Labor 1: Einführung in SAGE

Berechnung in SAGE

Grundlegende algebraische Operationen

```
Addierung, Subtrahierung, Multiplizierung, Division       a+b, a-b, a*b, a/b

Potenzierung                                              a^b or a**b

Wurzel                                                    sqrt(a)

Wurzel n-ter Ordnung                                      a^(1/n)
```

Ein Sage Worksheet besteht aus sogenannten Zellen, in die eine oder mehrere Anweisungen eingegeben werden können. Zum Beispiel kann man Sage als ganz normalen Taschenrechner nutzen. Hierzu gibt man einfach den zu berechnenden Ausdruck in die Zelle ein und druckt gleichzeitig "Alt+Enter" oder "Shift+Enter".

Das Ergebnis erscheint dann unter der Zelle. Es ist auch möglich, mehrere Anweisungen in eine Zelle einzugeben. Bei mehreren Eingaben werden alle Werte berechnet, aber nur das Ergebnis der letzten Rechnung wird angezeigt.

Eine neue Zelle wird eingeführt.

Sage hat eine umfassende eingebaute Dokumentation, auf die zugegriffen werden kann, indem der Name der Funktion oder Konstanten (zum Beispiel) gefolgt von einem Fragezeichen eingegeben wird.

In [75]:

```
1+2-3
```

Out[75]:

```
0
```

In [76]:

```
2*3/7+3^2
```

Out[76]:

```
69/7
```

In [77]:

```
(1 + 2*(3 + 5^2))*sqrt(9)
```

Out[77]:

```
171
```

Um die numerische Approximation zu bestimmen, benutzen wir den Symbol ".".

In [78]:

```
15/6
```

Out[78]:

```
5/2
```

In [79]:

```
15./6
```

Out[79]:

```
2.50000000000000
```

Exakte Ausdrücke können auch mithilfe der Funktion numerical_approx approximiert werden:

In [80]:

```
numerical_approx(15/6)
```

Out[80]:

```
2.50000000000000
```

```
Dabei kann die Anzahl der Dezimalstellen (Parameter digits) festgelegt werden:
```

In [81]:

```
numerical_approx(44/13, digits=60)
```

Out[81]:

```
3.38461538461538461538461538461538461538461538461538461538462
```

Andere Operationen mit ganzen Zahlen

Ganzzahldivision	a // b
Modulo-Rechnung	a % b
Quotient und Rest	divmod(a,b)
n!	factorial(n)
Binomialkoeffizient	binomial(n,k)

Mathematische Funktionen

Modul (Betrag)	abs(a)
Exponential und Logarithmus	exp, log
Logarithmus in Basis a	log(x, a)
Trigonometrische Funktionen	sin, cos, tan
Arkusfunktionen	arcsin, arccos, arctan
Hyperbelfunktionen	sinh, cosh, tanh
Areafunktionen	arcsinh, arccosh, arctanh
Deckenfunktion, usw	floor, ceil, trunc, round
Wurzel und Wurzel n-ter Ordnung	sqrt, nth_root

Variablen

```
Um Ergebnisse weiterverwenden zu können, kann man es einer Variable zuweisen (hier mit
dem Namen "a"). Da das Ergebnis einer Zuweisung standardmäßig nicht anzeigt wird,
schreiben wir noch eine Zeile, in der das Ergebnis von "a" (also sein Wert) angezeigt wird. Anders als z.B. in C muss man den
Typ der Variablen nicht explizit eingeben, sondern er ergibt sich aus dem Wert dessen, was zugewiesen wird. Man kann den Wert
von a jetzt auch mit Werten eines anderen Types überschreiben.
```

In [82]:

```
a=5*2
a
```

Out[82]:

```
10
```

Die Variablen müssen vor ihrer Anwendung explizit deklariert werden:

In [1]:

```
x = var('x')
p1=(x+1)^2
p1
```

Out[1]:

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

Um einen Ausdruck zu erweitern, kann man den Befehl variable.expand() benutzen.

In [2]:

```
p1.expand()
```

Out[2]:

```
x^2 + 2*x + 1
```

In [3]:

```
p2=x^2 + 2*x + 1
p2
```

Out[3]:

```
x^2 + 2*x + 1
```

Um einen Ausdruck zu faktorisieren, kann man den Befehl variable.factor() benutzen.

In [4]:

```
p2.factor()
```

Out[4]:

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

In [5]:

```
p3=x^2-5*x + 6
p3.factor()
```

Out[5]:

```
(x - 2)*(x - 3)
```

Man benutzt den Befehl subs um den Wert eines Ausdrucks für einen bestimmten Wert seiner Parameter zu berechnen.

In [6]:

```
p1.subs(x=-1)
```

Out[6]:

```
0
```

In [7]:

```
p3.subs(x=2)
```

Out[7]:

```
0
```

In [8]:

```
p3.subs(x=-3)
```

Out[8]:

```
30
```

```

```

In [9]:

```
x=var('x')
p=(2*x-1)^3
p.expand()
```

Out[9]:

```
8*x^3 - 12*x^2 + 6*x - 1
```

In [10]:

```
x,y=var('x,y')
p=(2*x+y)^2
p.expand()
```

Out[10]:

```
4*x^2 + 4*x*y + y^2
```

In [11]:

```
p.subs(x=1,y=1)
```

Out[11]:

```
9
```

Gleichungen und Systeme

In [ ]:

```

```

Eine Gleichung wird unter Verwendung der Symbole "==" definiert, z.B. f(x)==g(x).

Die am häufigsten verwendeten Befehle zum Lösen von Gleichungen sind:

```
Symbolische Losung                          solve

Wurzeln (mit der Ordnung der Vielfachheit)  roots

Numerische Lösung                           find_root
```

```
Die solve Funktion löst Gleichungen. Die Argumente von solve sind eine Gleichung (oder ein System von Gleichungen) zusammen
mit den Variablen, nach welchen Sie
auflösen möchten.
```

In [12]:

```
x=var('x')
eq1=x^2+x+1==0
solve(eq1,x)
```

Out[12]:

```
[x == -1/2*I*sqrt(3) - 1/2, x == 1/2*I*sqrt(3) - 1/2]
```

Nicht alle Gleichungen können mit Sage gelöst werden. Für das nächste Beispiel, Sage gibt uns keine Lösung zurück.

In [13]:

```
eq2=exp(-x)==x
solve(eq2,x)
```

Out[13]:

```
[x == e^(-x)]
```

Um eine numerische Approximation zu finden, benutzt man den Befehl find_root(equation,a,b). Dieser Befehl bestimmt die Lösung im Intervall [a,b].

In [14]:

```
find_root(eq2,0,2)
```

Out[14]:

```
0.5671432904098384
```

Der "solve" Befehl kann fur Gleichungssysteme verwendet werden. Die Systeme können mit [ ] definiert werden, z.B.:

[eq1,eq2,...,eqn]

In [15]:

```
x,y=var('x,y')
syst=[x+2*y==1,x-y==3]
solve(syst,x,y)
```

Out[15]:

```
[[x == (7/3), y == (-2/3)]]
```

Grenzwerte

Für das Berechnen von Grenzwerten benutzt man den Befehl limit

In [16]:

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

In [17]:

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

Out[17]:

```
0
```

In [18]:

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

Out[18]:

```
1
```

In [19]:

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

Out[19]:

```
Infinity
```

Man kann es merken dass der letzte Grenzwert unendlich ist, d.h. dass eine der Grenzen (linker oder rechter) unendlich ist. Um der linke [minus] (oder der rechte [plus]) Grenzwert zu berechnen, kann man die Option dir benutzen.

In [20]:

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

Out[20]:

```
-Infinity
```

In [21]:

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

Out[21]:

```
+Infinity
```

Nützliche Funktionen fur Analysis

```
      die Ableitung                          diff(f(x), x)

      die Ableitung n-ter Ordnung            diff(f(x), x, n)

      Unbestimmte Integral               integrate(f(x), x)

      Bestimmte Integral                 integrate(f(x),x a, b)

      Summe                              sum(f(i), i, imin, imax)

      Grenzwerte                         limit(f(x), x=a)

      Taylor-Reihen                      taylor(f(x), x, a, n)

      Potenz-Reihen                      f.series(x==a, n)
```

Die Ableitung

```
Mit Sage kann man die Ableitung ester Ordung oder höherer Ordungen berechnen.
```

In [22]:

```
f(x)=exp(x^2)+3
```

```
Um die zweiter Ordnung Ableitung zu berechen benutzen wir:
```

In [23]:

```
diff(f(x),x,2)
```

Out[23]:

```
4*x^2*e^(x^2) + 2*e^(x^2)
```

In [24]:

```
diff(f(x),x,3)
```

Out[24]:

```
8*x^3*e^(x^2) + 12*x*e^(x^2)
```

Das Integral

In [25]:

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

In [26]:

```
integrate(cos(x),x)
```

Out[26]:

```
sin(x)
```

In [27]:

```
f(x)=exp(x)*cos(x)
integrate(f(x),x)
```

Out[27]:

```
1/2*(cos(x) + sin(x))*e^x
```

Man benutzt für bestimmte Integrale den Befehl integrate(f(x),x,a,b).

In [28]:

```
integrate(cos(x),x,0,pi/2)
```

Out[28]:

```
1
```

Sage kann nicht immer den Wert des bestimmten Integrals berechnen.

In [29]:

```
integrate(sin(sqrt(1 - x^3)), x, 0,1)
```

Out[29]:

```
integrate(sin(sqrt(-x^3 + 1)), x, 0, 1)
```

Um den numerischen Wert eines Integrals in einem Intervall zu bestimmen, benutzt man die Funktion integral_numerical. Diese Funktion gibt ein Paar zuruck (der ungefahre Wert, der Fehler).

In [30]:

```
integral_numerical(sin(sqrt(1 - x^3)), 0, 1)
```

Out[30]:

```
(0.7315380084233594, 3.9533799816709084e-07)
```

2D Graphen

```
Sage kann in zwei Dimensionen Kreise, Geraden und Polygone zeichnen, sowie Plots von Funktionen in kartesischen
Koordinaten und Plots in Polarkoordinaten, Konturplots und Plots von Vektorfeldern.
```

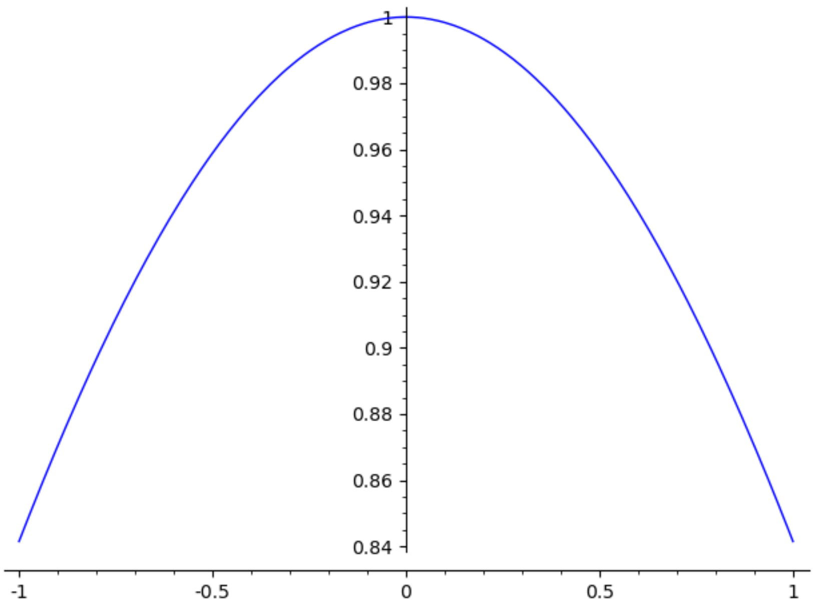
Graphische Darstellung der Funktionen

Um den Graphen der Funktion f(x) im Intervall [a,b] darzustellen, benutzt man den Befehl plot(f(x),a,b) oder den Befehl plot(f(x),x,a,b).

In [31]:

```
f(x)=sin(x)/x
plot(f(x),-1,1)
```

Out[31]:



Der plot Befehl hat mehrere Optionen. Die wichtigsten sind:

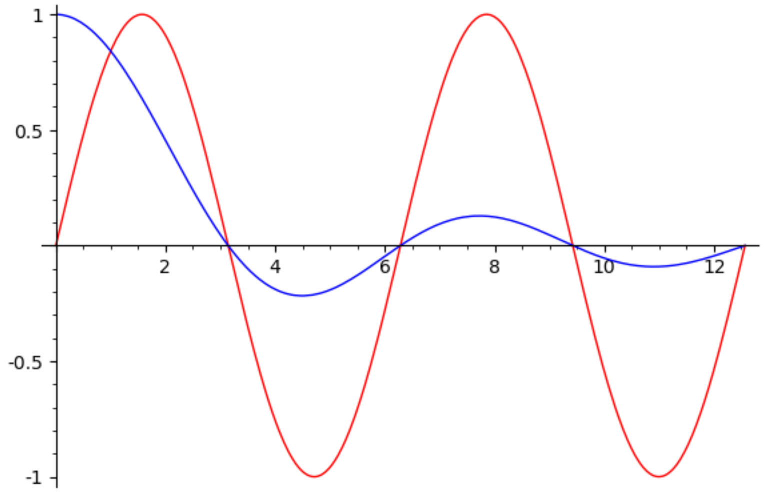
```
plot_points (default value 200): minimal number of computed points;

xmin and xmax: interval bounds over which the function is displayed;

color: colour of the graph, either a RGB triple, a character string such as 'blue', or an HTML colour like '#aaff0b';

detect_poles (default value False): enables to draw a vertical asymptote at poles of the function;

alpha: line transparency;

thickness: line thickness;

linestyle: style of the line, either dotted with ':', dash-dotted with '-.', or solid with the default value '-'.
```

Um mehrere Funktionen in denselben Koordinatensystem darzustellen, muss man die Liste der Funktionen und die Liste der Farben eingeben.

In [33]:

```
plot([sin(x),f(x)],0,4*pi,color=['red','blue'])
```

Out[33]:



Wenn wir mehrere Funktionen darstellen möchten, können wir einen Index benutzen umd den Befehl for um diese Liste zu erzeugen. z.B., Wenn die Funktion f_n(x)=x/(1+x^2)^n gegeben ist, und wir möchten die Funktionen f_1(x),...,f_10(x) grafisch darstellen, konstruieren wir zuerst die Liste. Danach stellen wir die Graphen der Funktionen mit dem Befehl plot dar:

In [34]:

```
x,n=var('x,n')
f(x,n)=x/(1+x^2)^n
```

In [35]:

```
f_list=[f(x,n) for n in [1..10]]
f_list
```

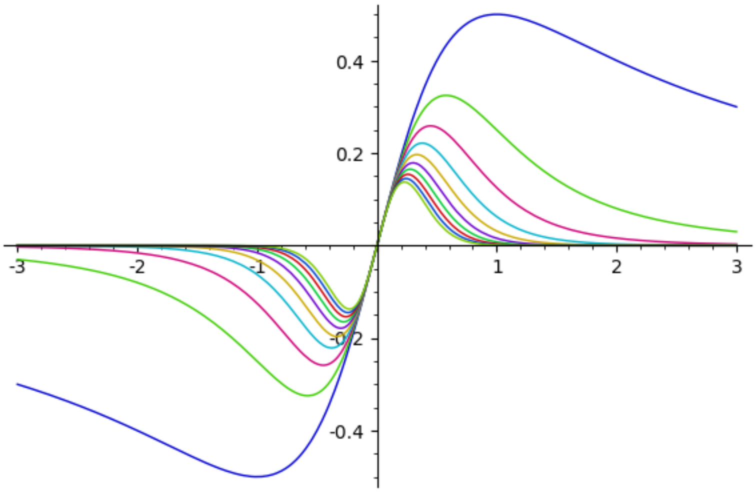
Out[35]:

```
[x/(x^2 + 1),
 x/(x^2 + 1)^2,
 x/(x^2 + 1)^3,
 x/(x^2 + 1)^4,
 x/(x^2 + 1)^5,
 x/(x^2 + 1)^6,
 x/(x^2 + 1)^7,
 x/(x^2 + 1)^8,
 x/(x^2 + 1)^9,
 x/(x^2 + 1)^10]
```

In [36]:

```
plot(f_list,-3,3)
```

Out[36]:

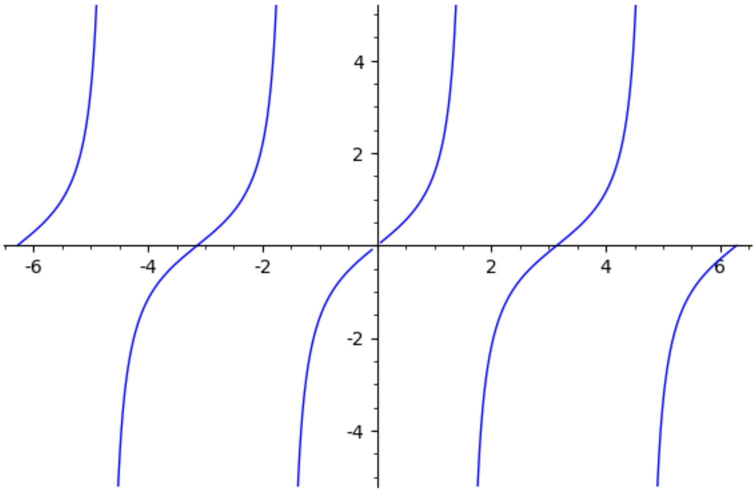


Wenn die Funktion nicht stetig ist, kann man die Option detect_poles=True benutzen. Um ein Intervall auf die OY- Achse festzulegen, kann man ymin und ymax angeben.

In [37]:

```
plot(tan(x), -2*pi,2*pi,detect_poles=True,ymin=-5,ymax=5)
```

Out[37]:



Parameterkurve

Wenn eine Kurve in parametrischer Form gegeben ist, z.B.

x(t)=f(t), y(t)=g(t), t in [a,b]

benutzt man den Befehl parametric_plot((f(t), g(t)), (t, a, b)). um sie graphisch darzustellen.

Z.B., im Fall des Halbkreises definiert durch

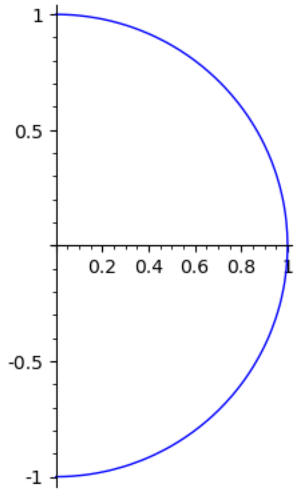
x(t)=cos(t), y(t)=sin(t), t in [-pi/2,b]

haben wir:

In [38]:

```
t=var('t')
x(t)=cos(t)
y(t)=sin(t)
parametric_plot((x(t), y(t)), (t, -pi/2, pi/2))
```

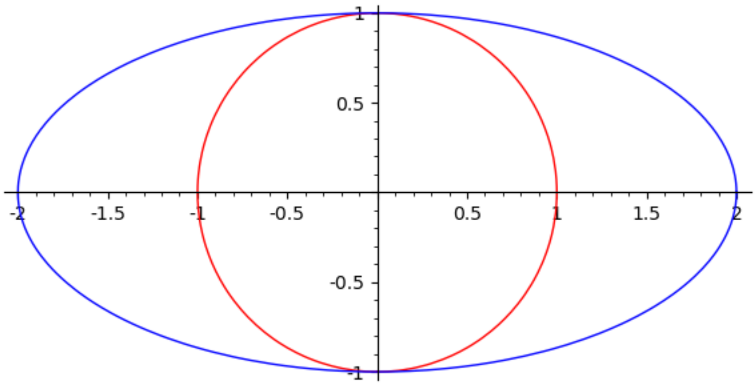
Out[38]:



Um in denselben Koordinatensystem mehrere parametrische Kurven darzustellen, weisen wir jedem Graph eine Variable zu und kombinieren wir sie mit dem Befehl plus(+). Zum Anzeigen verwenden wir den Befehl show.

In [39]:

```
g1=parametric_plot((x(t), y(t)), (t, -pi/2, 3*pi/2),color='red')
g2=parametric_plot((2*x(t), y(t)), (t, 0, 2*pi),color='blue')
show(g1+g2)
```



Implizite Kurve

Um den Graphen einer impliziten Kurve darzustellen, muss man den Befehl

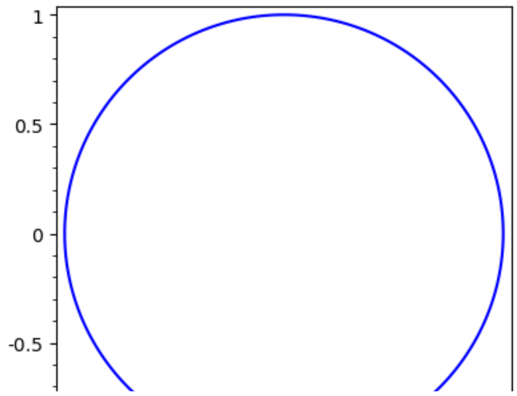
implicit_plot(f(x, y), (x, a, b), (y, c, d))

benutzen.

In [40]:

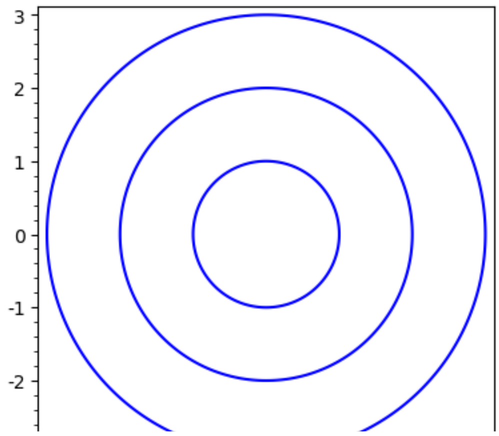
```
x,y=var('x,y')
f(x,y)=x^2+y^2
implicit_plot(f(x,y)==1,(x,-1,1),(y,-1,1))
```

Out[40]:



In [41]:

```
g1=implicit_plot(f(x,y)==1,(x,-1,1),(y,-1,1))
g2=implicit_plot(f(x,y)==4,(x,-2,2),(y,-2,2))
g3=implicit_plot(f(x,y)==9,(x,-3,3),(y,-3,3))
show(g1+g2+g3)
```



In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```