Package ‘m2r’

June 23, 2017

Type Package
Title Macaulay2 in R
Version 1.0.0
Maintainer David Kahle <david.kahle@gmail.com>
Description Persistent interface to Macaulay2 (<http://www.math.uiuc.edu/Macaulay2/>)
and front-end tools facilitating its use in the R ecosystem.
License GPL-2
LazyData TRUE
Depends mpoly (>= 1.0.5)
Imports stringr, memoise, pryr, gmp
SystemRequirements Macaulay2 <http://www.math.uiuc.edu/Macaulay2/>
Suggests knitr, rmarkdown, testthat
RoxygenNote 6.0.1
NeedsCompilation no
Author David Kahle [aut, cph, cre],
    Christopher O'Neill [aut, cph],
    Jeff Sommars [aut, cph]
Repository CRAN
Date/Publication 2017-06-23 06:40:14 UTC

R topics documented:

  enter_m2 .................................................. 2
  factor_n .................................................. 3
  factor_poly ............................................... 5
  gb .......................................................... 6
  ideal ..................................................... 9
  is .......................................................... 13
  m2r ........................................................ 14
  m2_call ................................................... 15
  m2_matrix .................................................. 16
enter_m2

Enter a Macaulay2 session

**Description**

Enter a Macaulay2 session

**Usage**

```
enter_m2(port = 27436L, timeout = 10)
```

**Arguments**

- **port**: port for Macaulay2 socket
- **timeout**: number of seconds before aborting

**Value**

```
TRUE invisibly
```

**Examples**

```
## Not run: requires Macaulay2 be installed and an interactive session

enter_m2()

# m2 code below
1 + 1
a = 1
a
R = QQ[t,x,y,z]
I = ideal(t^4 - x, t^3 - y, t^2 - z)
gens gb I
exit

# back in R, the variable persists using m2()
m2("a")
m2("I")
```
# we can also define variables in R that persist in m2
m2("b = 5")

enter_m2()
b
exit

## End(Not run)

---

**factor_n**

*Factor an integer into primes*

**Description**

Factor an integer into primes

**Usage**

```r
factor_n(n, code = FALSE, ...)
factor_n.(n, code = FALSE, ...)
```

**Arguments**

- `n`: an integer or a polynomial
- `code`: return only the M2 code? (default: FALSE)
- `...`: ...

**Value**

a data frame with integer columns `prime` and `power` or `m2_pointer` referencing the factorization in M2.

**Examples**

```r
## Not run: requires Macaulay2

#### basic usage

```
(df <- factor_n(218700))
df$prime
df$power
str(df)

factor_n(218700, code = TRUE)

#### other options

(integer_pointer <- m2.("218700"))
m2_name(integer_pointer)
factor_n(integer_pointer, code = TRUE)
factor_n(integer_pointer)

factor_n(3234432540)
factor_n(323443254223453)
factor_n(rpois(1, 1e4))

#### known issues

# R doesn't handle big ints well. note in the following
# the m2 code number is different than the supplied number
factor_n(32344325422364353453, code = TRUE)

# this can be circumvented by passing a string instead
factor_n("32344325422364353453", code = TRUE)

# but if the factors are large, R can't handle the parsing well
factor_n("32344325422364353453")

# here's a workaround:
factor_pointer <- factor_n("32344325422364353453")
m2_meta(factor_pointer, "ext_str")
extract_factors <- function(pointer) {
  require(stringr)
  str <- m2_meta(pointer, "ext_str")
  str <- str_sub(str, 19, -2)
  str <- str_extract_all(str, "\\([0-9]+,([0-9]+)\)\[[1]\])
  str <- str_sub(str, 2, -2)
  str <- str_split(str, ",")
  df <- as.data.frame(t(simplify2array(str)))
  names(df) <- c("prime", "power")
  df
}
(df <- extract_factors(factor_pointer))
factor_poly

Factor a polynomial

Description
Factor a polynomial

Usage
factor_poly(mpoly, code = FALSE)
factor_poly(mpoly, code = FALSE, ...)

Arguments
- mpoly: a character parseable by `mp`, an `mpoly` object, or a pointer to a polynomial in M2
- code: return only the M2 code? (default: FALSE)
- ...: ...

Value
a named list with elements factor (an `mpolyList` object) and power, an integer vector

Examples

# Not run: requires Macaulay2 be installed and an interactive session

#### basic usage

```
ring("x", "y", coefring = "QQ")
factor_poly("x^4 - y^4")
```

# reference function
```
factor_poly.("x^4 - y^4")
```

#### different inputs

```r
# using gmp (currently broken)
# factor_n("32344325422364353453", gmp = TRUE)
m2("11 * 479 * 6138607975396537")
11 * 479 * 6138607975396537
```

## End(Not run)
# factor_poly accepts mpoly objects:
(p <- mp("x^4 - y^4"))
factor_poly.(p)
factor_poly(p)
mp("(x-y) (x+y) (x^2+y^2)")

##### other examples

ring("x", "y", "z", coeefring = "QQ")
(p <- mp("(x^2 - y) (x^2 + y) (x + y)^2 (x - z)^2")
factor_poly.(p)
factor_poly(p)

(p <- mp("(x-1)^3 (y-1)^3")
factor_poly.(p)
factor_poly(p)

## End(Not run)

gb Compute a Grobner basis with Macaulay2

**Description**
Compute a Grobner basis with Macaulay2

**Usage**

```
gb(..., control = list(), raw_chars = FALSE, code = FALSE)
gb(..., control = list(), raw_chars = FALSE, code = FALSE)
gb_(x, control = list(), raw_chars = FALSE, code = FALSE, ...)
gb_(x, control = list(), raw_chars = FALSE, code = FALSE, ...)
```

**Arguments**

- `...`: ...
- `control`: a list of options, see examples
- `raw_chars`: if TRUE, the character vector will not be parsed by `mp`, saving time (default: FALSE). The downside is that the strings must be formatted for M2 use directly, as opposed to for `mp`. (E.g. "x*y+3" instead of "x y + 3")
`gb`  

code

return only the M2 code? (default: FALSE)

x

a character vector of polynomials to be parsed by `mp`, a `mpolyList` object, an `ideal` or pointer to an ideal

Details

gb uses nonstandard evaluation; `gb_` is the standard evaluation equivalent.

Value

an `mpolyList` object of class `m2_grobner_basis` or a `m2_grobner_basis_pointer` pointing to the same

See Also

`mp`, `use_ring`

Examples

```r
## Not run: requires Macaulay2

# basic usage

# the last ring evaluated is the one used in the computation
ring("t","x","y","z", coefring = "QQ")
gb("t^4 - x", "t^3 - y", "t^2 - z")

# the code it's running in M2
gb("t^4 - x", "t^3 - y", "t^2 - z", code = TRUE)

# different versions of gb

# standard evaluation version
poly_chars <- c("t^4 - x", "t^3 - y", "t^2 - z")
gb_(poly_chars)

# reference nonstandard evaluation version
gb.("t^4 - x", "t^3 - y", "t^2 - z")

# reference standard evaluation version
gb_.(poly_chars)

# different inputs to gb
```
# ideals can be passed to gb
I <- ideal("t^4 - x", "t^3 - y", "t^2 - z")
gb_(I)

# note that gb() works here, too, since there is only one input
gb(I)

# ideal pointers can be passed to gb
I. <- ideal("t^4 - x", "t^3 - y", "t^2 - z")
gb_(!i)

# setting raw_chars is a bit faster, because it doesn't use ideal()
gb("t^4 - x", "t^3 - y", "t^2 - z", raw_chars = TRUE, code = TRUE)

# more advanced usage

# the control argument accepts a named list with additional
# options
gb_(
  c("t^4 - x", "t^3 - y", "t^2 - z"),
  control = list(StopWithMinimalGenerators = TRUE),
  code = TRUE
)

gb_(
  c("t^4 - x", "t^3 - y", "t^2 - z"),
  control = list(StopWithMinimalGenerators = TRUE)
)

# potential issues

# when specifying raw_chars, be sure to add asterisks
# between variables to create monomials; that's the M2 way
ring("x", "y", "z", coefring = "QQ")

# errors without code = TRUE

# correct way
Create a new ideal in Macaulay2

Usage

ideal(..., raw_chars = FALSE, code = FALSE)

ideal.(..., raw_chars = FALSE, code = FALSE)

ideal_(x, raw_chars = FALSE, code = FALSE, ...)

ideal_.(x, raw_chars = FALSE, code = FALSE, ...)

## S3 method for class 'm2_ideal'
print(x, ...)

## S3 method for class 'm2_ideal_list'
print(x, ...)

radical(ideal, ring, code = FALSE, ...)

radical.(ideal, ring, code = FALSE, ...)

saturate(I, J, code = FALSE, ...)

saturate.(I, J, code = FALSE, ...)

quotient(I, J, code = FALSE, ...)

quotient.(I, J, code = FALSE, ...)

primary_decomposition(ideal, code = FALSE, ...)

primary_decomposition.(ideal, code = FALSE, ...)

dimension(ideal, code = FALSE, ...)

## S3 method for class 'm2_ideal'
el + e2
### S3 method for class 'm2_ideal'
```r
e1 * e2
```

### S3 method for class 'm2_ideal'
```r
e1 == e2
```

### S3 method for class 'm2_ideal'
```r
e1 ^ e2
```

**Arguments**

- `...`
- `raw_chars` if TRUE, the character vector will not be parsed by `mp`, saving time (default: FALSE). The downside is that the strings must be formatted for M2 use directly, as opposed to for `mp`. (e.g. "x*y+3" instead of "x y + 3")
- `code` return only the M2 code? (default: FALSE)
- `x` a listing of polynomials. Several formats are accepted, see examples.
- `ideal` an ideal object of class `m2_ideal` or `m2_ideal_pointer`
- `ring` the referent ring in Macaulay2
- `I, J` ideals or objects parsable into ideals
- `e1, e2` ideals for arithmetic

**Value**

A reference to a Macaulay2 ideal

**Examples**

```r
## Not run: requires Macaulay2

##### basic usage
```
```
```r
ingh("x", "y", coeefring = "QQ")
ideal("x + y", "x^2 + y^2")
```

##### different versions of gb
```
```r
# standard evaluation version
poly_chars <- c("x + y", "x^2 + y^2")
ideal_(poly_chars)
```

# reference nonstandard evaluation version
ideal.("x + y", "x^2 + y^2")

# reference standard evaluation version
ideal_.(poly_chars)

####### different inputs to gb

ideal_( c("x + y", "x^2 + y^2") )
ideal_(mp(c("x + y", "x^2 + y^2")))
ideal_(list("x + y", "x^2 + y^2") )

####### predicate functions

I <- ideal ("x + y", "x^2 + y^2")
I. <- ideal.("x + y", "x^2 + y^2")
is.m2_ideal(I)
is.m2_ideal(I.)
is.m2_ideal_pointer(I)
is.m2_ideal_pointer(I.)

####### ideal radical

I <- ideal("(x^2 + 1)^2 y", "y + 1")
radical(I)
radical.(I)

####### ideal dimension

I <- ideal,(c("(x^2 + 1)^2 y", "y + 1"))
dimension(I)

# dimension of a line
ring("x", "y", coeefring = "QQ")
I <- ideal("y - (x+1)")
dimension(I)

# dimension of a plane
ring("x", "y", "z", coeefring = "QQ")
I <- ideal("z - (x+y+1)")
dimension(I)
### ideal quotients and saturation

```plaintext
ring("x", "y", "z", cofring = "QQ")
(I <- ideal("x^2", "y^4", "z + 1"))
(J <- ideal("x^6"))

quotient(I, J)
quotient(I, J)

saturate(I)
saturate(I)
saturate(I, J)
saturate(I, mp("x"))
saturate(I, "x")

ring("x", "y", cofring = "QQ")
saturate(ideal("x y"), "x^2")

# saturation removes parts of varieties
# solution over R is x = -1, 0, 1
ring("x", cofring = "QQ")
I <- ideal("(x-1) x (x+1)")
saturate(I, "x") # remove x = 0 from solution
ideal("(x-1) (x+1)")
```

### primary decomposition

```plaintext
ring("x", "y", "z", cofring = "QQ")
I <- ideal("x^2 + 1) (x^2 + 2)"", "y + 1")
primary_decomposition(I)
primary_decomposition(I)

I <- ideal("x (x + 1)", "y")
primary_decomposition(I)

# variety = z axis union x-y plane
(I <- ideal("x z", "y z"))
dimension(I) # = max dimension of irreducible components
(Is <- primary_decomposition(I))
dimension(Is)
```

### ideal arithmetic

```plaintext
```

---
ring("x", "y", "z", coefring = "RR")

# sums (cox et al., 184)
(I <- ideal("x^2 + y"))
(J <- ideal("z"))
I + J

# products (cox et al., 185)
(I <- ideal("x", "y"))
(J <- ideal("z"))
I * J

# equality
(I <- ideal("x", "y"))
(J <- ideal("z"))
I == J
I == I

# powers
(I <- ideal("x", "y"))
I^3

## End(Not run)

---

**is**  
*Macaulay2 object tests*

**Description**

Predicate functions for Macaulay2 objects.

**Usage**

```r
is.m2(x)

is.m2_pointer(x)

is.ring(x)

is.m2_polynomialring(x)

is.m2_polynomialring_pointer(x)

is.m2_grobner_basis(x)

is.m2_ideal(x)

is.m2_ideal_pointer(x)
```
is.m2_ideal_list(x)

is.m2_ideal_list_pointer(x)

is.m2_module(x)

is.m2_option(x)

is.m2_matrix(x)

is.m2_matrix_pointer(x)

is.m2_list(x)

is.m2_array(x)

is.m2_sequence(x)

Arguments

x an object

Value

logical(1)

Examples

## Not run: requires Macaulay2

R <- ring(c("x1", "x2", "x3"))
is.m2(R)
is.ring(R)
is.ring(10)
is.ring(mp("x+1"))

## End(Not run)

---

m2r    Macaulay2 in R

Description

Back-end connections to Macaulay2 (http://www.math.uiuc.edu/Macaulay2/)
**m2_call**  
*Call and reset a Macaulay2 process*

**Description**

Call and reset a Macaulay2 process

**Usage**

```r
m2r_version_number()

start_m2(port = 27436L, timeout = 10, attempts = 10, cloud = FALSE,  
hostname = m2r_cloud_url())

stop_m2()

reset_m2(port = 27436L, timeout = 10, attempts = 10,  
hostname = "ec2-52-10-66-241.us-west-2.compute.amazonaws.com")

m2(code, timeout = -1)

m2.(code, timeout = -1)
```

```r
## S3 method for class 'm2_pointer'
print(x, ...)
```

**Arguments**

- `port` : port for Macaulay2 socket
- `timeout` : number of seconds before aborting
- `attempts` : number of times to try to make connection
- `cloud` : use a cloud?
- `hostname` : the remote host to connect to; defaults to the Amazon EC2 instance
- `code` : Macaulay2 code
- `x` : formal argument for print method
- `...` : ...

**Value**

`m2 return value`
Examples

```m2
## Not run: requires Macaulay2

m2("1 + 1")
m2.("1 + 1")

m2("factor 32004")

# run a chunk of m2 code, only pulling the end value back into R
m2("R = QQ[a..d]
   I = ideal(a^3-b^2*c, b*c^2-c*d^2, c^3)
   G = gens gb I
")

# illustrate the persistent connection
m2("a = 1 + 1")
m2("a")
reset_m2()
m2("a")

m2.("peek(QQ[x,y,z])")
m2("peek(QQ[x,y,z])")

# m2 returns in its ext_str position the result of running
# toExternalString on the return value of the chunk of code
# you run, in principle, toExternalString provides the code
# needed to recreate the m2 object of interest. however,
# does not work for all objects representable in the m2 language.
# in particular, mutable objects are not supported.
# this is what happens when you look at those:

m2("new MutableList from (1,2,3")
m2("new MutableList from (1,2,3")

## End(Not run)
```

---

**m2_matrix**

Create a new matrix in Macaulay2

**Description**

Create a new matrix in Macaulay2
Usage

\texttt{m2\_matrix(mat, ring, name, code = FALSE)}

\texttt{m2\_matrix.(mat, ring, name, code = FALSE)}

\texttt{m2\_numrows(x, code = FALSE, ...)}

\texttt{m2\_numcols(x, code = FALSE, ...)}

\texttt{m2\_length(x, code = FALSE, ...)}

## S3 method for class 'm2\_matrix'
\texttt{print(x, ...)}

## S3 method for class 'm2\_image'
\texttt{print(x, ...)}

\texttt{m2\_kernel(mat, name, code = FALSE)}

\texttt{m2\_kernel.(mat, name, code = FALSE)}

Arguments

- \texttt{mat} a matrix
- \texttt{ring} a ring containing the matrix entries
- \texttt{name} the \texttt{m2\_name} of the object, which is it’s name on the M2 side
- \texttt{code} return only the M2 code? (default: \texttt{FALSE})
- \texttt{x} formal argument for print method
- \texttt{...} ...

Value

a reference to a Macaulay2 ring

Examples

## Not run: requires Macaulay2

### basic usage

\begin{verbatim}
(mat <- m2\_matrix(matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2))
m2\_matrix(matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2))
m2\_name(mat)
m2(m2\_name(mat))
m2(sprintf("class\(\%s\)", m2\_name(mat)))
\end{verbatim}
(mat <- m2_matrix(matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)))

##### known issues

```{r}
ring("x", "y", "z", coefring = "QQ")
mat <- matrix(mp(c("x","y","x+y","y-2","x-3","y-z")), nrow = 2, ncol = 3)
m2_matrix(mat, code = TRUE)
m2_matrix(mat)
# the above is an mpoly problem, not a m2r problem
# mpoly does not have a data structure for matrices (as of 12/2016)
mat_chars <- sapply(m2_matrix(mat), print, silent = TRUE)
dim(mat_chars) <- c(2, 3)
mat_chars

m2_numrows(mat)
m2_numcols(mat)
m2_parse(mat)

(mat <- m2_matrix(matrix(c(1,2),nrow=1)))
m2_kernel(mat)
```

```
## End(Not run)
```

---

**m2_parser**  
*Convert a M2 object into an R object*

---

**Description**

Convert a M2 object into an R object

**Usage**

```r
m2_parse(s)
```

```r
# S3 method for class 'm2_integer'
print(x, ...)
```

```r
# S3 method for class 'm2_float'
print(x, ...)
```

```r
# S3 method for class 'm2_complex'
print(x, ...)
```

```r
# S3 method for class 'm2_string'
print(x, ...)
```
## S3 method for class 'm2_boolean'
print(x, ...)

## S3 method for class 'm2_list'
print(x, ...)

## S3 method for class 'm2_array'
print(x, ...)

## S3 method for class 'm2_sequence'
print(x, ...)

## S3 method for class 'm2_symbol'
print(x, ...)

## S3 method for class 'm2_option'
print(x, ...)

## S3 method for class 'm2_hashtable'
print(x, ...)

## S3 method for class 'm2_module'
print(x, ...)

m2_toggle_gmp()

gem2_gmp()

### Arguments

s
a character(1), typically the result of running toExternalString on an M2 object

x
an object to be printed

... ... ...

### Value

an R object

### Examples

## Not run: requires Macaulay2

m2("1+1")
m2.("1+1")
m2_parse(m2.("1+1"))
m2("QQ[x,y]")
Description
This function sets the path to external programs either by (1) passing it a character string or (2) using `file.choose`.

Usage

```r
set_m2_path(path = NULL)
get_m2_path()
get_m2_con()
get_m2_procid()
get_m2_port()
```

Arguments

- `path` A character string, the path to M2
Details

When m2r is loaded it attempts to find M2. How it looks depends on your operating system. If you’re using a Mac or Linux machine, it looks based on your system’s path. Unfortunately, R changes the system path in such a way that the path that R sees is not the same as the path that you’d see if you were working in the terminal. (You can open the Terminal app on a Mac by going to /Applications/Utilities/Terminal.) Consequently, m2r tries to guess the file in which your path is set. To do so, it first checks if your home directory (type echo ~/ in the terminal to figure out which directory this is if you don’t know) for the file named .bash_profile. If this file is present, it runs it and then checks your system’s path variable (echo $PATH). If it’s not present, it does the same for .bashrc and then .profile. In any case, once it has its best guess at your path, it looks for "M2".

On Windows, m2r just defaults to the cloud implementation. Local M2 instances are not currently supported on Windows.

Value

An invisible character string, the path found. More importantly, the function has the side effect of setting the global m2r option "m2_path"

Author(s)

David Kahle <david.kahle@gmail.com>

Examples

```r
## Not run: requires Macaulay2

getOption("m2r")
get_m2_path()
set_m2_path()

## each of these functions can be used statically as well
(m2_path <- get_m2_path())
set_m2_path("/path/to/m2/directory")
get_m2_path()
set_m2_path(m2_path) # undoes example

# if you'd like to use the cloud, after you library(m2r)
# and before you use m2() type
set_m2_path(NULL)

# alternatively, if you have already been using m2, do:
stop_m2()
set_m2_path(NULL)
m2("1+1")
```
m2_utility

Utility tools for M2

Description
Utility tools for M2

Usage

```r
m2_name(x)

m2_name(x) <- value

m2_meta(x, m2_attr)

m2_meta(x, m2_attr) <- value

m2_structure(x = NA, m2_name, m2_class, m2_meta, base_class)

m2_exists(name)

m2_ls(all.names = FALSE)

m2_rm(name)

m2_getwd()
```

Arguments

- `x` an object of class `m2`
- `value` the value to assign
- `m2_attr` the name of an M2 attribute
- `m2_name` `m2_name` M2 attribute
- `m2_class` `m2_class` M2 attribute
- `m2_meta` `m2_meta` M2 attribute
- `base_class` a base class; an R class to use for dispatching if there is no relevant method for the other classes (e.g. `m2`)
- `name` a string; the name of a M2 object
- `all.names` if `TRUE`, all registered Macaulay2 variables, including ones internally used by `m2r`, will be returned
**Examples**

```r
## Not run: requires Macaulay2
m2("a = 5")
m2_ls()
m2_exists("a")
m2("b = 1")
m2_exists(c("a","b","c"))

m2_getwd()
x <- 1
class(x) <- "m2"
attr(x, "m2_meta") <- list(a = 1, b = 2)
m2_meta(x)
m2_meta(x, "b")
m2_meta(x, "b") <- 5
m2_meta(x, "b")

# R <- ring(c("x1", "x2", "x3"))
# m2_name(R)
# m2(sprintf("class %s", m2_name(R)))
# m2_ls()
# m2_rm(m2_name(R))
# m2_ls()
# m2(paste("class", m2_name(R)))

m2_ls()
m2_ls(all.names = TRUE)

## End(Not run)
```

---

**phc**  
*PHCpack*

**Description**

Call PHCpack to solve a zero-dimensional system

**Usage**

```r
solve_system(mpolyList)
solve_system.(mpolyList)
mixed_volume(mpolyList)
```
Arguments

mpolyList       An mpolyList object

Details

Note that solve_system() doesn’t take in an input ring because the solver only works over the complex numbers.

Value

(currently) the output of an m2() call (string?)

Examples

```csharp
## Not run: requires Macaulay2

# for this to work, you need to have modified your
# init-PHCpack.m2 file instead of changing your .bashrc
# file to establish the path of phc
# (**clarify**, maybe checkout algstat::polySolve)

(mpolyList <- mp(c("t^4 - x", "t^3 - y", "t^2 - z", "x+y+z")))
solve_system(mpolyList)
mixed_volume(mpolyList)

## End(Not run)
```

---

**ring**

Create a new ring in Macaulay2

Description

Create a new ring in Macaulay2

Usage

```csharp
ring(..., coefring = m2_coefrings(), order = m2_termorders(),
     code = FALSE)

ring(..., coefring = m2_coefrings(), order = m2_termorders(),
     code = FALSE)

ring_(vars, coefring = m2_coefrings(), order = m2_termorders(),
     code = FALSE, ...)

ring_(vars, coefring = m2_coefrings(), order = m2_termorders(),
```
ring

    code = FALSE, ...)

m2_coefrings()

m2_termorders()

## S3 method for class 'm2_polynomialring'
print(x, ...)

Arguments

... ...  
coefring coefficient ring (default: "CC")
order a term order (default: "grevlex")
code return only the M2 code? (default: FALSE)
vars vector of variable names
x formal argument for print method

Value

a reference to a Macaulay2 ring

Examples

## Not run: requires Macaulay2

##### basic usage

ring("x", "y")
ring("x", "y", coeefring = "QQ")

##### standard evaluation

ring_(c("x", "y"))
ring_(c("x", "y"), code = TRUE)

(myring <- ring_(c("x1","x2","x3","y"), coeefring = "QQ", order = "lex"))

m2_name(myring)
m2_meta(myring, "vars")
m2_meta(myring, "coeefring")
m2_meta(myring, "order")

##### other options


ring_.(c("x", "y"))
ring_.(c("x", "y"), code = TRUE)

## End(Not run)

### snf  

**Smith normal form**

### Description

For an integer matrix M, this computes the matrices D, P, and Q such that $D = PMQ$, which can be seen as an analogue of the singular value decomposition. All are integer matrices, and P and Q are unimodular (have determinants ± 1).

### Usage

```r
snf(mat, code = FALSE)
```

### Arguments

- **mat** 
  a matrix (integer entries)
- **code** 
  return only the M2 code? (default: FALSE)

### Value

a list of integer matrices D, P, and Q

### Examples

```r
## Not run: requires Macaulay2

### basic usage

```R
M <- matrix(c(2, 4, 4,
  -6, 6, 12,
  10, -4, -16), nrow = 3, byrow = TRUE)

snf(M)
```

```r
(mats <- snf(M))
P <- mats$P; D <- mats$D; Q <- mats$Q
```


### str_m2

Give the structure of a Macaulay2 ring

#### Description

Give the structure of a Macaulay2 ring

#### Usage

str_m2(object, ...)

#### Arguments

- **object**: An m2 object

#### Value

Invisible the object passed in.

---

```r
P %*% M %*% Q
solve(P) %*% D %*% solve(Q)

det(P)
det(Q)

M <- matrix(c(
  1, 2, 3,
  1, 34, 45,
  2213, 1123, 6543,
  0, 0, 0
), nrow = 4, byrow = TRUE)
(mats <- snf(M))
P <- mats$P; D <- mats$D; Q <- mats$Q
P %*% M %*% Q
```

```r
####### other options

snf(M)
snf(M, code = TRUE)
```

```r
## End(Not run)
```
Examples

## Not run: requires Macaulay2

```r
a <- m2("1")
R <- ring(c("x1", "x2", "x3"))
str.m2(R)
str.m2.default(R)

## End(Not run)
```

### Description

`use_ring()` sets the default referent ring on the Macaulay2 side using the `use` function.

### Usage

```r
use_ring(ring)
```

### Arguments

- **ring**: a `m2_ring`(`ring`), `m2_ring_pointer`(`ring`), or a character string containing the name of a ring in Macaulay2

### Examples

## Not run: requires Macaulay2

```r

#### basic usage

```r
ring("x", coefring = "QQ")
factor_poly("x^4 + 1")

QQtxyz <- ring("t", "x", "y", "z", coefring = "QQ")
```

```r
gb("t^4 - x", "t^3 - y", "t^2 - z")

ring("x", "y", "z", "t", coefring = "QQ")
```

```r
gb("t^4 - x", "t^3 - y", "t^2 - z")

use_ring(QQtxyz)
```

```r```
use_ring

\texttt{gb("t^4 - x", "t^3 - y", "t^2 - z")}

\texttt{## End(Not run)}
Index

*.m2_ideal (ideal), 9
+.m2_ideal (ideal), 9
==.m2_ideal (ideal), 9
^.*.m2_ideal (ideal), 9
dimension (ideal), 9
enter_m2, 2
factor_n, 3
factor_poly, 5
file.choose, 20
gb, 6
gb_(gb), 6
get_m2_con (m2_path), 20
get_m2_gmp (m2_parser), 18
get_m2_path (m2_path), 20
get_m2_port (m2_path), 20
get_m2_procid (m2_path), 20
ideal, 7, 9
ideal_ (ideal), 9
is, 13
m2 (m2_call), 15
m2_call, 15
m2_coefrings (ring), 24
m2_exists (m2_utility), 22
m2_getwd (m2_utility), 22
m2_kernel (m2_matrix), 16
m2_length (m2_matrix), 16
m2_ls (m2_utility), 22
m2_matrix, 16
m2_meta (m2_utility), 22
m2_meta<- (m2_utility), 22
m2_name (m2_utility), 22
m2_name<- (m2_utility), 22
m2_numcols (m2_matrix), 16
m2_numrows (m2_matrix), 16
m2_parse (m2_parser), 18
m2_parser, 18
m2_path, 20
m2_rm (m2_utility), 22
m2_structure (m2_utility), 22
m2_termorders (ring), 24
m2_toggle_gmp (m2_parser), 18
m2_utility, 22
m2r, 14
m2r-package (m2r), 14
m2r_version_number (m2_call), 15
mixed_volume (phc), 23
mp, 5–7, 10
mpolyList, 5, 7
package-m2r (m2r), 14
phc, 23
primary_decomposition (ideal), 9
print.m2_array (m2_parser), 18
print.m2_boolean (m2_parser), 18
print.m2_complex (m2_parser), 18
print.m2_float (m2_parser), 18
print.m2_hashtable (m2_parser), 18
print.m2_ideal (ideal), 9
print.m2_ideal_list (ideal), 9
print.m2_image (m2_matrix), 16
print.m2_integer (m2_parser), 18
print.m2_list (m2_parser), 18
print.m2_matrix (m2_matrix), 16
print.m2_module (m2_parser), 18
print.m2_option (m2_parser), 18
print.m2_pointer (m2_call), 15
print.m2_polynomialring (ring), 24
print.m2_sequence (m2_parser), 18
print.m2_string (m2_parser), 18
print.m2_symbol (m2_parser), 18
quotient (ideal), 9
radical (ideal), 9
reset_m2 (m2_call), 15
INDEX

ring, 24, 28
ring., 28
ring_ (ring), 24

saturate (ideal), 9
set_m2_path (m2_path), 20
snf, 26
solve_system (phc), 23
start_m2 (m2_call), 15
stop_m2 (m2_call), 15
str_m2, 27

use_ring, 7, 28