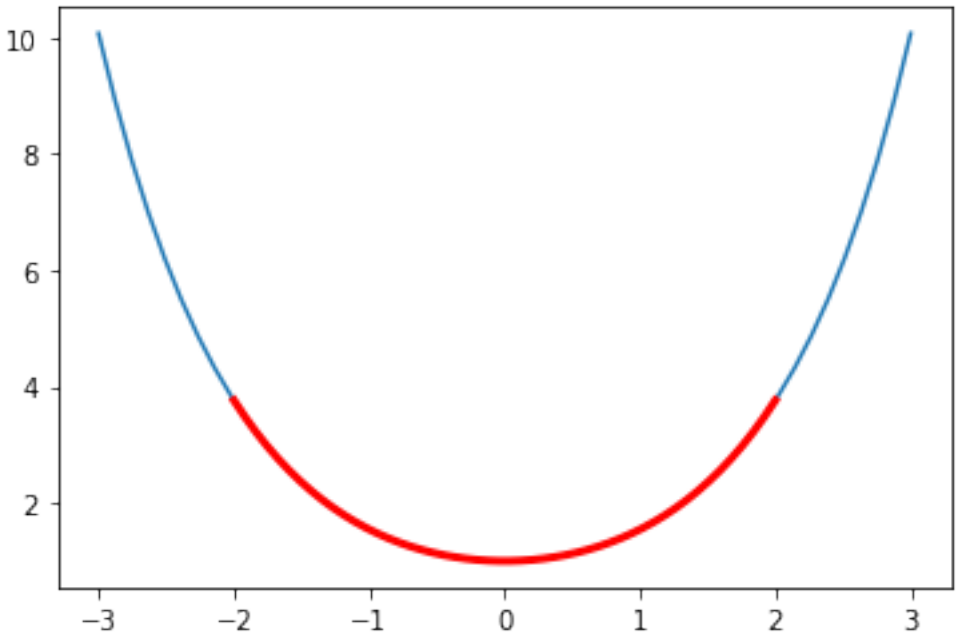
Wir wollen die Länge der Kurve für die Funktion $f(x) = \frac{1}{2}(e^x + e^{-x})$ im Bereich von $[-2; 2]$ berechnen.

[73]:
```
import matplotlib.pyplot as plt
import numpy as np

x  = np.linspace(-3,3,50)
xb = np.linspace(-2,2,50)

y  = [1/2*(np.exp(x)+np.exp(-x)) for x in x]
yb = [1/2*(np.exp(x)+np.exp(-x)) for x in xb]

plt.plot(x,y)
plt.plot(xb,yb, c="r", linewidth=3)
plt.show()
```



Die Formel für die Berechnung der Länge eines Kurvenstücks lautet:

$L = \int_a^b \sqrt{1 + (f'(x))^2}$

Mit *sympy* lässt sich die Länge einfach berechnen:

[74]:
```
from sympy import *

# Wir beschränken die symbolische Variable x auf reele Zahlen (also keine komplexen), dies beschleunigt die Berechnung.
x = symbols("x", real=True)

# Die Funktion für die Berechnung einer Kurve. Wir verwenden sympy.Rational, um den Bruch 1/2 nicht als Fließkommazahl, sondern als sympy-Objekt zu erstellen.
f = Rational(1,2) * (exp(x)+exp(-x))

# 1. Ableitung bestimmen. Die Methode doit() wertet den Ausdruck direkt aus.
deriv = diff(f,x).doit()

# Bestimme das Integral in den Grenzen von -2 bis 2
length = Integral(sqrt(1+deriv**2), (x,-2,2))
length
```

[74]:
$$\int_{-2}^{2} \sqrt{\left(\frac{e^{x}}{2} - \frac{e^{-x}}{2}\right)^{2} + 1}\, dx$$

Schließlich wandeln wir dieses Ergebnis noch in einen numerischen Wert um:

[75]:
```
N(length)
```

[75]:
7.25372081569404

[ ]:
```

```