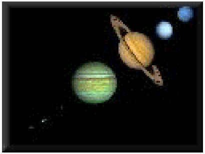

Macaulay2:

una breve introduzione

Luca Amata

7 luglio 2022

Dipartimento di Scienze Matematiche e Informatiche,
Scienze Fisiche e Scienze della Terra,
Università degli Studi di Messina

Introduzione

Introduzione

- Il *Macaulay2* [1] è un sistema di computer algebra (CAS) che permette di svolgere operazioni simboliche specializzate nel contesto dell'algebra commutativa o della combinatoria. Oltre alle istruzioni di base sono presenti varie librerie per la gestione di problemi specifici.
- Il sito ufficiale, che fornisce informazioni sia per la sintassi del linguaggio che per le caratteristiche tecniche, è il seguente:

  `faculty.math.illinois.edu/Macaulay2/`.
- La guida ufficiale contiene diversi percorsi tematici, di livello introduttivo o specialistico, non soltanto per chi vuole approcciarsi al linguaggio ma anche per chi è rivolto alla risoluzione di problemi avanzati. Può essere consultata alla pagina:

  `faculty.math.illinois.edu/Macaulay2/doc/Macaulay2-1.19.1/share/doc/Macaulay2/Macaulay2Doc/html/`.

Introduzione

- Per trovare istruzioni dettagliate per l'installazione di *Macaulay2* su diverse piattaforme, ci si può riferire alla pagina:

  `faculty.math.illinois.edu/Macaulay2/Downloads/`

  Per installare le versioni più recenti su sistemi *Microsoft Windows* è necessario emulare il sistema *Linux* (ad esempio tramite *Windows Subsystem for Linux*).
- È anche possibile utilizzare *Macaulay2* nella versione online senza installare alcun software (potrebbe essere richiesta un'iscrizione gratuita al portale):
  - `https://habanero.math.cornell.edu:3690/`
  - `www.unimelb-macaulay2.cloud.edu.au/` (by Paul Zinn-Justin [3])
  - `https://cocalc.com/` (Run CoCalc now, Notebook: Terminal, M2)

Linguaggio base

Ambiente

- Il *Macaulay2* utilizza un linguaggio procedurale e *case sensitive*, dispone dei comuni operatori e permette la costruzione di funzioni.
- Ad ogni accesso vengono richiamate alcune librerie non standard:

```
Macaulay2, version 1.19.0.1
with packages:  ConwayPolynomials, Elimination, IntegralClosure, InverseSystems,
LLLBases, MinimalPrimes, PrimaryDecomposition, ReesAlgebra, Saturation, TangentCone
i1 :
```

- Il prompt dei comandi è numerato e per inizializzarlo nuovamente bisogna eseguire il reset dell'ambiente di sviluppo.
- Per utilizzare **librerie** (packages) non caricate automaticamente all'avvio del client, si può utilizzare il seguente comando:

```
i1 :  loadPackage "<libreria>"
```

- Istruzioni di gestione dell'ambiente di sviluppo utili sono:
  - help <istr> (o viewhelp): restituisce la documentazione di aiuto relativa all'istruzione <istr>;
  - clearAll: elimina dalla memoria tutte le variabili e i risultati.
  - oo: variabile che viene aggiornata ad ogni operazione effettuata con successo e contiene l'ultimo valore restituito.
  - describe <istr>: restituisce informazioni dettagliate sulla struttura

Insiemi numerici

- *Macaulay2* gestisce gli insiemi numerici comuni tramite i simboli:

  `ZZ, QQ, RR, CC`.
- L'unità immaginaria è identificata da `ii`, π da `pi` (`+pi` per il valore numerico) e per e è possibile utilizzare la funzione `exp`.
- La priorità viene data agli insiemi più "piccoli": quella massima è data agli interi.
- Per riferirsi ad un elemento di un insieme specifico, ad esempio dei Reali, si può utilizzare l'operatore `_RR`: quindi `3_RR` (o equivalentemente `3.`) sarà un'unità di $\mathbb{R}$. Nel caso reale, per forzare la precisione è possibile scrivere `numeric_100 pi`.
- I campi finiti, in generale, possono essere ottenuti tramite l'istruzione `GF(p,n)` che genera il campo finito di ordine p^n (p primo).
- Le istruzioni `promote(x,R)`, `lift(x,R)` e `sub(x,R)` permettono di immergere/proiettare l'elemento `x` nell'anello `R`, quando possibile.
- Tutto ciò che non fa parte del linguaggio o che non è stato assegnato in precedenza viene identificato come simbolo esterno.

Operazioni

- Le operazioni (`+`, `-`, `*`, `/`, `^`) non sono mai sottintese e devono essere sempre esplicitate.
- Sono presenti le funzioni matematiche standard (`sqrt`, `sin`, ...). Sono disponibili anche alcune funzioni di calcolo combinatorio e per ottenere valori casuali: `partitions`, `subsets`, `binomial`, ..., `random`.
- L'operatore `^`, applicato ad un anello, permette di ottenere il modulo libero di rango specificato sull'anello (`ZZ^2`, `CC^4`, ...), mentre i quozienti di `ZZ` tramite numeri primi si ottengono tramite l'operatore `/` (`ZZ/2`, `ZZ/7`, etc...). L'operatore `**` permette di ottenere il prodotto cartesiano/tensoriale fra due strutture.
- L'operatore `=` serve ad eseguire assegnazioni, mentre l'operatore di confronto qualitativo è `==`. L'operatore `===` invece valuta l'uguaglianza delle strutture così come sono state definite. L'operatore binario `?` restituisce la relazione esistente fra due quantità, se confrontabili.
- Le stringhe sono identificate da virgolette `"str"`, l'operatore `_i` ne restituisce `i`-simo carattere, mentre `#` ne restituisce la lunghezza. Esistono varie funzioni per gestirle (`|`, `ascii`, `substring`, ...).

Macaulay2

RESET INTERRUPT UPLOAD FILE

```
i1 :  5+5*3^2
o1 =  50
i2 :  4/3
      4
o2 =  -
      3
o2 :  QQ
i3 :  numeric_100 oo
o3 =  1.333333333333333333333333333333
o3 :  RR (of precision 100)
i4 :  sqrt 2
o4 =  1.414213562373
o4 :  RR (of precision 53)
i5 :  sqrt(-1)
o5 =  ii
o5 :  CC (of precision 53)
i6 :  cos numeric pi
o6 =  -1
o6 :  RR (of precision 53)
i7 :  sqrt oo
o7 =  ii
o7 :  CC (of precision 53)
```

```
i8 :  x
o8 =  x
o8 :  Symbol
i9 :  s="abcdef"
o9 =  abcdef
i10 :  s_0 | s_(#s-1)
o10 =  af
i11 :  M=ZZ^3
o11 =  Z^3
o11 :  Z-module, free
i12 :  0_M
       |0|
o12 =  |0|
       |0|
o12 :  Z^3
i13 :  F=ZZ/5
       Z
o13 =  -
       5
o13 :  QuotientRing
i14 :  7_F
o14 =  2
o14 :  F
```

Funzioni

Il *Macaulay2* permette di definire funzioni personalizzate tramite l'operatore ->. La sintassi prevede l'utilizzo di un'assegnazione per determinare il nome della funzione. La parte che precede -> indica l'insieme delle variabili indipendenti (o parametri), quella che segue l'operatore specifica le operazioni che la funzione deve svolgere per ottenere l'output:

```
<nome>=(<p1,..., pn>) -> (<istr1>; ...; <istrk>; return <var>)
```

Esempio

```
i1 :  f=(a,b)->(r=sqrt(a^2+b^2); return r)
o1 =  f
o1 :  FunctionClosure
i2 :  f(1,1)
o2 =  1.4142135623731
o2 :  RR (of precision 53)
i3 :  f(3,4)
o3 =  5
o3 :  RR (of precision 53)
```

Strutture dati

- La struttura dati fondamentale è la **lista**, un vettore ordinato di elementi indicizzati a partire da 0. Esistono anche diverse varianti, come le sequenze `()`, i vettori `[]` o le liste aggiornabili.
- Per ottenere il numero di elementi di una lista `L`, basta premettere al suo nome l'operatore `#`: `#L`. Per ottenere invece l'elemento al posto *i*-esimo basta postporre al suo nome l'operatore `#i`: `L#i` (quest'ultimo funziona in modo circolare, modulo la lunghezza della lista). L'operatore `#i` in sola lettura può essere sostituito in alcuni casi da `_i`.
- L'istruzione `m..n` crea la sequenza `(m,m+1,...,n-1,n)` se $m \leq n$, altrimenti crea la sequenza nulla. L'struzione `k:n` crea la sequenza di lunghezza `k` i cui valori sono tutti `n`: `(n,...,n)`.
- Funzioni di conversione permettono di passare da una struttura ad un'altra: `toList`, `toSequence`, ...
- Il tipo di lista `MutableList` permette la modifica dei singoli elementi. Viene dichiarata tramite l'operatore `new` e visualizzata tramite `peek`. In questo caso il singolo elemento della lista `M` può essere utilizzato sia in lettura che in scrittura utilizzando il riferimento `M#i`.

Macaulay2

RESET INTERRUPT UPLOAD FILE

```
i1 :  l={a,1,ii}
o1 =  {a, 1, ii}
o1 :  List
i2 :  s=(a,1,ii)
o2 =  (a, 1, ii)
o2 :  Sequence
i3 :  v=[a,1,ii]
o3 =  [a, 1, ii]
o3 :  Array
i4 :  #l
o4 =  3
i5 :  s#2
o5 =  ii
o5 :  Constant
i6 :  v#(-1)
o6 =  ii
o6 :  Constant
```

```
i7 :  l_1
o7 =  1
i8 :  m = new MutableList from {a,b,c}
o8 =  MutableList{...3...}
o8 :  MutableList
i9 :  peek m
o9 =  MutableList{a, b, c}
i10 :  m#1=1
o10 =  1
i11 :  m#2=ii
o11 =  ii
o11 :  Constant
i12 :  peek m
o12 =  MutableList{a, 1, ii}
i13 :  toList(m)===l
o13 =  true
```

Gestione liste: elenco funzioni

Osservazione
Nessuna di esse modificherà automaticamente la lista `L`: per mantenere le modifiche si dovrà provvedere all'assegnazione.

- `append(L,x)`: aggiunge l'elemento `x` in coda
- `prepend(x,L)`: aggiunge l'elemento `x` in testa
- `insert(n,x,L)`: inserisce l'elemento `x` nella posizione `n`
- `switch(m,n,L)`: inverte l'elemento di posto `m` con quello di posto `n`
- `delete(x,L)`: elimina tutti gli elementi uguali a `x`
- `drop(L,m,n)`: elimina gli elementi fra le posizioni `m` ed `n`
- `take(L,m,n)`: restituisce gli elementi fra le posizioni `m` ed `n`
- `unique(L)`: restituisce gli elementi senza ripetizioni
- `reverse(L)`: inverte l'ordine degli elementi
- `sort(L), rsort(L)`: ordina in modo crescente/descrescente la lista
- `flatten L`: elimina le eventuali nidificazioni (di un livello)
- `join(L,M) o L|M`: data un'altra lista `M`, concatena le due liste

Gestione liste

Sulle liste è anche possibile effettuare delle operazioni logico-aritmetiche che tengano conto della natura degli elementi in esse contenuti.

- `max L (o maxPosition)`: restituisce il valore massimo (o l'indice)
- `min L (o minPosition)`: restituisce il valore minimo (o l'indice)
- `L±M`: restituisce le somme (differenze) termine a termine (`#L==#M`)
- `n*L`: restituisce i prodotti di `n` per ogni elemento di `L`
- `apply(L,f) o L/f o f\L`: applica la funzione `f` ad ogni elemento
- `select(L,cond)`: restituisce gli elementi che soddisfano `cond`. Esistono funzioni predefinite (`even`, `odd`, ...) o se ne possono costruire (ad es. `(i->i>0)`)
- `position(L,cond) (o positions)`: restituisce la prima posizione (o tutte le posizioni) dell'elemento soddisfa `cond`
- `number(L,cond)`: conta il numero di elementi che soddisfano `cond`
- `sum L`: somma fra loro gli elementi della lista, se possibile
- `product L`: moltiplica fra loro gli elementi della lista, se possibile
- `isSubset(L,M)`: verifica se `L` è contenuta in `M`

Macaulay2 RESET INTERRUPT UPLOAD FILE

```
i1 :  L = toList(-2..5)
o1 =  {-2, -1, 0, 1, 2, 3, 4, 5}
o1 :  List
i2 :  L=prepend(5,L)
o2 =  {5, -2, -1, 0, 1, 2, 3, 4, 5}
o2 :  List
i3 :  L=delete(5,L)
o3 =  {-2, -1, 0, 1, 2, 3, 4}
o3 :  List
i4 :  M=take(L,{1,5})
o4 =  {-1, 0, 1, 2, 3}
o4 :  List
i5 :  M=M|{{4,{5}}}
o5 =  {-1, 0, 1, 2, 3, {4, {5}}}
o5 :  List
i6 :  flatten M
o6 =  {-1, 0, 1, 2, 3, 4, {5}}
o6 :  List
i7 :  flatten flatten M
o7 =  {-1, 0, 1, 2, 3, 4, 5}
o7 :  List
```

```
i8 :  L = toList(-3..3)
o8 =  {-3, -2, -1, 0, 1, 2, 3}
o8 :  List
i9 :  L=L/(x->x^2)
o9 =  {9, 4, 1, 0, 1, 4, 9}
o9 :  List
i10 :  number(L,odd)
o10 =  4
i11 :  number(L,even)
o11 =  3
i12 :  M=select(L,x->x<5 and x%2==1)
o12 =  {1, 1}
o12 :  List
i13 :  positions(M,even)
o13 =  {}
o13 :  List
i14 :  isSubset(M,L)
o14 =  true
i15 :  isSubset(M|{1},L)
o15 =  true
```

Matrici

Il *Macaulay2* considera le matrici come trasformazioni lineari fra moduli. Per definire una matrice basta utilizzare l'istruzione:

`matrix` $\{\{a_{1,1}, \ldots, a_{1,n}\}, \ldots, \{a_{m,1}, \ldots, a_{m,n}\}\}$

- `+, -, *, ^`: operazioni estese all'anello delle matrici
- `**`: prodotto tensoriale fra due matrici
- `A|B (o A||B)`: concatena matrici compatibili per colonne (o righe)
- `A_{indici} (o A^{indici})`: estrae dalla matrice le colonne (o le righe) individuate dagli indici nella lista
- `entries A`: restituisce le entrate della matrice sotto forma di lista
- `rank A`: calcola il rango di una matrice
- `det A`: calcola il determinante di una matrice quadrata
- `transpose A`: restituisce la matrice trasposta
- `inverse A`: restituisce la matrice inversa, quando possibile
- `source A`: restituisce il dominio della trasformazione lineare
- `target A`: restituisce il codominio della trasformazione lineare
- `ring A`: restituisce l'anello a cui appartengono le entrate di `A`

Macaulay2 RESET INTERRUPT UPLOAD FILE

```
i1 :  A = matrix {{1,2,3},{4,5,6}}
o1 =  | 1 2 3 |
      | 4 5 6 |
o1 :  Matrix ZZ^2 <--- ZZ^3
i2 :  B=A|A
o2 =  | 1 2 3 1 2 3 |
      | 4 5 6 4 5 6 |
o2 :  Matrix ZZ^2 <--- ZZ^6
i3 :  C=B||B
o3 =  | 1 2 3 1 2 3 |
      | 4 5 6 4 5 6 |
      | 1 2 3 1 2 3 |
      | 4 5 6 4 5 6 |
o3 :  Matrix ZZ^4 <--- ZZ^6
i4 :  det C_{0..3}
o4 =  0
i5 :  rank C_{0..3} == rank B
o5 =  true
```

```
i6 :  A = random(ZZ^3,ZZ^2)
o6 =  | 9 6 |
      | 4 3 |
      | 2 4 |
o6 :  Matrix ZZ^3 <--- ZZ^2
i7 :  target A
o7 =  ZZ^3
o7 :  ZZ-module, free
i8 :  numgens target A
o8 =  3
i9 :  entries A
o9 =  {{9, 6}, {4, 3}, {2, 4}}
o9 :  List
i10 :  describe A
o10 =  map(ZZ^3 , ZZ^2 ,
          {{9, 6}, {4, 3}, {2, 4}})
```

Mappe

Per definire una **mappa**, $f: R \to S$, tramite la matrice M si può scrivere:

```
<f>=map(<S>,<R>,<M>)
```

Esempio

```
i1 :  R=ZZ^2;
i2 :  S=ZZ^3;
i3 :
M=id_R||matrix{toList(2:0)}
i4 :  f=map(S,R,M)
o4 =  | 1 0 |
      | 0 1 |
      | 0 0 |
o4 :  Matrix ZZ^3 <--- ZZ^2
i5 :  f(0_R)
o5 =  | 0 |
      | 0 |
      | 0 |
o5 :  ZZ^3
i6 :  f(R_0)
o6 =  | 1 |
      | 0 |
      | 0 |
o6 :  ZZ^3
```

```
i7 :  a=5*R_0-2*R_1
o7 =  |  5 |
      | -2 |
o7 :  ZZ^2
i8 :  f(a)
o8 =  |  5 |
      | -2 |
      |  0 |
o8 :  ZZ^3
i9 :  b=R_0+3*R_1
o9 =  | 1 |
      | 3 |
o9 :  ZZ^2
i10 :  f(2*a-3*b)==2*f(a)-3*f(b)
o10 =  true
```

Strutture di controllo

La struttura **decisionale** permette di eseguire un blocco di istruzioni o, in alternativa, un altro in base al verificarsi di una condizione. La sintassi è:

```
if <cond> then (<istr1>; ...<istrk>)
          else (<istr1>; ...<istrq>)
```

Le strutture di **ripetizione** permettono di ripetere un blocco di istruzioni. Le sintassi, completa dei parametri sono opzionali, sono le seguenti:

```
- for <i> from <m> to <n> [when <cond>]
                      list <funct>
                      do (<istr1>; ...<istrk>)

- for <x> in <L> [when <cond>]
                 list <funct>
                 do (<istr1>; ...<istrk>)

- while <cond>
        list <funct>
        do (<istr1>; ...<istrk>)
```

Macaulay2 RESET INTERRUPT UPLOAD FILE

```
i1 :  s=0; p=1;
i2 :  for i from 1 to 20 when i<=4 list i^2 do (s=s+i; p=p*i)
o2 =  {1, 4, 9, 16}
o2 :  List
i3 :  {s, p}
o3 =  {10, 24}
o3 :  List
i4 :  pot=(a,b)->(p=1; for i in 1..b do p=p*a; p)
o4 =  pot
o4 :  FunctionClosure
i5 :  pot(2,4)
o5 =  16
i6 :  fatt=n->(p=1; while n>1 do (p=p*n; n=n-1); return p)
o6 =  fatt
o6 :  FunctionClosure
i7 :  fatt\toList(0..5)
o7 =  {1, 1, 2, 6, 24, 120}
07 :  List
i8 :  fact=n->(p=1; if n>1 then p=n*fact(n-1); p)
o8 =  fact
o8 :  FunctionClosure
i9 :  fact\toList(0..5)
o9 =  {1, 1, 2, 6, 24, 120}
o9 :  List
```

Anelli dei polinomi

Definizioni standard

- Per definire un anello di polinomi basta far seguire all'identificativo di un anello/campo il vettore (finito) delle indeterminate `[a,b,...,z]`.
- È buona norma assegnare un nome agli anelli creati. In caso siano stati definiti più anelli verrà data precedenza all'ultimo definito.

Osservazione
Anelli definiti con istanze diverse, anche se con le medesime istruzioni, saranno considerati differenti ed i loro elementi non saranno confrontabili. Per poter effettuare confronti bisognerà operare delle conversioni tramite mappe.

Esempio

```
i1 :  R=QQ[x,y,z]
o1 =  R
o1 :  PolynomialRing
i2 :  S=QQ[x,y,z]
o2 =  S
o2 :  PolynomialRing
i3 :  R===S
o3 =  false
i4 :  clearAll
```

```
i5 :  R=QQ[x_1..x_6]
o5 =  R
o5 :  PolynomialRing
i6 :  x_1*x_4
o6 =  x_1 x_4
o6 :  R
i7 :  p=oo+x_1*x_5*x_6
o7 =  x_1 x_4 + x_1 x_5 x_6
o7 :  R
```

Anelli particolari

- È possibile lavorare anche su particolari anelli di polinomi (parzialmente) non commutativi. Per definirli basta utilizzare il parametro opzionale `SkewCommutative` tramite l'operatore =>:

```
R=QQ[x,y,z,SkewCommutative=>true] (o =>{x,y})
```

- Per lavorare su un'*algebra di Weyl* si può utilizzare il parametro opzionale `WeylAlgebra`:

```
R=QQ[x,y,dx,dy,WeylAlgebra=>{x=>dx, y=>dy}]
```

- Per costruire l'*algebra simmetrica* su un modulo si può utilizzare la seguente istruzione:

```
R=symmetricAlgebra (M, VariableBaseName=>x]
```

- Per costruire un anello dei *polinomi di Laurent* si può utilizzare la seguente istruzione (attualmente a causa di un bug tale costruzione non è possibile con l'ordinamento GRevLex):

```
R=QQ[x,y,z,Inverses=>true,MonomialOrder=>Lex]
```

Gestione anelli

Segue un elenco di funzioni per la gestione di un anello dei polinomi `R`.

- `**`: prodotto tensoriale fra due anelli
- `use S`: forza l'utilizzo dell'anello `S` prima definito
- `vars R`: restituisce una matrice riga contenente le indeterminate
- `gens R`: restituisce la lista delle indeterminate dell'anello
- `numgens R`: restituisce il numero delle indeterminate dell'anello
- `index <var>`: restituisce l'indice associato all'indeterminata `<var>`
- `0_R e 1_R`: restituiscono, rispettivamente, lo zero e l'unità di `R`
- `basis(d,R)`: restituisce la base, come spazio vettoriale sul campo di definizione, della componente di grado `d` di `R`
- `p % vars R (o p % 1_R)`: verifica l'appartenenza di `p` ad `R`
- `describe R`: fornisce informazioni complete (che è possibile reperire dinamicamente attraverso l'istruzione `options`)

Esempio

```
i1 :  describe (QQ[x,y,z])
o1 =  QQ[x..z, Degrees => {3:1}, Heft => {1}, MonomialOrder =>
                                 {MonomialSize => 32}
                                 {GRevLex => {3:1}}
                                 {Position => Up}, DegreeRank => 1]
```

Ordinamenti monomiali

- L'anello dei polinomi contiene informazioni riguardanti la graduazione delle indeterminate (standard di default) e l'**ordinamento monomiale** (lessicografico graduato inverso di default).
- L'ordinamento monomiale associato all'anello influenzerà non soltanto la scrittura dei polinomi ma anche molte operazioni che riguardano le elaborazioni (soprattuto tramite **basi di Gröbner**).

Per definire l'ordinamento monomiale di un anello si può assegnare il parametro opzionale `MonomialOrder` tramite l'operatore =>:

```
R=QQ[x,y,z,MonomialOrder=>Lex]
```

Come valori per il `MonomialOrder` è possibile scegliere fra:

- `GRevLex`: ordinamento lessicografico graduato inverso
- `Lex`: ordinamento lessicografico
- `GLex`: ordinamento lessicografico graduato
- `RevLex`: ordinamento lessicografico inverso
- `Weights`: ordinamento per assegnazione di pesi alle indeterminate
- `Eliminate`: ordinamento di eliminazione delle indeterminate

Gestione polinomi

I polinomi possono essere elaborati tramite l'utilizzo delle operazioni standard su anelli. Sono inoltre presenti funzioni a loro dedicate:

- `terms p`: restituisce una lista contenente i termini del polinomio `p`
- `monomials p`: restituisce una matrice riga contenente i monomi di `p`
- `support p`: restituisce la lista delle indeterminate presenti in `p`
- `degree p`: restituisce il grado di `p`
- `degree(x,p)`: restituisce il grado dell'indeterminata `x` in `p`
- `exponents p`: restituisce la lista dei multigradi dei monomi di `p`
- `size p`: restituisce il numero dei monomi di `p`
- `homogenize(p,x)`: omogenizza `p` utilizzando l'indeterminata `x`
- `leadMonomial p (o leadTerm, leadCoefficient)`: restituisce il monomio (o termine, coefficiente) **iniziale** di `p` (per l'ordinamento monomiale dell'anello di appartenenza)
- `factor p`: restituisce la fattorizzazione del polinomio in irriducibili
- `roots p`: restituisce le radici del polinomio (in una variabile)

Macaulay2

RESET INTERRUPT UPLOAD FILE

```
i1 :  R=QQ[x,y,z]
o1 =  R
o1 :  PolynomialRing
i2 :  p=5*x*y+x*z^2-3*y*z
o2 =  x z^2 + 5x y - 3y z
o2 :  R
i3 :  S=QQ[x,y,z,MonomialOrder=>Lex]
o3 =  S
o3 :  PolynomialRing
i4 :  q=5*x*y+x*z^2-3*y*z
o4 =  5x y + x z^2 - 3y z
o4 :  S
i5 :  leadMonomial p
o5 =  x z^2
o5 :  R
i6 :  leadMonomial q
o6 =  x y
o6 :  S
i7 :  f=map(S,R)
o7 =  map(S,R,x, y, z)
o7 :  RingMap S <--- R
i8 :  leadMonomial f(p)
o8 =  x y
o8 :  S
```

```
i9 :  R=QQ[w..z]
o9 =  R
o9 :  PolynomialRing
i10 :  random(R^3,R^{2:-1})
o10 =  | 5/8w+x+9y+1/4z        2/7w+1/2x+1/8y+3/2z |
       | w+8/7x+2/5y+2/3z      3w+1/4x+3/5y+7/10z  |
       | 1/10w+7/2x+9/5y+1/2z 3/8w+9/5x+1/3y+z     |
o10 :  Matrix R^3 <--- R^2
i11 :  isHomogeneous oo
o11 =  true
```

Ideali

Per costruire un **ideale** di un anello di polinomi si usa l'istruzione:

`ideal L`,

dove `L` è la lista (o sequenza o matrice) dei generatori dell'ideale.

Come per gli anelli, per ottenere informazioni sugli ideali è possibile utilizzare le istruzioni:

- `gens I`: restituisce la matrice dei generatori di `I` (come costruito)
- `numgens I`: restituisce il numero dei generatori di `I`
- `mingens I`: restituisce la matrice riga dei generatori minimali di `I`
- `trim I`: restituisce l'ideale generato dai generatori minimali di `I`
- `ideal 0_R` e `ideal 1_R`: restituiscono, rispettivamente, l'ideale nullo e l'ideale improprio di `R`
- `basis(d,I)`: restituisce la base, come spazio vettoriale sul campo di definizione, della componente di grado `d` di `I`
- `p % I`: verificare l'appartenenza di un polinomio `p` all'ideale `I`
- `isSubset(I,J)`: verifica se l'ideale `I` è contenuto nell'ideale `J`.

Gestione ideali

Segue una liste delle funzioni per la gestione degli ideali di un anello dei polinomi:

- `+, *, ^`: operazioni estese agli ideali di somma, prodotto e potenza
- `I/J`: calcola l'ideale quoziente (l'ideale delle classi di equivalenza)
- `I:J` (o `quotient(I,J)`): calcola il colon o trasportatore di `J` in `I`
- `saturate(I,J)`: calcola la saturazione di I rispetto all'ideale J
- `intersect L`: calcola l'intersezione degli ideali nella lista `L`
- `radical I`: calcola il radicale di `I`
- `dim I`: calcola la dimensione di Krull di `I`
- `codim I`: calcola la codimensione (altezza) di `I`
- `isHomogeneous I`: verifica che `I` sia omogeneo (o graduato)
- `degrees I`: restituisce la lista dei gradi delle componenti di `I`
- `minors(k,A)`: calcola l'ideale generato dai minori di ordine `k` della matrice `A`
- `res I`: calcola la risoluzione minimale graduata di `S/I`
- `betti res I`: calcola la tabella di Betti di `S/I`

Macaulay2 RESET INTERRUPT UPLOAD FILE

```
i1 :  R=QQ[w..z];
i2 :  I = ideal (w^2*y-x^2, x^5-w)
o2 =  ideal (w^2y-x^2, x^5-w)
o2 :  Ideal of R
i3 :  S=R/I
o3 =  S
o3 :  QuotientRing
i4 :  x^5
o4 =  w
o4 :  S
i5 :  clearAll
i6 :  R=QQ[x_1..x_5];
i7 :  I=ideal(x_1*x_4+x_5,x_2)
o7 =  ideal (x_1x_4+x_5, x_2)
o7 :  Ideal of R
i8 :  J=ideal(x_2*x_3,x_1*x_4,x_5);
i9 :  I+J
o9 =  ideal (x_1x_4+x_5, x_2, x_2x_3,
             x_1x_4, x_5)
o9 :  Ideal of R
i10 :  trim(I+J)
o10 =  ideal (x_5, x_2, x_1x_4)
o10 :  Ideal of R

i11 :  intersect(I,J)
o11 =  ideal (x_2x_5, x_1x_4+x_5, x_2x_3)
o11 :  Ideal of R
i12 :  trim(I*J)
o12 =  ideal (x_2x_5, x_1x_4x_5+x_5^2,
              x_2^2x_3, x_1^2x_4^2-x_5^2,
              x_1x_2x_4)
o12 :  Ideal of R
i13 :  isSubset(I*J,intersect(I,J))
o13 =  true
i14 :  isSubset(intersect(I,J),I*J)
o14 =  false
i15 :  radical I
o15 =  ideal (x_2, x_1x_4+x_5)
o15 :  Ideal of R
i16 :  radical (J*I)==(J*I)
o16 =  false
i17 :  res I
o17 =  R^1 <-- R^2 <-- R^1 <-- 0
        0       1       2      3
o17 :  ChainComplex
```

Ideali monomiali

Un ideale si dice *monomiale* se ammette un insieme di generatori formato da monomi. Nel *Macaulay2* gli ideali monomiali possono essere gestiti dalla *classe* `MonomialIdeal`. L'istruzione `monomialIdeal` permette di ottenere un ideale monomiale in base all'input che viene fornito.

Segue un elenco di istruzioni utili per la gestione degli ideali monomiali:

- `monomialIdeal <struct>`:
  - `L`: (lista di monomi) restituisce l'ideale monomiale generato dai monomi della lista
  - `L`: (lista o matrice di polinomi) restituisce l'ideale monomiale generato dai monomi iniziali (in base all'ordinamento monomiale fissato) dei polinomi della lista (o matrice)
  - `I`: (ideale non monomiale) restituisce l'ideale monomiale generato dai monomi iniziali dei generatori di una base di Gröbner di `I`, cioè l'**ideale iniziale** di `I`
- `isMonomialIdeal I`: controlla se l'ideale `I` è monomiale
- `monomialSubideal I`: restituisce il più grande ideale monomiale contenuto in `I`
- `isSquareFree I`: controlla se l'ideale monomiale `I` è squarefree

Basi di Gröbner

Dato un ideale I, il suo l'*ideale iniziale* è l'ideale

$$\mathrm{in}(I) = (\mathrm{LT}(I)) = (\mathrm{LT}(f) \;:\; f \in I)\,.$$

Un insieme $(g_1, \ldots, g_k)$ di elementi di I è una *base di Gröbner* per I se

$$\mathrm{in}(I) = (\mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_k))\,.$$

Ecco alcune istruzioni utili per lavorare con le basi di Gröbner:

- `gb I`: calcola una base di Gröbner dell'ideale `I`, restituisce una struttura che memorizza molte informazioni relative ai calcoli effettuati
- `gens gb I`: restituisce la matrice dei generatori di una base di Gröbner dell'ideale
- `syz gb(I,Syzygies=>true)`: restituisce la matrice dei generatori del modulo delle sizigie dell'ideale
- `p % G`: se `p` è un polinomio e `g` una base di Gröbner, restituisce la forma normale del polinomio rispetto alla base
- `leadTerm I`: restituisce la matrice dei generatori dell'ideale iniziale di `I`, cioè i termini iniziali di una base di Gröbner di `I`

Macaulay2 RESET INTERRUPT UPLOAD FILE

```
i1 :  R=QQ[x,y]
o1 =  R
o1 :  PolynomialRing
i2 :  I = ideal(x^3 - 2*x*y, x^2*y - 2*y^2 + x)
o2 =  ideal (x^3-2xy, x^2y-2y^2+x)
o2 :  Ideal of R
i3 :  g=gb I
o3 =  GroebnerBasis[status:  done; S-pairs encountered up to degree 8]
o3 :  GroebnerBasis
i4 :  gens g
o4 =  | 2y2-x xy x2 |
o4 :  Matrix R^1 <--- R^3
i5 :  ideal leadTerm I
o5 =  ideal (2y^2, xy, x^2 )
o5 :  Ideal of R
i6 :  ideal leadTerm g
o6 =  ideal (2y^2, x*y, x^2)
o6 :  Ideal of R
i7 :  syz gb(I,Syzygies=>true)
o7 =  {3} | -x2y+2y2-x |
      {3} | x3-2xy     |
o7 :  Matrix R^2 <--- R^1
```

Decomposizione primaria

Il *Macaulay2*, ad ogni avvio, richiama automaticamente un package che permette tramite semplici istruzioni di calcolare una decomposizione primaria di un ideale.

Seguono alcune delle istruzioni relative all'argomento:

- `isPrime I`: verifica se `I` è primo
- `isPrimary I`: verifica se `I` è primario
- `primaryDecomposition I`: restituisce la lista degli ideali che formano una decomposizione primaria per l'ideale `I`
- `associatedPrimes I`: restituisce la lista degli ideali primi associati all'ideale `I`
- `minimalPrimes I`: restituisce la lista degli ideali primi minimali associati all'ideale `I`
- `primaryComponent(I,P)`: restituisce una componente primaria relativa all'ideale primo `P` associato a `I`
- `localize(I,P)`: localizza l'ideale `I` rispetto all'ideale primo `P`
- `getMaxIdeal I`: restituisce l'ideale massimale che contiene `I`. È contenuta nel package `QuillenSuslin` e può essere utile per verificare se un ideale è massimale

Macaulay2

RESET INTERRUPT UPLOAD FILE

```
i1 :  R=QQ[x_1..x_4]
o1 =  R
o1 :  PolynomialRing
i2 :  I=ideal(x_1*x_4-x_2*x_3,x_3+x_4,x_2^2*x_3)
o2 =  ideal (-x_2x_3+x_1x_4, x_3+x_4, x_2^2x_3) o2 :  Ideal of R
i3 :  isPrime I
o3 =  false
i4 :  isPrimary I
o4 =  false
i5 :  primaryDecomposition I
o5 =  {ideal (x_3+x_4, x_1+x_2, x_2^2), ideal (x_4, x_3)}
o5 :  List
i6 :  associatedPrimes I
o6 =  {ideal (x_3+ x_4, x_2, x_1), ideal (x_4, x_3)}
o6 :  List
i7 :  (primaryDecomposition I)/(x->radical x)
o7 =  {ideal (x_3+x_4, x_2, x_1), monomialIdeal (x_3, x_4)}
o7 :  List
i8 :  loadPackage "QuillenSuslin"
o8 =  QuillenSuslin
o8 :  Package
i9 :  getMaxIdeal I
o9 =  ideal (x_4, x_3, x_2, x_1)
o9 :  Ideal of R
```

Bibliografia

Bibliografia i

[1] D. R. Grayson and M. E. Stillman.
Macaulay2, a software system for research in algebraic geometry.
Available at http://www.math.uiuc.edu/Macaulay2/, 2020.

[2] J. Herzog and T. Hibi.
Monomial ideals, volume 260 of Graduate texts in mathematics.
Springer-Verlag London, 1 edition, 2011.

[3] P. Zinn-Justin.
Academic webpage.
Available at
http://blogs.unimelb.edu.au/paul-zinn-justin/, 2021.