UNA VISITA RÁPIDA A SAGE

Juan Luis Varona (8 - febrero - 2010)

Sage Version 4.3.1

http://wiki.sagemath.org/quickref

Sage (http://www.sagemath.org) es un entorno de cálculos matemáticos de código abierto que, gracias a los diversos programas que incorpora, permite llevar a cabo cálculos algebraicos, simbólicos y numéricos. El objetivo de Sage es crear una alternativa libre y viable a Magma, Maple, Mathematica y Matlab, todos ellos potentes (y muy caros) programas comerciales.

Sage sirve como calculadora simbólica de precisión arbitraria, pero también puede efectuar cálculos y resolver problemas usando métodos numéricos (es decir, de manera aproximada). Para todo ello emplea algoritmos que tiene implementados él mismo o que toma prestados de alguno de los programas que incorpora, como Maxima, NTL, GAP, Pari/gp, R y Singular. Y para llevar a cabo algunas tareas puede utilizar paquetes especializados opcionales. Incluye un lenguaje de programación propio, que es una extensión de Python (Sage mismo está escrito en Python); es muy recomendable conocer Python para hacer un uso avanzado de Sage.

Sage no sólo consta del programa en sí mismo, que efectúa los cálculos, y con el que podemos comunicarnos a través de terminal, sino que incorpora un interfaz gráfico de usuario a través de cualquier navegador web; para representar las fórmulas y expresiones matemáticas utiliza jsMath, una implementación de LaTeX por medio de JavaScript. Sin necesidad de descargarlo e instalarlo en nuestro ordenador, podemos utilizar Sage en http://www.sagenb.org. Pero no nos preocupemos de ello; simplemente, ¡echemos un vistazo a su sintaxis y su funcionamiento!

1. Uso como calculadora:
```
5+4/3
```
2. Sage utiliza paréntesis ( ) para agrupar:
```
(5+4)/3
```
3. Y también los usa como argumentos de funciones:
```
cos(0)
```
4. Corchetes [ ] para formar listas (con sus elementos separados por comas):
```
v = [3,4,-6]       # Alternativa:  v = vector([3,4,-6])
```
5. También corchetes para acceder a elementos de listas (enumera contando desde 0, como en C y en Python):
```
v[2]
```
6. Como calculadora, Sage proporciona resultados exactos:
```
3^100              # Se usa ** o ^ para elevar a una potencia
factorial(1000)
```
7. Sin embargo, no ocurre así si alguno de los números involucrados en el cálculo tiene decimales (la parte que sigue al # es un comentario):
```
3.0^100       # 3.0 es un número real, no un entero.
```
8. También efectúa cálculos exactos cuando aparecen funciones:
```
arctan(1)
```
9. Con los comandos n o N conseguimos aproximaciones numéricas (ambos comandos son alias de numerical_approx). El símbolo _ alude al último resultado obtenido:
```
N(_)
```
10. Estas aproximaciones pueden tener la precisión que deseemos. Por ejemplo, evaluemos $\sqrt{10}$ con 50 cifras exactas:
```
N(sqrt(10), digits=50)
sqrt(10).n(digits=50)
N(sqrt(10), 170)       # Significa bits de precisión, no dígitos
```
11. Definición y uso de variables simbólicas (se puede usar " o ', y poner comas o no ponerlas):
```
var("alpha, x, y, z")              # Definimos alpha, x, y, z
z = sqrt(7*x + y^5 - sin(alpha))   # (z no hacía falta)
show(z)     # (o jsmath(z))   ¡LaTeX se encarga de dar formato!
latex(z)    # Proporciona el código LaTeX
```
12. Sage permite operar con números complejos (i o I es la unidad imaginaria):
```
(3+4*I)^10
e^(i*pi)       # Da iguar usar e o E
```
13. Podemos definir expresiones simbólicas y manipularlas (aquí, ; sirve para separar órdenes):
```
var('x'); p = (x+1)*(x-1)^2     # El * es importante
q = expand(p); q
```
14. En este ejemplo, el camino inverso lo recorreríamos con
```
factor(q)
```
15. Ahora, hallemos (numéricamente) una raíz de q que esté entre 0 y 3:
```
find_root(q, 0, 3)
```
16. Otro ejemplo de lo mismo:
```
var("theta")
find_root(cos(theta) == sin(theta)+1/5, 0, pi/2)
```
17. Para conocer el tiempo empleado por Sage en efectuar un cálculo:
```
time is_prime(2^127-1)
time factor(2^128-1)
```
18. Podemos librarnos de una asignación o definición previa mediante
```
reset("a")
reset()      # Reinicia todo Sage
```
19. Así se define la función $f(x) = \frac{1}{1+x^2}$:
```
f(x) = 1/(1+x^2)
```
20. Y así se usa:
```
var("r"); [f(x), f(x+1), f(3), f(r)]
```
21. La orden diff permite obtener la derivada (o derivadas parciales) de una función:
```
var("x,y")
diff(f(x))               # f la función definida antes
diff(sin(x^2), x, 4)     # Derivada cuarta
diff(x^2 + 17*y^2, y)    # También se puede usar derivative
```
22. Así calcularíamos una primitiva de f:
```
integrate(f(x),x)        # Da igual usar integral o integrate
```
23. La integral definida $\int_0^1 f(x)\,dx$ podemos evaluarla exactamente (mediante la regla de Barrow, por ejemplo) o numéricamente (mediante una fórmula de cuadratura):
```
var("x")
integral(x*sin(x^2), x)
show(integrate(x/(1-x^3)))
integral(x/(x^2+1), x, 0, 1)
```
24. También existe integración numérica, pero su sintaxis es diferente. En la respuesta que se obtiene, el primer elemento es el resultado, y el segundo una cota del error:
```
integral(x*tan(x), x)
integral(x*tan(x), x,0,1)     # Lo devuelve sin hacer
numerical_integral(x*tan(x), 0,1)
```
25. Cálculo de límites:
```
limit(sin(x)/abs(x), x=0)     # Se da cuenta de que no existe
limit(sin(x)/abs(x), x=0, dir="minus")
limit(sin(x)/abs(x), x=0, dir="plus")
```
26. Conoce la equivalencia de Stirling:
```
lim(factorial(x)*exp(x)/x^(x+1/2), x=oo)      # oo es lo mismo que infinity
```
27. Las funciones se pueden definir a trozos:
```
g = Piecewise([[(-5,1),(1-x)/2], [(1,8),sqrt(x-1)]],x)
```
28. Para representar funciones disponemos del comando plot:

```
plot(g)    #  o g.plot()
plot(cos(x^2), -5, 5, thickness=5, rgbcolor=(0.5,1,0.5), fill = 'axis')
plot(bessel_J(2,x,"maxima"), 0, 20)    # Funciona pero es muuuuuy lento
```

29. Así se guarda un gráfico en el disco duro:

```
save(plot(sin(x)/x, -5, 5), "ruta/dibujo.pdf")    # o plot(...).save("...")
```

30. También podemos representar funciones en paramétricas, gráficos en tres dimensiones, curvas de nivel...

```
automatic_names(true)    # Ya no necesitamos predefinir las variables (v. 4.3.1)
parametric_plot((cos(t),sin(t)), 0,2*pi).show(aspect_ratio=1, frame=true)
plot3d(4*x*exp(-x^2-y^2), (x,-2,2), (y,-2,2))
contour_plot(sin(x*y), (x,-3,3), (y,-3,3), contours=5, plot_points=80)
```

31. Incluso funciones en implícitas en dos y tres dimensiones:

```
implicit_plot(sin(x*y) + sin(x)*sin(y) == 1, (x,-5,5), (y,-5,5))
implicit_plot3d(x^4 + y^4 + z^4 == 16,
  (x, -2, 2), (y, -2, 2), (z, -2, 2), viewer='tachyon')
```

32. Con + se superponen gráficos:

```
plot(2*t^2/3+t, 0, 6) + plot(3*t+20, 0, 6, rgbcolor='red')
  + line([(0, 10), (6, 10)], rgbcolor='green')
```

33. Podemos hacer animaciones:

```
onda = animate([sin(x+k) for k in srange(0,10,0.5)], xmin=0, xmax=8*pi)
onda.show(delay=30, iterations=1)
```

34. Y gráficos interactivos:

```
f = sin(x)*e^(-x)
dibujof = plot(f,-1,5, thickness=2)
punto = point((0,f(x=0)), pointsize=80, rgbcolor=(1,0,0))
@interact
def _(orden=(1..12)):            # La variable de control
  ft = f.taylor(x,0,orden)
  dibujotaylor = plot(ft,-1, 5, color="green", thickness=2)
  show(punto + dibujof + dibujotaylor, ymin = -.5, ymax = 1)
```

35. Para buscar ayuda sobre un comando (especialmente, su sintaxis y ejemplos de uso), basta poner ? tras el nombre del comando; con ?? se obtiene información más técnica (sobre el código fuente):

```
plot?
numerical_integral??
```

36. También podemos buscar en la documentación:

```
search_doc("rgbcolor")
```

37. La orden solve sirve para resolver ecuaciones (obsérvese que se emplea ==) o sistemas:

```
solve(x^2-2 == 0, x)
f = x^4 + 2*x^3 - 4*x^2 - 2*x + 3
solve(f == 0, x, multiplicities=true)
soluciones = solve([9*x - y == 2, x^2 + 2*x*y + y == 7], x, y)
soluciones[0][0].rhs()    # Componente x de la primera solución
```

38. En la versión 4.3.1, Sage aún no sabe sumar series, pero se lo podemos pedir a Maxima:

```
sum(1/n^2 for n in (1..20))     # No sabe si en vez de 20 ponemos oo
maxima("sum(1/n^2,n,1,inf), simpsum")
```

39. Las matrices y vectores se crean así:

```
A = matrix([[-4,1,0],[3,5,-2],[6,8,3]]);
B = identity_matrix(3)
v = vector([3,-2,8]); w = vector([-1,1,1])
H = matrix([[1/(i+j+1) for i in [0..2]] for j in [0..2]])
```

40. Y con ellos se opera como sigue:

```
T = A^2*transpose(A) - 5*B - (1/20)*det(A)*exp(B)
v.dot_product(w)    # Producto escalar
H.inverse()         # También se puede usar ~H o H^(-1)
```

41. El sistema de ecuaciones lineales $Ax = w$ se resuelve con (si se hace simbólico con parámetros, no estudia casos)

```
x = A\w
```

42. Sage nos permite resolver ecuaciones diferenciales:

```
x = var("x"); y = function("y",x)
desolve(diff(y,x,2)-2*diff(y,x)-3*y == exp(x)*sin(x),y)
desolve(diff(y,x) + 2*y - 8 == 0, y, ics=[3,5])  # Condición inicial y(3) = 5
desolvers?   # Más órdenes para resolver ecuaciones diferenciales (o sistemas)
```

43. También podemos resolverlas mediante métodos numéricos (p.e., con un Runge-Kutta):

```
y = function('y',x)
sol = desolve_rk4(diff(y,x)+y*(y-2) == x-3, y, ics=[1,2], step=0.1, end_points=8)
list_plot(sol, plotjoined=True, color="purple")
```

44. Usando simplify, Sage simplifica expresiones (suele ser muy cuidadoso):

```
var("x"); sqrt(x^2)
sqrt(x^4)
simplify(_)    # Sigue sin hacer nada
assume(x>0); simplify(sqrt(x^2))   # Ya simplifica
```

45. También con expresiones trigonométricas:

```
sin(asin(y))  # Devuelve y
asin(sin(x))  # Lo devuelve "sin hacer"
simplify(_)   # Sigue sin hacer nada
assume(-pi/2 <= x <= pi/2); simplify(asin(sin(x)))
var('k t'); assume(k, 'integer'); simplify(sin(t+2*k*pi))
```

46. Pero Sage a veces hace chapuzas:

```
find_root(x*exp(-x), 2, 100)
```

47. Obsérvese también esto:

```
t=-40.0;    # Número real
sum([t^n/factorial(n) for n in [0..300]])
t = -40     # Número entero
N(sum([t^n/factorial(n) for n in [0..300]]))
```

48. Un ejemplo que muestra un programita hecho en Python (con """...""" ponemos la información que aparecerá al usar letraDelDNI?):

```
def letraDelDNI(n):
    """
    Esta funcion calcula la letra de un DNI espanol
    """
    letras = "TRWAGMYFPDXBNJZSQVHLCKE"
    return letras[n%23]
letraDelDNI(12345678)
```

49. Así se define una función de manera recursiva:

```
def f(n):
    if n <= 1: return 1
    elif n%2 == 0: return 2*f(n/2)
    else: return 3*f((n-1)/2)
f(12345678)
```

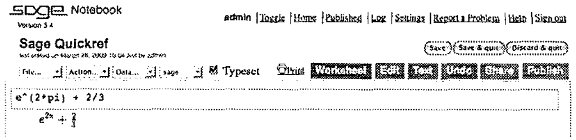
50. Concluyamos con otro programita, el test de Lucas-Lehmer (como s está definido módulo $2^p - 1$, las operaciones con s también son modulares):

```
def is_prime_lucas_lehmer(p):
    s = Mod(4,2^p-1)          # ¡Definimos s como un entero modular!
    for i in range(0, p-2):
        s = s^2 - 2
    return s==0
is_prime_lucas_lehmer(127)         # Nos dice si 2^127-1 es primo (Lucas, 1876)
time is_prime_lucas_lehmer(19937)  # El mayor primo conocido en 1971
```

Sage Quick Reference

William Stein (based on work of P. Jipsen)

Notebook



Evaluate cell: ⟨shift-enter⟩
Evaluate cell creating new cell: ⟨alt-enter⟩
Split cell: ⟨control-;⟩
Join cells: ⟨control-backspace⟩
Insert math cell: click blue line between cells
Insert text/HTML cell: shift-click blue line between cells
Delete cell: delete content then backspace

Command line

com⟨tab⟩ complete *command*
bar? list command names containing "bar"
command?⟨tab⟩ shows documentation
command??⟨tab⟩ shows source code
`a.`⟨tab⟩ shows methods for object `a` (more: `dir(a)`)
`a._`⟨tab⟩ shows hidden methods for object `a`
`search_doc("`*string or regexp*`")` fulltext search of docs
`search_src("`*string or regexp*`")` search source code
`_` is previous output

Numbers

Integers: $\mathbf{Z}$ = `ZZ` e.g. `-2 -1 0 1 10^100`
Rationals: $\mathbf{Q}$ = `QQ` e.g. `1/2 1/1000 314/100 -2/1`
Reals: $\mathbf{R} \approx$ `RR` e.g. `.5 0.001 3.14 1.23e10000`
Complex: $\mathbf{C} \approx$ `CC` e.g. `CC(1,1) CC(2.5,-3)`
Double precision: `RDF` and `CDF` e.g. `CDF(2.1,3)`
Mod n: $\mathbf{Z}/n\mathbf{Z}$ = `Zmod` e.g. `Mod(2,3) Zmod(3)(2)`
Finite fields: $\mathbf{F}_q$ = `GF` e.g. `GF(3)(2) GF(9,"a").0`
Polynomials: $R[x,y]$ e.g. `S.<x,y>=QQ[] x+2*y^3`
Series: $R[[t]]$ e.g. `S.<t>=QQ[[]] 1/2+2*t+O(t^2)`
p-adic numbers: $\mathbf{Z}_p \approx$`Zp`, $\mathbf{Q}_p \approx$`Qp` e.g. `2+3*5+O(5^2)`
Algebraic closure: $\overline{\mathbf{Q}}$ = `QQbar` e.g. `QQbar(2^(1/5))`
Interval arithmetic: `RIF` e.g. `sage: RIF((1,1.00001))`
Number field: `R.<x>=QQ[];K.<a>=NumberField(x^3+x+1)`

Arithmetic

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)`
Sums: $\sum_{i=k}^{n} f(i)$ = `sum(f(i) for i in (k..n))`
Products: $\prod_{i=k}^{n} f(i)$ = `prod(f(i) for i in (k..n))`

Constants and functions

Constants: π = `pi` e = `e` i = `i` ∞ = `oo`
ϕ = `golden_ratio` γ = `euler_gamma`
Approximate: `pi.n(digits=18)` = 3.14159265358979324
Functions: `sin cos tan sec csc cot sinh cosh tanh sech csch coth log ln exp` ...
Python function: `def f(x): return x^2`

Interactive functions

Put `@interact` before function (vars determine controls)

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

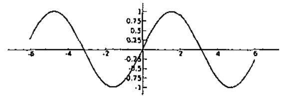
Symbolic expressions

Define new symbolic variables: `var("t u v y z")`
Symbolic function: e.g. $f(x) = x^2$ `f(x)=x^2`
Relations: `f==g f<=g f>=g f<g f>g`
Solve $f = g$: `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)` find $x \in [a,b]$ s.t. $f(x) \approx 0$

Calculus

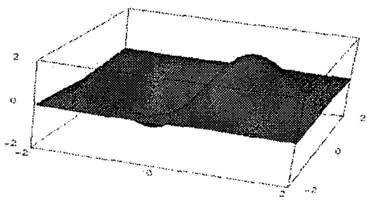
$\lim_{x\to a} f(x)$ = `limit(f(x), x=a)`
$\frac{d}{dx}(f(x))$ = `diff(f(x),x)`
$\frac{\partial}{\partial x}(f(x,y))$ = `diff(f(x,y),x)`
`diff` = `differentiate` = `derivative`
$\int f(x)dx$ = `integral(f(x),x)`
$\int_a^b f(x)dx$ = `integral(f(x),x,a,b)`
$\int_a^b f(x)dx \approx$ `numerical_integral(f(x),a,b)`
Taylor polynomial, deg n about a: `taylor(f(x),x,`a`,`n`)`

2D graphics



`line([(`x_1`,`y_1`),...,(`x_n`,`y_n`)],`*options*`)`
`polygon([(`x_1`,`y_1`),...,(`x_n`,`y_n`)],`*options*`)`
`circle((`x`,`y`),`r`,`*options*`)`
`text("txt",(`x`,`y`),`*options*`)`
options as in `plot.options`, e.g. `thickness=`*pixel*,
`rgbcolor=(`r`,`g`,`b`)`, `hue=`h where $0 \le r,b,g,h \le 1$
`show(`*graphic*, *options*`)`
use `figsize=[w,h]` to adjust size
use `aspect_ratio=`*number* to adjust aspect ratio
`plot(f(`x`),(`$x,x_{\min},x_{\max}$`),`*options*`)`
`parametric_plot((f(`t`),g(`t`)),(`$t,t_{\min},t_{\max}$`),`*options*`)`
`polar_plot(f(`t`),(`$t,t_{\min},t_{\max}$`),`*options*`)`
combine: `circle((1,1),1)+line([(0,0),(2,2)])`
`animate(`*list of graphics, options*`).show(delay=20)`

3D graphics



`line3d([(`x_1`,`y_1`,`z_1`),...,(`x_n`,`y_n`,`z_n`)],`*options*`)`
`sphere((`x`,`y`,`z`),`r`,`*options*`)`
`text3d("txt", (`x`,`y`,`z`), `*options*`)`
`tetrahedron((`x`,`y`,`z`),`*size*`,`*options*`)`
`cube((`x`,`y`,`z`),`*size*`,`*options*`)`
`octahedron((`x`,`y`,`z`),`*size*`,`*options*`)`
`dodecahedron((`x`,`y`,`z`),`*size*`,`*options*`)`
`icosahedron((`x`,`y`,`z`),`*size*`,`*options*`)`
`plot3d(f(`x`,`y`),(`x,x_b,x_e`), (`y,y_b,y_e`),`*options*`)`
`parametric_plot3d((f,g,h),(`t,t_b,t_e`),`*options*`)`
`parametric_plot3d((f(`u,v`),g(`u,v`),h(`u,v`)),`
`(`u,u_b,u_e`),(`v,v_b,v_e`),`*options*`)`
options: `aspect_ratio=[1,1,1], color="red"`
`opacity=0.5, figsize=6, viewer="tachyon"`

Discrete math

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

Remainder of n divided by k = `n%k` $k|n$ iff `n%k==0`

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

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

Strings: e.g. `s = "Hello" = "He"+'llo'`

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

Lists: e.g. `[1,"Hello",x] = []+[1,"Hello"]+[x]`

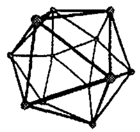
Tuples: e.g. `(1,"Hello",x)` (immutable)

Sets: e.g. $\{1,2,1,a\}$ = `Set([1,2,1,"a"])` $(=\{1,2,a\})$

List comprehension ≈ set builder notation, e.g.

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

Graph theory



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

Directed Graph: `DiGraph(`*dictionary*`)`

Graph families: `graphs.`⟨tab⟩

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

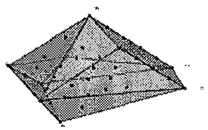
Paths: `G.shortest_path()`

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

Automorphisms: `G.automorphism_group()`,

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

Combinatorics



Integer sequences: `sloane_find(`*list*`)`, `sloane.`⟨tab⟩

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

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

Cartesian product: `CartesianProduct(P,C)`

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

Words: `W=Words("abc"); W("aabca")`

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

Root systems: `RootSystem(["A",3])`

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

Lattice Polytopes: `A=random_matrix(ZZ,3,6,x=7)`

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

Matrix algebra

$\begin{pmatrix}1\\2\end{pmatrix}$ = `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()`

Solve $Ax = v$: `A\v` or `A.solve_right(v)`

Solve $xA = v$: `A.solve_left(v)`

Reduced row echelon form: `A.echelon_form()`

Rank and nullity: `A.rank()` `A.nullity()`

Hessenberg form: `A.hessenberg_form()`

Characteristic polynomial: `A.charpoly()`

Eigenvalues: `A.eigenvalues()`

Eigenvectors: `A.eigenvectors_right()` (also left)

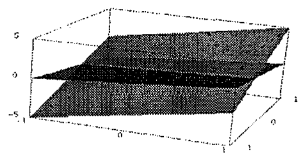
Gram-Schmidt: `A.gram_schmidt()`

Visualize: `A.plot()`

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

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

Linear algebra



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

Subspace: `span(vectors, `*field*` )`

E.g., `span([[1,2,3], [2,3,5]], QQ)`

Kernel: `A.right_kernel()` (also left)

Sum and intersection: `V + W` and `V.intersection(W)`

Basis: `V.basis()`

Basis matrix: `V.basis_matrix()`

Restrict matrix to subspace: `A.restrict(V)`

Vector in terms of basis: `V.coordinates(`*vector*`)`

Numerical mathematics

Packages: `import numpy, scipy, cvxopt`

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

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

Number theory

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

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

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

Continued fractions: `continued_fraction(x)`

Bernoulli numbers: `bernoulli(n)`, `bernoulli_mod_p(p)`

Elliptic curves: `EllipticCurve([`a_1,a_2,a_3,a_4,a_6`])`

Dirichlet characters: `DirichletGroup(`*N*`)`

Modular forms: `ModularForms(`*level*, *weight*`)`

Modular symbols: `ModularSymbols(`*level*, *weight*, *sign*`)`

Brandt modules: `BrandtModule(`*level*, *weight*`)`

Modular abelian varieties: `J0(`*N*`)`, `J1(`*N*`)`

Group theory

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

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

Abelian groups: `AbelianGroup([3,15])`

Matrix groups: `GL, SL, Sp, SU, GU, SO, GO`

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

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

Noncommutative rings

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

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

Python modules

`import` *module_name*

module_name`.`⟨tab⟩ and `help(module_name)`

Profiling and debugging

`time` *command*: show timing information

`timeit("command")`: accurately time command

`t = cputime(); cputime(t)`: elapsed CPU time

`t = walltime(); walltime(t)`: elapsed wall time

`%pdb`: turn on interactive debugger (command line only)

`%prun command`: profile command (command line only)

Sage Quick Reference: Calculus

William Stein

Sage Version 3.4

http://wiki.sagemath.org/quickref

Builtin constants and functions

Constants: $\pi =$ `pi` $e =$ `e` $i =$ `I` $=$ `i`

$\infty =$ `oo` $=$ `infinity` NaN=`NaN` log(2) =`log2`

$\phi =$ `golden_ratio` $\gamma =$ `euler_gamma`

$0.915 \approx$ `catalan` $2.685 \approx$ `khinchin`

$0.660 \approx$ `twinprime` $0.261 \approx$ `merten` $1.902 \approx$ `brun`

Approximate: `pi.n(digits=18)` $= 3.14159265358979324$

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

Defining symbolic expressions

Create symbolic variables:

`var("t u theta")` or `var("t,u,theta")`

Use `*` for multiplication and `^` for exponentiation:

$2x^5 + \sqrt{2}$ = `2*x^5 + sqrt(2)`

Typeset: `show(2*theta^5 + sqrt(2))` $\longrightarrow 2\theta^5 + \sqrt{2}$

Symbolic functions

Symbolic function (can integrate, differentiate, etc.):

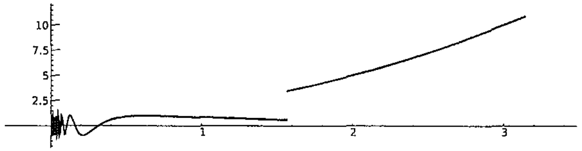
`f(a,b,theta) = a + b*theta^2`

Also, a "formal" function of theta:

`f = function('f',theta)`

Piecewise symbolic functions:

`Piecewise([[(0,pi/2),sin(1/x)],[(pi/2,pi),x^2+1]])`



Python functions

Defining:

```
def f(a, b, theta=1):
    c = a + b*theta^2
    return c
```

Inline functions:

```
f = lambda a, b, theta = 1: a + b*theta^2
```

Simplifying and expanding

Below f must be symbolic (so **not** a Python function):

Simplify: `f.simplify_exp()`, `f.simplify_full()`, `f.simplify_log()`, `f.simplify_radical()`, `f.simplify_rational()`, `f.simplify_trig()`

Expand: `f.expand()`, `f.expand_rational()`

Equations

Relations: $f = g$: `f == g`, $f \neq g$: `f != g`,
$f \leq g$: `f <= g`, $f \geq g$: `f >= g`,
$f < g$: `f < g`, $f > g$: `f > g`

Solve $f = g$: `solve(f == g, x)`, and
`solve([f == 0, g == 0], x,y)`

`solve([x^2+y^2==1, (x-1)^2+y^2==1],x,y)`

Solutions:

`S = solve(x^2+x+1==0, x, solution_dict=True)`

`S[0]["x"]` `S[1]["x"]` are the solutions

Exact roots: `(x^3+2*x+1).roots(x)`

Real roots: `(x^3+2*x+1).roots(x,ring=RR)`

Complex roots: `(x^3+2*x+1).roots(x,ring=CC)`

Factorization

Factored form: `(x^3-y^3).factor()`

List of (factor, exponent) pairs:

`(x^3-y^3).factor_list()`

Limits

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

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

$\lim_{x \to a^+} f(x) =$ `limit(f(x), x=a, dir='plus')`

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

$\lim_{x \to a^-} f(x) =$ `limit(f(x), x=a, dir='minus')`

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

Derivatives

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

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

`diff` = `differentiate` = `derivative`

`diff(x*y + sin(x^2) + e^(-x), x)`

Integrals

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

`integral(x*cos(x^2), x)`

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

`integral(x*cos(x^2), x, 0, sqrt(pi))`

$\int_a^b f(x)dx \approx$ `numerical_integral(f(x),a,b)[0]`

`numerical_integral(x*cos(x^2),0,1)[0]`

`assume(...)`: use if integration asks a question

`assume(x>0)`

Taylor and partial fraction expansion

Taylor polynomial, deg n about a:

`taylor(f,x,a,n)`$\approx c_0 + c_1(x-a) + \cdots + c_n(x-a)^n$

`taylor(sqrt(x+1), x, 0, 5)`

Partial fraction:

`(x^2/(x+1)^3).partial_fraction()`

Numerical roots and optimization

Numerical root: `f.find_root(a, b, x)`

`(x^2 - 2).find_root(1,2,x)`

Maximize: find (m, x_0) with $f(x_0) = m$ maximal

`f.find_maximum_on_interval(a, b, x)`

Minimize: find (m, x_0) with $f(x_0) = m$ minimal

`f.find_minimum_on_interval(a, b, x)`

Minimization: `minimize(f,` *start_point*`)`

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

Multivariable calculus

Gradient: `f.gradient()` or `f.gradient(`*vars*`)`

`(x^2+y^2).gradient([x,y])`

Hessian: `f.hessian()`

`(x^2+y^2).hessian()`

Jacobian matrix: `jacobian(f,` *vars*`)`

`jacobian(x^2 - 2*x*y, (x,y))`

Summing infinite series

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

Not yet implemented, but you can use Maxima:

`s = 'sum (1/n^2,n,1,inf), simpsum'`

`SR(sage.calculus.calculus.maxima(s))` $\longrightarrow \pi^2/6$

Sage Quick Reference:
Elementary Number Theory

William Stein

Sage Version 3.4

http://wiki.sagemath.org/quickref

Everywhere $m, n, a, b, etc.$ are elements of ZZ

ZZ = $\mathbf{Z}$ = all integers

Integers

$\ldots, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \ldots$

n divided by m has *remainder* `n % m`

`gcd(n,m)`, `gcd(`*list*`)`

extended gcd $g = sa + tb = \gcd(a, b)$: `g,s,t=xgcd(a,b)`

`lcm(n,m)`, `lcm(`*list*`)`

binomial coefficient $\binom{m}{n}$ = `binomial(m,n)`

digits in a given base: `n.digits(`*base*`)`

number of digits: `n.ndigits(`*base*`)`

(*base* is optional and defaults to 10)

divides $n \mid m$: `n.divides(m)` if $nk = m$ some k

divisors all d with $d \mid n$: `n.divisors()`

factorial $n!$ = `n.factorial()`

Prime Numbers

$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, \ldots$

factorization: `factor(n)`

primality testing: `is_prime(n)`, `is_pseudoprime(n)`

prime power testing: `is_prime_power(n)`

$\pi(x) = \#\{p : p \leq x \text{ is prime}\}$ = `prime_pi(x)`

set of prime numbers: `Primes()`

$\{p : m \leq p < n \text{ and } p \text{ prime}\}$ =`prime_range(m,n)`

prime powers: `prime_powers(m,n)`

first n primes: `primes_first_n(n)`

next and previous primes: `next_prime(n)`, `previous_prime(n)`, `next_probable_prime(n)`

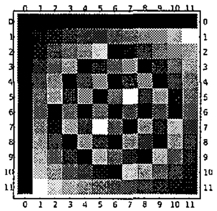
prime powers: `next_prime_power(n)`, `pevious_prime_power(n)`

Lucas-Lehmer test for primality of $2^p - 1$

```
def is_prime_lucas_lehmer(p):
    s = Mod(4, 2^p - 1)
    for i in range(3, p+1): s = s^2 - 2
    return s == 0
```

Modular Arithmetic and Congruences

`k=12; m = matrix(ZZ, k, [(i*j)%k for i in [0..k-1] for j in [0..k-1]]); m.plot(cmap='gray')`



Euler's $\phi(n)$ function: `euler_phi(n)`

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

Quadratic residues: `quadratic_residues(n)`

Quadratic non-residues: `quadratic_residues(n)`

ring $\mathbf{Z}/n\mathbf{Z}$ = `Zmod(n)` = `IntegerModRing(n)`

a modulo n as element of $\mathbf{Z}/n\mathbf{Z}$: `Mod(a, n)`

primitive root modulo n = `primitive_root(n)`

inverse of $n \pmod m$: `n.inverse_mod(m)`

power $a^n \pmod m$: `power_mod(a, n, m)`

Chinese remainder theorem: `x = crt(a,b,m,n)`

finds x with $x \equiv a \pmod m$ and $x \equiv b \pmod n$

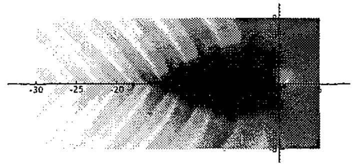
discrete log: `log(Mod(6,7), Mod(3,7))`

order of $a \pmod n$ =`Mod(a,n).multiplicative_order()`

square root of $a \pmod n$ =`Mod(a,n).sqrt()`

Special Functions

`complex_plot(zeta, (-30,5), (-8,8))`



$\zeta(s) = \prod_p \frac{1}{1-p^{-s}} = \sum \frac{1}{n^s}$ = `zeta(s)`

$\mathrm{Li}(x) = \int_2^x \frac{1}{\log(t)} dt$ = `Li(x)`

$\Gamma(s) = \int_0^\infty t^{s-1} e^{-t} dt$ = `gamma(s)`

Continued Fractions

`continued_fraction(pi)`

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cdots}}}}$$

continued fraction: `c=continued_fraction(x,` *bits*`)`

convergents: `c.convergents()`

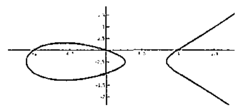
convergent numerator p_n = `c.pn(n)`

convergent denominator q_n = `c.qn(n)`

value: `c.value()`

Elliptic Curves

`EllipticCurve([0,0,1,-1,0]).plot(plot_points=300,thickness=3)`



E = EllipticCurve($[a_1, a_2, a_3, a_4, a_6]$)

$$y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$$

conductor N of E =`E.conductor()`

discriminant Δ of E =`E.discriminant()`

rank of E = `E.rank()`

free generators for $E(\mathbf{Q})$ = `E.gens()`

j-invariant = `E.j_invariant()`

$N_p = \#\{\text{solutions to } E \text{ modulo } p\}$ = `E.Np(`*prime*`)`

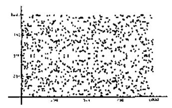
$a_p = p + 1 - N_p$ =`E.ap(`*prime*`)`

$L(E, s) = \sum \frac{a_n}{n^s}$ = `E.lseries()`

$\mathrm{ord}_{s=1} L(E, s)$ = `E.analytic_rank()`

Elliptic Curves Modulo p

`EllipticCurve(GF(997), [0,0,1,-1,0]).plot()`



E = EllipticCurve(GF(p), $[a_1, a_2, a_3, a_4, a_6]$)

$\#E(\mathbf{F}_p)$ = `E.cardinality()`

generators for $E(\mathbf{F}_p)$ = `E.gens()`

$E(\mathbf{F}_p)$ =`E.points()`

Sage Quick Reference: Linear Algebra

Robert A. Beezer
Sage Version 4.8
http://wiki.sagemath.org/quickref

Based on work by Peter Jipsen, William Stein

Vector Constructions

Caution: First entry of a vector is numbered 0
`u = vector(QQ, [1, 3/2, -1])` length 3 over rationals
`v = vector(QQ, {2:4, 95:4, 210:0})`
211 entries, nonzero in entry 4 and entry 95, sparse

Vector Operations

`u = vector(QQ, [1, 3/2, -1])`
`v = vector(ZZ, [1, 8, -2])`
`2*u - 3*v` linear combination
`u.dot_product(v)`
`u.cross_product(v)` order: u×v
`u.inner_product(v)` inner product matrix from parent
`u.pairwise_product(v)` vector as a result
`u.norm() == u.norm(2)` Euclidean norm
`u.norm(1)` sum of entries
`u.norm(Infinity)` maximum entry
`A.gram_schmidt()` converts the rows of matrix `A`

Matrix Constructions

Caution: Row, column numbering begins at 0
`A = matrix(ZZ, [[1,2],[3,4],[5,6]])`
3×2 over the integers
`B = matrix(QQ, 2, [1,2,3,4,5,6])`
2 rows from a list, so 2×3 over rationals
`C = matrix(CDF, 2, 2, [[5*I, 4*I], [I, 6]])`
complex entries, 53-bit precision
`Z = matrix(QQ, 2, 2, 0)` zero matrix
`D = matrix(QQ, 2, 2, 8)`
diagonal entries all 8, other entries zero
`E = block_matrix([[P,0],[1,R]])`, very flexible input
`II = identity_matrix(5)` 5×5 identity matrix
$\mathtt{I} = \sqrt{-1}$, do not overwrite with matrix name
`J = jordan_block(-2,3)`
3×3 matrix, -2 on diagonal, 1's on super-diagonal
`var('x y z'); K = matrix(SR, [[x,y+z],[0,x^2*z]])`
symbolic expressions live in the ring `SR`
`L = matrix(ZZ, 20, 80, {(5,9):30, (15,77):-6})`
20×80, two non-zero entries, sparse representation

Matrix Multiplication

`u = vector(QQ, [1,2,3]), v = vector(QQ, [1,2])`
`A = matrix(QQ, [[1,2,3],[4,5,6]])`
`B = matrix(QQ, [[1,2],[3,4]])`
`u*A, A*v, B*A, B^6, B^(-3)` all possible
`B.iterates(v, 6)` produces $vB^0, vB^1, \ldots, vB^5$
`rows = False` moves `v` to the right of matrix powers
`f(x)=x^2+5*x+3` then `f(B)` is possible
`B.exp()` matrix exponential, i.e. $\sum_{k=0}^{\infty} \frac{1}{k!} B^k$

Matrix Spaces

`M = MatrixSpace(QQ, 3, 4)` is space of 3×4 matrices
`A = M([1,2,3,4,5,6,7,8,9,10,11,12])`
coerce list to element of `M`, a 3×4 matrix over `QQ`
`M.basis()`
`M.dimension()`
`M.zero_matrix()`

Matrix Operations

`5*A+2*B` linear combination
`A.inverse(), A^(-1), ~A`, singular is `ZeroDivisionError`
`A.transpose()`
`A.conjugate()` entry-by-entry complex conjugates
`A.conjugate_transpose()`
`A.antitranspose()` transpose + reverse orderings
`A.adjoint()` matrix of cofactors
`A.restrict(V)` restriction to invariant subspace `V`

Row Operations

Row Operations: (change matrix in place)
Caution: first row is numbered 0
`A.rescale_row(i,a)` a*(row i)
`A.add_multiple_of_row(i,j,a)` a*(row j) + row i
`A.swap_rows(i,j)`
Each has a column variant, `row`→`col`
For a new matrix, use e.g. `B = A.with_rescaled_row(i,a)`

Echelon Form

`A.rref(), A.echelon_form(), A.echelonize()`
Note: `rref()` promotes matrix to fraction field
`A = matrix(ZZ,[[4,2,1],[6,3,2]])`
`A.rref()` `A.echelon_form()`

$$\begin{pmatrix} 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

`A.pivots()` indices of columns spanning column space
`A.pivot_rows()` indices of rows spanning row space

Pieces of Matrices

Caution: row, column numbering begins at 0
`A.nrows(), A.ncols()`
`A[i,j]` entry in row `i` and column `j`
`A[i]` row `i` as immutable Python tuple. Thus,
Caution: OK: `A[2,3] = 8`, Error: `A[2][3] = 8`
`A.row(i)` returns row `i` as Sage vector
`A.column(j)` returns column `j` as Sage vector
`A.list()` returns single Python list, row-major order
`A.matrix_from_columns([8,2,8])`
new matrix from columns in list, repeats OK
`A.matrix_from_rows([2,5,1])`
new matrix from rows in list, out-of-order OK
`A.matrix_from_rows_and_columns([2,4,2],[3,1])`
common to the rows and the columns
`A.rows()` all rows as a list of tuples
`A.columns()` all columns as a list of tuples
`A.submatrix(i,j,nr,nc)`
start at entry `(i,j)`, use `nr` rows, `nc` cols
`A[2:4,1:7], A[0:8:2,3::-1]` Python-style list slicing

Combining Matrices

`A.augment(B)` `A` in first columns, matrix `B` to the right
`A.stack(B)` `A` in top rows, `B` below; `B` can be a vector
`A.block_sum(B)` Diagonal, `A` upper left, `B` lower right
`A.tensor_product(B)` Multiples of `B`, arranged as in `A`

Scalar Functions on Matrices

`A.rank(), A.right_nullity()`
`A.left_nullity() == A.nullity()`
`A.determinant() == A.det()`
`A.permanent(), A.trace()`
`A.norm() == A.norm(2)` Euclidean norm
`A.norm(1)` largest column sum
`A.norm(Infinity)` largest row sum
`A.norm('frob')` Frobenius norm

Matrix Properties

`.is_zero(); .is_symmetric(); .is_hermitian();`
`.is_square(); .is_orthogonal(); .is_unitary();`
`.is_scalar(); .is_singular(); .is_invertible();`
`.is_one(); .is_nilpotent(); .is_diagonalizable()`

Eigenvalues and Eigenvectors

Note: Contrast behavior for exact rings (QQ) vs. RDF, CDF
A.charpoly('t') no variable specified defaults to x
  A.characteristic_polynomial() == A.charpoly()
A.fcp('t') factored characteristic polynomial
A.minpoly() the minimum polynomial
  A.minimal_polynomial() == A.minpoly()
A.eigenvalues() unsorted list, with mutiplicities
A.eigenvectors_left() vectors on left, _right too
  Returns, per eigenvalue, a triple: e: eigenvalue;
  V: list of eigenspace basis vectors; n: multiplicity
A.eigenmatrix_right() vectors on right, _left too
  Returns pair: D: diagonal matrix with eigenvalues
  P: eigenvectors as columns (rows for left version)
    with zero columns if matrix not diagonalizable
Eigenspaces: see "Constructing Subspaces"

Decompositions

Note: availability depends on base ring of matrix,
  try RDF or CDF for numerical work, QQ for exact
  "unitary" is "orthogonal" in real case
A.jordan_form(transformation=True)
  returns a pair of matrices with: A == P^(-1)*J*P
    J: matrix of Jordan blocks for eigenvalues
    P: nonsingular matrix
A.smith_form() triple with: D == U*A*V
  D: elementary divisors on diagonal
  U, V: with unit determinant
A.LU() triple with: P*A == L*U
  P: a permutation matrix
  L: lower triangular matrix, U: upper triangular matrix
A.QR() pair with: A == Q*R
  Q: a unitary matrix, R: upper triangular matrix
A.SVD() triple with: A == U*S*(V-conj-transpose)
  U: a unitary matrix
  S: zero off the diagonal, dimensions same as A
  V: a unitary matrix
A.schur() pair with: A == Q*T*(Q-conj-transpose)
  Q: a unitary matrix
  T: upper-triangular matrix, maybe 2×2 diagonal blocks
A.rational_form(), aka Frobenius form
A.symplectic_form()
A.hessenberg_form()
A.cholesky() (needs work)

Solutions to Systems

A.solve_right(B) _left too
  is solution to A*X = B, where X is a vector or matrix
A = matrix(QQ, [[1,2],[3,4]])
b = vector(QQ, [3,4]), then A\b is solution (-2, 5/2)

Vector Spaces

VectorSpace(QQ, 4) dimension 4, rationals as field
VectorSpace(RR, 4) "field" is 53-bit precision reals
VectorSpace(RealField(200), 4)
  "field" has 200 bit precision
CC^4 4-dimensional, 53-bit precision complexes
Y = VectorSpace(GF(7), 4) finite
  Y.list() has $7^4 = 2401$ vectors

Vector Space Properties

V.dimension()
V.basis()
V.echelonized_basis()
V.has_user_basis() with non-canonical basis?
V.is_subspace(W) True if W is a subspace of V
V.is_full() rank equals degree (as module)?
Y = GF(7)^4, T = Y.subspaces(2)
  T is a generator object for 2-D subspaces of Y
  [U for U in T] is list of 2850 2-D subspaces of Y,
  or use T.next() to step through subspaces

Constructing Subspaces

span([v1,v2,v3], QQ) span of list of vectors over ring

For a matrix A, objects returned are
  vector spaces when base ring is a field
  modules when base ring is just a ring
A.left_kernel() == A.kernel() right_ too
A.row_space() == A.row_module()
A.column_space() == A.column_module()
A.eigenspaces_right() vectors on right, _left too
  Pairs: eigenvalues with their right eigenspaces
A.eigenspaces_right(format='galois')
  One eigenspace per irreducible factor of char poly

If V and W are subspaces
V.quotient(W) quotient of V by subspace W
V.intersection(W) intersection of V and W
V.direct_sum(W) direct sum of V and W
V.subspace([v1,v2,v3]) specify basis vectors in a list

Dense versus Sparse

Note: Algorithms may depend on representation
Vectors and matrices have two representations
  Dense: lists, and lists of lists
  Sparse: Python dictionaries
.is_dense(), .is_sparse() to check
A.sparse_matrix() returns sparse version of A
A.dense_rows() returns dense row vectors of A
Some commands have boolean sparse keyword

Rings

Note: Many algorithms depend on the base ring
<object>.base_ring(R) for vectors, matrices,...
  to determine the ring in use
<object>.change_ring(R) for vectors, matrices,...
  to change to the ring (or field), R
R.is_ring(), R.is_field(), R.is_exact()
Some common Sage rings and fields
  ZZ integers, ring
  QQ rationals, field
  AA, QQbar algebraic number fields, exact
  RDF real double field, inexact
  CDF complex double field, inexact
  RR 53-bit reals, inexact, not same as RDF
  RealField(400) 400-bit reals, inexact
  CC, ComplexField(400) complexes, too
  RIF real interval field
  GF(2) mod 2, field, specialized implementations
  GF(p) == FiniteField(p) p prime, field
  Integers(6) integers mod 6, ring only
  CyclotomicField(7) rationals with 7^{th} root of unity
  QuadraticField(-5, 'x') rationals with $x=\sqrt{-5}$
  SR ring of symbolic expressions

Vector Spaces versus Modules

Module "is" a vector space over a ring, rather than a field
Many commands above apply to modules
Some "vectors" are really module elements

More Help

"tab-completion" on partial commands
"tab-completion" on <object.> for all relevant methods
<command>? for summary and examples
<command>?? for complete source code

Sage Quick Reference: Abstract Algebra

B. Balof, T. W. Judson, D. Perkinson, R. Potluri
version 1.0, Sage Version 5.0.1
latest version: `http://wiki.sagemath.org/quickref`

Based on work by P. Jipsen, W. Stein, R. Beezer

Basic Help

com⟨`tab`⟩ complete *command*
`a.`⟨`tab`⟩ all methods for object `a`
`<command>?` for summary and examples
`<command>??` for complete source code
`*`*foo*`*?` list all commands containing *foo*
`_` underscore gives the previous output
`www.sagemath.org/doc/reference` online reference
`www.sagemath.org/doc/tutorial` online tutorial
`load foo.sage` load commands from the file `foo.sage`
`attach foo.sage`
loads changes to `foo.sage` automatically

Lists

`L = [2,17,3,17]` an ordered list
`L[i]` the ith element of L
Note: lists begin with the 0th element
`L.append(x)` adds x to L
`L.remove(x)` removes x from L
`L[i:j]` the i-th through $(j-1)$-th element of L
`range(a)` list of integers from 0 to $a-1$
`range(a,b)` list of integers from a to $b-1$
`[a..b]` list of integers from a to b
`range(a,b,c)`
every c-th integer starting at a and less than b
`len(L)` length of L
`M = [i^2 for i in range(13)]`
list of squares of integers 0 through 12
`N = [i^2 for i in range(13) if is_prime(i)]`
list of squares of prime integers between 0 and 12
`M + N` the concatenation of lists `M` and `N`
`sorted(L)` a sorted version of `L` (`L` is not changed)
`L.sort()` sorts `L` (`L` is changed)
`set(L)` an unordered list of unique elements

Programming Examples

Print the squares of the integers $0,\ldots,14$:

```
for i in range(15):
    print i^2
```

Print the squares of those integers in $\{0,\ldots,14\}$ that are relatively prime to 15:

```
for i in range(13):
    if gcd(i,15)==1:
      print i^2
```

Preliminary Operations

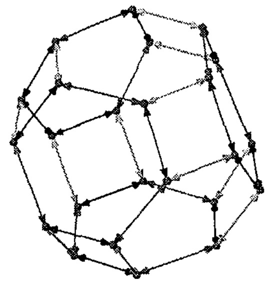
`a = 3; b = 14`
`gcd(a,b)` greatest common divisor a, b
`xgcd(a,b)`
triple (d,s,t) where $d = sa + tb$ and $d = \gcd(a,b)$
`next_prime(a)` next prime after a
`previous_prime(a)` prime before a
`prime_range(a,b)` primes p such that $a \le p < b$
`is_prime(a)` is a prime?
`b % a` the remainder of b upon division by a
`a.divides(b)` does a divide b?

Group Constructions

Permutation multiplication is left-to-right.
`G = PermutationGroup([[(1,2,3),(4,5)],[(3,4)]])`
perm. group with generators $(1,2,3)(4,5)$ and $(3,4)$
`G = PermutationGroup(["(1,2,3)(4,5)","(3,4)"])`
alternative syntax for defining a permutation group
`S = SymmetricGroup(4)` the symmetric group, S_4
`A = AlternatingGroup(4)` alternating group, A_4
`D = DihedralGroup(5)` dihedral group of order 10
`Ab = AbelianGroup([0,2,6])` the group $\mathbb{Z}\times\mathbb{Z}_2\times\mathbb{Z}_6$
`Ab.0, Ab.1, Ab.2` the generators of `Ab`
`a,b,c = Ab.gens()`
shorthand for `a = Ab.0; b = Ab.1; c = Ab.2`
`C = CyclicPermutationGroup(5)`
`Integers(8)` the group $\mathbb{Z}_8$
`GL(3,QQ)` general linear group of 3×3 matrices
`m = matrix(QQ,[[1,2],[3,4]])`
`n = matrix(QQ,[[0,1],[1,0]])`
`MatrixGroup([m,n])`
the (infinite) matrix group with generators `m` and `n`
`u = S([(1,2),(3,4)]); v = S((2,3,4))` elements of `S`
`S.subgroup([u,v])`
the subgroup of `S` generated by `u` and `v`
`S.quotient(A)` the quotient group `S/A`
`A.cartesian_product(D)` the group `A`×`D`
`A.intersection(D)` the intersection of groups `A` and `D`
`D.conjugate(v)` the group `v`$^{-1}$`Dv`
`S.sylow_subgroup(2)` a Sylow 2-subgroup of `S`
`D.center()` the center of `D`
`S.centralizer(u)` the centralizer of `x` in `S`
`S.centralizer(D)` the centralizer of `D` in `S`
`S.normalizer(u)` the normalizer of `x` in `S`
`S.normalizer(D)` the normalizer of `D` in `S`
`S.stabilizer(3)` subgroup of `S` fixing 3

Group Operations

`S = SymmetricGroup(4); A = AlternatingGroup(4)`
`S.order()` the number of elements of `S`
`S.gens()` generators of `S`
`S.list()` the elements of `S`
`S.random_element()` a random element of `S`
`u*v` the product of elements `u` and `v` of `S`
`v^(-1)*u^3*v` the element `v`$^{-1}$`u`^{3}`v` of `S`
`u.order()` the order of `u`
`S.subgroups()` the subgroups of `S`
`S.normal_subgroups()` the normal subgroups of `S`
`A.cayley_table()` the multiplication table for `A`
`u in S` is `u` an element of `S`?
`u.word_problem(S.gens())`
write `u` as a product of the generators of `S`
`A.is_abelian()` is `A` abelian?
`A.is_cyclic()` is `A` cyclic?
`A.is_simple()` is `A` simple?
`A.is_transitive()` is `A` transitive?
`A.is_subgroup(S)` is `A` a subgroup of `S`?
`A.is_normal(S)` is `A` a normal subgroup of `S`?
`S.cosets(A)` the right cosets of `A` in `S`
`S.cosets(A,'left')` the left cosets of `A` in `S`
`g = S.cayley_graph()` Cayley graph of `S`
`g.show3d(color_by_label=True, edge_size=0.01, vertex_size=0.03)` see below:



Ring and Field Constructions

`ZZ` integral domain of integers, $\mathbb{Z}$
`Integers(7)` ring of integers mod 7, $\mathbb{Z}_7$
`QQ` field of rational numbers, $\mathbb{Q}$
`RR` field of real numbers, $\mathbb{R}$
`CC` field of complex numbers, $\mathbb{C}$
`RDF` real double field, inexact
`CDF` complex double field, inexact
`RR` 53-bit reals, inexact, not same as `RDF`
`RealField(400)` 400-bit reals, inexact
`ComplexField(400)` complexes, too
`ZZ[I]` the ring of Gaussian integers
`QuadraticField(7)` the quadratic field, $\mathbb{Q}(\sqrt{7})$
`CyclotomicField(7)`
smallest field containing $\mathbb{Q}$ and the zeros of $x^7 - 1$
`AA, QQbar` field of algebraic numbers, $\overline{\mathbb{Q}}$
`FiniteField(7)` the field $\mathbb{Z}_7$
`F.<a> = FiniteField(7^3)`
finite field in a of size 7^3, $GF(7^3)$
`SR` ring of symbolic expressions
`M.<a>=QQ[sqrt(3)]` the field $\mathbb{Q}[\sqrt{3}]$, with $a = \sqrt{3}$.
`A.<a,b>=QQ[sqrt(3),sqrt(5)]`
the field $\mathbb{Q}[\sqrt{3}, \sqrt{5}]$ with $a = \sqrt{3}$ and $b = \sqrt{5}$.
`z = polygen(QQ,'z'); K = NumberField(x^2 - 2,'s')`
the number field in s with defining polynomial $x^2 - 2$
`s = K.0` set `s` equal to the generator of `K`
`D = ZZ[sqrt(3)]`
`D.fraction_field()`
field of fractions for the integral domain `D`

Ring Operations

Note: Operations may depend on the ring

`A = ZZ[I]; D = ZZ[sqrt(3)]` some rings
`A.is_ring()` is A a ring?
`A.is_field()` is A a field?
`A.is_commutative()` is A commutative?
`A.is_integral_domain()`
`True` is A an integral domain?
`A.is_finite()` is A is finite?
`A.is_subring(D)` is A a subring of D?
`A.order()` the number of elements of A
`A.characteristic()` the characteristic of A
`A.zero()` the additive identity of A
`A.one()` the multiplicative identity of A
`A.is_exact()`
`False` if `A` uses a floating point representation
`a, b = D.gens(); r = a + b`
`r.parent()` the parent ring of r (in this case, `D`)
`r.is_unit()` is r a unit?

Polynomials

`R.<x> = ZZ[ ]` R is the polynomial ring $\mathbb{Z}[x]$
`R.<x> = QQ[ ]; R = PolynomialRing(QQ,'x'); R = QQ['x']`
`R` is the polynomial ring $\mathbb{Q}[x]$
`S.<z> = Integers(8)[ ]` S is the polynomial ring $\mathbb{Z}_8[z]$
`S.<s, t> = QQ[ ]` S is the polynomial ring $\mathbb{Q}[s,t]$
`p = 4*x^3 + 8*x^2 - 20*x - 24`
a polynomial in `R` ($= \mathbb{Q}[x]$)
`p.is_irreducible()` is p irreducible over $\mathbb{Q}[x]$?
`q = p.factor()` factor p
`q.expand()` expand `q`
`p.subs(x=3)` evaluates p at $x = 3$
`R.ideal(p)` the ideal in R generated by p
`R.cyclotomic_polynomial(7)`
the cyclotomic polynomial $x^6+x^5+x^4+x^3+x^2+x+1$
`q = x^2-1`
`p.divides(q)` does p divide q?
`p.quo_rem(q)`
the quotient and remainder of `p` upon division by `q`
`gcd(p, q)` the greatest common divisor of p and q
`p.xgcd(q)` the extended gcd of p and q
`I = S.ideal([s*t+2,s^3-t^2])`
the ideal $(st+2, s^3-t^2)$ in `S` ($= \mathbb{Q}[s,t]$)
`S.quotient(I)` the quotient ring, S/I

Field Operations

`A.<a,b>=QQ[sqrt(3),sqrt(5)]`
`C.<c> = A.absolute_field()`
"flattens" a relative field extension
`A.relative_degree()`
the degree of the relative extension field
`A.absolute_degree()`
the degree of the absolute extension
r = a + b; `r.minpoly()`
the minimal polynomial of the field element `r`
`C.is_galois()` is `C` a Galois extension of Q?

Sage Quick Reference: Graph Theory

Steven Rafael Turner

Sage Version 4.7

http://wiki.sagemath.org/quickref

Constructing

Adjacency Mapping:

`G=Graph([GF(13), lambda i,j: conditions on i,j])`

Input is a list whose first item are vertices and the other is some adjacency function: [list of vertices, function]

Adjacency Lists:

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

`G=Graph({0:{1:"x",2:"z",3:"a"}, 2:{5:"out"}})`

x, z, a, and out are labels for edges and be used as weights.

Adjacency Matrix:

`A = numpy.array([[0,1,1],[1,0,1],[1,1,0]])`

Don't forget to import numpy for the NumPy matrix or ndarray.

`M = Matrix([(....), (....), . . . ])`

Edge List with or without labels:

`G = Graph([(1,3,"Label"),(3,8,"Or"),(5,2)])`

Incidence Matrix:

`M = Matrix(2, [-1,0,0,0,1, 1,-1,0,0,0])`

Graph6 Or Sparse6 string

`G=':IgMoqoCUOqeb\n:I'EDOAEQ?PccSsge\N\n'`

`graphs_list.from_sparse6(G)`

Above is a list of graphs using sparse6 strings.

NetworkX Graph

`g = networkx.Graph({0:[1,2,3], 2:[4]})`

`DiGraph(g)`

`g_2 = networkx.MultiGraph({0:[1,2,3], 2:[4]})`

`Graph(g_2)`

Don't forget to import networkx

Centrality Measures

`G.centrality_betweenness(normalized=False)`

`G.centrality_closeness(v=1)`

`G.centrality_degree()`

Graph Deletions and Additions

`G.add_cycle([vertices])`

`G.add_edge(edge)`

`G.add_edges(iterable of edges)`

`G.add_path`

`G.add_vertex(Name of isolated vertex)`

`G.add_vertices(iterable of vertices)`

`G.delete_edge( v_1, v_2, 'label')`

`G.delete_edges(iterable of edges)`

`G.delete_multiedge(v_1, v_2)`

`G.delete_vertex(v_1)`

`G.delete_vertices(iterable of vertices)`

`G.merge_vertices([vertices])`

Connectivity and Cuts

`G.is_connected()`

`G.edge_connectivity()`

`G.edge_cut(source, sink`

`G.blocks_and_cut_vertices()`

`G.max_cut()`

`G.edge_disjoint_paths(v1,v2, method='LP')`

This method can us LP (Linear Programming) or FF (Ford-Fulkerson)

`vertex_disjoint_paths(v1,v2)`

`G.flow(1,2)`

There are many options to this function please check the documentation.

Conversions

`G.to_directed()`

`G.to_undirected()`

`G.sparse6_string()`

`G.graph6_string()`

Products

`G.strong_product(H)`

`G.tensor_product(H)`

`G.categorical_product(H)`

Same as the tensor product.

`G.disjunctive_product(H)`

`G.lexicographic_product(H)`

`G.cartesian_product(H)`

Boolean Queries

`G.is_tree()`

`G.is_forest()`

`G.is_gallai_tree()`

`G.is_interval()`

`G.is_regular()`

`G.is_chordal()`

`G.is_eulerian()`

`G.is_hamiltonian()`

`G.is_interval()`

`G.is_independent_set([vertices])`

`G.is_overfull()`

`G.is_regular(k)`

Can test for being k-regular, by default k=None.

Common Invariants

`G.diameter()`

`G.average_distance()`

`G.edge_disjoint_spanning_trees(k)`

`G.girth()`

`G.size()`

`G.order()`

`G.radius()`

Graph Coloring

`G.chromatic_polynomial()`

`G.chromatic_number(algorithm="DLX")`

You can change DLX (dancing links) to CP (chromatic polynomial coefficients) or MILP (mixed integer linear program)

`G.coloring(algorithm="DLX")`

You can change DLX to MILP

`G.is_perfect(certificate=False)`

Planarity

`G.is_planar()`

`G.is_circular_planar()`

`G.is_drawn_free_of_edge_crossings()`

`G.layout_planar(test=True, set_embedding=True`

`G.set_planar_positions()`

Search and Shortest Path

`list(G.depth_first_search([vertices], distance=4)`

`list(G.breadth_first_search([vertices])`

`dist,pred = graph.shortest_path_all_pairs(by_weig`

Choice of algorithms: BFS or Floyd-Warshall-Python

`G.shortest_path_length(v_1,v_2, by_weight=True`

`G.shortest_path_lengths(v_1)`

`G.shortest_path(v_1,v_2)`

Spanning Trees

`G.steiner_tree(g.vertices()[:10])`

```
G.spanning_trees_count()
G.edge_disjoint_spanning_trees(2, root vertex)
G.min_spanning_tree(weight_function=somefunction,
algorithm='Kruskal',starting_vertex=3)
```

Kruskal can be change to Prim_fringe, Prim_edge, or NetworkX

Linear Algebra

Matrices

```
G.kirchhoff_matrix()
G.laplacian_matrix()
```

Same as the kirchoff matrix

```
G.weighted_adjacency_matrix()
G.adjacency_matrix()
G.incidence_matrix()
```

Operations

```
G.characteristic_polynomial()
G.cycle_basis()
G.spectrum()
G.eigenspaces(laplacian=True)
G.eigenvectors(laplacian=True)
```

Automorphism and Isomorphism Related

```
G.automorphism_group()
G.is_isomorphic(H)
G.is_vertex_transitive()
G.canonical_label()
G.minor(graph of minor to find)
```

Generic Clustering

```
G.cluster_transitivity()
G.cluster_triangles()
G.clustering_average()
G..clustering_coeff(nbunch=[0,1,2],weights=True)
```

Clique Analysis

```
G.is_clique([vertices])
G.cliques_vertex_clique_number(vertices=[(0, 1), (1, 2)],algorithm="networkx")
```

networkx can be replaced with cliquer.

```
G.cliques_number_of()
G.cliques_maximum()
G.cliques_maximal()
G.cliques_get_max_clique_graph()
G.cliques_get_clique_bipartite()
G.cliques_containing_vertex()
G.clique_number(algorithm="cliquer")
```

cliquer can be replaced with networkx.

```
G.clique_maximum()
G.clique_complex()
```

Component Algorithms

```
G.is_connected()
G.connected_component_containing_vertex(vertex)
G.connected_components_number()
G.connected_components_subgraphs()
G.strong_orientation()
G.strongly_connected_components()
G.strongly_connected_components_digraph()
G.strongly_connected_components_subgraphs()
G.strongly_connected_component_containing_vertex(vertex)
G.is_strongly_connected()
```

NP Problems

```
G.vertex_cover(algorithm='Cliquer')
```

The algorithm can be changed to MILP (mixed integer linear program. Note that MILP requires packages GLPK or CBC.

```
G.hamiltonian_cycle()
G.traveling_salesman_problem()
```