# Sage 4: Grafik

12.12.2018

# Inhaltsverzeichnis



# Ebene Geometrie

Wir zeichnen ein Dreieck mit seinen Schwerlinien.

```
A=[1,1]
B=[5,3]
C=[-1,5]
```

Die Befehle **line**, **point**, **circle** etc. erzeugen die entsprechenden graphischen Objekte und man kann das Bild der Reihe nach aufbauen. Koordinaten können als Listen, Tupel oder Vektoren übergeben werden.

```
g = line([A,B,C])
g
```

```
g = g+line([A,C])
g.show()
```

Die Schwerlinien verbinden die Mittelpunkte der Seiten mit den gegenüberliegenden Scheiteln.

```
(A+B)/2
```

```
Traceback (click to the left of this block for traceback)
...
TypeError: unsupported operand parent(s) for /: '<type
'list'>' and 'Integer Ring'
```

Zunächst aber müssen die Punkte in Vektoren umgewandelt werden.

```
A=vector(A)
B=vector(B)
C=vector(C)
```

```
(A+B)/2
```

$(3, 2)$

```
g = g + line([(A+B)/2,C],color='green')
g
```

```
g = g+line([(B+C)/2,A],color='green')
g
```

```
g = g+line([(C+A)/2,B],color='green')
g
```

Um den Schwerpunkt zu bestimmen, müssen die Schwerlinien geschnitten werden.

```
var('s,t')
```

$(s, t)$

```
sA = A + s * ((B+C)/2-A)
sA
```

$(s + 1, 3\,s + 1)$

```
sB = B + t * ((A+C)/2-B)
sB
```

$(-5\,t + 5, 3)$

Gleichungen mit Vektoren kann man nicht direkt lösen:

```
solve(sA-sB,[s,t])
```

```
Traceback (click to the left of this block for traceback)
...
TypeError: The first argument must be a symbolic expression o
list of symbolic expressions.
```

wir müssen sie zuerst in Listen zurückverwandeln.

```
list(sA-sB)
```

$[s + 5\,t - 4, 3\,s - 2]$

```
st = solve(list(sA-sB),[s,t])
```

```
st
```

$$\left[\left[s = \left(\frac{2}{3}\right), t = \left(\frac{2}{3}\right)\right]\right]$$

Wie immer wird eine Liste von Lösungen zurückgegeben, auch wenn die Lösung eindeutig ist.

```
S = sA.subs(st[0])
```

Die Fehlermeldung gibt einen Hinweis, daß die Substitution so nicht funktioniert. Wir versuchen es mit einem *solution dictionary*.

```
st = solve(list(sA-sB), [s,t], solution_dict=True)
st
```

$$\left[\left\{t : \frac{2}{3}, s : \frac{2}{3}\right\}\right]$$

Damit funktioniert die Substitution und wir können den Schwerpunkt einzeichnen.

```
S = sA.subs(st[0])
S
```

$$\left(\frac{5}{3}, 3\right)$$

oder

```
S = A + st[0][s] * ((B+C)/2-A)
S
```

$$\left(\frac{5}{3}, 3\right)$$

```
g += point(S, color = 'red')
g
```

Um ihn besser sichtbar zu machen, ein Kreis.

```
g=g+ circle(S, 0.01,color='red')
g
```

# 2d-Grafik

Wir haben bereits Funktionsgraphen gesehen.

```
g1 = plot(sin(x), x, 0, 2*pi)
g1.show()
```

Abspeichern des Bildes (das Format wird anhand der Endung erkannt):

```
g1.save('plot.pdf')
```

```
g2 = plot(cos(x), x, 0, 2*pi, color='red')
g2.show()
```

Grafiken können beliebig kombiniert werden.

```
h = g1+g2
h.show()
```

Polarkoordinaten

```
polar_plot(exp(t/5),t,0,4*pi)
```

Analog kann man mit `list_plot` auch Punktmengen und geometrische Objekte zeichnen lassen. Um ein regelmäßiges *n*-Eck zu Zeichnen, braucht man die Liste der Eckpunkte.

```
def ngon(n):
    return [[cos(2*pi*k/n), sin(2*pi*k/n)] for k in
range(0,n+1)]
```

```
g12 = list_plot(ngon(12), aspect_ratio=1, plotjoined=True)
g12.show()
```

Das Bild kann zu den anderen hunzugefügt werden.

```
h += g12
h.show()
```

Um verschiedene Farben automatisch erzeugen zu lassen, empfiehlt es sich, im sogenannten HSV-Raum (hue-saturation-value) zu arbeiten, siehe unten.

```
for k in range(12):
```

```
    g12 += line([[0,0], [cos(2*pi*k/12),sin(2*pi*k/12)]],
hue=k/12)
```

```
g12.show()
```

# 3d-Grafik

```
var('y')
plot3d(exp(x)*sin(2*pi*y),(x,-1,1),(y,-1,1))
```

sphaerische Koordinaten

```
var('th,psi')
g = spherical_plot3d(1,(th,0,pi),(psi,pi/4,3*pi
/4),aspect_ratio=1)
show(g)
```

```
g+= dodecahedron(color='red')
show(g)
```

# Grafik: Farben und HSV-Koordinaten

Die Farbe von Graphen kann mit der Option **color** festgelegt werden.

```
p = plot(x^2,0,1, color='red')
p
```

Wenn man viele verschiedene Farben benötigt, ist es besser, eine Parametrisierung des Farbenraums zu verwenden. Entweder, indem die drei Farbkanäle direkt angegeben werden,

```
plot(x^2, x, -1, 1, rgbcolor = [0.5,0.7,0.1])
```

oder intuitiver anhand anhand des Farbtons in HSV-Koordinaten (engl. *hue-saturation-value*). Alle drei Koordinaten verlaufen in sage im Intervall [0..1]. Reine Grundfarben erhält man mit s=v=1:

```
p = Graphics()
for x in srange(0,1,0.001):
    p = p+line([(x,0),(x,1)], hue = x)
p
```

Verringerung der Sättigung entspricht dem Zumischen von weißer Farbe:

```
p = Graphics()
for x in srange(0,1,0.001):
    p = p+line([(x,0),(x,1)], rgbcolor = hue(x,s=0.5))
p
```

```
p = Graphics()
for x in srange(0,1,0.02):
    for y in srange(0,1,0.02):
        p = p+point([x,y], rgbcolor = hue(x,s=y))
p
```

In dieser Variante wird der hue-Wert direkt in RGB-Koordinaten umgewandelt.

```
hue(0.6)
```

(0.0, 0.40000000000000036, 1.0)

Der Parameter *value* variiert die Helligkeit (=Mischung mit schwarzer Farbe).

```
p = Graphics()
for x in srange(0,1,0.001):
    p = p+line([(x,0),(x,1)], rgbcolor = hue(x,v=0.5))
p
```