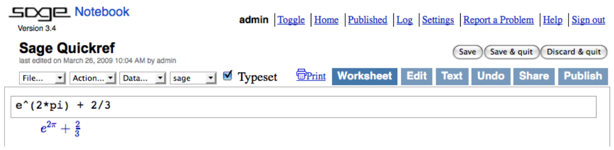
Sage Referenzkarte

Michael Mardaus (based on work of W. Stein)

Sage-„Notebook“



Zelle auswerten: ⟨Umschalt-Enter⟩

Zelle auswerten und neue Zelle einfügen: ⟨Alt-Enter⟩

Zelle teilen: ⟨Strg-;⟩

Zellen verbinden: ⟨Strg-Rücktaste⟩

Math. Zelle einfügen: blaue Linie zwischen Zellen klicken

Text/HTML Zelle einfügen: blaue Linie Umschalt-klicken

Zelle löschen: Inhalt löschen, dann Rücktaste

Kommandozeile

Bef⟨Tab⟩ *Befehl* vervollständigen

`*`*bar*`*?` Alle Befehle auflisten, die "bar" enthalten

Befehl`?` zeigt Dokumentation von *Befehl*

Befehl`??` zeigt Quelltext von *Befehl*

`a.`⟨Tab⟩ zeigt Methoden für Objekt `a` (mehr: `dir(a)`)

`a._`⟨Tab⟩ zeigt versteckte Methoden für Objekt `a`

`search_doc("`*reg. Ausdr.*`")` Suche in Dokumentation

`search_src("`*reg. Ausdr.*`")` Suche in Quelltext

`_` ist die letzte Ausgabe

Zahlen

ganze: $\mathbb{Z}$ = `ZZ` z.B. `-2 -1 0 1 10^100`

rationale: $\mathbb{Q}$ = `QQ` z.B. `1/2 1/1000 314/100 -2/1`

reelle: $\mathbb{R} \approx$ `RR` z.B. `.5 0.001 3.14 1.23e10000`

komplexe: $\mathbb{C} \approx$ `CC` z.B. `CC(1,1) CC(2.5,-3)`

doppelte Genauigkeit: `RDF` und `CDF` z.B. `CDF(2.1,3)`

Modulo n: $\mathbb{Z}/n\mathbb{Z}$ = `Zmod` z.B. `Mod(2,3) Zmod(3)(2)`

endliche Körper: $\mathbb{F}_q$ = `GF` z.B. `GF(3)(2) GF(9,"a").0`

Polynome: $K[x, y]$ z.B. `S.<x,y>=QQ[] x+2*y^3`

Reihen: $R[[t]]$ z.B. `S.<t>=QQ[[]] 1/2+2*t+O(t^2)`

p-adische Zahlen: $\mathbb{Z}_p \approx$`Zp`, $\mathbb{Q}_p \approx$`Qp` z.B. `2+3*5+O(5^2)`

Algebraische Abschlüsse: $\overline{\mathbb{Q}}$ =`QQbar` z.B. `QQbar(2^(1/5))`

Intervallarithmetik: `RIF` z.B. `sage: RIF((1,1.00001))`

Zahlkörper: `R.<x>=QQ[];K.<a>=NumberField(x^3+x+1)`

Arithmetik

ab = `a*b` $\frac{a}{b}$ = `a/b` a^b = `a^b` $\sqrt{x}$ = `sqrt(x)`

$\sqrt[n]{x}$ = `x^(1/n)` $|x|$ = `abs(x)` $\log_b(x)$ = `log(x,b)`

Summen: $\sum_{i=k}^{n} f(i)$ = `sum(f(i) for i in (k..n))`

Produkte: $\prod_{i=k}^{n} f(i)$ = `prod(f(i) for i in (k..n))`

Konstanten und Funktionen

Konstanten: π = `pi` e = `e` i = `i` ∞ = `oo`

ϕ = `golden_ratio` γ = `euler_gamma`

Approximieren: `pi.n(digits=18)` = 3.14159265358979324

Funktionen: `sin cos tan sec csc cot sinh cosh tanh sech csch coth log ln exp` ...

Python Funktionen: `def f(x): return x^2`

Interaktive Funktionen

Mit `@interact` (Parameter steuern die Kontrolle)

```
@interact
def f(n=[0..4], s=(1..5), c=Color("red")):
  var("x");show(plot(sin(n+x^s),-pi,pi,color=c))
```

Symbolische Ausdrücke

Neue symbolische Variablen definieren: `var("t u v y z")`

Symbolische Funktionen: z.B. $f(x) = x^2$ `f(x)=x^2`

Relationen: `f==g f<=g f>=g f<g f>g`

Löse $f = g$: `solve(f(x)==g(x), x)`

`solve([f(x,y)==0, g(x,y)==0], x,y)`

`factor(...)` `expand(...)` `(...).simplify_...`

`find_root(f(x), a, b)` finde $x \in [a, b]$ mit $f(x) \approx 0$

Analysis

$\lim_{x\to a} f(x)$ = `limit(f(x), x=a)`

$\frac{d}{dx}(f(x))$ = `diff(f(x),x)`

$\frac{\partial}{\partial x}(f(x,y))$ = `diff(f(x,y),x)`

`diff` = `differentiate` = `derivative`

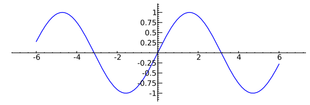
$\int f(x)dx$ = `integral(f(x),x)`

$\int_a^b f(x)dx$ = `integral(f(x),x,a,b)`

$\int_a^b f(x)dx \approx$ `numerical_integral(f(x),a,b)`

Taylor-Polynom, Grad n bei a: `taylor(f(x),x,`a`,`n`)`

2D Grafiken



`line([(`x_1`,`y_1`),...,(`x_n`,`y_n`)],`*Optionen*`)`

`polygon([(`x_1`,`y_1`),...,(`x_n`,`y_n`)],`*Optionen*`)`

`circle((`x`,`y`),`r`,`*Optionen*`)`

`text("txt",(`x`,`y`),`*Optionen*`)`

Optionen wie in `plot.options`, z.B. `thickness=`*Pixel*,

`rgbcolor=(`r`,`g`,`b`)`, `hue=`h mit $0 \le r, b, g, h \le 1$

`show(`*Grafik*, *Optionen*`)`

Größe ändern: `figsize=[w,h]`

Seitenverhältnis ändern: `aspect_ratio=`*Zahl*

`plot(f(`x`),(`x`,`$x_{\min}$`,`$x_{\max}$`),`*Optionen*`)`

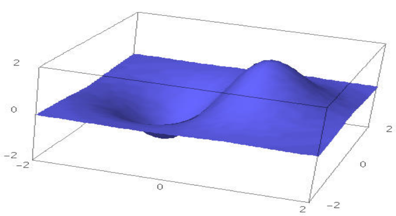
`parametric_plot((f(`t`),g(`t`)),(`t`,`$t_{\min}$`,`$t_{\max}$`),`*Optionen*`)`

`polar_plot(f(`t`),(`t`,`$t_{\min}$`,`$t_{\max}$`),`*Optionen*`)`

Vereinigen: `circle((1,1),1)+line([(0,0),(2,2)])`

`animate(`*Liste von Grafiken*, *Optionen*`).show(delay=20)`

3D Grafiken



`line3d([(`x_1`,`y_1`,`z_1`),...,(`x_n`,`y_n`,`z_n`)],`*Optionen*`)`

`sphere((`x`,`y`,`z`),`r`,`*Optionen*`)`

`text3d("txt", (`x`,`y`,`z`),`*Optionen*`)`

`tetrahedron((`x`,`y`,`z`),`*Größe*, *Optionen*`)`

`cube((`x`,`y`,`z`),`*Größe*, *Optionen*`)`

`octahedron((`x`,`y`,`z`),`*Größe*, *Optionen*`)`

`dodecahedron((`x`,`y`,`z`),`*Größe*, *Optionen*`)`

`icosahedron((`x`,`y`,`z`),`*Größe*, *Optionen*`)`

`plot3d(f(`x`,`y`),(`x`,`x_b`,`x_e`), (`y`,`y_b`,`y_e`),`*Optionen*`)`

`parametric_plot3d((f,g,h),(`t`,`t_b`,`t_e`),`*Optionen*`)`

`parametric_plot3d((f(`u,v`),g(`u,v`),h(`u,v`)),`
`(`u`,`u_b`,`u_e`),(`v`,`v_b`,`v_e`),`*Optionen*`)`

Optionen: `aspect_ratio=`$[1, 1, 1]$, `color="red"`

`opacity=0.5, figsize=6, viewer="tachyon"`

Diskrete Mathematik

$\lfloor x \rfloor$ = `floor(x)` $\lceil x \rceil$ = `ceil(x)`

Rest von n geteilt durch k = `n%k` $k|n$ falls `n%k==0`

$n!$ = `factorial(n)` $\binom{x}{m}$ = `binomial(x,m)`

$\phi(n)$ = `euler_phi(`n`)`

Zeichenketten: z.B. `s = "Hallo"` = `"Ha"+'llo'`

`s[0]="H"` `s[-1]="o"` `s[1:3]="al"` `s[3:]="lo"`

Listen: z.B. `[1,"Hallo",x]` = `[]+[1,"Hallo"]+[x]`

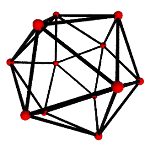
Tupel: z.B. `(1,"Hallo",x)` (unveränderbar)

Mengen: z.B. $\{1,2,1,a\}$ = `Set([1,2,1,"a"])`$(=\{1,2,a\})$

Sprechweise ≈ Mengenschreibweise, z.B.

$\{f(x) : x \in X, x > 0\}$ = `Set([f(x) for x in X if x>0])`

Graphentheorie



Graphen: `G = Graph({0:[1,2,3], 2:[4]})`

Gerichtete Graphen: `DiGraph(`*Ausrichtung*`)`

Graphenfamilien: `graphs.`⟨Tab⟩

Invarianten: `G.chromatic_polynomial()`, `G.is_planar()`

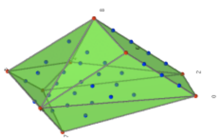
Pfade: `G.shortest_path()`

Zeichnen: `G.plot()`, `G.plot3d()`

Automorphismen: `G.automorphism_group()`,

`G1.is_isomorphic(G2)`, `G1.is_subgraph(G2)`

Kombinatorik



Folgen: `sloane_find(`*Liste*`)`, `sloane.`⟨Tab⟩

Partitionen: `P=Partitions(`n`)` `P.count()`

Kombinationen: `C=Combinations(`*Liste*`)` `C.list()`

Kartesisches Produkt: `CartesianProduct(P,C)`

Tableau: `Tableau([[1,2,3],[4,5]])`

Wörter: `W=Words("abc"); W("aabca")`

Teilgeordnete Mengen: `Poset([[1,2],[4],[3],[4],[]])`

Wurzelsysteme: `RootSystem(["A",3])`

Kristalle: `CrystalOfTableaux(["A",3], shape=[3,2])`

Verbände/Polytope: `A=random_matrix(ZZ,3,6,x=7)`

`L=LatticePolytope(A)` `L.npoints()` `L.plot3d()`

Matrixalgebra

$\begin{pmatrix}1\\2\end{pmatrix}$ = `vector([1,2])`

$\begin{pmatrix}1&2\\3&4\end{pmatrix}$ = `matrix(QQ,[[1,2],[3,4]], sparse=False)`

$\begin{pmatrix}1&2&3\\4&5&6\end{pmatrix}$ = `matrix(QQ,2,3,[1,2,3, 4,5,6])`

$\begin{vmatrix}1&2\\3&4\end{vmatrix}$ = `det(matrix(QQ,[[1,2],[3,4]]))`

Av = `A*v` A^{-1} = `A^-1` A^t = `A.transpose()`

Löse $Ax = v$: `A\v` oder `A.solve_right(v)`

Löse $xA = v$: `A.solve_left(v)`

reduzierte Stufenform: `A.echelon_form()`

Rang und Defekt: `A.rank()` `A.nullity()`

Hessenberg-Form: `A.hessenberg_form()`

Charakteristisches Polynom: `A.charpoly()`

Eigenwerte: `A.eigenvalues()`

Eigenvektoren: `A.eigenvectors_right()` (auch `left`)

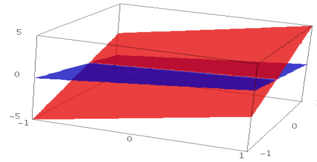
Gram-Schmidt-Orthogonalisierung: `A.gram_schmidt()`

Zeichnen: `A.plot()`

LLL Reduktion: `matrix(ZZ,...).LLL()`

Hermite Normalform: `matrix(ZZ,...).hermite_form()`

Lineare Algebra



Vektorraum K^n = `K^n` e.g. `QQ^3` `RR^2` `CC^4`

Unterraum: `span(`*Vektoren, Körper*`)`

Z.B., `span([[1,2,3], [2,3,5]], QQ)`

Kern: `A.right_kernel()` (auch `left_kernel()`)

Vereinigung und Schnitt: `U + V` and `U.intersection(V)`

Basis: `U.basis()`

Basismatrix: `U.basis_matrix()`

Einschränkung auf den Unterraum: `A.restrict(U)`

Vektor in Basisdarstellung: `U.coordinates(`*Vektor*`)`

Numerik

Pakete: `import numpy, scipy, cvxopt`

Minimalisierung: `var("x y z")`

  `minimize(x^2+x*y^3+(1-z)^2-1, [1,1,1])`

Zahlentheorie

Primzahlen: `prime_range(n,m)`, `is_prime`, `next_prime`

Faktorisierung: `factor(n)`, `qsieve(n)`, `ecm.factor(n)`

Kronecker Symbol: $\left(\frac{a}{b}\right)$ = `kronecker_symbol(`a,b`)`

Kettenbrüche: `continued_fraction(x)`

Bernoulli-Zahlen: `bernoulli(n)`, `bernoulli_mod_p(p)`

Elliptische Kurven: `EllipticCurve(`$[a_1, a_2, a_3, a_4, a_6]$`)`

Dirichlet-Charaktere: `DirichletGroup(`N`)`

Modulformen: `ModularForms(`*Level, Gewicht*`)`

Modulsymbole: `ModularSymbols(`*Level, Gewicht, Zeichen*`)`

Brandt Moduln: `BrandtModule(`*Level, Gewicht*`)`

Abelsche Varietäten: `J0(`N`)`, `J1(`N`)`

Gruppentheorie

`G = PermutationGroup([[(1,2,3),(4,5)],[(3,4)]])`

`SymmetricGroup(`n`)`, `AlternatingGroup(`n`)`

Abelsche Gruppen: `AbelianGroup([3,15])`

Matrixgruppen: `GL, SL, Sp, SU, GU, SO, GO`

Funktionen: `G.sylow_subgroup(p)`, `G.character_table()`,

`G.normal_subgroups()`, `G.cayley_graph()`

Nichtkommutative Ringe

Quaternionen: `Q.<i,j,k> = QuaternionAlgebra(a,b)`

Freie Algebren: `R.<a,b,c> = FreeAlgebra(QQ, 3)`

Python Module

`import` *Modul_Name*

`Modul_Name.`⟨Tab⟩ und `help(Modul_Name)`

Laufzeitanalyse

`time` *befehl*: Zeigt Laufzeitinformationen

`timeit("`*befehl*`")`: genaue Zeitmessung von *befehl*

`t = cputime(); cputime(t)`: vergangene CPU Zeit

`t = walltime(); walltime(t)`: vergangene Echtzeit

`%pdb`: Interaktiven Debugger anschalten (Kommandozeile)

`%prun` *befehl*: Analysiere *befehl* (Kommandozeile)