Introducció de Sage

Release 8.4

The Sage Development Team

Oct 18, 2018

CONTENTS

Sage és un programari matemàtic gratuït i de codi obert que dóna suport a la recerca i l'ensenyament en àlgebra, geometria, teoria de nombres, criptografia, càlcul numèric, i àrees relacionades. Tant el model de desenvolupament de Sage com la tecnologia pròpia de Sage es distingeixen per un èmfasi extremadament fort en el fet de ser lliure, en la comunitat, la cooperació i la col.laboració: estem construint el cotxe, no pas reinventant la roda. L'objectiu global de Sage és el de crear una alternativa viable, lliure, i de codi obert als paquets de Maple, Mathematica, Magma, i MATLAB.

Aquest tutorial és una introducció al Sage basada en un manual escrit per Maria Bras-Amorós. Es pot llegir bé sigui en HTML o en PDF.

CHAPTER

ONE

INDICACIONS PER COMENÇAR

- La pàgina oficial de Sage és http://sagemath.org. Des d'allà podeu descarregar-vos el programa.
- Es comença amb `sage` i s'acaba amb `quit`.
- Totes les ordres han d'acabar en salt de línia.
- Per interrompre un càlcul: `Ctrl + C`.
- Es pot fer servir la tecla ↑ per recuperar codi escrit anteriorment.
- Per escriure ordres de sistema: `!ordre`.
- Per llegir el fitxer `fin.sage`: `load fin.sage`.
- Per llegir el fitxer `fin.sage` i escriure els resultats al fitxer `fout`: `sage fin.sage > fout`.
- Per comentar una línia utilitzem `# ...`. Tot el que hi hagi després de # no serà llegit per Sage.
- Per comentar tot un paràgraf o part d'un fitxer escrivim tres vegades cometes al principi i al final i el que quedi entremig no serà llegit per Sage:

```
"""
bla
bla
bla
"""
```

- Per buscar ordres, escrivim les primeres lletres i utilizem el tabulador per veure les possibles complecions.

CHAPTER

TWO

ACCIONS SIMPLES I EXPRESSIONS

2.1 Assignació de variables i escriptura

- Per assignar variables: `nom_variable = valor`.
- Per mostrar per pantalla:
  - El valor de les variables `var1, var2`: `var1, var2`.
  - Text: `print("Un text")`.

2.2 Operadors relacionals

- < (menor que), > (major que), <= (menor o igual que), >= (major o igual que),
- == (igual que), != (diferent de),
- `in` (pertany), `not in` (no pertany).

2.3 Operadors booleans

- `x and y` (x i y),
- `x or y` (x o y),
- `not x` (no x).

2.4 Funcions definides pel Sage

Sage té definides les seves pròpies funcions que depenen d'una o vàries variables (o cap). Per cridar-les escrivim el nom de la funció seguit de les variables entre parèntesis. Per exemple:

```
sage: floor(3.141592)
3
sage: gcd(12,8)
4
sage: sum([3,2,5])
10
```

Hi ha funcions que retornen més d'un valor. Per exemple la funció `divmod(a,b)` ens retorna el quocient i el residu de dividir a entre b:

```
sage: divmod(19,7)
(2, 5)
```

Per assignar aquests valors a variables ho fem de la manera següent:

```
sage: q,r = divmod(19,7)
sage: q
2
sage: r
5
```

També podríem assignar el parell de valors a una sola variable:

```
sage: D = divmod(19,7)
sage: D
(2, 5)
sage: D[0]
2
sage: D[1]
5
```

CHAPTER

THREE

COMPOSICIÓ D'ACCIONS

3.1 Composició condicional

L'estructura següent executarà `bloc1` només si `condicio1` és certa. El bloc `bloc2` només s'executarà si `condicio1` és falsa, i `condicio2` és certa. Si ambdues condicions són falses, aleshores s'executarà `bloc3`.

Cal tenir en compte que la indentació determina l'inici i fi dels blocs i que, per tant, és obligatòria.

```
sage: if condicio1:
....:     bloc1
....: elif condicio2:
....:     bloc2
....: else:
....:     bloc3
```

3.2 Composició iterativa

Les estructures següents executaran `bloc` repetides vegades. La primera ho farà $i_0 - i_1 + 1$ vegades, i la variable `i` prendrà valors consecutius $i_0, \ldots, i_1$. El segon bloc és similar, però `i` prendrà els valors que apareguin a `llista`, en el mateix ordre en què hi apareguin. Finalment, el tercer bloc s'executarà mentre `condicio` sigui certa. Si `condicio` és falsa al començament, `bloc` no s'executarà cap vegada.

```
sage: for i in [i0..i1]:
....:     bloc

sage: for i in llista:
....:     bloc

sage: while condicio:
....:     bloc
```

Exemples:

```
sage: for i in [2..5]:
....:     print(i + 1)
3
4
5
6
```

```
sage: for i in [-1,"foo",3.4]:
....:     print(i)
-1
foo
3.40000000000000
```

```
sage: i = 1
sage: while i < 4:
....:     print(i)
....:     i += 1
1
2
3
```

CHAPTER

FOUR

CONJUNTS I SEQÜÈNCIES

4.1 Conjunts i seqüències

Tant els conjunts (`set`) com les seqüències (`list`) són col.leccions d'objectes. Un conjunt no és ordenat, per tant, un element pot ser en un conjunt com a molt una vegada. Una seqüència, en canvi, és ordenada i, per tant, la repetició és possible. Les seqüències s'escriuen entre `[` i `]`. Els conjunts es construeixen a partir de seqüències fent servir `Set(sequencia)`.

Per exemple:

```
sage: S = [ (-11)^2, (-7)^2, (-5)^2, (-3)^2, 3^2, 5^2, 7^2, 11^2 ]
sage: C = Set(S)
sage: S
[121, 49, 25, 9, 9, 25, 49, 121]
sage: C    # random
{121, 9, 49, 25}
sage: sorted(C)
[9, 25, 49, 121]
```

La i-èssima entrada d'una seqüència (o d'un conjunt) C és `C[i]`. Però compte perquè Sage enumera les posicions des de 0!:

```
sage: S[1] = 1000
sage: S
[121, 1000, 25, 9, 9, 25, 49, 121]
sage: S[3]
9
```

Sage té constructors especials per a seqüències. Les expressions `[a..b]` i `[a,a+k..b]` designen respectivament les progressions `[a,a+1,a+2,...,b]` i `[a,a+k,a+2k,...,b']`, on b' és el màxim enter de la forma $a + ik$ menor o igual que b.

D'altra banda,

```
[ expressio(x) for x in D ]
[ expressio(x,y) for x in D for y in E ]
```

denota la seqüència de valors `expressio(x)` (resp. `expressió(x,y)`) avaluada per tot $x \in D$ (resp. $x \in D, y \in E$).

Així mateix,

```
[ expressio(x) for x in D if condicio ]
```

denota la seqüència de valors `expressio(x)` avaluada per tot $x \in D$ tals que el booleà `condicio` és cert. Per exemple, hauríem pogut crear `S` de la manera següent:

```
sage: S = [ n^2 for n in [-11,-9..11] if is_prime(abs(n)) ]
sage: print(S)
[121, 49, 25, 9, 9, 25, 49, 121]
```

Podem aplicar una funció a tots els elements d'una llista de la manera següent:

```
sage: [cos(t) for t in [0,pi..6*pi]]
[1, -1, 1, -1, 1, -1, 1]
```

4.2 Operacions per conjunts i seqüències

- `len(S)` retorna el cardinal de S.
- `sum(S)` retorna la suma dels elements de S.
- `prod(S)` retorna el producte dels elements de S.
- `S + T` retorna S concatenat amb T.
- `min(S)`, `max(S)` retorna el mínim i el màxim de `S`, que ha de contenir elements amb un ordre.

4.3 Operacions només per conjunts

- `A.union(B)` retorna $A \cup B$.
- `A.intersection(B)` retorna $A \cap B$.
- `A.difference(B)` retorna $A \setminus B$.

4.4 Operacions només per seqüències

- `S.append(x)` afegeig x al final de S.
- `S.remove(x)` esborra x de S.
- `S.index(x)` retorna la posició de l'element x dins de S.
- `S.insert(i,x)` mou una posició els elements amb índex igual o superior a i i afegeix x a la posició i.
- `S.reverse()` capgira la seqüència S.
- `S.sort()` ordena la seqüència S.
- `S[randint(0,len(S))]` retorna un element aleatori de S.

4.4.1 Booleans

- `x in C`, `x not in C` determina si x pertany o no a C.
- `A.issubset(B)`, `A.issuperset(B)` determina si A és un subconjunt de B, o si A conté a B com a subconjunt (només per conjunts).
- `D == C`, `D != C` determina si D és igual a C o no.

CHAPTER

FIVE

FUNCIONS

La declaració general d'una funció de n arguments per la que s'hagi de fer una sèrie d'operacions és:

```
def f(x1, x2, ..., xn):
    ...
    return (val1, val2, ...)
```

Per exemple:

```
sage: def solucions_reals_equacio_segon_grau(a,b,c):
....:     discriminant = b^2 - 4*a*c
....:     arrel_discr = sqrt(discriminant)
....:     if discriminant > 0:
....:         print("hi ha dues solucions reals")
....:         sol1 = (-b + arrel_discr) / (2*a)
....:         sol2 = (-b - arrel_discr) / (2*a)
....:         return (sol1, sol2)
....:     elif discriminant == 0:
....:         print("hi ha una solucio real")
....:         sol = -b / (2*a)
....:         return (sol)
....:     else:
....:         print("no hi ha solucions reals")
....:         return ()
```

Observem que l'indentat és important. Ara podem cridar-la:

```
sage: solucions_reals_equacio_segon_grau(1,0,-1)
hi ha dues solucions reals
(1, -1)
```

Com que en l'exemple donat la funció retorna dos valors, podem assignar-los a dues variables:

```
sage: a,b = solucions_reals_equacio_segon_grau(1,0,-1)
hi ha dues solucions reals
sage: a
1
sage: b
-1
```

Aquesta crida, però, ens donaria un error si la solució no existís o fos única.

Observem que totes les variables utilitzades són per defecte locals i que no ha calgut declarar-les. Una variable externa a la funció amb nom igual a una de les variables locals no es modificarà a causa de la funció. Per exemple, considerem la funció f:

```
sage: def f():
....:     a = 5
....:     return a
```

Observem el resultat del codi següent:

```
sage: a = 3
sage: print(f())
5
sage: print(a)
3
```

CHAPTER

SIX

ENTERS, ANELLS $\mathbb{Z}/N\mathbb{Z}$, RACIONALS I REALS

6.1 Enters

L'anell dels enters el cridem o bé amb `Integers()` o bé amb `ZZ`. A continuació llistem algunes funcions pels enters:

- Operacions bàsiques: `+, -, *, /, ^` (suma, resta, producte, quocient (possiblement racional) i potència).
- `n // m` (quocient de dividir n entre m)
- `n % m` (n mòdul m, el residu de dividir n entre m).
- `divmod(a,b)` (quocient i residu de dividir a entre b).
- `abs(n)` (valor absolut).
- `randint(a,b)` (un enter aleatori entre a i b, ambdós inclosos).
- `random_prime(a)` (un primer aleatori menor o igual que a).
- `divisors(n)`, `prime_divisors(n)`, `number_of_divisors(n)` (la seqüència de divisors de n, o només els divisors primers, o el nombre total de divisors).
- `gcd(m,n)`, `gcd(S)`, `lcm(m,n)`, `lcm(S)` (el màxim comú divisor i el mínim comú múltiple de m i n o bé de la seqüència d'enters S).
- `xgcd(m,n)` (retorna tres valors d, a i b on d és el màxim comú divisor de m i n i on $am + bn = d$).
- `euler_phi(n)` (el nombre d'unitats de l'anell $\mathbb{Z}/n\mathbb{Z}$).
- `factorial(n)` (el factorial de l'enter n).

6.1.1 Booleans

- `is_odd(i)`, `is_even(i)` retornen si l'enter i és senar (o parell).
- `is_prime(i)`, `is_prime_power(i)` retornen si l'enter i és primer (o una potència d'un primer).
- `is_square(n)` retorna si l'enter n és un quadrat perfecte.

6.2 Anells $\mathbb{Z}/n\mathbb{Z}$

L'anell de residus mòdul n el cridem amb `Integers(n)` o amb `Zmod(n)`. Si posem `R = Zmod(n)` per algun n aleshores els elements de R els cridem utilitzant `R(1)`, `R(2)`, etc. Algunes funcions que ens poden ser útils són

- Operacions bàsiques: `+, -, *, ^`.
- `inverse_mod(x,m)` retorna l'invers de $x \pmod m$.

- `solve_mod(expr1 == expr2,m)` resol equacions amb congruències; és important que les incògnites s'hagin alliberat abans, per exemple amb `var('x')`. Per exemple:

```
sage: x, y = var('x','y')
sage: solve_mod(3*x + 2*y == 1, 5)
[(0, 3), (1, 4), (2, 0), (3, 1), (4, 2)]
```

- `primitive_root(n)` retorna un enter que genera el grup multiplicatiu dels enters mòdul n, si existeix.
- `multiplicative_order(n)`, `additive_order(n)` retornen l'ordre multiplicatiu i additiu de l'enter n.

6.2.1 Booleans

- `is_field()` determina si un conjunt és un cos.
- `is_unit()` determina si un element és invertible.

L'exemple que segueix pot ser il.lustratiu:

```
sage: is_prime(19)
True
sage: primitive_root(19)
2
sage: Z19 = Zmod(19)
sage: [x for x in Z19]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
sage: Z19(2).additive_order()
19
sage: Z19(2).multiplicative_order()
18
sage: Z19(5).multiplicative_order()
9
sage: Set([2^i % 19 for i in [1..18]])
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18}
sage: Set([5^i % 19 for i in [1..18]])
{1, 4, 5, 6, 7, 9, 11, 16, 17}
```

6.3 Racionals

El cos dels nombres racionals es denota amb `RationalField()` o `QQ`. Algunes operacions que podem fer amb els racionals són

- Operacions bàsiques: `+, -, *, /, ^`.
- `numerator(q)`, `denominator(q)` (el numerador i denominador de q).

6.4 Reals

El cos dels reals el denotem amb `RealField()` o `RR`. Algunes funcions per als reals són:

- Operacions bàsiques: `+, -, *, /, ^`.
- `ceil(r)`, `floor(r)` retorna la part entera superior o inferior de r.

- `abs(r)` retorna el valor absolut de r.
- `sqrt(r)` retorna l'arrel quadrada de r.

6.5 Equacions

Es poden resoldre equacions utilitzant l'ordre `solve()`. Escrivint `?solve` el sage ens dóna una explicació molt extensa. Aquí en repetim els primers exemples:

```
sage: x, y = var('x, y')
sage: solve([x + y == 6, x - y == 4], x, y)
[[x == 5, y == 1]]
sage: solve([x^2 + y^2 == 1, y^2 == x^3 + x + 1], x, y)
[[x == -1/2*I*sqrt(3) - 1/2, y == -sqrt(-1/2*I*sqrt(3) + 3/2)], [x == -1/2*I*sqrt(3) -
↪ 1/2, y == sqrt(-1/2*I*sqrt(3) + 3/2)], [x == 1/2*I*sqrt(3) - 1/2, y == -sqrt(1/
↪2*I*sqrt(3) + 3/2)], [x == 1/2*I*sqrt(3) - 1/2, y == sqrt(1/2*I*sqrt(3) + 3/2)], [x␣
↪== 0, y == -1], [x == 0, y == 1]]
sage: solutions = solve([sqrt(x) + sqrt(y) == 5, x + y == 10], x, y,solution_dict =␣
↪True)
sage: len(solutions)
2
sage: type(solutions[0])
<... 'dict'>
sage: for sol in solutions: print((sol[x].n(digits=3), sol[y].n(digits=3)))
(5.00 - 5.59*I, 5.00 + 5.59*I)
(5.00 + 5.59*I, 5.00 - 5.59*I)
```

CHAPTER

SEVEN

ANELL DE POLINOMIS

7.1 Definició de polinomis

Per generar l'anell de polinomis sobre un anell R ho fem així:

```
sage: P.<x> = PolynomialRing(R)
```

A més, així queda definida x com la indeterminada dels polinomis de P. Per escriure un polinomi de P ho podem fer de dues maneres:

```
sage: P.<x> = PolynomialRing(QQ)
sage: x^3-7*x^2+5
x^3 - 7*x^2 + 5
sage: P([5,0,-7,1])
x^3 - 7*x^2 + 5
```

7.2 Operacions

- Operacions bàsiques: `+, -, *, /, ^`.
- `f(a)` per avaluar un polinomi f en a.
- `f.degree()` (el grau del polinomi).
- `f.coeffs()` retorna tots els coeficients de f mentre que `f.coefficients()` només retorna els coeficients no-nuls.
- `f.leading_coefficient(f)` retorna el coeficient principal de f.
- `parent(f)` ens diu on pertany el polinomi f.
- `divmod(f,g)` (quocient i residu de dividir f entre g), que es poden obtenir per separat amb `f // g` i `f % g`.
- `gcd(f,g), xgcd(f,g), lcm(f,g)` (vegeu l'apartat d'enters).
- `factor(f)` (la factorització de f).
- `f.roots(f)`.

7.2.1 Booleans

- `f.is_irreducible()` retorna si f és irreductible.

- `f.is_primitive()` retorna si f és primitiu.

Vegem un exemple:

```
sage: P.<x> = PolynomialRing(GF(49,'a'))
sage: f = 4*x^5+5*x+2
sage: g = x^8 + 6*x^7 + x^2
sage: d,a,b = xgcd(f,g)
sage: d
1
sage: d == a*f + b*g
True
```

CHAPTER

EIGHT

COSSOS FINITS

8.1 Construcció de cossos finits i extensions

Per crear el cos finit de p^n elements escrivim:

```
sage: K = GF(p^n,'a')
```

o bé:

```
sage: K.<a> = GF(p^n)
```

Queda així definida també a com la classe de la indeterminada dels polinomis sobre $F = \mathbb{Z}/p\mathbb{Z}$ tal que $K = F[x]/(f(x))$. Només podem obviar la variable a quan $n = 1$.

També podem definir cossos finits forçant un determinat polinomi f de l'anell de polinomis sobre $\mathbb{Z}/p\mathbb{Z}$ i de grau n utilitzant:

```
sage: F = GF(q, modulus = f)
```

Donat un cos finit sempre podem conèixer-ne el cos primer escrivint `F.prime_subfield()`. Si hem definit F com a extensió d'un cos K direm que K és el cos base de F i el podrem conèixer escrivint `F.base_ring()`. Per a un cos que haguem definit directament es considera que el cos base és el cos primer.

8.2 Operacions

- Operacions bàsiques: `+, -, *, /, ^`.
- `K.polynomial()` retorna el polinomi que defineix K.
- `b.minimal_polynomial()` (o `b.minpoly()`) retorna el polinomi mínim de b.
- `b.multiplicative_order()` retorna l'ordre multiplicatiu, el mínim enter positiu n tal que $b^n = 1$.
- `len(K)` és equivalent a `K.cardinality()` retorna el nombre d'elements del cos K.
- `K.random_element()` retorna un element aleatori de K.

8.2.1 Booleans

- `f.is_primitive()` retorna si f és un polinomi primitiu.

CHAPTER

NINE

ALGEBRA LINEAL

9.1 Construcció de vectors

Donat un cos K, creem l'espai vectorial K^n mitjançant `VectorSpace(K,n)`, o escrivint `K^n`. Per crear vectors podem fer una coerció dins l'espai dels vectors corresponent o bé els podem definir directament:

```
sage: K = GF(9,'a')
sage: V3 = VectorSpace(K,3)
sage: v = V3([1,0,1])
sage: v
(1, 0, 1)
sage: v[1]
0

sage: w = vector(K,[1,2,3])
sage: w
(1, 2, 0)
sage: w[2]
0
```

9.2 Operacions

- Operacions bàsiques: `+, -, *`.
- `v.inner_product(w)` (producte escalar),
- `v.pairwise_product(w)` (producte vectorial a K^3).

9.3 Construcció de matrius

Donat un anell R, creem el conjunt de matrius de m files i n columnes mitjançant `MatrixSpace(R,m,n)`. Per crear matrius podem fer una coerció dins l'espai de les matrius corresponent o bé les podem definir directament:

```
sage: F9.<alpha> = GF(9)
sage: M = MatrixSpace(F9,2,3)
sage: M([alpha, 2*alpha, 3*alpha, alpha, alpha^2, alpha^3])
[      alpha      2*alpha            0]
[      alpha    alpha + 1  2*alpha + 1]
sage: matrix(2,3,[alpha,2*alpha,3*alpha,alpha,alpha^2,alpha^3])
```

(continues on next page)

(continued from previous page)

```
[      alpha     2*alpha           0]
[      alpha   alpha + 1 2*alpha + 1]
sage: matrix(3,2,[alpha,2*alpha,3*alpha,alpha,alpha^2,alpha^3])
[      alpha     2*alpha]
[          0       alpha]
[  alpha + 1 2*alpha + 1]
```

Si volem, podem especificar l'anell sobre el qual està definida una matriu. Així, les dues ordres següents ens donarien matrius diferents:

```
sage: m1 = matrix(Zmod(5),[[1,2],[3,4]]); m1
[1 2]
[3 4]
sage: m2 = matrix(Zmod(7),[[1,2],[3,4]]); m2
[1 2]
[3 4]
sage: m1 == m2
False
```

Per obtenir els elements d'una matriu m utilitzarem `m[i,j]` i per obtenir les seves files utilitzarem `m[i]`. Seguint l'exemple anterior,

```
sage: m = matrix(F9,[[alpha,2*alpha,3*alpha],[alpha,alpha^2,alpha^3]])
sage: m[0,1]
2*alpha
sage: m[1]
(alpha, alpha + 1, 2*alpha + 1)
```

Per sumar, restar i multiplicar per escalars es fa amb la notació usual. Per trobar la matriu inversa d'una matriu invertible m escriurem `m^-1`.

Per trobar les solucions d'un sistema lineal $xm = v$ on m és una matriu i v és un vector tenim `m.solve_left()` mentre que per resoldre el sistema $xm = v$ tenim `m.solve_right()`.

```
sage: m = matrix(Integers(),[[0,1],[2,0]])
sage: m
[0 1]
[2 0]
sage: v = vector(Integers(),[2,2])
sage: v
(2, 2)
sage: m.solve_left(v)
(2, 1)
sage: m.solve_right(v)
(1, 2)
```

També podem calcular submatrius. Per obtenir la submatriu 3×3 començant a l'entrada $(1, 1)$ d'una matriu 4×4:

```
sage: m = matrix(4, [1..16])
sage: m.submatrix(1, 1)
[ 6  7  8]
[10 11 12]
[14 15 16]
```

Ara només n'agafem dues files:

```
sage: m.submatrix(1, 1, 2)
[ 6  7  8]
[10 11 12]
```

I ara només una columna:

```
sage: m.submatrix(1, 1, 2, 1)
[ 6]
[10]
```

També es poden escollir 0 files o columnes:

```
sage: m.submatrix(1, 1, 0)
[]
```

9.4 Operacions amb matrius en general

- Operacions bàsiques: `+, -, *`.
- `A.nrows()` i `A.ncols()` retorna el nombre de files i columnes de la matriu.
- `m.transpose()` retorna la matriu transposada.
- `m.rank()`, `m.kernel()`, `m.image()` retorna el rang, nucli i imatge de la matriu.

9.4.1 Booleans

- `m.is_square()` (per saber si la matriu és quadrada).

9.5 Operacions amb matrius quadrades

- Operacions bàsiques: `+, -, *, ^`.
- `m.determinant()`, `m.trace()` i `m.inverse()` retorna el determinant, la traça i la inversa de la matriu (aquesta última, només en cas que existeixi).

9.5.1 Booleans

- `m.is_symmetric()`, `m.is_invertible()`, `m.is_singular()` (si la matriu és simètrica, invertible o singular (és a dir, no invertible)).
- `m.is_zero()`, `m.is_one()` (si la matriu és zero, o la matriu identitat).

CHAPTER

TEN

CÀLCUL DIFERENCIAL

Amb Sage es poden calcular les derivades i integrals de moltes funcions. Per exemple, per calcular la derivada de $\sin(u)$ respecte u, fem:

```
sage: u = var('u')
sage: diff(sin(u), u)
cos(u)
```

La quarta derivada de $\sin(x^2)$ s'obté mitjançant:

```
sage: diff(sin(x^2), x, 4)
16*x^4*sin(x^2) - 48*x^2*cos(x^2) - 12*sin(x^2)
```

Per calcular les derivades parcials de $x^2 + 17y^2$ respecte x i y fem:

```
sage: x, y = var('x,y')
sage: f = x^2 + 17*y^2
sage: f.diff(x)
2*x
sage: f.diff(y)
34*y
```

També podem calcular integrals, tant definides com indefinides. Per calcular $\int x\sin(x^2)\,dx$ i $\int_0^1 \frac{x}{x^2+1}\,dx$ fem:

```
sage: integral(x*sin(x^2), x)
-1/2*cos(x^2)
sage: integral(x/(x^2+1), x, 0, 1)
1/2*log(2)
```

També podem aconseguir una descomposició en fraccions parcials, per exemple de $\frac{1}{x^2-1}$:

```
sage: f = 1/((1+x)*(x-1))
sage: f.partial_fraction(x)
-1/2/(x + 1) + 1/2/(x - 1)
```

CHAPTER

ELEVEN

GRÀFICS

Amb Sage podem produir gràfics 2-D i 3-D.

11.1 Gràfics bidimensionals

Podem dibuixar cercles, línies, polígons; gràfics de funcions en coordenades cartesianes; i també en coordenades polars, corbes de nivell i gràfics de camps vectorials. Podem trobar més exemples de les funcions gràfiques del Sage a la documentació Sage Constructions

Per dibuixar un cercle groc de radi 1, centrat a l'origen, fem:

```
sage: circle((0,0), 1, rgbcolor=(1,1,0))
Graphics object consisting of 1 graphics primitive
```

I un cercle sòlid:

```
sage: circle((0,0), 1, rgbcolor=(1,1,0), fill=True)
Graphics object consisting of 1 graphics primitive
```

També podem assignar el cercle a una variable. Llavors no obtindrem cap dibuix:

```
sage: c = circle((0,0), 1, rgbcolor=(1,1,0))
```

I el podem visualitzar mitjançant `c.show()` o `show(c)`:

```
sage: c.show()
```

L'ordre `c.save('fitxer.png')` desa el dibuix al disc.

Amb l'opció `aspect_ratio = 1` podem aconseguir que els dos eixos estiguin a la mateixa escala:

```
sage: c.show(aspect_ratio=1)
```

També hauriem pogut utilitzar `show(c, aspect_ratio=1)`, o desar-ho amb l'ordre `c.save('fitxer.png', aspect_ratio=1)`.

Així obtenim el gràfic d'una funció bàsica:

```
sage: plot(cos, (-5,5))
Graphics object consisting of 1 graphics primitive
```

Si primer alliberem una variable, podem obtenir gràfics de funcions paramètriques:

```
sage: x = var('x')
sage: parametric_plot((cos(x),sin(x)^3),(x,0,2*pi),rgbcolor=hue(0.6))
Graphics object consisting of 1 graphics primitive
```

Podem combinar diversos gràfics en una de sol:

```
sage: x = var('x')
sage: p1 = parametric_plot((cos(x),sin(x)),(x,0,2*pi),rgbcolor=hue(0.2))
sage: p2 = parametric_plot((cos(x),sin(x)^2),(x,0,2*pi),rgbcolor=hue(0.4))
sage: p3 = parametric_plot((cos(x),sin(x)^3),(x,0,2*pi),rgbcolor=hue(0.6))
sage: show(p1+p2+p3, axes=false)
```

També podem crear polígons. Comencem amb una llista `L` de punts, i després utilitzem la funció `polygon` per dibuixar el polígon que té aquests punts com a frontera. Per exemple, un deltoide verd:

```
sage: L = [[-1+cos(pi*i/100)*(1+cos(pi*i/100)),\
....: 2*sin(pi*i/100)*(1-cos(pi*i/100))] for i in range(200)]
sage: p = polygon(L, rgbcolor=(1/8,3/4,1/2))
sage: p
Graphics object consisting of 1 graphics primitive
```

Si no volem veure els eixos, utilitzem `show(p, axes=false)`.

També podem afegir text a un gràfic:

```
sage: L = [[6*cos(pi*i/100)+5*cos((6/2)*pi*i/100),\
....: 6*sin(pi*i/100)-5*sin((6/2)*pi*i/100)] for i in range(200)]
sage: p = polygon(L, rgbcolor=(1/8,1/4,1/2))
sage: t = text("hipotrocoide", (5,4), rgbcolor=(1,0,0))
sage: show(p+t)
```

Un gràfic més complicat és el que mostra múltiples branques de la funció arcsinus: és a dir, el gràfic de $y = \sin(x)$ on x entre -2π i 2π, i rotat un angle de 45 graus. Així és com ho construirem en Sage:

```
sage: v = [(sin(x),x) for x in srange(-2*float(pi),2*float(pi),0.1)]
sage: line(v)
Graphics object consisting of 1 graphics primitive
```

Finalment, un exemple d'un gràfic de corbes de nivell:

```
sage: f = lambda x,y: cos(x*y)
sage: contour_plot(f, (-4, 4), (-4, 4))
Graphics object consisting of 1 graphics primitive
```

11.2 Gràfics tridimensionals

També podem aconseguir gràfics 3-D amb Sage. Per defecte, aquests gràfics es visualitzen amb el paquet *[Jmol]*, que permet rotar i ampliar l'objecte de manera interactiva, utilitzant el ratolí.

Per visualitzar una funció de la forma $f(x, y) = z$ utilitzem `plot3d`:

```
sage: x, y = var('x,y')
sage: plot3d(x^2 + y^2, (x,-2,2), (y,-2,2))
Graphics3d Object
```

També podem utilitzar `parametric_plot3d` per visualitzar una superfície paramètrica on cadascuna de les coordenades x, y, z ve determinada per una funció d'una o dues variables (normalment denotades per u i v). Així, el gràfic anterior també es pot visualitzar paramètricament de la manera següent:

```
sage: u, v = var('u, v')
sage: f_x(u, v) = u
sage: f_y(u, v) = v
sage: f_z(u, v) = u^2 + v^2
sage: parametric_plot3d([f_x, f_y, f_z], (u, -2, 2), (v, -2, 2))
Graphics3d Object
```

Finalment, podem utilitzar `implicit_plot3d`, per visualitzar les corbes de nivell d'una funció com $f(x, y, z)$. Aquesta ordre visualitza una esfera utilitzant la fórmula clàssica:

```
sage: x, y, z = var('x, y, z')
sage: implicit_plot3d(x^2 + y^2 + z^2 - 4, (x,-2, 2), (y,-2, 2), (z,-2, 2))
Graphics3d Object
```

CHAPTER

TWELVE

ÍNDEXS I TAULES

- genindex
- search

BIBLIOGRAPHY

[Jmol] Jmol: un visualitzador d'estructures químiques en 3D, de codi obert i escrit en Java i Javascript. http://www.jmol.org/.