Einführung in SAGE

```
##############
# FUNKTIONEN #
##############
```

```
# Definition der Funktion sin(n*x)

def f(x,n):
    return sin(n*x)
```

```
f(pi/3,2)
```

```
# alternative Definition der Funktion sin(n*x)

g(x,n) = sin(n*x)
```

```
g(pi/5,3)
```

```
# Drucken von Funktionsgraphen

n=2
plot(g(x,n),-1,8,figsize=5,aspect_ratio=1.2)
```

```
# ueber "range"

for i in range(11):
    print "i =",i," "
```

```
for i in range(11):
    print "i =",i," ", # "," am Schluss der Printzeile
```

```
# mit 1 beginnen in 3er-Schritten

for i in range(1,11,3):
    print "i =",i," ",
```

```
# Definition der nten Dreieckszahl

def Dreieck(n):
    sum=0
    for i in range(n+1):
        sum=sum+i
    return sum
```

```
Dreieck(1222000)
```

```
# Definition von n^2

def quad(n):
    return sum(i for i in range(1,2*n,2))
```

```
quad(12)
```

```
# Drucken mehrerer Funktionsgraphen

for n in range(1,7):
    plot(f(x,n),(-4,7),figsize=5,aspect_ratio=1)
```

```
# Drucken mehrerer Funktionsgraphen im selben Koordinatensystem

p=plot(f(x,1),(-4,7),figsize=9,aspect_ratio=1)
for n in range(2,7,2):
    p+=plot(f(x,n),(-4,7),figsize=9,aspect_ratio=1)
p
```

```
# ein Beispiel mit "sum(f(i) for i in range(n))"

p=plot(x, (0,10))
for n in range(1,15):
    p += plot(sum((-1)^i*x^(2*i+1)/factorial(2*i+1) for i in
range(n)), (0,10), ymin=-2,ymax=2,aspect_ratio=1,figsize=8)
p
```

```
plot(1/(x^3-x),(-2,2),figsize=5,)
```

```
plot(1/(x^3-x),(-2,2),ymin=-5,ymax=5,figsize=5,detect_poles='show')
#detect_poles='show'/True
```

```
# Ableiten

df(x,n)=diff(f(x,n),x)
```

```
df(x,n)
```

```
# Integrieren

integrate(df(x,n),(x,1,2))
```

```
f(x)=cos(x)*sin(x^2)
F=plot(f(x),(-2.5,2.5),color='green')
dF=plot(diff(f(x),x),(-2.5,2.5),ymin=-3,ymax=3,color='#ff55ff')
show(F+dF)
```

```
var('m')
integrate(sin(n*x)*cos(m*x),(x,-pi,pi))
```

```
n=3
integrate(sin(n*x)^2,(x,-pi,pi))
```

```
integrate(exp(-x^2),x)
```

```
erf?
```

```
n=100
plot(sum(-1/k*sin(k*x) for k in range(1,n)),(-4,7),figsize=5)
```

```
# 3d-Graphiken mit "implicit_plot3d"

var('x,y,z')
```

```
implicit_plot3d(x^2+y^2+z^2==4, (x,-3,3), (y,-3,3), (z,-3,3),
mesh=True)
```

```
var('x,y,z')
K=implicit_plot3d(x^2+y^2==(4-z)^2/4, (x,-3,3), (y,-3,3), (z,0,4),
plot_points=200,mesh=False)
K
```

```
E=implicit_plot3d(x+y+2*z==4, (x,-3,3), (y,-3,3), (z,0,4),
color="lightblue", opacity=0.7)
E
```

```
show(K+E)
```

```
# 3d-Graphiken mit "plot3d"

var('x,y,z')
g(x,y)=sin(x-2*y)*cos(x*y)
plot3d(g,(-2.5,3),(-2.5,3),plot_points=100)
```

```
gradient = diff(g)
```

```
# Vektorfelder

VekFeld = plot_vector_field(gradient, (x,-4,4),
(y,-4,4),figsize=7,plot_points=30,aspect_ratio=1)
VekFeld
```

```
Contour = contour_plot(g, (x,-4,4),(y,-4,4),
fill=False,figsize=7,aspect_ratio=1)
Contour
```

```
show(VekFeld+Contour)
```

```
# interaktive Zelle

@interact
```

```
def _(a=(-3,3),b=(-3,3),c=(-3,3)):
    show(plot(a*x^2+b*x+c,(-5,5),figsize=6,ymin=-30,ymax=30))
```

```
###############
# GLEICHUNGEN #
###############
```

```
solve([x^4+2*x-3==0],x)
```

```
# Differenzialgleichungen

y = function('y',x)
desolve(diff(y,x)+y==7,y)
```

```
h(x)=desolve(diff(y,x)+y==7,y,[0,2])
h
```

```
plot(h,(-2,2),figsize=5)
```

```
# Kontrolle, ob die Loesung richtig ist

plot(diff(h,x)+h,(-2,2),figsize=3,ymin=-1)
```

```
# gedaempfte harmonische Schwingung

y = function('y',x)
harm(x) = desolve(diff(y,x,2)+1/3*diff(y,x)+5*y==0,y,[0,5,0])
harm
```

```
ploth=plot(harm,(0,20),aspect_ratio=1,figsize=5)
plotE=plot(5*e^(-1/6*x),(0,20),aspect_ratio=1,figsize=8,color='red')
plotEE=plot(-5*e^(-1/6*x),
(0,20),aspect_ratio=1,figsize=8,color='green')
show(ploth+plotE+plotEE)
```

```
############
# MATRIZEN #
```

```
############
```

```
A=matrix([[1,2,0],[2,4,6],[1,3,5]])
A
```

```
transpose(A)
```

```
det(A)
```

```
Ainv=A.inverse()
Ainv
```

```
A*Ainv
```

```
v=vector((1,2,3))
v
```

```
A*v,v*A
```

```
############
# NUMERIK #
############
```

```
pi
```

```
sin(pi/3)
```

```
sin(349)
```

```
sin(349.)
```

```
numerical_approx(sin(349))
# es geht auch n(sin(349), wenn n noch nicht anders gebraucht wurde
```

```
numerical_approx(sin(349),digits=50)
```

```
numerical_approx(pi,digits=3000)
```

```
# etwas zu Kettenbruechen

continued_fraction(pi)
```

```
convergents([3, 7, 15, 1])
```

```
print numerical_approx(355/113,digits=9)
print numerical_approx(pi,digits=9)
```

```
####################
# FOLGEN & REIHEN #
####################
```

```
var('k')
sum(1/k, k,1,oo)
```

```
var('k')
sum((5/7)^k, k,0,oo)
```

```
print sum(1/k^2, k,1,oo)
print sum(1/k^3, k,1,oo)
print sum(1/k^4, k,1,oo)
```

```
continued_fraction(sum(1/k^3, k,1,oo))
```

```
###############
# ODDS & ENDS #
```

```
###############
```

```
# Primzahltest

is_prime(2^16+1)
```

```
factor(93875983475)
```

```
# ggT

gcd(9261943626501,2377462613171339)
```

```
# verallgemeinerter euklidischer Algorithmus

xgcd(9261943626501,2377462613171339)
```

```
# kgV

print lcm(12,54)
print LCM([1,2,3,4,5,6,7])
```

```
LCM(range(1,8))
```

```
# vollkommene (perfekte) Zahlen

def perfect(n):
    sum=0
    for i in range(1,n):
        if n%i==0:
            sum=sum+i
    if sum==n:
        return True
    else:
        return False
```

```
perfect(28)
```

```
# gerade vollkommene Zahlen sind immer von der Form 2^(m-1)*(2^m-1)
# wobei 2^(m-1) eine Primzahl ist

for i in range(1,100):
    if is_prime(2^i-1):
        print 2^(i-1)*(2^i-1)
```

```
perfect(33550336)
```

```
# sage-output als latex-code

continued_fraction(sqrt(77))
```

```
latex(continued_fraction(sqrt(77)))
```

```
latex(Ainv)
```

```
# "oeis" steht fuer "On-Line Encyclopedia of Integer Sequences"

search = oeis([23,27,31,37], max_results=7) ; search
```

```
# die Sequenz A089649

def a(k):
    if k==1:
        return 1
    if k==2:
        return 2
    else:
        return a(k-1) + a(floor((k+1)/3))
```

```
for k in range(1,20):
    print a(k),
```

```
search = oeis("Goodstein", max_results=7) ; search
```