Introducció al SAGE

Maria Bras Amorós

Índex

1 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`
- Tots els comandaments 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 comandes de sistema: `!`*comandes*
- Per llegir d'un fitxer *fin.sage*: `load fin.sage`
- Per llegir d'un fitxer *fin.sage* i escriure els resultats en un fitxer *fout*: `./sage fin.sage > fout`
- Per comentar una línia: `# ...`
  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 buscat comandes: `?`primeres lletres

2 Accions simples i expressions

2.1 Assignació de variables i escriptura

- Per assignar variables: `nom_variable=valor_variable`
- Per mostrar per pantalla:
  - El valor de les variables $var1, var2, \ldots$: `print var1, var2, ..`
  - Text: `print "text"`

2.2 Operadors relacionals

- $<, >$
- $=, !=$
- $>=, <=$
- `in`, `not in`

2.3 Operadors booleans

- `x and y`
- `x or y`
- `not x`

2.4 Funcions definides pel SAGE

SAGE té definides les seves pròpies funcions que depenen d'una o vàries variables (o cap variable). 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 següent manera:

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

També podríem assignar els dos valors de cop:

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

3 Composició d'accions

3.1 Composició condicional

- ```
  if ():
       ..
  elif ():
       ..
  else:
       ..
  ```

3.2 Composició iterativa

- `for i in [i_inicial..i_final]:`
- `for i in llista:`
- `while condicio:`
  ```

4 Conjunts i seqüències

4.1 Conjunts i seqüències

Tant els conjunts com les seqüències 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 [ ]. Els conjunts es construeixen a partir de seqüències $set([..])$.

Per exemple,

```
sage: t = [ (-11)^2, (-7)^2, (-5)^2, (-3)^2, 3^2, 5^2, 7^2, 11^2 ]
sage: q = set(t)
sage: t
[121, 49, 25, 9, 9, 25, 49, 121]
sage: q
{121, 25, 9, 49}
```

La i-èssima entrada d'una seqüència (o d'un conjunt) `t` és `t[i]`. Però compte perquè SAGE enumera les posicions des de 0!!!

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

SAGE té constructors especials per a seqüències. Les expressions

`[a..b]` i `[a,a+k..b]`

designen respectivament les progressions $[a, a+1, a+2, \ldots, b]$ i $[a, a+k, a+2k, \ldots, \tilde{b}]$ on $\tilde{b}$ 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 condició és cert. Per exemple, hauríem pogut crear `t` de la següent manera:

```
sage: t = [ n^2 for n in [-11,-9..11] if is_prime(abs(n)) ]
```

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

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

4.2 Operacions

- `len(S)` (cardinal de S) per conjunts i seqüències.
- `sum(S)` (suma dels elements de S) per conjunts i seqüències.
- `prod(S)` (producte dels elements de S) per conjunts i seqüèncie
- `A.union(B),A.intersection(B),A.difference(B)` ($A \cup B$, $A \cap B$, $A \setminus B$) per conjunts.

- `S + T` (`S` concatenat amb `T`) per conjunts i seqüències.
- `min(S),max(S)` per conjunts i seqüències.
- `S.append(x)` (afegir x al final de S) per seqüències.
- `S.remove(x)` (esborrar x de S) per seqüències.
- `S.index(x)` (posició de l'element x dins de S) per seqüències.
- `S.insert(i,x)` (moure una posició els elements amb índex igual o superior a i i afegir x a la posició i) per seqüències.
- `S.reverse()` (invertir) per seqüències.
- `S.sort()` (ordenar) per seqüències.
- `S[randint(0,len(S))]` (element aleatori de S) per seqüències.

Booleans

- `x in C; x not in C` per conjunts i seqüències.
- `A.issubset(B); A.issuperset(B)` per conjunts.
- `D == C,D != C` per conjunts i seqüències.

5 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, ..):
  operacions necessaries
  return (val1,val2,..)
```

Per exemple,

```
def solucions_reals_equacio_segon_grau(a,b,c):
  discriminant=b^2-4.0*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/(2.0*a)
    return (sol)
  else:
    print "no hi ha solucions reals"
    return ()
```

Observem que els espais són rellevants. Ara podem cridar-la:

```
sage: solucions_reals_equacio_segon_grau(1,0,-1)
hi ha dues solucions reals
(1.00000000000000, -1.00000000000000)
```

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
1.00000000000000
sage: b
-1.00000000000000
```

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. Per tant una variable externa a la funció amb nom igual a una de les variables locals no es modificarà a causa de la funció.

6 Enters, anells $\mathbb{Z}_n$, racionals i reals

6.1 Enters

L'anell dels enters el cridem `Integers()` A continuació llistem algunes funcions pels enters:

- Operacions bàsiques: `+, -, *, /, ^`
- `n // m` (quocient de dividir n entre m)
- `n % m` (n mòdul 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, incloent a i b)
- `random_prime(a)` (un primer aleatori menor o igual que a)
- `divisors(n), prime_divisors(n), number_of_divisors(n)`
- `gcd(m,n), gcd(S), lcm(m,n), lcm(S)` (on `S` és una seqüència d'enters)
- `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)`
- `factorial(n)`

Booleans

- `is_odd(i), is_even(i)`
- `is_prime(i), is_prime_power(i)`
- `is_square(n)`

6.2 Anells $\mathbb{Z}_n$

L'anell de residus mòdul n el cridem `Integers(n)`. Si posem $R = Integers(n)$ 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)` (invers de x mod m)
- `solve_mod(expr1 == expr2,m)` (resol equacions amb congruències; és important que les incògnites s'hagin alliberat abans, per exemple amb `var('x')`)
- `primitive_root(n)`
- `multiplicative_order(x), additive_order(x)`

Booleans

- is_field() per saber si un conjunt és un cos.
- is_unit() per saber si un element és invertible.

El següent exemple us pot ser il·lustratiu.

```
sage: is_prime(19)
True
sage: Z19=Integers(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: primitive_root(19)
2
sage: additive_order(Z19(2))
19
sage: multiplicative_order(Z19(2))
18
sage: multiplicative_order(Z19(5))
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 de les fraccions a/b es crida `RationalField()`. Algunes operacions que podem fer amb els racionals són

- Operacions bàsiques: `+, -, *, /, ^`
- `numerator(q)`
- `denominator(q);`

6.4 Reals

El cos dels reals el cridem `RealField()`. Algunes funcions pels reals són:

- Operacions bàsiques: `+, -, *, /, ^`
- `ceil(r), floor(r)`
- `abs(r)`
- `sqrt(r)`

6.5 Equacions

Es poden resoldre equacions utilitzant `solve`. Escrivint ?`solve` el sage ens dóna una explicació molt extensa. Aquí 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 == -1/2*sqrt(-I*sqrt(3) + 3)*sqrt(2)],
 [x == -1/2*I*sqrt(3) - 1/2, y == 1/2*sqrt(-I*sqrt(3) + 3)*sqrt(2)],
 [x == 1/2*I*sqrt(3) - 1/2, y == -1/2*sqrt(I*sqrt(3) + 3)*sqrt(2)],
```

```
 [x == 1/2*I*sqrt(3) - 1/2, y == 1/2*sqrt(I*sqrt(3) + 3)*sqrt(2)],
 [x == 0, y == -1],
 [x == 0, y == 1]]
sage: solve([sqrt(x) + sqrt(y) == 5, x + y == 10], x, y)
[[x == -5/2*I*sqrt(5) + 5, y == 5/2*I*sqrt(5) + 5], [x == 5/2*I*sqrt(5) + 5, y == -5/2*I*
sage: for solution in solutions: print solution[x].n(digits=3), ",", solution[y].n(digits
-0.500 - 0.866*I , -1.27 + 0.341*I
-0.500 - 0.866*I , 1.27 - 0.341*I
-0.500 + 0.866*I , -1.27 - 0.341*I
-0.500 + 0.866*I , 1.27 + 0.341*I
0.000 , -1.00
0.000 , 1.00
```

7 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: 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()`
- `f.coeffs()` retorna tots els coeficients de f mentres que `f.coefficients()` només retorna els coeficients no-nuls.
- `f.leading_coefficient(f)`
- `parent(f)` (ens diu a on pertany el polinomi f)
- `divmod(f,g)` (quocient i residu de dividir f entre g)
- `f // g, f % g`
- `gcd(f,g), xgcd(f,g), lcm(f,g)` (vegeu l'apartat d'enters)
- `factor(f)`
- `f.roots(f)`

Booleans

- `f.is_irreducible()`
- `f.is_primitive()`

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
```

8 Cossos finits

8.1 Construcció de cossos finits i extensions

Per crear els cossos finits de q o p^n elements escrivim

```
sage: K:=GF(q,'a')
```

o bé

```
sage: K.<a>=GF(q)
```

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

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

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

8.2 Operacions

- Operacions bàsiques: `+, -, *, /, ^`
- `K.polynomial()`
- `minimal_polynomial(b)` equivalent a `minpoly(b)`
- `multiplicative_order(b)`
- `len(K)` equivalent a `K.cardinality()`
- `K.random_element()`

Booleans

- `f.is_primitive()`

9 Algebra lineal

9.1 Construcció de vectors

Donat un cos `K`, denotem el conjunt de vectors de K^n per `VectorSpace(K,n)`. Per crear vectors podem fer una coerció dins l'espai dels vectors corresponent o bé els podem definir directament. Per exemple,

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

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

```
sage: v
(1, 2, 0)

sage: v[1]
0
sage: w[1]
2
```

9.2 Operacions

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

9.3 Construcció de matrius

Donat un anell `R`, denotem el conjunt de matrius de m files i n columnes `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])
[      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 que està definida una matriu. Així les dues comandes següents ens donarien matrius diferents.

```
sage: matrix(Integers(3),[[1,2],[3,4]])
[1 2]
[0 1]
sage: matrix(Integers(4),[[1,2],[3,4]])
[1 2]
[3 0]
```

Per cridar els elements d'una matriu utilitzarem `[,]` i per cridar les seves files utilitzarem `[]`. 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)
```

Per calcular submatrius vegem l'exemple de la guia de SAGE:

```
Take the 3 x 3 submatrix starting from entry (1,1)
in a 4 x 4 matrix:

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

Same thing, except take only two rows:

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

And now take only one column:

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

You can take zero rows or columns if you want:

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

9.4 Operacions amb matrius en general

- Operacions bàsiques: `+, -, *`
- `A.nrows(); A.ncols()`
- `m.transpose()`
- `rank(m), kernel(m), image(m)`

Booleans

- `m.is_square()`

9.5 Operacions amb matrius quadrades

- Operacions bàsiques: `+`, `-`, `*`, `^`
- `m.determinant()`
- `m.trace()`
- `m^-1`

Booleans

- `m.is_symmetric()`
- `m.is_invertible(),m.is_singular()`
- `m.is_zero(),m.is_one()`