Journal of Software for Algebra and Geometry

vol 13 2023

```
gap> g:= SymmetricGroup( 4 );
Sym( [ 1 .. 4 ] )
gap> tbl:= CharacterTable( g );;  HasIrr( tbl );
false
gap> tblmod2:= CharacterTable( tbl, 2 );
BrauerTable( Sym( [ 1 .. 4 ] ), 2 )
gap> tblmod2 = CharacterTable( tbl, 2 );
true
gap> tblmod2 = BrauerTable( tbl, 2 );
true
gap> tblmod2 = BrauerTable( g, 2 );
true
gap> libtbl:= CharacterTable( "M" );
CharacterTable( "M" )
gap> CharacterTableRegular( libtbl, 2 );
BrauerTable( "M", 2 )
gap> BrauerTable( libtbl, 2 );
fail
gap> CharacterTable( "Symmetric", 4 );
CharacterTable( "Sym(4)" )
gap> ComputedBrauerTables( tbl );
[ , BrauerTable( Sym( [ 1 .. 4 ] ), 2 ) ]
```

```
i5 : betti(t,Weights=>{1,0})

            0 1  2  3 4
o5 = total: 1 4 13 14 4
         0: 1 .  .  . .
         1: . 2  2  4 2
         2: . 2  5  6 .
         3: . .  4  . 2
         4: . .  .  4 .
         5: . .  2  . .

o5 : BettiTally
i6 : betti(t,Weights=>{0,1})

            0 1  2  3 4
o6 = total: 1 4 13 14 4
         0: 1 .  .  . .
         1: . 2  2  4 2
         2: . 2  5  6 .
         3: . .  4  . 2
         4: . .  .  4 .
         5: . .  2  . .

o6 : BettiTally
i7 : t1 = betti(t,Weights=>{1,1})

            0 1  2  3 4
o7 = total: 1 4 13 14 4
         0: 1 .  .  . .
         1: . .  .  . .
         2: . .  .  . .
         3: . 2  .  . .
         4: . .  .  . .
         5: . 2  .  . .
         6: . .  1  . .
         7: . .  8  6 .
         8: . .  4  8 4

o7 : BettiTally
i8 : peek t1

o8 = BettiTally{(0, {0, 0}, 0) => 1 }
                (1, {2, 2}, 4) => 2
                (1, {3, 3}, 6) => 2
                (2, {3, 7}, 10) => 2
                (2, {4, 4}, 8) => 1
                (2, {4, 5}, 9) => 4
                (2, {5, 4}, 9) => 4
                (2, {7, 3}, 10) => 2
                (3, {4, 7}, 11) => 4
                (3, {5, 5}, 10) => 6
                (3, {7, 4}, 11) => 4
                (4, {5, 7}, 12) => 2
                (4, {7, 5}, 12) => 2
```

```
ring r1 = 32003,(x,y,z),ds;
int a,b,c,t=11,5,3,0;
poly f = x^a+y^b+z^(3*c)+x^(c+2)*y^(c-1)+x^
         x^(c-2)*y^c*(y^2+t*x)^2;
option(noprot);
timer=1;
ring r2 = 32003,(x,y,z),dp;
poly f=imap(r1,f);
ideal j=jacob(f);
vdim(std(j));
==> 536
vdim(std(j+f));
==> 195
timer=0;  // reset timer
```

*JSAG* **13** (2023), 1–12
https://doi.org/10.2140/jsag.2023.13.1

The Journal of Software for
Algebra and Geometry

# Algorithms for computing mixed multiplicities, mixed volumes, and sectional Milnor numbers

KRITI GOEL, VIVEK MUKUNDAN, SUDESHNA ROY AND JUGAL VERMA

ABSTRACT: We present a package `MixedMultiplicity` for computing mixed multiplicities of ideals in a Noetherian ring which is either local or a standard graded algebra over a field. This enables us to find mixed volumes of convex lattice polytopes and sectional Milnor numbers of hypersurfaces with an isolated singularity. The algorithms make use of the defining equations of the multi-Rees algebra of ideals, which are obtained by generalising a result of Cox, Lin and Sosa.

**1.** INTRODUCTION. This article describes the [Macaulay2] package `MixedMultiplicity` which computes the mixed multiplicities of ideals having positive grade in a Noetherian ring, the mixed volume of a collection of convex lattice polytopes, and sectional Milnor numbers of hypersurfaces with an isolated singularity. One of the main steps of the algorithms is the computation of the defining equations of the multi-Rees algebra

$$\mathcal{R}(I_1, \ldots, I_s) = R[I_1 t_1, \ldots, I_s t_s] = \bigoplus_{a_1,\ldots,a_s \geq 0} I_1^{a_1} \cdots I_s^{a_s} t_1^{a_1} \cdots t_s^{a_s} \subseteq R[t_1, \ldots, t_s]$$

of ideals $I_1, \ldots, I_s$ in a Noetherian ring $R$. For the case when $R$ is a polynomial ring over any field $k$ and $I_1, \ldots, I_s$ are monomial ideals in $R$, an explicit formula for the defining equations of $\mathcal{R}(I_1, \ldots, I_s)$ has been given by D. Cox, K.-N. Lin, and G. Sosa in [Cox et al. 2019]. In Theorem 2.1, we obtain an analogue of their result for any set of ideals $I_1, \ldots, I_s$ in a Noetherian ring $R$ such that each $I_i$ has positive grade. The latter condition is always satisfied when $R$ is a domain or for any ideal of positive height in a reduced ring or in a Cohen–Macaulay ring. Using this result, we write a function `multiReesIdeal` to compute the defining ideal of a multi-Rees algebra in Macaulay2. It should be noted that the command `reesIdeal` in the Macaulay2 package `ReesAlgebra` [Eisenbud 2018] is already available to compute the defining ideal of the Rees algebra of a module [Eisenbud et al. 2003], in particular, the defining ideal of a multi-Rees algebra. But, the algorithm presented in this article runs faster in many cases. When the ring is not a domain, the algorithm follows an analogue of the `reesIdeal`, albeit for the multiple ideal setting.

Several authors (see for instance, [Ribbe 1999; Lin and Polini 2014; Sosa 2014; Jabarnejad 2018]) have proposed algorithms to determine the defining equations of the multi-Rees algebra for specific classes of ideals. The algorithm for computing defining equations of $\mathcal{R}(I_1, \ldots, I_s)$ helps us to construct algorithms

*MSC2020:* 13-04, 13A30, 13H15.
*Keywords:* multi-Rees algebras, mixed multiplicities, sectional Milnor numbers, mixed volume.
`MixedMultiplicity` version 3.0

© 2023 The Authors, under license to MSP (Mathematical Sciences Publishers).

to compute *mixed multiplicities* (Section 3), *mixed volume* (Section 4) and *sectional Milnor numbers* (Section 5) in the general setting. Observe that to compute mixed multiplicities of ideals one can always assume each ideal to have positive grade using a standard trick (see Remark 3.3).

For any ideal $J$ in a Noetherian ring $R$, we denote the common length of the maximal $R$-sequences in $J$ by grade$(J)$. Let $I_0, I_1, \ldots, I_r$ be a set of ideals in a Noetherian ring of dimension $d \geq 1$, which is either local or a standard graded algebra over a field. Further assume that $I_0$ is primary to the maximal ideal (resp. maximal graded ideal) and grade$(I_j) > 0$ for all $j$. Let $\underline{a} = (a_0, a_1, \ldots, a_r) \in \mathbb{N}^{r+1}$ with $|\underline{a}| = d - 1$. The function `mixedMultiplicity` computes the mixed multiplicity $e_{\underline{a}}(I_0 \mid I_1, \ldots, I_r)$. Using the results of N. V. Trung, J. K. Verma and B. Teissier, mixed volumes and sectional Milnor numbers can be identified with mixed multiplicities of ideals over polynomial rings. Let $Q_1, \ldots, Q_n$ be a collection of lattice polytopes in $\mathbb{R}^n$. The function `mMixedVolume` computes the mixed volume of $Q_1, \ldots, Q_n$. Let $R = k[x_1, \ldots, x_n]$ be a polynomial ring in $n$ variables, $\mathfrak{m}$ be the maximal graded ideal and $f \in R$ be any polynomial. The function `secMilnorNumbers` computes the sectional Milnor numbers by calculating the mixed multiplicities $e(\mathfrak{m}^{[n-i]}, J(f)^{[i]})$, $0 \leq i \leq n-1$, where $J(f) = (f_{x_1}, \ldots, f_{x_n})$ is the ideal generated by the partial derivatives of $f$. Many researchers, including M. Herrmann et al. [1997], J. K. Verma [1992], and C. D'Cruz [2003], have expressed the multiplicities of Rees algebras, extended Rees algebras, and certain form rings in terms of mixed multiplicities. The `MixedMultiplicity` package is also helpful in this regard. For any unexplained invariants and definitions used in this article, the reader may refer to [Bruns and Herzog 1993], [Eisenbud et al. 2003], and [Huneke and Swanson 2006].

**2.** DEFINING IDEAL OF MULTI-REES ALGEBRA OF IDEALS. An explicit formula for the defining ideal of the multi-Rees algebra of a finite collection of monomial ideals in a polynomial ring was given in [Cox et al. 2019]. In this section, we generalize their result to find the defining ideal of the multi-Rees algebra of a collection of ideals with positive grade in a Noetherian ring. We use this result to write a Macaulay2 algorithm to compute the defining ideal when the base ring is a domain. We provide another algorithm for the nondomain case.

Let $R$ be a Noetherian ring and $I_1, \ldots, I_s \subseteq R$ be ideals. Suppose that $I_i = \langle f_{ij} \mid j = 1, \ldots, n_i \rangle$ for all $i = 1, \ldots, s$. Let $\mathcal{R}(I_1, \ldots, I_s)$ be the multi-Rees algebra of ideals $I_1, \ldots, I_s$. Consider the set of indeterminates $\underline{Y} = \{Y_{ij} \mid i = 1, \ldots, s, j = 1, \ldots, n_i\}$ and $\underline{T} = (T_1, \ldots, T_s)$ over $R$. Define an $R$-algebra homomorphism $R[\underline{Y}] \xrightarrow{\varphi} \mathcal{R}(I_1, \ldots, I_s) \subseteq R[\underline{T}]$ such that $\varphi(Y_{ij}) = f_{ij}T_i$, for all $i = 1, \ldots, s$, $j = 1, \ldots, n_i$ and $\varphi(r) = r$ for all $r \in R$. Then $\mathcal{R}(I_1, \ldots, I_s) \simeq R[\underline{Y}]/\ker(\varphi)$. The ideal $\ker\varphi$ is called the defining ideal of $\mathcal{R}(I_1, \ldots, I_s)$. We give an explicit description of $\ker(\varphi)$.

**Theorem 2.1.** *Let $R$ be a Noetherian ring and $I_1, \ldots, I_s \subseteq R$ be ideals of positive grade. For each $i$, consider a generating set $\{f_{ij} \mid j = 1, \ldots, n_i\}$ of $I_i$ which contains at least one nonzerodivisor $f_{ij_i}$. We set $h = \prod_{i=1}^{s} f_{ij_i}$ and set*

$$\Gamma = \langle Y_{ij} f_{ij'} - Y_{ij'} f_{ij} \mid i = 1, \ldots, s \text{ and } j, j' \in \{1, \ldots, n_i\} \rangle : h^{\infty} \subseteq R[\underline{Y}].$$

*Then $\Gamma \subseteq R[\underline{Y}]$ is the defining ideal of $\mathcal{R}(I_1, \ldots, I_s)$.*

*Proof.* Without loss of generality, we may assume that $j_i = 1$ for all $i = 1, \ldots, s$ and $h = \prod_{i=1}^{s} f_{i1}$. Consider the ring homomorphism $\phi : R \to R[f_{11}^{-1}, f_{21}^{-1}, \ldots, f_{s1}^{-1}] \cong R[h^{-1}]$, which induces a natural map $\tilde{\phi} : R[\underline{Y}] \to R[h^{-1}][\underline{Y}] \cong R[\underline{Y}]_h$. The defining ideal of $\mathcal{R}(\phi(I_1), \ldots, \phi(I_s))$ of the ideals $\phi(I_i) = (1, f_{i2}/f_{i1}, \ldots, f_{in_i}/f_{i1})$ for $i = 1, \ldots, s$, is $J := J_1 + \cdots + J_s$ in $R[h^{-1}][\underline{Y}] \cong R[\underline{Y}]_h$, where $J_i := (Y_{i2} - f_{i2}/f_{i1}Y_{i1}, \ldots, Y_{in_i} - f_{in_i}/f_{i1}Y_{i1})$. We claim that $\tilde{\phi}^{-1}(J) = \Gamma$. Observe that for all $j \neq j'$, and for all $i$,

$$f_{ij}Y_{ij'} - f_{ij'}Y_{ij} = f_{ij}\left(Y_{ij'} - \frac{f_{ij'}}{f_{i1}}Y_{i1}\right) - f_{ij'}\left(Y_{ij} - \frac{f_{ij}}{f_{i1}}Y_{i1}\right) \in J_i.$$

So $\Gamma \subseteq \tilde{\phi}^{-1}(J)$. Now let $r \in \tilde{\phi}^{-1}(J)$. Then $\tilde{\phi}(r) \in J$, i.e.,

$$\frac{r}{1} = \sum_{i=1}^{s}\sum_{j=2}^{n_i} \frac{a_{ij}}{f_{i1}^{m_{ij}}}\left(Y_{ij} - \frac{f_{ij}}{f_{i1}}Y_{i1}\right)$$

for some $a_{ij} \in R$. Thus we have

$$h^m r \in (f_{i1}Y_{ij} - f_{ij}Y_{i1} \mid 1 \leq i \leq s,\ 1 \leq j \leq n_i) \subseteq (f_{ij}Y_{ij'} - f_{ij'}Y_{ij} \mid 1 \leq i \leq s,\ 1 \leq j, j' \leq n_i)$$

for some $m \geq \max\{m_{ij} \mid 1 \leq i \leq s, 1 \leq j \leq n_i\} + 1$. Therefore, $r \in \Gamma$ and hence the claim holds. From [Atiyah and Macdonald 2016, Proposition 3.11 (iii)], we get that $\tilde{\phi}^{-1}(J)$ is the defining ideal of $\mathcal{R}(I_1, \ldots, I_s)$, as $h^t$ is a nonzerodivisor on $R[\underline{Y}]/\Gamma$ for all $t \geq 1$. □

When $R$ is a domain or when a list of nonzerodivisors (one each from the list of ideals with positive grades) is provided by the user, the function `multiReesIdeal` computes the defining ideal of the multi-Rees algebra using Theorem 2.1.

**Algorithm** (version I: `multiReesIdeal`, set of ideals with positive grade). Let $I_1, \ldots, I_s$ be ideals of a Noetherian ring $R$ with grade $I_i > 0$ for all $i$ and let $a_1, \ldots, a_s$ be a set of nonzerodivisors, where $a_i$ belongs to the generating set of $I_i$ for all $i$. When $R$ is a domain, the function picks $a_i$ to be the first element in the generating set of $I_i$ for each $i$.

**Input**: The list $W = \{\{I_1, \ldots, I_s\}, \{a_1, \ldots, a_s\}\}$, or $W = \{I_1, \ldots, I_s\}$ if $R$ is a domain.

(1) Define a polynomial ring $S$ by attaching $m$ indeterminates to the ring $R$, where $m$ is the sum of the number of generators of all the ideals.

(2) For each ideal $I_i$, construct a matrix $M(i)$ whose first row consists of the generators of the ideal and the second row consists of the indeterminates.

(3) Add the $2 \times 2$ minors of these matrices to get an ideal $L$.

(4) To get the defining ideal, saturate $L$ with the product of $a_i$'s.

**Output**: The defining ideal of the Rees algebra $\mathcal{R}(I_1, \ldots, I_s)$.

The elements of the defining ideal are assigned $\mathbb{N}^{s+1}$ degree by the function, where the first $\mathbb{N}^s$ coordinates point to the component of $\mathcal{R}(I_1, \ldots, I_s)$ where the element lies and the last coordinate is the

degree of the element. In order to compute the multi-Rees ideal $\mathcal{R}(I, J)$ using the function `reesIdeal`, one needs to enter $I \oplus J$ and this routine is sometimes slower than the `multiReesIdeal` routine.

**Example 2.2.**

```
i1 : R = QQ[x,y,z];

i2 : I = ideal(x^4+y^2*z^2,x*y^2*z);

i3 : J = ideal(y^3+z^3, x^2*y+x*z^2);

i4 : time multiReesIdeal {I,J};
      -- used 0.0131127 seconds

i5 : transpose gens oo

o5 = {0,-1,-6} | (x2y+xz2)X_2+(-y3-z3)X_3 |
     {-1,0,-8} | xy2zX_0+(-x4-y2z2)X_1    |

i6 : (first entries gens o4)/degree

o6 : {{0, 1, 6}, {1, 0, 8}}

i7 : time multiReesIdeal({I,J},{I_1, J_0});
     -- used 0.0629737 seconds

i8 : M = directSum(module I, module J);

i9 : time reesIdeal M;
     -- used 0.40043 seconds
```

In the following example, our algorithm works faster than the function `reesIdeal`.

**Example 2.3.**

```
i1 : ZZ/32003[y_0..y_4];

i2 : C = trim monomialCurveIdeal(R, {3, 5, 7, 12});

i3 : time multiReesIdeal (C, C_0);
          -- used 167.118 seconds

i4 : time reesIdeal (C, C_0);
          -- used 295.675 seconds
```

**2.1.** ***Routine for the nondomain case.*** In this section we present an algorithm to find the defining ideal of the Rees algebra using the definitions of Rees algebra. This method does not have any requirements on the grade of the ideals or the domain-ness of the ring, but it seems to be slower than the previous method.

We can construct the Rees algebra of $I_i$ as the kernel of the map $\varphi_i : R[Y_{i1}, \ldots, Y_{in_i}] \to R[T_i]$ where $\varphi_i(Y_{ij}) = f_{ij}T_i$ for $j = 1, \ldots, n_i$. Notice that $(\ker \varphi_i)R[\underline{Y}] \subseteq \ker \varphi$. Suppose that $\phi_i$ is the presentation matrix of $I_i$. Then the symmetric algebra $\mathrm{Sym}(I_i)$ has a presentation $R[Y_{i1}, \ldots, Y_{in_i}]/\mathscr{L}_i$ where $\mathscr{L}_i = I_1([Y_{i1}, \ldots, Y_{in_i}] \cdot \phi_i)$. Clearly, $\mathscr{L}_i \subseteq \ker \varphi_i \subset \ker \varphi$. So the map $\varphi_i$ factors through the symmetric algebra $\mathrm{Sym}(I_i)$. Now $\mathrm{Sym}(I_1) \otimes \cdots \otimes \mathrm{Sym}(I_s)$ has the presentation $R[\underline{Y}]/(\mathscr{L}_1 + \cdots + \mathscr{L}_s)$. Since $\mathscr{L}_i \subseteq \ker \varphi$, the map $\varphi$ also factors through $\mathrm{Sym}(I_1) \otimes \cdots \otimes \mathrm{Sym}(I_s)$. Thus to find the defining ideal of the multi Rees algebra $\mathcal{R}(I_1, \ldots, I_s)$ it is enough to find the kernel of the surjective map $\mathrm{Sym}(I_1) \otimes \cdots \otimes \mathrm{Sym}(I_s) \to \mathcal{R}(I_1, \ldots, I_s)$.

**Algorithm** (version II: `multiReesIdeal`, no assumptions). Let $I_1, \ldots, I_s$ be ideals in the Noetherian ring $R$.

**Input**: The list $W = \{I_1, \ldots, I_s\}$.

(1) For each ideal $I_i$, compute the presentation $F_i' \xrightarrow{\phi_i} F_i \to R_i \to 0$ where $\phi_i$ is the presentation matrix of $I_i$.

(2) Now compute the source symmetric algebra $\mathrm{Sym}(F_1') \otimes \cdots \otimes \mathrm{Sym}(F_s')$ and the target symmetric algebra $\mathrm{Sym}(F_1) \otimes \cdots \otimes \mathrm{Sym}(F_s)$ of the map $\phi_1 \otimes \cdots \otimes \phi_s$.

(3) Compute the map between the symmetric algebra of the source and target and return the kernel of the above map.

**Output**: The defining ideal of the Rees algebra $\mathcal{R}(I_1, \ldots, I_s)$.

In the following example, the ring $U$ is not a domain and hence the algorithm uses the above method. As expected, the computational time in the case where a nonzerodivisor is given as an optional input is faster than the case where no optional input is given.

**Example 2.4.**

```
i1 : T = QQ[a,b,c];

i2 : m = matrix{{a,b,c},{b,c,a}};

i3 : U = T/minors(2,m);

i4 : J = ideal vars U;

i5 : time multiReesIdeal J;
-- used 0.0977545 seconds

i6 : time multiReesIdeal (J, a);
-- used 0.0142101 seconds
```

**3.** COMPUTATION OF MIXED MULTIPLICITIES OF IDEALS. Let $I_1, \ldots, I_r$ be ideals of positive height in a local ring $(A, \mathfrak{m})$ (or a standard graded algebra over a field and $\mathfrak{m}$ be the maximal graded ideal) and let $I_0$ be an $\mathfrak{m}$-primary ideal. In [Katz and Verma 1989], the authors prove $\ell(I_0^{u_0} I_1^{u_1} \cdots I_r^{u_r} / I_0^{u_0+1} I_1^{u_1} \cdots I_r^{u_r})$ is a polynomial $P(\underline{u})$, for $u_i$ large, where $\underline{u} = (u_0, \ldots, u_r)$. Write this polynomial in the form

$$P(\underline{u}) = \sum_{\substack{\alpha \in \mathbb{N}^{r+1} \\ |\alpha| = t}} \frac{1}{\alpha!} e_\alpha(I_0 \mid I_1, \ldots, I_r) u^\alpha + \text{ lower degree terms},$$

where $t = \deg P(\underline{u})$, $\alpha = (\alpha_0, \ldots, \alpha_r) \in \mathbb{N}^{r+1}$, $\alpha! = \prod_{i=0}^r \alpha_i!$ and $|\alpha| = \sum_{i=0}^r \alpha_i$. If $|\alpha| = t$, then $e_\alpha(I_0 \mid I_1, \ldots, I_r)$ are called the *mixed multiplicities* of the ideals $I_0, I_1, \ldots, I_r$.

**Theorem 3.1** [Trung and Verma 2007, Theorem 1.2]. *Set*

$$R = R(I_0 \mid I_1, \ldots, I_r) = \bigoplus_{(u_0, u_1, \ldots, u_r) \in \mathbb{N}^{r+1}} \frac{I_0^{u_0} I_1^{u_1} \cdots I_r^{u_r}}{I_0^{u_0+1} I_1^{u_1} \cdots I_r^{u_r}}.$$

*Assume that* $d = \dim A/(0 : I^\infty) \geq 1$, *where* $I = I_1 \cdots I_r$. *Then* $\deg P_R(\underline{u}) = d - 1$, *where* $P_R(\underline{u})$ *is the Hilbert polynomial of* $R$.

In [Verma et al. 1994], D. Katz, S. Mandal, and J. K. Verma found a precise formula for the Hilbert polynomial of the quotient of a bigraded algebra over an Artinian local ring. This result can be generalized

to the case of the quotient of a multigraded algebra over an Artinian local ring and we skip the proof as the technique is similar. Let $S$ be an Artinian local ring and $A = S[X_1, \ldots, X_r]$ be an $\mathbb{N}^r$-graded ring over $S$, where for $1 \le i \le r$, $X_i = \{X_i(0), \ldots, X_i(s_i)\}$ is a set of indeterminates. Set $\underline{u} = (u_1, \ldots, u_r) \in \mathbb{N}^r$ and $|u| = u_1 + \cdots + u_r$. Then $A = \bigoplus_{\underline{u} \in \mathbb{N}^r} A_{\underline{u}}$, where $A_{\underline{u}}$ is the $S$-module generated by monomials of the form $P_1 \cdots P_r$, where $P_i$ is a monomial of degree $u_i$ in $X_i$. An element in $A_{\underline{u}}$ is called multihomogeneous of degree $\underline{u}$. An ideal $I \subseteq A$ generated by multihomogeneous elements is called a multihomogeneous ideal. Then $R = A/I$ is an $\mathbb{N}^r$-graded algebra with $\underline{u}$-graded component $R_{\underline{u}} = A_{\underline{u}}/I_{\underline{u}}$. The Hilbert function of $R$ is defined as $H(\underline{u}) = \lambda(R_{\underline{u}})$, where $\lambda$ denotes the length as an $S$-module. Set $\underline{t}^{\underline{u}} = t_1^{u_1} \cdots t_r^{u_r}$. The Hilbert series of $R$ is given by $HS(R, \underline{t}) = \sum_{\underline{u} \in \mathbb{N}^r} \lambda(R_{\underline{u}}) \underline{t}^{\underline{u}}$. Then there exists a polynomial $N(t_1, \ldots, t_r) \in \mathbb{Z}[t_1, \ldots, t_r]$ so that $HS(R, \underline{t}) = N(t_1, \ldots, t_r)/((1-t_1)^{s_1+1} \cdots (1-t_r)^{s_r+1})$.

**Theorem 3.2.** *Write the Hilbert polynomial of $R$ as*

$$P(\underline{u}, R) = \sum_{\alpha=\underline{0}}^{\underline{s}} c_\alpha \binom{u_1+\alpha_1}{\alpha_1} \cdots \binom{u_r+\alpha_r}{\alpha_r}. \tag{1}$$

*Then*

$$c_\alpha = \frac{(-1)^{|\underline{s}-\alpha|}}{(s_1-\alpha_1)! \cdots (s_r-\alpha_r)!} \cdot \frac{\partial^{|\underline{s}-\alpha|} N}{\partial t_1^{s_1-\alpha_1} \cdots \partial t_r^{s_r-\alpha_r}} \Bigg|_{(t_1,\ldots,t_r)=\underline{1}}.$$

Note that

$$\binom{u_i+\alpha_i}{\alpha_i} = \frac{1}{\alpha_i!} u_i^{\alpha_i} + \text{ lower degree terms.}$$

So if we write $P(\underline{u})$ as in (1), then $c_\alpha = e_\alpha$ for all $\alpha \in \mathbb{N}^{r+1}$ with $|\alpha| = d-1$. Therefore, Theorem 3.2 gives an expression for $e_\alpha$.

**Remark 3.3.** Let $I_0', I_1', \ldots, I_r'$ denote the images of ideals $I_0, I_1, \ldots, I_r$ in the ring $A/(0 : I^\infty)$, where $I = I_1 \cdots I_r$. Put $R' = R(I_0' | I_1', \ldots, I_r')$. Then for $\underline{u}$ large, $P_R(\underline{u}) = P_{R'}(\underline{u})$ (see [Trung and Verma 2007, Theorem 1.2] for details). Therefore, in the case where grade $I_i = 0$ for some $i$, the user needs to work in the quotient ring $A/(0 : I^\infty)$ and input the images of the ideals in the quotient ring.

**Algorithm.** The algorithm for the function `mixedMultiplicity` uses the above ideas. Let $I_0, I_1, \ldots, I_r$ be a set of ideals of a Noetherian ring $R$ of dimension $d \ge 1$, where $I_0$ is primary to the maximal ideal and grade$(I_i) > 0$ for all $i$; $\underline{a} = (a_0, a_1, \ldots, a_r) \in \mathbb{N}^{r+1}$ with $|\underline{a}| = d-1$.

**Input**: The sequence $W = ((I_0, I_1, \ldots, I_r), (a_0, a_1, \ldots, a_r))$.

(1) Compute the defining ideal of the multi-Rees algebra using the function `multiReesIdeal` and use it to find the Hilbert series of $R(I_0 \mid I_1, \ldots, I_r)$.

(2) Extract the powers of $(1 - T_i)$ in the denominator of the Hilbert series.

(3) Calculate $e_{\underline{a}}$ using the formula given in Theorem 3.2.

**Output**: The mixed multiplicity $e_{\underline{a}}(I_0 \mid I_1, \ldots, I_r)$.

**Example 3.4.**

```
i1 : R = QQ[x,y,z,w];
i2 : I = ideal(x*y*w^3,x^2*y*w^2,x*y^3*w,x*y*z^3); m = ideal vars R;
i3 : mixedMultiplicity ((m,I,I,I),(0,1,1,1))
o3 = 6
```

When some ideal has grade zero, the following example explains how to compute the mixed multiplicity by using the fact that $(0 : I^\infty) = (0 : (I^t)^\infty)$ for all $t \geq 1$.

**Example 3.5.** Let $S = \mathbb{Q}[x, y, z, w]/(xz, yz)$, $\mathfrak{m} = (x, y, z, w)$, and $I = (x, y)$. Notice that grade $I = 0$, since $I \in \operatorname{Ass} S$.

```
i1 : S = QQ[x,y,z,w]/ideal(x*z, y*z);
i2 : I = ideal(x,y);
i3 : m = ideal vars S;
i4 : L = saturate(sub(ideal 0, S), I);
i5 : T = S/L;
i6 : J = sub(I,T); n = sub(m,T);
i7 : dim T
o7 = 3
i8 : mixedMultiplicity((n,J,J,J),(1,0,1,0))
o8 = 1
```

To calculate mixed multiplicities, the function `mixedMultiplicity` computes the Hilbert polynomial of the graded ring $\bigoplus I_0^{u_0} I_1^{u_1} \cdots I_r^{u_r}/I_0^{u_0+1} I_1^{u_1} \cdots I_r^{u_r}$. In particular, if $I_1, \ldots, I_r$ are also $\mathfrak{m}$-primary ideals, then $e_{(a_0,a_1,\ldots,a_r)}(I_0 \mid I_1, \ldots, I_r) = e(I_0^{[a_0+1]}, I_1^{[a_1]}, \ldots, I_r^{[a_r]})$ (see [Huneke and Swanson 2006, Definition 17.4.3]). So to compute the $(a_0 + 1, a_1, \ldots, a_r)$-th mixed multiplicity of $I_0, I_1, \ldots, I_r$, one needs to enter the sequence $(a_0, a_1, \ldots, a_r)$ in the function. The same is illustrated in the following example.

**Example 3.6.**

```
i1 : R = QQ[x,y,z];
i2 : m = ideal vars R;
i3 : f = z^5 + x*y^7 + x^15;
i4 : I = ideal(apply(0..2, i -> diff(R_i,f)));
i5 : mixedMultiplicity ((m,I),(2,0))
o5 = 1
i6 : mixedMultiplicity ((m,I),(1,1))
o6 = 4
```

**4.** MIXED VOLUME OF LATTICE POLYTOPES. The Minkowski sum of two polytopes $P$ and $Q$ in $\mathbb{R}^n$ is defined as the polytope $P + Q = \{a + b \mid a \in P, b \in Q\}$. The $n$-dimensional mixed volume of a collection of $n$ polytopes $Q_1, \ldots, Q_n$ in $\mathbb{R}^n$, denoted by $MV_n(Q_1, \ldots, Q_n)$, is the coefficient of $\lambda_1 \cdots \lambda_n$ in $\operatorname{vol}_n(\lambda_1 Q_1 + \cdots + \lambda_n Q_n)$. Given a collection of lattice polytopes in $\mathbb{R}^n$, Trung and Verma proved that $MV_n(Q_1, \ldots, Q_n)$ is equal to a mixed multiplicity of a set of homogeneous ideals.

**Corollary 4.1** [Trung and Verma 2007, Corollary 2.5]. *Let $Q_1, \ldots, Q_n$ be an arbitrary collection of lattice polytopes in $\mathbb{R}^n$. Let $R = k[x_0, x_1, \ldots, x_n]$ and let $\mathfrak{m}$ be the maximal graded ideal of $R$. Let $M_i$ be any set of monomials of the same degree in $R$ such that $Q_i$ is the convex hull of the lattice points of their dehomogenized monomials in $k[x_1, \ldots, x_n]$. Let $I_j$ be the ideal of $R$ generated by the monomials of $M_j$. Then $MV_n(Q_1, \ldots, Q_n) = e_{(0,1,\ldots,1)}(\mathfrak{m} \mid I_1, \ldots, I_n)$.*

We use this result to construct an algorithm which calculates the mixed volume of a collection of lattice polytopes. We also give an algorithm which outputs the homogeneous ideal corresponding to the vertices of a lattice polytope.

Let $Q$ be a lattice polytope in $\mathbb{R}^n$ with the set of vertices $\{p_1, \ldots, p_r\} \subseteq \mathbb{N}^n$. We first compute the corresponding homogeneous ideal $I$ in the ring $R = k[x_1, \ldots, x_{n+1}]$. We write a function `homIdealPolytope` which requires as input the list $W = \{p_1, p_2, \ldots, p_r\}$ and produces as output the homogeneous ideal corresponding to the lattice points of $Q$.

We write a function `mMixedVolume` to calculate the mixed volume of a collection of $n$ lattice polytopes in $\mathbb{R}^n$. Let $Q_1, \ldots, Q_n$ be an arbitrary collection of lattice polytopes in $\mathbb{R}^n$. Let $R = k[x_1, \ldots, x_{n+1}]$ and let $I_i$ be the homogeneous ideal of $R$ such that the polytope $Q_i$ is the convex hull of the lattice points of the dehomogenization of a set of monomials that generates $I_i$ in $k[x_1, \ldots, x_n]$, for all $i$. Each of these homogeneous ideals can be obtained by giving the lattice points of each polytope as input in the function `homIdealPolytope`. The function `mMixedVolume` takes the list $\{I_1, \ldots, I_n\}$ as input and produces the mixed volume of $Q_1, \ldots, Q_n$ as output. The function can also take the list of lists of vertices of the polytope as input to compute their mixed volume. Since calculating the mixed volume is the same as calculating a mixed multiplicity, the algorithm of the function `mMixedVolume` is similar to the algorithm of the function `mixedMultiplicity`.

**Example 4.2.** We calculate the mixed volume of a *cross polytope*. An $n$-cross polytope $\beta_n$ is the convex hull of the points formed by permuting the coordinates of $(\pm 1, 0, \ldots, 0) \in \mathbb{R}^n$:

$$\begin{aligned}\beta_n &= \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid |x_1| + \cdots + |x_n| \leq 1\} \\ &= \text{conv}\{(\pm 1, 0, \ldots, 0), (0, \pm 1, 0, \ldots, 0), \ldots, (0, 0, \ldots, 0, \pm 1)\}.\end{aligned}$$

The volume of an $n$-cross polytope is $2^n/n!$ [Betke and Henk 1993, Theorem 2.1] and hence the mixed volume is $2^n$. We say that a polytope is a $(0, 1)$-*polytope* if its vertex set is a subset of $\{0, 1\}^d$ of the unit cube. In this example, we calculate the mixed volume of a 2-cross polytope and a 2-dimensional $(0, 1)$-polytope.

```
i1 : A = {(0,1),(1,0),(0,-1),(-1,0)};
i2 : mMixedVolume {A,A}
o2 = 4
i3 : I = homIdealPolytope A;
i4 : B = {(0,0),(0,1),(1,0),(1,1)};
i5 : J = homIdealPolytope B;
i6 : mMixedVolume {I, sub(J, vars ring I)}
o6 = 4
```

The proposed function `mMixedVolume` takes less time to compute the mixed volume of a 3-cross polytope than the existing function `mixedVolume` in the `Polyhedra` package.

```
i1 : needsPackage "Polyhedra";

i2 : Q = crossPolytope 3;

i3 : time mixedVolume {Q,Q,Q};
        -- used 238.277 seconds

------- n-cross polytope

i4 : CP = n -> ( U = (i,p) -> (1..n)/(j -> if j == i then p else 0);
                 flatten toList apply(1..n, i -> toList(U(i,1), U(i,-1))) );

i5 : time mMixedVolume {CP(3),CP(3),CP(3)}
        -- used 3.71303 seconds

o5 = 8
```

**5.** SECTIONAL MILNOR NUMBERS. In this section, we give an algorithm to compute the sectional Milnor numbers. We use Teissier's observation of identifying the sectional Milnor numbers with mixed multiplicities to achieve this task. Teissier [1973] conjectured that invariance of the Milnor number implies invariance of the sectional Milnor numbers. The conjecture was disproved by Joël Briançon and Jean-Paul Speder. We verify their example using our algorithm.

Suppose the origin is an isolated singular point of a complex analytic hypersurface $H = V(f) \subset \mathbb{C}^{n+1}$. Let $f_{z_i}$ denote the partial derivative of $f$ with respect to $z_i$. Set

$$\mu = \dim_{\mathbb{C}} \frac{\mathbb{C}\{z_0, z_1, \ldots, z_n\}}{(f_{z_0}, f_{z_1}, \ldots, f_{z_n})}.$$

The number $\mu$ is called the *Milnor number* of the hypersurface $H$ at the origin. Teissier, in his Cargèse paper [1973], refined the notion of Milnor number by replacing it with a sequence of Milnor numbers of intersections with general linear subspaces. Let $(X, x)$ be a germ of a hypersurface in $\mathbb{C}^{n+1}$ with an isolated singularity. The Milnor number of $X \cap E$, where $E$ is a general linear subspace of dimension $i$ passing through $x$, is called the $i^{\text{th}}$-sectional Milnor number of $X$. It is denoted by $\mu^{(i)}(X, x)$. Let $J(f) = (f_{z_0}, \ldots, f_{z_n})$ be the Jacobian ideal. In 1973, Teissier proved that the $i^{\text{th}}$-mixed multiplicity, $e(\mathfrak{m}^{[n-i]}, J(f)^{[i]})$, is equal to the $i^{\text{th}}$-sectional Milnor number of the singularity. Here $\mathfrak{m} = (z_0, z_1, \ldots, z_n)$.

Let $R = \mathbb{C}[x_1, \ldots, x_n]$ be a polynomial ring in $n$ variables, let $\mathfrak{m}$ be the maximal graded ideal and let $f \in R$ be any polynomial with an isolated singularity at the origin. Using Theorem 3.2, one can now calculate the mixed multiplicities of $\mathfrak{m}$ and $J(f)$. We use the ideas in the previous section to write a function `secMilnorNumbers` for computing the first $n-1$ sectional Milnor numbers. With a polynomial $f$ given as input, the algorithm calculates the Jacobian ideal of $f$ and then using the function `multiReesIdeal`, it finds the defining ideal of $\mathcal{R}(\mathfrak{m}, J(f))$. This helps to compute the Hilbert series of the special fiber $\mathcal{F}(\mathfrak{m}, J(f)) = \mathcal{R}(\mathfrak{m}, J(f)) \otimes_R R/\mathfrak{m}$. Using the formula given in Theorem 3.2, it then calculates the mixed multiplicities. Note that the $n^{\text{th}}$-sectional Milnor number is the Milnor number of the hypersurface $f$ at the origin. So under the extra assumption that the ideal $J(f)$ is $\mathfrak{m}$-primary, we have $\mu^{(n)}(X, 0) = \dim_{\mathbb{C}} R/J(f)$. Together, the function `secMilnorNumbers` outputs $(\mu^{(0)}(X, 0), \mu^{(1)}(X, 0), \ldots, \mu^{(n)}(X, 0))$.

**Example 5.1.**

```
i1 : R = QQ[x,y,z];
i2 : f = x^2*y+y^2*z+z^3;
i3 : secMilnorNumbers(f)
o3 = HashTable{0 => 1}
               1 => 2
               2 => 4
               3 => 8
```

Joël Briançon and Jean-Paul Speder [1975] considered the family of hypersurfaces $\boldsymbol{X}_t \in \mathbb{C}^3$ defined by $F_t(x, y, z) = z^5 + ty^6z + xy^7 + x^{15} = 0$. They proved that the topological type of $\boldsymbol{X}_t$ is constant whereas the topological type of the section of $\boldsymbol{X}_t$ by a general plane varies. One can verify the example using the methods discussed above. For instance, consider the ideals

$$\mathfrak{m} = (x, y, z) \quad \text{and} \quad J(F_t) = (\partial F_t/\partial x, \partial F_t/\partial y, \partial F_t/\partial z)$$

in the ring $\mathbb{C}[x, y, z]$. In [Goel et al. 2023], an expository version of this article, we show that $e(\mathfrak{m}^{[1]}, J(F_t)^{[2]})$ depends on $t$ although $e(J(F_t))$ is independent of $t$. The following Macaulay2 session verifies the example given by Briançon and Speder.

```
i1 : QQ[t];
i2 : k = frac oo;
i3 : R = k[x,y,z];
i4 : f = z^5 + t*y^6*z + x*y^7 + x^15;
i5 : secMilnorNumbers (f)
o5 = HashTable{0 => 1 }
               1 => 4
               2 => 26
               3 => 364
i6 : g = z^5 + x*y^7 + x^15;
i7 : secMilnorNumbers (g)
o7 = HashTable{0 => 1 }
               1 => 4
               2 => 28
               3 => 364
```

ACKNOWLEDGEMENTS. We thank the reviewers for their comments to improve this article. The authors thank Wolfram Decker and David Eisenbud for their help and encouragement. The first author thanks D. Grayson and M. Stillman for their comments and suggestions to improve the exposition and fix the grading scheme in the algorithm. The first author is supported by a Fulbright–Nehru Postdoctoral Research Fellowship. The third author is grateful to the Infosys Foundation for providing partial financial support. At the beginning of the project, the first and third author were Ph.D. students at Indian Institute of Technology Bombay, Mumbai, India and were supported by UGC-SRF fellowship from the Government of India.

SUPPLEMENT. The online supplement contains version 3.0 of `MixedMultiplicity`.

## REFERENCES.


[Atiyah and Macdonald 2016] M. F. Atiyah and I. G. Macdonald, *Introduction to commutative algebra*, Westview Press, Boulder, CO, 2016. MR Zbl

[Betke and Henk 1993] U. Betke and M. Henk, "Intrinsic volumes and lattice points of crosspolytopes", *Monatsh. Math.* **115**:1-2 (1993), 27–33. MR Zbl

[Briançon and Speder 1975] J. Briançon and J.-P. Speder, "La trivialité topologique n'implique pas les conditions de Whitney", *C. R. Acad. Sci. Paris Sér. A-B* **280**:6 (1975), 365–367. MR Zbl

[Bruns and Herzog 1993] W. Bruns and J. Herzog, *Cohen–Macaulay rings*, Cambridge Studies in Advanced Mathematics **39**, Cambridge University Press, 1993. MR

[Cox et al. 2019] D. A. Cox, K.-N. Lin, and G. Sosa, "Multi-Rees algebras and toric dynamical systems", *Proc. Amer. Math. Soc.* **147**:11 (2019), 4605–4616. MR Zbl

[D'Cruz 2003] C. D'Cruz, "A formula for the multiplicity of the multi-graded extended Rees algebra", *Comm. Algebra* **31**:6 (2003), 2573–2585. MR Zbl

[Eisenbud 2018] D. Eisenbud, "The ReesAlgebra package in Macaulay2", *J. Softw. Algebra Geom.* **8** (2018), 49–60. MR Zbl

[Eisenbud et al. 2003] D. Eisenbud, C. Huneke, and B. Ulrich, "What is the Rees algebra of a module?", *Proc. Amer. Math. Soc.* **131**:3 (2003), 701–708. MR Zbl

[Goel et al. 2023] K. Goel, V. Mukundan, S. Roy, and J. K. Verma, "Computing mixed multiplicities, mixed volumes, and sectional Milnor numbers", 2023. arXiv 2307.10124

[Herrmann et al. 1997] M. Herrmann, E. Hyry, J. Ribbe, and Z. Tang, "Reduction numbers and multiplicities of multigraded structures", *J. Algebra* **197**:2 (1997), 311–341. MR Zbl

[Huneke and Swanson 2006] C. Huneke and I. Swanson, *Integral closure of ideals, rings, and modules*, London Mathematical Society Lecture Note Series **336**, Cambridge University Press, 2006. MR Zbl

[Jabarnejad 2018] B. Jabarnejad, "Equations defining the multi-Rees algebras of powers of an ideal", *J. Pure Appl. Algebra* **222**:7 (2018), 1906–1910. MR Zbl

[Katz and Verma 1989] D. Katz and J. K. Verma, "Extended Rees algebras and mixed multiplicities", *Math. Z.* **202**:1 (1989), 111–128. MR Zbl

[Lin and Polini 2014] K.-N. Lin and C. Polini, "Rees algebras of truncations of complete intersections", *J. Algebra* **410** (2014), 36–52. MR Zbl

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

[Ribbe 1999] J. Ribbe, "On the defining equations of multi-graded rings", *Comm. Algebra* **27**:3 (1999), 1393–1402. MR Zbl

[Sosa 2014] G. Sosa, "On the Koszulness of multi-Rees algebras of certain strongly stable ideals", 2014. arXiv 1406.2188v1

[Teissier 1973] B. Teissier, "Cycles évanescents, sections planes et conditions de Whitney", pp. 285–362 in *Singularités à Cargèse* (*Rencontre Singularités Géom. Anal.*, *Inst. Études Sci.*) (Cargèse, 1972), Astérisque **7 et 8**, Soc. Math. France, Paris, 1973. MR Zbl

[Trung and Verma 2007] N. V. Trung and J. Verma, "Mixed multiplicities of ideals versus mixed volumes of polytopes", *Trans. Amer. Math. Soc.* **359**:10 (2007), 4711–4727. MR Zbl

[Verma 1992] J. K. Verma, "Multigraded Rees algebras and mixed multiplicities", *J. Pure Appl. Algebra* **77**:2 (1992), 219–228. MR Zbl

[Verma et al. 1994] J. K. Verma, D. Katz, and S. Mandal, "Hilbert functions of bigraded algebras", pp. 291–302 in *Commutative algebra* (Trieste, 1992), World Sci. Publ., River Edge, NJ, 1994. MR


RECEIVED: 14 Aug 2020 REVISED: 19 May 2023 ACCEPTED: 6 Jun 2023

KRITI GOEL:

kritigoel.maths@gmail.com
Department of Mathematics, University of Utah, Salt Lake City, UT, United States

VIVEK MUKUNDAN:

vmukunda@iitd.ac.in
Department of Mathematics, Indian Institute of Technology Delhi, Delhi, India

SUDESHNA ROY:

sudeshnaroy.11@gmail.com
Department of Mathematics, Chennai Mathematical Institute, Kelambakkam, India

JUGAL VERMA:

jkv@iitb.ac.in
Department of Mathematics, Indian Institute of Technology Bombay, Mumbai, India

PROOFS - PAGE NUMBERS ARE TEMPORARY

*JSAG* **13** (2023), 13–31
https://doi.org/10.2140/jsag.2023.13.13

The Journal of Software for
Algebra and Geometry

# FastMinors package for Macaulay2

BOYANA MARTINOVA, MARCUS ROBINSON, KARL SCHWEDE AND YUHUI YAO

ABSTRACT: In this article, we present `FastMinors.m2`, a package in *Macaulay*2 designed to introduce new methods focused on computations in function field linear algebra. Some key functionality that our package offers includes: finding a submatrix of a given rank in a provided matrix (when present), verifying that a ring is regular in codimension n, recursively computing the ideals of minors in a matrix, and finding an upper bound of the projective dimension of a module.

**1.** INTRODUCTION. We start with some motivation. Suppose that $I = (f_1, \ldots, f_m) \subseteq k[x_1, \ldots, x_n]$ is a prime ideal. The corresponding variety $X := V(I)$ is nonsingular if and only if $I$ plus the ideal generated by the minors of size $n - \dim X$ of the Jacobian matrix

$$\operatorname{Jac}(X) = \left(\frac{\partial f_i}{\partial x_j}\right)$$

generates the unit ideal. Unfortunately, even for relatively small values of $m$ and $n$, the number of such submatrices is prohibitive. Suppose for instance that $n = 10$, $m = 15$ and $\dim X = 5$. Then there are

$$\binom{10}{5} \cdot \binom{15}{5} = 756756$$

such submatrices. We cannot reasonably compute all of their determinants. This package attempts to fix this in several ways.

(1) We try to compute just a portion of the determinants, in a relatively smart way.

(2) We offer some tools for computing determinants that are sometimes faster.

Our techniques have also been applied to the related problem of showing that the singular locus has a certain codimension (for example, checking that a variety is R1 in order to prove normality). Of course, computing the singular locus is not the only potential application. We provide a function for giving a better upper bound on the projective dimension of a non-homogeneous module. Finally, this package has also been applied in the `RationalMaps` *Macaulay*2 package.

Martinova was supported by a University of Utah Mathematics REU fellowship and by the University of Utah ACCESS program. Robinson was supported by NSF RTG grant #1840190. Schwede was supported by NSF CAREER grant #1501102, NSF grants #1801849 and #2101800, NSF FRG Grant #1952522 and a fellowship from the Simons Foundation. Yao was supported by a University of Utah Mathematics REU fellowship.

*MSC2020:* 13D02, 14B05, 14H05, 15A15.

*Keywords:* FastMinors, Macaulay2.

`FastMinors.m2` version 1.2.6

© 2023 The Authors, under license to MSP (Mathematical Sciences Publishers).

We provide the following functions:

- `getSubmatrixOfRank`, which tries to find a submatrix of a given rank; see Section 4.
- `isRankAtLeast`, which uses `getSubmatrixOfRank` to try to find lower bounds for the rank of a matrix; see Section 5.
- `regularInCodimension`, which tries to verify if an integral domain is regular in codimension n; see Section 6.
- `projDim`, which tries to find upper bounds for the projective dimension of a non-homogeneous module; see Section 7.
- `recursiveMinors`, which computes the ideal of minors of a matrix via a recursive cofactor algorithm, as opposed to the included non-recursive cofactor algorithm; see Section 8.

Version 1.2.6 of this package is available as an online supplement to this paper. Later versions will be available at https://github.com/kschwede/M2/blob/master/M2/Macaulay2/packages/FastMinors.m2

This paper refers to `FastMinors` version 1.2.2. Earlier versions are also available in the *Macaulay*2 build tree.

## 2. Finding interesting submatrices.

A lot of the speedups available in the package come down to finding interesting square submatrices of a given matrix. For example, it is often useful to compute a square submatrix whose determinant has small degree. The idea is that the determinant of this submatrix will be less likely to vanish.

### 2.1. *How are the submatrices chosen?*

Consider the following matrix defined over $\mathbb{Q}[x, y]$:

$$\begin{bmatrix} x & xy & 0 \\ xy^2 & x^6 & 0 \\ 0 & x^2y^3 & xy^4 \end{bmatrix}.$$

Suppose we want to choose a submatrix of size $2 \times 2$. Consider the monomial order `Lex` where $x < y$. We find, in the matrix, the nonzero element of smallest order. In this case, that is $x$. We choose this element to be a part of our submatrix. Hence our submatrix will include the first row and column as well:

$$\begin{bmatrix} \boldsymbol{x} & xy & 0 \\ xy^2 & x^6 & 0 \\ 0 & x^2y^3 & xy^4 \end{bmatrix}.$$

To find the next element, we discard that row and column containing this term. Now, the next smallest element with respect to our monomial order is $xy^4$:

$$\begin{bmatrix} x^6 & 0 \\ x^2y^3 & xy^4 \end{bmatrix} \quad \begin{bmatrix} x^6 & 0 \\ x^2y^3 & \boldsymbol{xy^4} \end{bmatrix}.$$

Since we are only looking for a $2 \times 2$ submatrix, we stop here. We have selected the submatrix with rows 0 and 2 and columns 0 and 2:

$$\begin{bmatrix} x & 0 \\ 0 & xy^4 \end{bmatrix}.$$

The determinant of this submatrix is $x^2y^4$. This happens to be the smallest $2 \times 2$ minor with respect to the given monomial order (which frequently happens, although it is certainly not always the case).

If we choose a different monomial order, we get a different submatrix, with a different determinant. For example,

- `Lex`, $x > y$. We obtain the submatrix with rows 0 and 1 and columns 0 and 1, whose determinant is $x^7 - xy^3$.
- `GRevLex`, $x < y$. We obtain the submatrix with rows 0 and 2 and columns 0 and 1, whose determinant is $x^3y^3$.

For any of these strategies, in this package, we randomize the order of the variables before choosing a submatrix.

**2.2.** ***Ways of choosing submatrices.*** In the end we have the following methods to select submatrices:

`GRevLexLargest:` Choose entries which are largest with respect to a random `GRevLex` order.

`GRevLexSmallest:` Choose nonzero entries which are smallest with respect to a random `GRevLex` order.

`GRevLexSmallestTerm:` Choose nonzero entries which have the smallest terms with respect to a random `GRevLex` order.

`LexLargest:` Choose entries which are largest with respect to a random `Lex` order.

`LexSmallest:` Choose nonzero entries which are smallest with respect to a random `Lex` order.

`LexSmallestTerm:` Choose nonzero entries which have the smallest terms with respect to a random `GRevLex` order.

`Points:` Choose a submatrix whose determinant does not vanish at a random point found on a given ideal.

`Random:` Choose random entries.

`RandomNonzero:` Choose random nonzero entries.

However, from the end user's perspective, normally we find multiple minors, and the strategy will combine several of these methods (one typically does not know which method will work best in a given situation). For instance, the first minor might be selected by `GRevLexSmallest` and the second minor by `Random`. How to arrange what method is used (and with what probability) is described in Section 3.1.

We now describe each of these methods for selecting a submatrix in more detail. Note we have already described `Lex` and given an initial description of `GRevLex`.

**2.3.** `LexSmallestTerm` ***and*** `GRevLexSmallestTerm`**.** If we have a matrix whose entries are not monomial, then we could reasonably either pick the submatrix of the smallest entries with respect to our monomial order: `LexSmallest` or `GRevLexSmallest`.

Alternatively, we can pick the submatrix whose entries have the smallest terms via `LexSmallestTerm` or `GRevLexSmallestTerm`.

For example, consider the matrix

$$\begin{bmatrix} x^2+y^2 & 0 & xy+2x \\ y^4-x & 0 & 3x^5 \\ x^3 & x^4y^5-y^8 & 0 \end{bmatrix}.$$

In this case, if we are choosing the entries with the smallest terms, we first replace each entry in the matrix with the smallest term. For example, if we are using `LexSmallestTerm` with $x < y$, we would obtain

$$\begin{bmatrix} x^2 & 0 & 2x \\ -x & 0 & 3x^5 \\ x^3 & x^4y^5 & 0 \end{bmatrix}.$$

Then we proceed as before. Notice that if there is a tie, it is broken randomly.

**Remark.** Different strategies work differently on different examples. When working with a non-homogeneous matrix, with some entries that have constant terms, those entries will always be chosen first in `LexSmallestTerm` or `GRevLexSmallestTerm`, regardless of the monomial order. On the other hand, for homogeneous matrices, choosing the smallest term is frequently very effective.

**2.4.** `GRevLexLargest` ***and*** `LexLargest`**.** While we can imagine uses for these, in most cases these strategies appear to be worse than random. Indeed, submatrices picked this way seem likely to already vanish everywhere of interest.

**2.5.** `Points`**.** Instead of finding interesting submatrices by inspection, we can alternatively find submatrices by trying to find rational points. In that case, typically we are trying to find a submatrix with full rank on a certain subvariety, defined by an ideal $J$. We use the package `RandomRationalPoints` [Bisui et al.] to find a (rational) point $Q$ on $V(J)$ (or a point over some finite extension of our base field). We then evaluate our entire matrix at that point. Because we now have a matrix over a field, we use the very fast built-in commands to find pivot rows and columns, and thus find a submatrix of the desired rank. To use this functionality, use the strategy `Points`.

Currently, this functionality only works over a finite field. In characteristic zero, the `Points` strategy returns random submatrices.

For example, suppose we are working over $\mathbb{F}_5[x, y, z]$ with an ideal $I = (z^2y - x(x-z)(x+z))$, with the matrix

$$M = \begin{bmatrix} x^2 & xy & 3y^2 \\ x^3+y^3 & x^2+z^2 & y^2+z^2 \\ x^2*z & z^2*y & y^2*x \end{bmatrix}.$$

Suppose we found the point $(2, 0, 2)$ on this elliptic curve. We then evaluate our matrix at that point to obtain

$$M = \begin{bmatrix} 4 & 0 & 0 \\ 3 & 3 & 4 \\ 3 & 0 & 0 \end{bmatrix}.$$

We would then identify a submatrix with nonzero determinant, for instance the top left $2 \times 2$ submatrix $\left[\begin{smallmatrix} 4 & 0 \\ 3 & 3 \end{smallmatrix}\right]$ and then return the top determinant of the top $2 \times 2$ submatrix of the original matrix:

$$\det \begin{bmatrix} x^2 & xy \\ x^3 + y^3 & x^2 + z^2 \end{bmatrix} = x^4 + x^2z^2 - x^4y - xy^4.$$

Typically, the `Points` strategy will find much better matrices than any of the heuristic methods `LexSmallest`, `GRevLexSmallestTerm`, etc., allowing it to compute fewer determinants. However, there is still a substantial amount of work needed to find each submatrix. In our experience, the heuristic methods above perform better than `Points` roughly half of the time. If the user is implementing the `Points` strategy, we make two recommendations to optimize performance:

(i) Set the option `MaxMinors` (the maximum number of minors to be computed) to a relatively low number.

(ii) Set the option `CodimCheckFunction` to a linear function (such as `t -> t`). This will force the dimension of the ideal of minors computed so far to be checked more frequently (in our example, after adding every new minor to the ideal of minors)

The tutorial `RegularInCodimensionTutorial` in the package documentation contains further discussion of `MaxMinors`, `CodimCheckFunction` and other options.

**2.6.** `Random` ***and*** `RandomNonzero`**.**

`Random`: With this strategy, a random submatrix is chosen.

`RandomNonzero`: With this strategy, a random nonzero element is chosen in each step following the method used by the other strategies. This guarantees a submatrix where no row or column is zero, which can be very useful when dealing with relatively sparse matrices.

*More on* `GRevLex`*: modifying the underlying matrix.* Finally, when using the `GRevLexSmallest` and `GRevLexSmallestTerm` methods, we periodically change the underlying matrix by replacing terms of small order with terms of larger order in order to avoid recomputing the same submatrix. For example, in the matrix

$$\begin{bmatrix} x^2 & 0 & xy \\ y^4 & 0 & x^5 \\ x^3 & x^4y^5 & 0 \end{bmatrix},$$

after several iterations, we might replace the $x^2$ term with

$$x^2 \cdot (\text{a random degree 1 polynomial}).$$

which might look something like

$$\begin{bmatrix} x^2(2x-7y) & 0 & xy \\ y^4 & 0 & x^5 \\ x^3 & x^4y^5 & 0 \end{bmatrix}.$$

This forces the algorithm to make different choices. After several minors are selected, the matrix is reset again to its original form.

**3.** `chooseGoodMinors` AND SUBMATRIX SELECTION CONTROL. The function `chooseGoodMinors` tries to choose interesting submatrices of a given matrix. This is done by running the command

```
i1 : R = QQ[x, y, z];
     chooseGoodMinors()
```

**3.1.** ***The*** `Strategy` ***option.*** The core features included in the package allow the user to choose which methods from Section 2.2 should be used when selecting submatrices. This is done most easily by setting a `Strategy` option to one of the ways of choosing submatrices as above: `GRevLexSmallest`, `GRevLexSmallestTerm`, `GRevLexLargest`, `LexSmallest`, `LexSmallestTerm`, `LexLargest`, `Points`, `Random`, `RandomNonzero`. However, most of the time it is best to choose several strategies simultaneously, as one doesn't know which strategy will perform the best (in some cases, a combination works best). Hence instead of choosing a strategy which uses only one method, by default we use several. Thus you can set the `Strategy` option to one of the following:

- `StrategyDefault`: This strategy uses `LexSmallest`, `LexSmallestTerm`, `GRevLexSmallest`, `GRevLexSmallestTerm`, `Random` and `RandomNonzero` with equal probability.
- `StrategyDefaultNonRandom`: This uses `LexSmallest`, `LexSmallestTerm`, `GRevLexSmallest` and `GRevLexSmallestTerm` with equal probability.
- `StrategyDefaultWithPoints`: This strategy uses `Points` one third of the time and `LexSmallest`, `LexSmallestTerm`, `GRevLexSmallest` and `GRevLexSmallestTerm` with equal probability the rest of the time.
- `StrategyLexSmallest`: This strategy chooses 50% of the submatrices using `LexSmallest` and 50% using `LexSmallestTerm`.
- `StrategyGRevLexSmallest`: This chooses 50% of the submatrices using `GRevLexSmallest` and 50% using `GRevLexLargest`.
- `StrategyPoints`: This chooses submatrices by finding rational points, evaluating the submatrix at those points, and then doing a computation.
- `StrategyRandom`: This chooses submatrices by using 50% `Random` and 50% `RandomNonzero`.

The user can also create their own custom strategy by setting the `Strategy` parameter to a `HashTable` with the following keys: `LexLargest`, `LexSmallestTerm`, `LexSmallest`, `GRevLexSmallestTerm`, `GrevLexSmallest`, `GRevLexLargest`, `Random`, `RandomNonzero`, each with value an integer (the values

need not sum to 100). If one value is twice the size of another, that strategy will be employed twice as often. For example, `StrategyDefaultNonRandom` was created by the command

```
StrategyDefaultNonRandom = new HashTable from {
    LexLargest => 0,
    LexSmallestTerm => 25,
    LexSmallest => 25,
    GRevLexSmallestTerm => 25,
    GRevLexSmallest => 25,
    GRevLexLargest => 0,
    Random => 0,
    RandomNonzero => 0,
    Points => 0
};
```

For a tutorial on strategy choice, see the documentation, particularly `FastMinorsStrategyTutorial`.

**4.** FIND A SUBMATRIX OF A GIVEN RANK: `getSubmatrixOfRank`. This method examines the submatrices of an input matrix and attempts to find one of a given rank. If a submatrix with the specified rank is found, a list of two lists is returned. The first is the list of row indices and the second is the list of column indices, which describe the desired submatrix of the desired rank. If no such submatrix is found, the function will return `null`.

The option `MaxMinors` allows the user to control how many minors to consider before giving up. If left `null`, the number considered is based on the size of the matrix. This method will choose the indicated amount of minors using one of the strategy options described above. If one of the chosen submatrices has the desired rank, the function will terminate and return its rows and columns. This process continues until a submatrix is found or `MaxMinors` submatrices have been unsuccessfully checked. The strategy can be controlled using the `Strategy` option as described above; the default value is `StrategyDefaultNonRandom`.

**4.1.** ***Examples of*** `getSubmatrixOfRank`**.** In the following example, we first create a $3 \times 4$ matrix $M$ over $\mathbb{Q}[x, y, z]$. We execute two calls to `getSubmatrixOfRank`; the first has no `Strategy` parameter and the second utilizes `StrategyGRevLexSmallest`. Note that these calls return different indices, but both find valid rank 3 submatrices.

```
i1 : loadPackage "FastMinors";

i2 : R = QQ[x,y];

i3 : M = random(R^{2,2,2},R^4)

o3 = {-2} | x2+2/3xy+9/2y2    3/10x2+2/3xy+1/5y2 2x2+5/3xy+7/5y2    4/3x2+1/3xy+10/9y2 |
     {-2} | 3/2x2+2/3xy+2y2   1/2x2+3/2xy+3/4y2  6x2+5xy+4y2        9/5x2+1/5xy+7/2y2  |
     {-2} | 1/4x2+1/7xy+5/6y2 7/5x2+4xy+4/5y2    10/9x2+3/7xy+5/9y2 5/2x2+xy+7/6y2     |

               3       4
o3 : Matrix R  <--- R

i4 : getSubmatrixOfRank(3,M)

o4 = {{2, 0, 1}, {0, 1, 3}}

o4 : List

i5 : getSubmatrixOfRank(3, M, Strategy=>StrategyGRevLexSmallest)

o5 = {{0, 2, 1}, {1, 2, 0}}

o5 : List
```

In our next example, over a ring with 6 variables, we create a Jacobian matrix out of an ideal generated by 8 random forms of various degrees. We display the time needed for the `rank` function to return, followed by the time elapsed during a call to `getSubmatrixOfRank` when searching for a rank 6 submatrix. We find that `getSubmatrixOfRank` significantly outperformed `rank`:

```
i6 : R = ZZ/103[x_1..x_6]

o6 = R

o6 : PolynomialRing

i7 : J = jacobian ideal apply(8, i -> random(2+random(2), R));

              6       8
o7 : Matrix R  <--- R

i8 : time rank J
     -- used 21.8251 seconds

o8 = 6

i9 : time getSubmatrixOfRank(6, J)
     -- used 0.00714912 seconds

o9 = {{5, 1, 3, 4, 2, 0}, {5, 2, 6, 0, 4, 7}}
```

In one of the core examples from the `RationalMaps` package, before using this package a function would look at several thousands of submatrices (randomly) typically before finding a submatrix of the desired rank, whereas this package finds one after looking at fewer than half a dozen (typically only looking at 1 or 2 submatrices). Using this package sped up the computation of that example by more than one order of magnitude; see the non-maximal linear rank example from [Bott et al. 2022, page 7].

**5.** FINDING LOWER BOUNDS FOR MATRIX RANKS: `isRankAtLeast`. This method is a direct implementation of `getSubmatrixOfRank`. This function returns a boolean value indicating whether the rank of an input matrix, $M$, is greater than or equal to an input integer, $n$. In order to do so, the function first performs some basic checks to ensure a rank of $n$ is possible given $M$'s dimensions, then executes a call to `getSubmatrixOfRank`. If `getSubmatrixOfRank` returns a matrix, then this function will return true. However, if `getSubmatrixOfRank` does not return a matrix, a conclusive answer can not be reached. As such, the method will then evaluate the rank of $M$ and return the appropriate boolean value.

However, the function `isRankAtLeast`, which is efficient when `getSubmatrixOfRank` returns quickly, may be costly if the results are inconclusive and a rank evaluation is necessary. As such, the described implementation is not optimized. In order to lead to time improvements, we developed a multithreaded version of this function that simultaneously evaluates the rank of $M$ and invokes `getSubmatrixOfRank`. Once a thread has terminated with a usable answer, the other threads are canceled and the appropriate value is returned. During the implementation of this functionality, we discovered that *Macaulay*2 becomes unstable when canceling threads and thus users are not currently allowed to invoke the multithreaded version. However, this functionality is included in the package and can be made easily accessible once the stability issue is resolved.

**5.1.** ***Example of*** `isRankAtLeast`**.** The following example first creates a $9 \times 9$ matrix, $N$, and calls `isRankAtLeast` to determine whether its rank is at least 7. Directly calling `rank N` on a matrix of this

size would take multiple seconds, whereas `isRankAtLeast` returns in a fraction of the time:

```
i1 : loadPackage "FastMinors";

i2 : N = random(R^{6,6,6,6,6,6,6,7,7},R^9);
                9       9
o2 : Matrix R  <--- R

i3 : elapsedTime isRankAtLeast(7,N)
-- 0.0654172 seconds elapsed

o3 = true
```

**6.** REGULAR IN CODIMENSION $n$: `regularInCodimension`. Using the `getSubmatrixOfRank` routines, we provide a function for checking whether a variety is regular in codimension $n$, or $R_n$. The default strategy is `Strategy=>Default`.

The function `regularInCodimension(ZZ, Ring)` returns `true` if it verifies that the ring is regular in codimension $n$. This only works if the ring is equidimensional, as it is using a Jacobian criterion. If it cannot make a determination, it returns `null`. If it ended up computing all minors of the matrix, and it still doesn't have the desired codimension, it will return `false` (note this will likely only occur for small matrices).

**6.1.** ***Example of*** `regularInCodimension`**.** We begin with an example of a 3-dimensional ring that is regular in codimension 1, but not in codimension 2. It is generated by 12 equations in 7 variables:

```
i3 :  T = ZZ/101[x1,x2,x3,x4,x5,x6,x7];

i4 :  I =  ideal(x5*x6-x4*x7,x1*x6-x2*x7,x5^2-x1*x7,x4*x5-x2*x7,x4^2-x2*x6,x1*x4-x2*x5,
                 x2*x3^3*x5+3*x2*x3^2*x7+8*x2^2*x5+3*x3*x4*x7-8*x4*x7+x6*x7,
                 x1*x3^3*x5+3*x1*x3^2*x7+8*x1*x2*x5+3*x3*x5*x7-8*x5*x7+x7^2,
                 x2*x3^3*x4+3*x2*x3^2*x6+8*x2^2*x4+3*x3*x4*x6-8*x4*x6+x6^2,
                 x2^2*x3^3+3*x2*x3^2*x4+8*x2^3+3*x2*x3*x6-8*x2*x6+x4*x6,
                 x1*x2*x3^3+3*x2*x3^2*x5+8*x1*x2^2+3*x2*x3*x7-8*x2*x7+x4*x7,
                 x1^2*x3^3+3*x1*x3^2*x5+8*x1^2*x2+3*x1*x3*x7-8*x1*x7+x5*x7);

o4 : Ideal of T

i5 : S = T/I; dim S

o6 = 3

i7 : time regularInCodimension(1, S)
     -- used 0.150734 seconds

o7 = true

i8 : time regularInCodimension(2, S)
     -- used 2.12777 seconds

i9 : time singularLocus S;
     -- used 8.29746 seconds

i10 : time dim o9
-- used 23.2483 seconds

o10 = 1
```

As seen above, the function `regularInCodimension` verified that $S$ was regular in codimension 1 in a fraction of a second. When `regularInCodimension(2, S)` was called, nothing was returned, indicating that nothing was found (our function could not make a determination). Computing the Jacobian ideal however took more than 8 seconds and verifying that it had dimension 1 took more than 23 seconds.

**6.2.** ***Options and strategies for*** `regularInCodimension`**.** We consider the same example using some different strategies. For another look at options in this function, see the tutorial in the document under the key `RegularInCodimensionTutorial`.

One might think that it might be just as effective to choose random matrices as to use our strategies, and sometimes it is, but this is not the typical behavior we have observed.

```
i11 :  time regularInCodimension(1, S, Strategy=>StrategyRandom, Verbose=>true)
regularInCodimension: ring dimension =3, there are 17325 possible minors,
                      we will compute up to 317.599 of them.
regularInCodimension: About to enter loop
internalChooseMinor: Choosing Random
regularInCodimension:  Loop step, about to compute dimension.  Submatrices considered: 7,
                       and computed = 7
regularInCodimension:  isCodimAtLeast failed, computing codim.
regularInCodimension:  partial singular locus dimension computed, = 2
regularInCodimension:  Loop step, about to compute dimension.  Submatrices considered: 9,
                       and computed = 9
regularInCodimension:  isCodimAtLeast failed, computing codim.
regularInCodimension:  partial singular locus dimension computed, = 2
regularInCodimension:  Loop step, about to compute dimension.  Submatrices considered: 11,
                       and computed = 11
regularInCodimension:  isCodimAtLeast failed, computing codim.
regularInCodimension:  partial singular locus dimension computed, = 2
regularInCodimension:  Loop step, about to compute dimension.  Submatrices considered: 14,
                       and computed = 14
regularInCodimension:  isCodimAtLeast failed, computing codim.
regularInCodimension:  partial singular locus dimension computed, = 2
regularInCodimension:  Loop step, about to compute dimension.  Submatrices considered: 18,
                       and computed = 18
regularInCodimension:  isCodimAtLeast failed, computing codim.
regularInCodimension:  partial singular locus dimension computed, = 2
regularInCodimension:  Loop step, about to compute dimension.  Submatrices considered: 24,
                       and computed = 24
regularInCodimension:  isCodimAtLeast failed, computing codim.
regularInCodimension:  partial singular locus dimension computed, = 2
regularInCodimension:  Loop step, about to compute dimension.  Submatrices considered: 31,
                       and computed = 31
regularInCodimension:  isCodimAtLeast failed, computing codim.
regularInCodimension:  partial singular locus dimension computed, = 2
regularInCodimension:  Loop step, about to compute dimension.  Submatrices considered: 40,
                       and computed = 40
regularInCodimension:  isCodimAtLeast failed, computing codim.
regularInCodimension:  partial singular locus dimension computed, = 2
regularInCodimension:  Loop step, about to compute dimension.  Submatrices considered: 52,
                       and computed = 52
regularInCodimension:  isCodimAtLeast failed, computing codim.
regularInCodimension:  partial singular locus dimension computed, = 2
regularInCodimension:  Loop step, about to compute dimension.  Submatrices considered: 67,
                       and computed = 67
regularInCodimension:  isCodimAtLeast failed, computing codim.
regularInCodimension:  partial singular locus dimension computed, = 2
regularInCodimension:  Loop step, about to compute dimension.  Submatrices considered: 87,
                       and computed = 87
regularInCodimension:  isCodimAtLeast failed, computing codim.
regularInCodimension:  partial singular locus dimension computed, = 1
regularInCodimension:  Loop completed, submatrices considered = 87,
                       and computed = 87.
singular locus dimension appears to be = 1
     -- used 1.04945 seconds

o11 = true
```

Above, we have deleted 86 of the 87 times the verbose output displays `internalChooseMinor: Choosing Random`.

In this particular example, the `StrategyRandom` option looked at 87 submatrices of the Jacobian matrix. Note it does not check to see whether we have obtained the desired codimension after considering each new random submatrix. Instead, it only computes the codimension periodically, with the space between checks increasing. The `considered` values on each line tell how many submatrices have been considered. The `computed` value tells how many were not repeats (`computed` will be nearly the same as `considered` with a random strategy).

Running `Rn(1, S, Strategy=>StrategyRandom, Verbose=>true)` 50 times yielded:

(1) an average of 61.3 submatrices of the Jacobian matrix considered.

(2) a median value of 40 or 52 submatrices of the Jacobian matrix considered.

(3) a minimum value of 7 submatrices of the Jacobian matrix considered (1 time).

(4) a maximum value of 248 submatrices of the Jacobian matrix considered (1 time).

Due to certain settings, we do not check the codimension of the singular locus until 7 submatrices have been considered. Users can control this behavior via the `MinMinorsFunction` and `CodimCheckFunction` options; see the tutorial in the documentation.

The default strategy `Rn(1, S, Strategy=>StrategyDefaultNonRandom, Verbose=>true)`, on the other hand, run 50 times yielded

(1) an average of 12.1 submatrices of the Jacobian matrix considered.

(2) a median value of 7 or 9 submatrices of the Jacobian matrix considered.

(3) a minimum value of 7 submatrices of the Jacobian matrix considered (25 times).

(4) a maximum value of 40 submatrices of the Jacobian matrix considered (1 time).

In the above example, `Strategy=>StrategyLexSmallest` yields even better performance.

Using `Strategy=>StrategyPoints` (combined with the options `MinMinorsFunction=>(t->t)` and `CodimCheckFunction=>(t->t)`) to check codimension after computing every submatrix, produces:

(1) an average of 4.96 submatrices of the Jacobian matrix considered.

(2) a median value of 5 submatrices of the Jacobian matrix considered.

(3) a minimum value of 4 submatrices of the Jacobian matrix considered (3 times).

(4) a maximum value of 6 submatrices of the Jacobian matrix considered (1 time).

In this case, `StrategyPoints` considers very few submatrices, but it actually does the computation substantially slower than `StrategyDefaultNonRandom` since finding each submatrix can be a lot of work as rational points must be found. However, `StrategyPoints` is still faster than `StrategyRandom`.

Note that larger matrices tend to exhibit even larger disparities between the strategies.

**6.3.** ***Notes on implementation.*** As mentioned above, this function computes minors (based on the passed `Strategy` option) until either it finds that the singular locus has the desired dimension, or until it has considered too many minors. By default, it considers up to $10 \cdot$ (minimum number of minors needed) $+$ $8 \cdot \log_{1.3}$(possible minors). This value was simply chosen by experimentation. If the user is trying to show a singular locus has a certain codimension, they will need a minimum number of minors. The multiplication by 10 is due to our default strategy using multiple strategies, but only considering one might work well on a given matrix. The user can set the option `MaxMinors` to a function $F$ with two inputs, $x =$ (minors needed) and $y =$ (possible minors), where $F$ outputs the maximum number of matrices to compute. More simply, one may simply set `MaxMinors` to be a number.

These matrices are considered in a loop. We begin with computing a constant number of minors, by default $2 \cdot$(minimum number of minors needed)$+3$, and check whether the output has the right dimension. The user can also set the option `MinMinorsFunction` to a function $G$ with one input, $x =$ (minors needed), which will output how many minors to compute before first checking the codimension. After those initial minors are found, we compute additional minors, checking periodically (based on an exponential function, $1.3^k$ minors considered before the next reset) whether our minors define a subset of the desired codimension. New functions can be provided via the option `CodimCheckFunction`; see the tutorial for more details. If in this loop, a submatrix is considered again, it is not recomputed, but the counter is still increased.

**6.4.** ***Other options.*** This function also includes other options including the option `Modulus` which handles switching the coefficient field for a field of characteristic $p > 0$ (which is specified with `Modulus=>p`.)

One can also control how determinants are computed with the `DetStrategy` option; valid values are `Bareiss`, `Cofactor` and `Recursive`.

**7.** PROJECTIVE DIMENSION: `projDim`. In April of 2019, it was pointed out in a thread on github (https://github.com/Macaulay2/M2/issues/936) that the command `pdim` sometimes provides an incorrect value (an overestimate) for the projective dimension for non-homogeneous modules over polynomial rings. There, it was also suggested that this could be addressed by looking at appropriate minors of the matrices in a possibly non-minimal resolution, but that in practice these matrices have too many minors to compute. We have implemented a function `projDim` that tries to address this by looking at only *some* minors. Our function does not solve the problem as it also gives only an upper bound on the projective dimension. However, this upper bound is frequently correct.

The idea is as follows. Take a free resolution of a module $M$ over a polynomial ring $R$,

$$0 \longleftarrow M \longleftarrow F_0 \xleftarrow{d_1} F_1 \longleftarrow \cdots \xleftarrow{d_{n-1}} F_{n-1} \xleftarrow{d_n} F_n \longleftarrow 0.$$

Each $d_i$ is given by a matrix. The term $F_n$ is unnecessary (i.e., $d_n$ splits) exactly when the rank $F_n$ minors of $d_n$ generate the unit ideal. In that case, we know our projective dimension is at most $n-1$. Continuing in this way, we can compute the (rank $F_{n-1}$ $-$ rank $F_n$)-minors of $d_{n-1}$, and see whether they generate the unit ideal. Our algorithm of course only computes a subset of those minors.

**7.1.** ***Example of*** `projDim`**.** In the below example, we take a monomial ideal of projective dimension 2, compute a non-homogeneous change of coordinates, and observe that `pdim` returns an incorrect answer that `projDim` corrects:

```
i1 : R = QQ[x,y,z,w];
i2 : I = ideal(x^4,x*y,w^3, y^4);
i3 : pdim module I
o3 = 2
i4 : f = map(R, R, {x+x^2+1, x+y+1, z+z^4+x-2, w+w^5+y+1});
i5 : pdim module f I
o5 = 3
i6 : time projDim module f I
     -- used 3.43851 seconds
o6 = 2
i7 : time projDim(module f I, MinDimension=>2)
     -- used 0.0503165 seconds
o7 = 2
```

**7.2.** ***Options.*** As seen in the previous example, setting `MinDimension` can substantially speed up the computation, as otherwise the function will try to determine whether the projective dimension is actually 1.

The option `MaxMinors` can be set to be the number of minors computed at each step. Alternatively, it can be set to be a list of numbers, one for each step in the above algorithm. Finally, it can be set to be a function of the dimension $d$ of the polynomial ring $R$ and the number $t$ of possible minors. This is the default option, and the function is $5 * d + 2 * \log_{1.3}(t)$. The option `Strategy` is also available and it works as in the above functions with the default value being `StrategyDefault`.

**8.** COMPUTING IDEALS OF MINORS: `recursiveMinors`. *Macaulay*2 contains a `minors` method that returns the ideal of minors of a certain size, $n$, in a given matrix, a necessary step in locating singularities. However, the current implementation's default is to evaluate determinants using the Bareiss algorithm, which is efficient when the entries in the matrix have a low degree and few variables, but very slow otherwise. The current minors method also allows users to compute determinants using cofactor expansion, but this strategy performs some unnecessary calculations, causing it to be quite costly as well. We improved the current cofactor expansion method to find the determinants of minors by adding recursion and multithreading throughout. We also eliminated said unnecessary calculations by ensuring that only the required determinants are being computed at each step of the recursion, rather than all possible determinants of the given size.

In order to do so, we programmed a method in *Macaulay*2's software that recursively finds all $n \times n$ minors by first computing the $2 \times 2$ minors and storing them in a hash table. Then we use the $2 \times 2$ minors to compute the necessary $3 \times 3$ minors, and so on, with the process repeated recursively until the minors of size $n \times n$ are evaluated. At each step, we only compute the determinants that will be needed when performing a cofactor expansion on the following size minor.

To allow for further time improvements, we also utilized *Macaulay*2's existing parallel programming methods to multithread our code so different computations at each step of the recursion can occur simultaneously in separate threads. We divide the list of all determinants to be evaluated into different available threads and wait for them to finish before consolidating the results in a hash table and proceeding with the recursion. In order to more effectively utilize *Macaulay*2's multithreading methods, we also created a nanosleep method that waits a given number of nanoseconds, rather than full seconds. This function has already been incorporated into the software.

**8.1.** ***Example of*** `recursiveMinors`**.** Below, we first create a simple matrix, M, of polynomials in a single variable with rational coefficients and execute the `recursiveMinors` method to find the ideal of all $3 \times 3$ minors. As can be seen, the result is equivalent to the output of the minors method when called with the same parameters. We then create a new, larger matrix, *N*, with two dimensional rational coefficients and return the computation time for `recursiveMinors` and minors utilizing both the Bareiss and Cofactor strategies. The `recursiveMinors` method finished executing approximately six times faster than the Bareiss algorithm and almost seven times faster than the Cofactor expansion, while yielding the same results.

```
i1 : loadPackage "FastMinors";

i2 : allowableThreads => 8;

i3 : R = QQ[x];

i4 : M = random(R^{2,2,2}, R^4)

o4 = {-2} | x2    3x2   5/8x2 7/10x2 |
     {-2} | 3/4x2 2x2   7/4x2 9x2    |
     {-2} | x2    2/9x2 1/2x2 4/3x2  |
              3       4
o4 : Matrix R  <--- R

i5 : recursiveMinors(3,M)

             1403 6  449 6    292 6  517 6
o5 = ideal (----x , ---x , - ---x , ---x )
              60     240       45    144

o5 : Ideal of R

i6 : recursiveMinors(3,M) == minors(3,M)

o6 = true

i7 : Q = QQ[x,y];

i8 : N = random(Q^{5,5,5,5,5,5}, Q^7);
               6       7
o8 : Matrix Q  <--- Q

i9 : elapsedTime minors(5,N, Strategy => Bareiss);
-- 1.42867 seconds elapsed

o9 : Ideal of Q

i10 : elapsedTime minors(5,N, Strategy => Cofactor);
-- 1.82251 seconds elapsed

o10 : Ideal of Q

i11 : elapsedTime recursiveMinors(5,N);
-- 0.273007 seconds elapsed;

o11 : Ideal of Q

i12 : recursiveMinors(5,N) == minors(5,N)

o12 = true
```

PROOFS - PAGE NUMBERS ARE TEMPORARY



| Degree | Bareiss | Cofactor | RecursiveMinors | RecursiveMinors, Threads=>4 |
|---|---|---|---|---|
| 8 | 3.465 | 4.443 | 0.632 | 0.408 |
| 10 | 5.771 | 6.799 | 0.971 | 0.560 |
| 12 | 7.405 | 8.935 | 1.220 | 0.699 |
| 15 | 12.187 | 12.007 | 1.687 | 1.007 |
| 20 | 21.007 | 22.615 | 2.819 | 1.854 |
| 25 | 31.915 | 34.865 | 4.233 | 2.635 |
| 40 | 83.583 | 77.198 | 10.585 | 6.296 |
| 60 | 181.179 | 192.911 | 23.875 | 13.062 |

**Table 1.** Time to compute the $5 \times 5$ minors of a $6 \times 7$ random matrix over $\mathbb{Q}[x, y]$.

| Degree | Bareiss | Cofactor | RecursiveMinors | RecursiveMinors, Threads=>4 |
|---|---|---|---|---|
| 2 | 5.998 | 3.785 | 0.588 | 0.519 |
| 3 | 17.397 | 8.781 | 1.730 | 1.535 |
| 4 | 49.615 | 22.575 | 4.582 | 3.833 |
| 5 | 115.412 | 45.088 | 8.364 | 6.394 |

**Table 2.** Time to compute the $5 \times 5$ minors of a $6 \times 7$ random matrix over $\mathbb{Q}[x, y, z]$.

We briefly show the limits of this package in Tables 1 and 2. All times in the paper are given in seconds. We consider a random $6 \times 7$ matrix over $\mathbb{Q}[x, y]$ as above, and then also for $\mathbb{Q}[x, y, z]$. We compare the single-threaded and 4-threaded versions of `recursiveMinors` in this package with the `Bareiss` and `Cofactor` strategies with `recursiveMinors` for different degrees of the terms.

Generally speaking, `recursiveMinors` performs best when the matrix one is looking at has very-expensive-to-compute minors (such as with the random matrices we consider above). In sparse examples and examples with easy-to-compute determinants, other strategies tend to perform better.

**9.** PERFORMANCE AND LIMITS OF THE PACKAGE. We conclude by providing some tables showing how long various computations take in several different strategies. We limit ourselves to the function `regularInCodimension` as other functions such as `projDim` have roughly similar performance. Note that we have already discussed some of the performance behavior of taking determinants (including via a recursive algorithm). Again, we recommend the interested user also see the tutorial in the package documentation.

The Successful column shows what percentage of the time the function verified that the given equation was regular in a certain codimension (depending on the strategy, it doesn't always succeed). All computations were run in *Macaulay*2 version 1.18 on a machine running Ubuntu 20.04 with 64 gigabytes of memory.

In Table 3, we verify that the cone over a product of elliptic curves (an Abelian surface) embedded in $\mathbb{P}^8$ is regular in codimension 1. Note that `StrategyRandom` does not tend to work well on this or other examples, and so we generally do not consider it further. In Table 4 we verify the same example is regular in codimension 2. When we make the elliptic curves defined by less sparse equations, `Points` tends to perform much better, as can be seen in Table 5.

| Strategy | Attempts | Average time | Successful |
|---|---|---|---|
| `StrategyDefault` | 100 | 1.7 | 100% |
| `StrategyDefaultNonRandom` | 100 | 0.9 | 100% |
| `Points` | 100 | 4.0 | 100% |
| `StrategyDefaultWithPoints` | 100 | 2.2 | 100% |
| `StrategyRandom` | 100 | 6.1 | 4% |
| `StrategyRandom, MaxMinors=>2000` | 20 | 49.0 | 15% |
| `StrategyRandom, MaxMinors=>5000` | 10 | 238.1 | 50% |

Regular in codimension 1, 9 variables, 28 equations, 31646160 possible 6×6 minors

**Table 3.** We check $R$ is regular in codimension 1 where $R$ is the cone over a product of two elliptic curves in positive characteristic given with a Segre embedding. One of the curves is diagonal, the other is in Weierstrass form. This has a relatively sparse Jacobian matrix.

| Strategy | Attempts | Average time | Successful |
|---|---|---|---|
| `StrategyDefault` | 10 | 10.9 | 0% |
| `StrategyDefault, MaxMinors=>5000` | 10 | 30.1 | 100% |
| `StrategyDefaultNonRandom` | 10 | 7.7 | 0% |
| `StrategyDefaultNonRandom, MaxMinors=>5000` | 10 | 13.7 | 100% |
| `Points` | 10 | 4.6 | 100% |
| `StrategyDefaultWithPoints` | 10 | 5.8 | 100% |

Regular in codimension 2, 9 variables, 28 equations, 31646160 possible 6×6 minors

**Table 4.** We check $R$ is regular in codimension 2 where $R$ is the cone over a product of two elliptic curves in positive characteristic given with a Segre embedding. One of the curves is diagonal, the other is in Weierstrass form. This has a relatively sparse Jacobian matrix. Using `StrategyDefault` and `StrategyDefaultNonRandom` did not work with the default number of minors, but increasing `MaxMinors` led to successful verification that the ring was regular in codimension 2.

We next consider a relatively sparse higher dimension example in Table 6. Here we are taking a cone over a product of an elliptic curve with a diagonal equation, an elliptic curve in Weierstrass form and a copy of $\mathbb{P}^1$. This is a cone over a 3-dimensional smooth projective variety embedded in $\mathbb{P}^{17}$.

We now move on to computing dimensions of singular loci of varieties that are not cones. We constructed several non-normal (non-S2) varieties using the `Pullback` package. First, in Table 7 we took 3 coordinate axes through the origin in $\mathbb{A}^3$ and randomly glued them to a single line. In Table 8 we did the same with three random lines through the origin (creating a less sparse Jacobian matrix). Finally, in Table 9, we consider a similar example in $\mathbb{A}^4$ (except now it is regular in codimension 2), first verifying it is regular in codimension 1. Finally, we verify it is regular in codimension 2 in Table 10.

SUPPLEMENT. The online supplement contains version 1.2.6 of `FastMinors.m2`.

| Strategy | Attempts | Average time | Successful |
|---|---|---|---|
| `StrategyDefault` | 10 | $\infty$? | 0% |
| `StrategyDefaultNonRandom` | 10 | $\infty$? | $<$10% |
| `Points, CodimCheckFunction => t->t+1` | 10 | 7.7 | 100% |
| `StrategyDefaultWithPoints` | 10 | ? | about 50% |

Regular in codimension 1, 9 variables, 28 equations, 31646160 possible $6\times6$ minors

**Table 5.** We check $R$ is regular in codimension 1 where $R$ is the cone over a product of two elliptic curves in positive characteristic given with a Segre embedding. One of the curves is in Weierstrass form, the other is given by a random degree 3 equation. This has a relatively complicated (non-sparse) Jacobian matrix. The other strategies generally do not work. The one exception is `StrategyDefaultWithPoints` which sometimes is very fast (faster than `Points`), and other times gets stuck trying to compute a point. Setting `CodimCheckFunction => t->t+1` forces the codimension to be checked at every step, which provides better and more consistent performance. Without that, sometimes this function will hang trying to find a point after on a 1-dimensional scheme where it has already verified that $R$ is regular in codimension 1, but has not computed that codimension yet.

| Strategy | Attempts | Average time | Successful |
|---|---|---|---|
| `StrategyDefaultNonRandom` | 10 | $\infty$? | 0% |
| `Points` | 10 | 58.5 | 100% |
| `StrategyDefaultWithPoints` | 10 | 27.1 | 100% |

Regular in codimension 1, 18 variables, 139 equations, 17927476818965522386560 possible $14\times14$ minors

**Table 6.** We check $R$ is regular in codimension 1 where $R$ is the cone over a product of two elliptic curves plus a $\mathbb{P}^1$ in positive characteristic given with a Segre embedding. One of the curves is in Weierstrass form, the other is given by a random degree 3 equation. This has a relatively sparse Jacobian matrix. The other strategies (not involving points) generally do not work. Using `StrategyDefaultNonRandom` took more than 30 minutes and computed more than 5000 minors, but still did not finish.

| Strategy | Attempts | Average time | Successful |
|---|---|---|---|
| `StrategyDefault` | 100 | 0.8 | 100% |
| `StrategyDefaultNonRandom` | 100 | 0.5 | 100% |
| `Points` | 100 | 3.0 | 100% |
| `StrategyDefaultWithPoints` | 100 | 2.4 | 100% |

Regular in codimension 1, 8 variables, 26 equations, 3683680 possible $5\times5$ minors

**Table 7.** We check $R$ is regular in codimension 1 where $R$ is obtained by gluing three coordinate axis lines through the origin in $\mathbb{A}^3$ together to a single line. This is a 3-dimensional ring that is regular in codimension 1, but not codimension 2. The Jacobian matrix is fairly sparse, but has some quite complicated sections.

| Strategy | Attempts | Average time | Successful |
|---|---|---|---|
| `StrategyDefault` | 20 | 2.0 | 100% |
| `StrategyDefaultNonRandom` | 20 | 0.5 | 100% |
| `Points` | 10 | 11.6 | 100% |
| `StrategyDefaultWithPoints` | 10 | 6.9 | 100% |

Regular in codimension 1, 8 variables, 34 equations, 15582336 possible 5×5 minors

**Table 8.** We check $R$ is regular in codimension 1 where $R$ is obtained by gluing random lines through the origin in $\mathbb{A}^3$ together to a single line. This is a 3-dimensional ring that is regular in codimension 1, but not codimension 2. The Jacobian matrix is substantially less sparse than when we glued the three *coordinate axes*.

| Strategy | Attempts | Average time | Successful |
|---|---|---|---|
| `StrategyDefault` | 100 | 5.2 | 100% |
| `StrategyDefaultNonRandom` | 100 | 1.3 | 100% |
| `Points` | 20 | 7.3 | 100% |
| `StrategyDefaultWithPoints` | 20 | 4.0 | 100% |

Regular in codimension 1, 11 variables, 52 equations, 44148904800 possible 7×7 minors

**Table 9.** We check $R$ is regular in codimension 1 where $R$ is obtained by gluing three coordinate axis lines through the origin in $\mathbb{A}^4$ together to a single line. This is a 4-dimensional ring that is regular in codimension 2, but not codimension 3. The Jacobian matrix is fairly sparse, but has some quite complicated sections.

| Strategy | Attempts | Average time | Successful |
|---|---|---|---|
| `StrategyDefault` | 20 | 14.9 | 100% |
| `StrategyDefaultNonRandom` | 20 | 5.5 | 100% |
| `Points` | 10 | ∞? | 0% |
| `StrategyDefaultWithPoints` | 10 | ∞? | 0% |

Regular in codimension 2, 11 variables, 52 equations, 44148904800 possible 7×7 minors

**Table 10.** We check $R$ is regular in codimension 2 where $R$ is obtained by gluing three coordinate axis lines through the origin in $\mathbb{A}^4$ together to a single line. This is a 4-dimensional ring that is regular in codimension 2, but not codimension 3. The Jacobian matrix is fairly sparse, but has some quite complicated sections. Strategies involving `Points` fail quickly as they use more than 64 gigabytes of RAM.

ACKNOWLEDGEMENTS: The authors thank David Eisenbud, Dan Grayson, Eloísa Grifo and Zhuang He for valuable conversations and feedback.

## REFERENCES.

[Bisui et al.] S. Bisui, Z. Jiang, S. Maitra, T. Nguyen, F.-O. Schreyer, and K. Schwede, "RandomPoints: A *Macaulay*2 package", preprint.

PROOFS - PAGE NUMBERS ARE TEMPORARY



[Bott et al. 2022] C. J. Bott, S. H. Hassanzadeh, K. Schwede, and D. Smolkin, "RationalMaps, a package for Macaulay2", *J. Software for Algebra and Geometry* **12** (2022), 17–26. arXiv 1908.04337

[Pullback] D. Ellingson and K. Schwede, "Pullback: pullback in the category of rings", *Macaulay*2 package, version 1.03, available at http://www2.macaulay2.com/Macaulay2/doc/Macaulay2-1.18/share/doc/Macaulay2/Pullback/html/index.html.

RECEIVED: 24 Nov 2020 REVISED: 2 Aug 2021 ACCEPTED: 8 May 2023

BOYANA MARTINOVA:
martinova@wisc.edu
Department of Mathematics, University of Wisconsin, Madison, WI, United States

MARCUS ROBINSON:
mrobinso@reed.edu
Department of Mathematics, Reed College, Portland, OR, United States

KARL SCHWEDE:
schwede@math.utah.edu
Department of Mathematics, The University of Utah, Salt Lake City, UT, United States

YUHUI YAO:
weiy@math.uchicago.edu
Department of Mathematics, University of Chicago, Eckhart Hall, Chicago, IL, United States

msp

*JSAG* **13** (2023), 33–43
https://doi.org/10.2140/jsag.2023.13.33

The Journal of Software for
Algebra and Geometry

# Finding points on varieties with Macaulay2

SANKHANEEL BISUI, ZHAN JIANG, SARASIJ MAITRA,
THÁI THÀNH NGUYỄN AND KARL SCHWEDE

ABSTRACT: We present `RandomPoints`, a package in Macaulay2 designed mainly to identify rational and geometric points in a variety over a finite field. We provide tools to estimate the dimension of a variety. We also present methods to obtain nonvanishing minors of a given size in a given matrix, by evaluating the matrix at a point.

**1.** INTRODUCTION. Let $I$ be an ideal in a polynomial ring $k[x_1, \ldots, x_n]$ over a finite field $k$. Let $X := V(I)$ denote the corresponding set of rational points in affine $n$-space. Finding one such rational point or geometric point (geometric meaning a point over some finite field extension), in an algorithmically efficient manner, is our primary motivation for this package. The authors of the package are Sankhaneel Bisui, Zhan Jiang, Sarasij Maitra, Thái Thành Nguyễn, Frank-Olaf Schreyer, and Karl Schwede.

There is an existing package [RationalPoints], which we took inspiration from, which aims to find *all* the rational points of a variety; our aim here is to find one or more random rational or geometric points on a variety quickly. We also note that the package [Cremona] can find rational points on projective varieties, as can the core function `randomKRationalPoint` in [Macaulay2]. Our methods frequently appear to be faster and apply in the affine setting as well.

We develop functions that apply various strategies to generate random rational and geometric points on the given variety. We also provide functions that will expedite the process of determining properties of the singular locus of $X$.

We provide the following core functions:

- `randomPoints`: This tries to find a point in the vanishing set of an ideal. (Section 2)
- `dimViaBezout`: This tries to compute the dimension of an algebraic set by intersecting with hyperplanes. (Section 3.1)
- `projectionToHypersurface` and `genericProjection`: These functions provide customizable projection. (Section 4)

Schwede was supported by NSF Grants #1801849, #2101800, FRG #1952522 and a Fellowship from the Simons Foundation.
*MSC2020:* 13C99, 14G05.
*Keywords:* RandomPoints, Macaulay2.
`RandomPoints` version 1.5.3

© 2023 The Authors, under license to MSP (Mathematical Sciences Publishers).

- `findANonZeroMinor` and `extendIdealByNonZeroMinor`: The first of these finds a submatrix of a given matrix that is nonsingular at a point of a given ideal. The second adds said submatrix to an ideal, which is useful for computing partial Jacobian ideals. (Section 5.1)

All polynomial rings considered here will be over finite fields. In the subsequent sections, we explain the core and helper functions and describe the strategies that we have implemented.

**2.** OUR PRIMARY PURPOSE: `randomPoints`. We start with the core function `randomPoints`, which is a function to find rational or geometric points in a variety. The typical usage is `randomPoints(n, I)` where $n$ is a positive integer denoting the number of points desired, and $I$ is an ideal inside a polynomial ring. If $n$ is omitted, it is assumed to be 1.

**2.1.** ***Options.*** The user may also choose to provide some additional information, which may accelerate the computation and improve the probability that a point is found.

`Strategy`: This parameter can have the value `BruteForce`, `LinearIntersection` or `Default`.

- `BruteForce` simply tries random points and sees if they are on the variety.
- `LinearIntersection` intersects with a random linear space.
- `Default` performs the above strategies in sequence, beginning with `BruteForce`, then moving to `LinearIntersections` with particularly simple linear forms, and gradually ramping up the randomness of the linear forms.

The speed and the probability of success depend on the strategy (see also Section 3).

**Example 2.1.** Consider the following example.

```
i2 : R = ZZ/101[x, y, z];
i3 : J = ideal(x^3 + y^2 + 1, z^3 - x^2 - y^2 + 2);
o3 : Ideal of R
i4 : time randomPoints(J,Strategy=>BruteForce, PointCheckAttempts=>10)
        -- used 0.00186098 seconds
o4 = {}
o4 : List
i5 : time randomPoints(J)
        -- used 0.0205099 seconds
o5 = {{-1, 0, -1}}
o5 : List
i6 : time randomPoints(J, Strategy=>LinearIntersection)
        -- used 0.0334881 seconds
o6 = {{0, 10, 48}}
```

`ExtendField`: Intersection with a general linear space will naturally find scheme theoretic points that are not rational over the base field. Setting the boolean parameter `ExtendField` to be `true` will tell the function that such points are valid. Setting it to be `false` will tell the function to ignore such points. In fact, setting `ExtendField` to be `true` will also tell Macaulay2 to use linear spaces defined over a field extension, which can improve randomness properties. This sometimes can slow computation, and other

times can substantially speed it up when the variety has few rational points. For some applications, points over extended fields may also have better randomness properties.

`DecompositionStrategy`: Within the `LinearIntersection` strategy, one can also specify the option `DecompositionStrategy`. Valid values are `Decompose` and `MultiplicationTable`, the latter of which is currently only implemented for homogeneous ideals. The point is, after intersecting the linear space and obtaining an ideal defining a set of (possibly thickened) points, we need to find the minimal associated primes. By default we use Macaulay2's built-in `decompose` command. We also have implemented a `MultiplicationTable` algorithm, as provided by Frank-Olaf Schreyer, which utilizes the action of a variable on the residue fields of these points computed in more than one way. This method is frequently faster for rings with smaller numbers of variables.

The `Default` strategy switches back and forth between `Decompose` and `MultiplicationTable` for homogeneous ideals (starting with one the function thinks will be fastest). Setting this to `Decompose` in the default strategy will force only `Decompose` to be used; setting it to `MultiplicationTable` will force only `MultiplicationTable` to be used (if the ideal is homogeneous).

`Homogeneous`: Setting this to be `true` specifies that the origin (corresponding to the irrelevant ideal) is not a valid point.

`Replacement`: When intersecting with a random linear space, it is frequently much faster to use a linear space defined by relatively sparse equations (i.e., equations that do not involve all variables). Specifying this parameter to have the value `Monomial` will mean linear forms such as $ax+b$ are used (for constants $a$ and $b$), involving only one variable. `Binomial` means forms like $ax+by+c$, using two variables. `Trinomial` means forms like $ax+by+cz+d$. `Full` means all variables will have coefficients.

`DimensionFunction`: Our current implementation does not need to know the dimension of $V(I)$. However, there are places where we try to verify the dimension of an ideal before we decompose the ideal. You can pass the function `dim` (the default), or our probabilistic `dimViaBezout` or any other dimension function you might prefer.

`PointCheckAttempts`: When calling `randomPoints` with a `BruteForce` strategy, this denotes the number of trials for brute force point checking. It also controls how many linear spaces to simultaneously study in the `LinearIntersection` strategy.

**Example 2.2.** We re-compute Example 2.1 this time specifying more attempts.

```
i7 : time randomPoints(J, Strategy => BruteForce, PointCheckAttempts => 10000)
-- used 1.16294 seconds
o7 = {{-43, 25, 29}}
```

`NumThreadsToUse`: When calling `randomPoints` and functions that call it with a `BruteForce` strategy, this option allows the user to specify the number of threads to use in brute force point checking.

**2.2.** ***Comments on performance and implementation.*** When working over very small fields, especially with hypersurfaces, frequently `BruteForce` is most efficient. This is not surprising as there may not be

many points to check. However, if the field size is larger, `BruteForce` will perform poorly. Even for some simple examples, it could not provide any rational points if the number of trials is not large enough. Other strategies work differently on different examples, and the same strategy can sometimes work very quickly even if it typically works very slowly.

The current version of the `LinearIntersection` strategy no longer computes the dimension of the algebraic set. Instead, it first finds a point defined by linear equations. If the point is on the algebraic set, we are done. If not, we throw away one of the forms and so now have a line and we see if this line intersects our algebraic set. We continue in this way until we find a point. This appears to avoid a number of bottlenecks in our previous implementation since Macaulay2 is relatively fast at identifying when a linear space and a variety do *not* intersect.

**Example 2.3.** We begin with an example over a small field.

```
i2 : R = ZZ/7[x_1..x_10];
i3 : I = ideal(random(2, R), random(3, R));
o3 : Ideal of R
i4 : time randomPoints(I, Strategy => BruteForce, PointCheckAttempts => 20000)
        -- used 0.00311884 seconds
o4 = {{-1, -1, 0, 2, 2, -2, -2, -3, -3, -3}}
o4 : List
i5 : time randomPoints(I, Strategy => Default)
        -- used 0.081349 seconds
o5 = {{3, 0, 3, 3, 2, -2, 1, -1, 3, 1}}
```

**Example 2.4.** Now we work over a larger field.

```
i6 : S = ZZ/211[x_1..x_10];
i7 : J = ideal(random(2, S), random(3, S));
o7 : Ideal of S
i8 : time randomPoints(J, Strategy => BruteForce, PointCheckAttempts => 2000000)
        -- used 17.7988 seconds
o8 = {{15, 67, -27, -103, 56, 66, -23, 28, -50, 13}}
o8 : List
i9 : time randomPoints(J, Strategy => Default)
        -- used 0.0864013 seconds
o9 = {{0, 0, 0, 0, 34, 76, 51, 0, 1, 0}}
```

**Example 2.5.** Finally, we can allow our functions to extend our field.

```
i11 : time randomPoints(J, Strategy => Default, ExtendField => true)
           -- used 0.144332 seconds
                3      2                          3      2
o11 = {{0, - a  + 62a  - 47a - 76, 0, 0, 13a  - 18a  + 63a - 31, 0, 0,
             3      2                   3      2
       - 20a  - 82a  + 35a - 19,  55a  - 64a  - 8a - 50, 1}}
i12 : coefficientRing ring first first o11
o12 = GF 1982119441
i13 : log_211 1982119441
o13 = 4
```

In this case, we found a degree 4 point.

**Remark 2.2.1** (comments on the probability of finding a point). In the case of an absolutely irreducible hypersurface in $\mathbb{A}^n_{\mathbb{F}_q}$ (defined by $f$ say), there is significant discussion in the literature estimating lower bounds of number of rational points (see for instance, [Lang and Weil 1954; Ghorpade and Lachaud 2002; Cafure and Matera 2006]) all of which point to the fact that there is "good probability" of finding a rational point in this case when we intersect with a random line. Heuristically, we can make the following rough estimation. We expect that each equation $f = \lambda$ for $\lambda \in \mathbb{F}_q$ has approximately the same number of solutions. Since each point on $\mathbb{F}_q^n$ solves exactly one of these equations, we expect that $f = 0$ has approximately $q^{n-1}$ solutions, or in other words, our hypersurface has $q^{n-1}$ points. Now, a random line $L$ has $q$ points. We want to find the probability that one of these points is rational for $V(f)$. We would expect that if these points are randomly distributed, then the probability that our line contains one of those points $1 - \left(1 - \frac{1}{q}\right)^q$ which tends to $1 - e^{-1} \approx 0.63$ for $q$ large. Alternatively, one can use the proof of [v. Bothmer and Schreyer 2005, Proposition 2.12] for a more precise statement. For each point of $L$, we see that the probability that the chosen point does not lie in the intersection, $L \cap V(f)$, is $1 - \frac{1}{q}$. We then exhaust this search over all the points on $L$ to get the probability that there is indeed a successful intersection is $1 - \left(1 - \frac{1}{q}\right)^q$. As $q$ gets larger, this value tends to $1 - e^{-1} \approx 0.63$.

Of course, there are schemes over $\mathbb{F}_q$ with no rational points at all, even for plane curves.

**Remark 2.2.2** (projecting to a hypersurface first). Suppose $X \subseteq \mathbb{A}^n$ is an algebraic set. In a number of existing algorithms, one first does a generic (or even not very generic) projection $h : \mathbb{A}^n \to \mathbb{A}^m$ and so that $h(X)$ is a hypersurface (at least set theoretically). Then one finds a point $x \in h(X)$ (say as above), and computes $h^{-1}(\{x\})$, which is a linear space in $\mathbb{A}^n$ that typically intersects $X$ in a rational point. For example, this is done in `randomKRationalPoint` in core Macaulay2. Note that projecting to a hypersurface still is intersecting with a linear space, since $h^{-1}(\{x\})$ is linear, but it tries to choose the linear space intelligently.

However, in our experience, doing this generic projection first yields slower results. First, one has to compute the dimension. There are also numerous cases where computing this hypersurface $h(X)$ can be quite slow. This particularly appears in cases when one is computing successive minors to identify the locus where some variety is nonsingular.

On the other hand, instead of using a truly random linear space to intersect with, in the default strategy we initially try linear spaces whose defining equations have as few terms as possible. For example, in a ring with 10 variables, we first try binomial linear forms like

$$-27x_2 + 38x_7$$

instead of a random linear form like

$$-28x_1 - 27x_2 + 29x_3 + 27x_4 - 28x_5 + 27x_6 + 38x_7 - 13x_8 + 21x_9 - 3x_{10}.$$

Such simple linear spaces are the ones implicitly considered in `randomKRationalPoint`, for instance, since that generic projection is so simple. In practice, our approach seems to perform at least as well as projecting to a hypersurface, without the chance of the code hanging on the generic projection or dimension computations. We also do successive intersections in a way that avoids computing the dimension as described above in Section 2.2.

## 3. USEFUL FUNCTIONS: dimViaBezout AND randomCoordinateChange.

**3.1.** dimViaBezout: We thank Frank-Olaf Scheyer for pointing out that in most of the computations, computing the codimension of the given ideal is a significant bottleneck. While we have avoided most dimension computations in our current implementation, we have also implemented a probabilistic dimension computation of $V(I)$. This function takes as input an ideal $I$ in a polynomial ring over a field and intersects $V(I)$ with random linear spaces of increasing dimension until there is an intersection. For example, if the intersection of $V(I)$ with a random line has a point, then we expect that $V(I)$ contains a hypersurface. If there is no intersection, this function tries a 2-dimensional linear space, and so on. This can speed up a number of computations. The function also takes in optional inputs as described below:

- DimensionIntersectionAttempts: Our function actually estimates dimension several times and then averages the result (rounding down) since we tend to overestimate the dimension due to the nature of dimViaBezout as described above. By default it does this three times unless the Homogeneous flag is set, in which case it is done five times.
- MinimumFieldSize: If the ambient field is smaller than this integer value, it will automatically be replaced with an extension field. For instance, there are relatively few linear spaces over a field of characteristic 2, and this can cause incorrect results to be returned to the user. The user may set the MinimumFieldSize to ensure that the field being worked over is big enough. If this is not set, the program tries to choose a reasonable minimum field size based on the ambient ring.
- Homogeneous: If the ideal is homogeneous, we can use homogeneous linear spaces to compute dimension. Sometimes this is faster and other times slower.

**Example 3.1.** We illustrate the speed difference in this example.

```
i2 : S = ZZ/101[y_0..y_9];
i3 : I=ideal random(S^1,S^{-2,-2,-2,-3})+(ideal random(2,S))^2;
o3 : Ideal of S
i4 : time dimViaBezout I
     -- used 0.837359 seconds
o4 = 5
i5 : time dim I
     -- used 36.8496 seconds
o5 = 5
i6 : time dimViaBezout(I, DimensionIntersectionAttempts=>1)
     -- used 0.280803 seconds
o6 = 5
```

As you can see, doing a single intersection attempt is about three times faster, and it usually gives the right answer (far more than 99% of the time in this particular example, but in others doing the computation in triplicate avoids returning incorrect answers).

**3.2.** randomCoordinateChange: This function takes a polynomial ring as an input and outputs a coordinate change map, i.e., given a polynomial ring, this will produce a linear automorphism of the ring. This function checks whether the map is an isomorphism by computing the Jacobian.

In some applications, a full random change of coordinates is not desired, as it might cause code to run very slowly. A binomial change of coordinates might be preferred, or we might only replace some monomials by other monomials. This is controlled with the following options.

- `Replacement`: This works like the `Replacement` option for `RandomPoints`.
- `MaxCoordinatesToReplace`: The user can specify that only a specified number of coordinates should be nonmonomial (assuming `Homogeneous` is set to `true`).
- `Homogeneous`: Setting `Homogeneous` to `false` will cause degree zero terms to be added to modified coordinates (including monomial coordinates).

**Example 3.2.** We demonstrate some of these options.

```
i3 : R = ZZ/11[x, y, z];
i4 : randomCoordinateChange(R)
             ZZ
o4 = map(R,--[x, y, z],{4x + 5y - 5z, 3x - 4y - 3z, 4x})
             11

                      ZZ
o4 : RingMap R <--- --[x, y, z]
                      11
i5 : matrix randomCoordinateChange(R, MaxCoordinatesToReplace => 1)
o5 = | x -x-4y-5z y |
i6 : matrix randomCoordinateChange(R, MaxCoordinatesToReplace => 1,
        Homogeneous => false)
o6 = | x-3 z-5 -x+3y-4z+2 |
```

**4.** OTHER FUNCTIONS: `genericProjection` AND `projectionToHypersurface`. We include two functions providing customizable projections. We describe them here.

**4.1.** `genericProjection`. This function finds a random (somewhat, depending on options) generic projection of the ring or ideal. The typical usages are

- `genericProjection(n, I)`
- `genericProjection(n, R)`

where $I$ is an ideal in a polynomial ring, $R$ can denote a quotient of a polynomial ring and $n \in \mathbb{Z}$ is an integer specifying how many dimensions to drop. Note that this function makes no attempt to verify that the projection is actually generic with respect to the ideal.

This gives the projection map from $\mathbb{A}^N \mapsto \mathbb{A}^{N-n}$ and the defining ideal of the projection of $V(I)$. If no integer $n$ is provided then it acts as if $n = 1$.

**Example 4.1.** We project a curve in 4-space to one in 2-space.

```
i1 : R = ZZ/5[x, y, z, w];
i2 : I = ideal(x, y^2, w^3 + x^2);
i3 : genericProjection(2, I)
             ZZ                                          2          2
o3 = (map(R,--[z, w],{- x - 2y - z, - y - 2z}), ideal(z  - z*w - w ))
             5
```

Alternatively, instead of $I$, we may pass it a quotient ring. It will then return the inclusion of the generic projection ring into the given ring, followed by the source of that inclusion.

This method works by calling `randomCoordinateChange` (Section 3) before dropping variables. It passes the options `Replacement`, `MaxCoordinatesToReplace`, `Homogeneous` to that function.

**4.2.** `projectionToHypersurface`. This function creates a projection to a hypersurface. The typical usages are

- `projectionToHypersurface I`
- `projectionToHypersurface R`

where $I$ is an ideal in a polynomial ring and $R$ is a quotient of a polynomial ring. The output is a list with two entries: the generic projection map and the ideal (respectively the ring).

It differs from `genericProjection(codim I - 1, I)` as it only tries to find a hypersurface equation that vanishes along the projection, instead of finding one that vanishes exactly at the projection. This can be faster and can be useful for finding points. The same approach was used in the `point` command in the package [Cremona]. If we already know the codimension is `c`, we can set `Codimension` to be `c` so the function does not compute it.

**5.** AN APPLICATION: `findANonZeroMinor` AND `extendIdealByNonZeroMinor`. As mentioned in the introduction, the two functions in this section will provide further tools for computing singular locus, in addition to those available in the package `FastLinAlg`.

**5.1.** `findANonZeroMinor`: The typical usage of this function is

- `findANonZeroMinor(n, M, I)`

where $I$ is an ideal in a polynomial ring over `QQ` or `ZZ/p` for $p$ prime, $M$ is a matrix over the polynomial ring and $n \in \mathbb{Z}$ denotes the size of the minors of interest.

The function outputs the following:

- A randomly chosen point $P$ in $V(I)$ which it finds using `randomPoints`.
- The indexes of the columns of $M$ that stay linearly independent upon plugging $P$ into $M$.
- The indices of the linearly independent rows of the matrix extracted from $M$ in the above step.
- A random $n \times n$ submatrix of $M$ that has full rank at $P$.

Besides the options from `randomPoints` which are automatically passed to that function, the user may also provide the following additional information:

`MinorPointAttempts`: This controls how many points at which to check the rank.

**Example 5.1.** We demonstrate this `findANonZeroMinor` function.

```
i3 : R = ZZ/5[x, y, z];
i4 : I = ideal(random(3, R) - 2, random(2, R))
             3    2        2   3   2              2        2      2   3                    2
o4 = ideal(2x - 2x y + 2x*y + y + x z - 2x*y*z + y z - 2x*z + 2y*z - z - 2, - 2x*y - x*z - z )
o4 : Ideal of R
i5 :  M = jacobian(I)
o5 = {1} | x2+xy+2y2+2xz-2yz-2z2   -2y-z |
     {1} | -2x2-xy-2y2-2xz+2yz+2z2 -2x   |
     {1} | x2-2xy+y2+xz-yz+2z2     -x-2z |
              3       2
o5 : Matrix R  <--- R
i6 : findANonZeroMinor(2, M, I, Strategy => GenericProjection)
o6 = ({-2, 1, 1}, {0, 1}, {0, 1}, {1} | x2+xy+2y2+2xz-2yz-2z2   -2y-z |)
                                  {1} | -2x2-xy-2y2-2xz+2yz+2z2 -2x   |
```

**5.2.** `extendIdealByNonZeroMinor`: The typical usage is

- `extendIdealByNonZeroMinor(n, M, I)`

where $n$, $M$, $I$ are the same as before. This finds a submatrix of size $n \times n$ using `findANonZeroMinor`; it extracts the last entry of the output, finds its determinant and adds it to the ideal $I$, thus extending $I$. It has the same options as `findANonZeroMinor`.

One can use this function to show that rings are regular in codimension 1, that is, satisfy Serre's condition (R1).

**Example 5.2.** Consider the following 3-dimensional example which is regular in codimension 1. Note, in this example, computing the dimension of the singular locus takes around 30 seconds as there are 15500 minors of size $4 \times 4$ coming from the associated $7 \times 12$ Jacobian matrix. However, we can use our function to quickly find interesting minors.

```
i2 : T = ZZ/101[x1, x2, x3, x4, x5, x6, x7];
i3 : I =  ideal(x5*x6-x4*x7,x1*x6-x2*x7,x5^2-x1*x7,x4*x5-x2*x7,x4^2-x2*x6,x1*x4-x2*x5,
               x2*x3^3*x5+3*x2*x3^2*x7+8*x2^2*x5+3*x3*x4*x7-8*x4*x7+x6*x7,
               x1*x3^3*x5+3*x1*x3^2*x7+8*x1*x2*x5+3*x3*x5*x7-8*x5*x7+x7^2,
               x2*x3^3*x4+3*x2*x3^2*x6+8*x2^2*x4+3*x3*x4*x6-8*x4*x6+x6^2,
               x2^2*x3^3+3*x2*x3^2*x4+8*x2^3+3*x2*x3*x6-8*x2*x6+x4*x6,
               x1*x2*x3^3+3*x2*x3^2*x5+8*x1*x2^2+3*x2*x3*x7-8*x2*x7+x4*x7,
               x1^2*x3^3+3*x1*x3^2*x5+8*x1^2*x2+3*x1*x3*x7-8*x1*x7+x5*x7);
o3 : Ideal of T
i4 : M = jacobian I;
             7       12
o4 : Matrix T  <--- T
i5 : i = 0; J = I;
o6 : Ideal of T
i7 : elapsedTime(while (i < 10) and dim J > 1 do (
                                                  i = i + 1;
                                                  J = extendIdealByNonZeroMinor(4, M, J)));
          -- 0.640164 seconds elapsed
i8 : dim J
o8 = 1
i9 : i
o9 = 5
```

In this particular example, there tend to be about five associated primes when adding the first minor to $J$, and so one expects about five steps as each minor will typically eliminate one of those primes.

There is some similar functionality for computing partial Jacobian ideals obtained via heuristics (as opposed to actually finding rational or geometric points) in the package [FastLinAlg]. That package now uses the functionality contained here in `RandomPoints` in some of its functions.

**6.** THE LATEST VERSION. The latest version of the package is available here.

SUPPLEMENT. The online supplement contains version 1.5.3 of `RandomPoints`.

ACKNOWLEDGEMENTS. The authors would like to thank David Eisenbud and Mike Stillman for useful conversations and comments on the development of this package. The authors began work on this package at the virtual Cleveland 2020 Macaulay2 workshop. The authors are also grateful to the reviewers for suggesting and providing preliminary codes to speed up computations, thereby improving the efficacy of the package substantially.

Frank-Olaf Schreyer is also an author on this package, as he provided some code related to the `MultiplicationTable` decomposition strategy and suggested using a probabilistic approach to compute dimension.

REFERENCES.

[v. Bothmer and Schreyer 2005] H.-C. G. v. Bothmer and F.-O. Schreyer, "A quick and dirty irreducibility test for multivariate polynomials over $\mathbb{F}_q$", *Experiment. Math.* **14**:4 (2005), 415–422. MR Zbl

[Cafure and Matera 2006] A. Cafure and G. Matera, "Improved explicit estimates on the number of solutions of equations over a finite field", *Finite Fields Appl.* **12**:2 (2006), 155–185. MR Zbl

[Cremona] G. Staglianò, "Cremona: rational maps between projective varieties", Macaulay2 package, available at https://github.com/Macaulay2/M2/tree/master/M2/Macaulay2/packages.

[FastLinAlg] B. Martinova, M. Robinson, K. Schwede, and Y. W. Yao, "FastLinAlg: faster linear algebra operations", Macaulay2 package, available at https://github.com/Macaulay2/M2/tree/master/M2/Macaulay2/packages.

[Ghorpade and Lachaud 2002] S. R. Ghorpade and G. Lachaud, "Number of solutions of equations over finite fields and a conjecture of Lang and Weil", pp. 269–291 in *Number theory and discrete mathematics* (Chandigarh, 2000), Birkhäuser, Basel, 2002. MR Zbl

[Lang and Weil 1954] S. Lang and A. Weil, "Number of points of varieties in finite fields", *Amer. J. Math.* **76** (1954), 819–827. MR Zbl

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

[RationalPoints] N. Stapleton, "RationalPoints", Macaulay2 package, available at https://github.com/Macaulay2/M2/tree/master/M2/Macaulay2/packages.

RECEIVED: 8 Feb 2021 REVISED: 18 Aug 2021 ACCEPTED: 8 May 2023

SANKHANEEL BISUI:

sankhaneel.bisui@umanitoba.ca
Department of Mathematics, The University of Manitoba, Winnipeg MB, Canada

ZHAN JIANG:

zoeng@umich.edu
Department of Mathematics, The University of Michigan, Ann Arbor MI, United States

SARASIJ MAITRA:

maitra@math.utah.edu
Department of Mathematics, The University of Utah, Salt Lake City UT, United States

THÁI THÀNH NGUYỄN:

nguyt161@mcmaster.ca
Department of Mathematics, McMaster University, Hamilton ON, Canada

KARL SCHWEDE:

schwede@math.utah.edu
Department of Mathematics, The University of Utah, Salt Lake City UT, United States

*JSAG* **13** (2023), 45–51
https://doi.org/10.2140/jsag.2023.13.45

The Journal of Software for
Algebra and Geometry

# Setting the scene for Betti characters

FEDERICO GALETTO

ABSTRACT: Finite group actions on free resolutions and modules arise naturally in many interesting examples. Understanding these actions amounts to describing the terms of a free resolution or the graded components of a module as group representations which, in the nonmodular case, are completely determined by their characters. With this goal in mind, we introduce a Macaulay2 package for computing characters of finite groups on free resolutions and graded components of finitely generated graded modules over polynomial rings.

**1.** INTRODUCTION. Let $R$ be a polynomial ring over a field $\Bbbk$, and $M$ be a finitely generated graded $R$-module. Let $G$ be a linearly reductive group acting $\Bbbk$-linearly on $R$ and $M$, and assume these actions preserve degrees and distribute over $R$-multiplication. If $F_\bullet$ is a minimal graded free resolution of $M$, then the action of $G$ extends to $F_\bullet$. More precisely, if $\mathfrak{m}$ denotes the irrelevant maximal ideal of $R$, then the finite-dimensional vector spaces $F_i/\mathfrak{m}F_i$ carry a natural structure of graded $G$-representations (see [Galetto 2016, Proposition 2.4.9 and Remark 2.4.10] for details). This additional structure makes a resolution more rigid as the differentials must commute with the group action; in some cases, this makes it possible to construct the differentials explicitly using representation theory (see, for example, [Sam 2009; Sam and Weyman 2011]). Understanding how $G$ acts on the modules $F_i/\mathfrak{m}F_i$ may also lead to interesting combinatorial descriptions of the Betti numbers of $M$, such as in [Galetto 2020, Corollary 4.12]. Free resolutions equipped with group actions (also known as equivariant resolutions) have found many important applications, such as the computation of Betti numbers of determinantal varieties [Lascoux 1978], and a proof of the existence of pure free resolutions [Eisenbud et al. 2011] (a central aspect of Boij-Söderberg theory).

From a computational perspective, the [Macaulay2] package `HighestWeights` [Galetto 2015] allows users to determine the representation theoretic structure of an equivariant resolution with the action of a semisimple Lie group in characteristic zero. Recent publications [Zamaere et al. 2014; Efremenko et al. 2018; Galetto et al. 2018; Bauer et al. 2019; Galetto 2020; Biermann et al. 2020; Murai 2020; Shibata and Yanagawa 2023; Raicu 2021; Murai and Raicu 2022] point to an interest in equivariant resolutions with actions of finite groups, particularly symmetric groups. However, at the time of writing, no software solution is available to compute such actions. The present article introduces the Macaulay2 package `BettiCharacters` to fill this gap. In the nonmodular case (i.e., when the characteristic of the field does

*MSC2020:* primary 13-04; secondary 13A50, 13D02, 13P20, 20C15.
*Keywords:* Macaulay2, equivariant resolution, finite group, Betti character.
`BettiCharacters` version 2.1

© 2023 The Author, under license to MSP (Mathematical Sciences Publishers).

not divide the order of the group), finite-dimensional representations of finite groups are determined, up to isomorphism, by their characters (see [Serre 1977, Chapter 2] for an introduction to the subject). Thus understanding the $G$-action on a minimal free resolution $F_\bullet$ amounts to describing the graded characters of the representations $F_i/\mathfrak{m}F_i$ or, equivalently, the characters of the graded components $(F_i/\mathfrak{m}F_i)_j$. The uniqueness of minimal free resolutions implies $(F_i/\mathfrak{m}F_i)_j \cong \mathrm{Tor}_i^R(M,\Bbbk)_j$ as $G$-representations. Moreover, the character of $G$ on $\mathrm{Tor}_i^R(M,\Bbbk)_j$ evaluated at the identity of $G$ is the dimension of $\mathrm{Tor}_i^R(M,\Bbbk)_j$ as a $\Bbbk$-vector space, i.e., the $(i,j)$-th Betti number of $M$. Therefore we adopt the following definition, after which the package is named.

**Definition.** The *$(i,j)$-th Betti character of $G$ on $M$*, denoted $\beta_{i,j}^G(M)$, is the character of $G$ on $\mathrm{Tor}_i^R(M,\Bbbk)_j$.

The package `BettiCharacters` implements the algorithm described in [Galetto 2022, Algorithm 1], which in essence propagates the group action from the module $M$ through a (previously computed) minimal free resolution $F_\bullet$. In addition, `BettiCharacters` also allows users to compute the characters of $G$ on the graded components of $M$. The rest of this article illustrates the main functionalities of the package.[1] The author thanks the anonymous referee for carefully reviewing this work.

**2.** EXAMPLE: A SYMMETRIC SHIFTED IDEAL. Consider the ideal generated by all quadratic squarefree monomials in a ring with four variables.

```
i1 : R = QQ[x_1..x_4];

i2 : I = ideal apply(subsets(gens R,2),product)

o2 = ideal (x x , x x , x x , x x , x x , x x )
            1 2   1 3   2 3   1 4   2 4   3 4

o2 : Ideal of R

i3 : RI = res I

      1      6      8      3
o3 = R  <-- R  <-- R  <-- R  <-- 0
     0      1      2      3      4

o3 : ChainComplex
```

The symmetric group $\mathfrak{S}_4$ acts by permuting the ring variables and, in doing so, preserves the ideal. Thus the action passes to the quotient and its minimal free resolution. The equivariant structure of the resolution is described in [Galetto 2020, Theorem 4.11] and [Efremenko et al. 2018, Theorem 4.1]. The ideal also belongs to the larger class of symmetric shifted ideals, whose equivariant resolutions are described in [Biermann et al. 2020, Theorem 6.2]. To verify these results computationally, we first define the group action. Since characters are class functions (i.e., constant on conjugacy classes) it is enough to define a single group element per conjugacy class. In the case of a symmetric group, the method `symmetricGroupActors` returns the desired elements as one-row matrices of substitutions for the ring variables. Then we set up the action on the resolution as an object of type `ActionOnComplex` using the `action` method.

[1] All computations performed in Macaulay2 version 1.19.1 on Fedora 36, using `BettiCharacters` 2.0.

```
i4 : needsPackage "BettiCharacters";

i5 : S4 = symmetricGroupActors R

o5 = {| x_2 x_3 x_4 x_1 |, | x_2 x_3 x_1 x_4 |, | x_2 x_1 x_4 x_3 |,
     ------------------------------------------------------------
      | x_2 x_1 x_3 x_4 |, | x_1 x_2 x_3 x_4 |}

o5 : List

i6 : A = action(RI,S4)

o6 = ChainComplex with 5 actors

o6 : ActionOnComplex
```

Now we can use the `character` method to compute Betti characters.

```
i7 : a = character A

o7 = Character over R

     (0, {0}) => | 1  1  1  1 1 |
     (1, {2}) => | 0  0  2  2 6 |
     (2, {3}) => | 0 -1  0  0 8 |
     (3, {4}) => | 1  0 -1 -1 3 |

o7 : Character
```

The output is of type `Character`, a new kind of hash table introduced by the package. The keys are pairs containing the homological degree and internal (multi)degree of the nonzero components of a graded character. The values are one-row matrices whose entries are the traces of the previously defined group elements.

Finally, we decompose this character against the character table of the symmetric group, which can be obtained using the method `symmetricGroupTable`. The irreducible characters $\chi^\lambda$ of $\mathfrak{S}_4$ are in bijection with the partitions $\lambda$ of 4 (see [Fulton 1997, §7]), which appear as row labels in the character table. The column labels are the cardinalities of the conjugacy classes represented by the permutations in `o5`.

```
i8 : T = symmetricGroupTable R

o8 = Character table over R

             |  6   8   3   6  1
     --------+-------------------
        (4)  |  1   1   1   1  1
      (3,1)  | -1   0  -1   1  3
         2   |
       (2 )  |  0  -1   2   0  2
          2  |
     (2,1 )  |  1   0  -1  -1  3
        4    |
       (1 )  | -1   1   1  -1  1

o8 : CharacterTable
```

The decomposition is achieved with the method `decomposeCharacter`. The output is a table whose rows are labeled by pairs of homological degree and internal (multi)degree, and whose columns are labeled by irreducible characters of the group. The entry in a given cell is the coefficient of the irreducible character labeled by the column in the degree labeled by the row.

```
i9 : decomposeCharacter(a,T)

o9 = Decomposition table

                |                    2       2
                |  (4)  (3,1)  (2 )  (2,1 )
     -----------+---------------------------
     (0, {0})   |    1      0     0       0
     (1, {2})   |    1      1     1       0
     (2, {3})   |    0      1     1       1
     (3, {4})   |    0      0     0       1

o9 : CharacterDecomposition
```

The computation above shows that the Betti character $\beta_{1,2}^{\mathfrak{S}_4}$ of the quotient by the ideal in the example is $\chi^{(4)} + \chi^{(3,1)} + \chi^{(2,2)}$.

The methods in `BettiCharacters` are completely independent of the group. However, the package contains methods that simplify working with symmetric groups, as shown in the current example.

**3.** EXAMPLE: KLEIN POINT CONFIGURATION. We consider the Klein configuration of points in the projective plane. The defining ideal $I$ is explicitly constructed in [Bauer et al. 2019, Proposition 7.3]. Although $I$ is defined over the rationals, we work over the cyclotomic field obtained by adjoining a primitive seventh root of unity for the purpose of defining a group action.

```
i1 : kk=toField(QQ[a]/ideal(sum apply(7,i->a^i)));

i2 : R=kk[x,y,z];

i3 : f4=x^3*y+y^3*z+z^3*x

      3    3      3
o3 = x y + y z + x*z

o3 : R

i4 : f6=-1/54*det(jacobian transpose jacobian f4)

        5    5      2 2 2      5
o4 = x*y  + x z - 5x y z  + y*z

o4 : R

i5 : I=minors(2,jacobian matrix{{f4,f6}});

o5 : Ideal of R
```

The unique simple group $G$ of order 168 acts on the projective plane preserving the Klein configuration. This induces an action on our polynomial ring preserving the ideal $I$. The action (which is minimally defined over our cyclotomic field) is explicitly described in [Bauer et al. 2019, §2.2]. In particular, the group is generated by elements `g` of order 7, `h` of order 3, and `i` of order 2. Since we are interested in some characters of $G$, we need a representative for each conjugacy class; therefore, in addition to `g`, `h`, and `i`, we also consider the identity element, the inverse of `g`, and an element `j` of order 4. We define all these group elements as matrices.

```
i6 : g=matrix{{a^4,0,0},{0,a^2,0},{0,0,a}};

              3        3
o6 : Matrix kk  <--- kk
```

```
i7 : h=matrix{{0,1,0},{0,0,1},{1,0,0}};

              3        3
o7 : Matrix ZZ  <--- ZZ

i8 : i=(2*a^4+2*a^2+2*a+1)/7 * matrix{{a-a^6,a^2-a^5,a^4-a^3},
                                      {a^2-a^5,a^4-a^3,a-a^6},
                                      {a^4-a^3,a-a^6,a^2-a^5}};

              3        3
o8 : Matrix kk  <--- kk

i9 : j=-1/(2*a^4+2*a^2+2*a+1) * matrix{{a^5-a^4,1-a^5,1-a^3},
                                       {1-a^5,a^6-a^2,1-a^6},
                                        {1-a^3,1-a^6,a^3-a}};

              3        3
o9 : Matrix kk  <--- kk

i10 : G={id_(R^3),i,h,j,g,inverse g};
```

As proved in [Seceleanu 2015, Theorem 4.4] and [Bauer et al. 2019, Proposition 8.1], the symbolic cube $I^{(3)}$ is not contained in the square $I^2$. The second proof of [Bauer et al. 2019, Proposition 8.1] reduces the failure of containment to showing the graded component of degree 21 in the quotient $I^{(2)}/I^2$ is a trivial $G$-representation. By local duality, this is equivalent to showing that the last module in a minimal free resolution of $I^2$ is generated in degree 24 by a one-dimensional trivial $G$-module. We proceed to compute the character of $G$ on the last module of the resolution of $I^2$.

```
i11 : I2=I^2;

o11 : Ideal of R

i12 : RI2=res I2

       1      6      6      1
o12 = R  <-- R  <-- R  <-- R  <-- 0
      0      1      2      3      4

o12 : ChainComplex

i13 : needsPackage "BettiCharacters";

i14 : A=action(RI2,G,Sub=>false)

o14 = ChainComplex with 6 actors

o14 : ActionOnComplex
```

The action is defined with the option `Sub=>false`, which allows passing group elements as square matrices rather than one-row matrices of substitutions as in Section 2. Next we compute the character of the $G$-action on the resolution of $I^2$ in homological degree 3.

```
i15 : character(A,3)

o15 = Character over R

      (3, {24}) => | 1 1 1 1 1 1 |

o15 : Character
```

As expected, we obtain a trivial character concentrated in degree 24.

The BettiCharacters package can also compute the characters of a finite group on the graded components of a module. Using the package [SymbolicPowers], we can directly establish that the character of $G$ on the graded component of degree 21 in $I^{(2)}/I^2$ is trivial.

```
i16 : needsPackage "SymbolicPowers";

i17 : Is2 = symbolicPower(I,2);

o17 : Ideal of R

i18 : M = Is2 / I2;

i19 : B = action(M,G,Sub=>false)

o19 = Module with 6 actors

o19 : ActionOnGradedModule

i20 : character(B,21)

o20 = Character over R
      (0, {21}) => | 1 1 1 1 1 1 |

o20 : Character
```

SUPPLEMENT. The online supplement contains version 2.1 of BettiCharacters.

REFERENCES.

[Bauer et al. 2019] T. Bauer, S. Di Rocco, B. Harbourne, J. Huizenga, A. Seceleanu, and T. Szemberg, "Negative curves on symmetric blowups of the projective plane, resurgences, and Waldschmidt constants", *Int. Math. Res. Not.* **2019**:24 (2019), 7459–7514. MR

[Biermann et al. 2020] J. Biermann, H. de Alba, F. Galetto, S. Murai, U. Nagel, A. O'Keefe, T. Römer, and A. Seceleanu, "Betti numbers of symmetric shifted ideals", *J. Algebra* **560** (2020), 312–342. MR

[Efremenko et al. 2018] K. Efremenko, J. M. Landsberg, H. Schenck, and J. Weyman, "On minimal free resolutions of sub-permanents and other ideals arising in complexity theory", *J. Algebra* **503** (2018), 8–20. MR

[Eisenbud et al. 2011] D. Eisenbud, G. Fløystad, and J. Weyman, "The existence of equivariant pure free resolutions", *Ann. Inst. Fourier* (*Grenoble*) **61**:3 (2011), 905–926. MR Zbl

[Fulton 1997] W. Fulton, *Young tableaux*, London Mathematical Society Student Texts **35**, Cambridge University Press, 1997. MR Zbl

[Galetto 2015] F. Galetto, "Free resolutions and modules with a semisimple Lie group action", *J. Softw. Algebra Geom.* **7** (2015), 17–29. MR Zbl

[Galetto 2016] F. Galetto, "Propagating weights of tori along free resolutions", *J. Symbolic Comput.* **74** (2016), 1–45. MR Zbl

[Galetto 2020] F. Galetto, "On the ideal generated by all squarefree monomials of a given degree", *J. Commut. Algebra* **12**:2 (2020), 199–215. MR Zbl

[Galetto 2022] F. Galetto, "Finite group characters on free resolutions", *J. Symbolic Comput.* **113** (2022), 29–38. MR Zbl

[Galetto et al. 2018] F. Galetto, A. V. Geramita, and D. L. Wehlau, "Symmetric complete intersections", *Comm. Algebra* **46**:5 (2018), 2194–2204. MR Zbl

[Lascoux 1978] A. Lascoux, "Syzygies des variétés déterminantales", *Adv. in Math.* **30**:3 (1978), 202–237. MR Zbl

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

[Murai 2020] S. Murai, "Betti tables of monomial ideals fixed by permutations of the variables", *Trans. Amer. Math. Soc.* **373**:10 (2020), 7087–7107. MR Zbl

[Murai and Raicu 2022] S. Murai and C. Raicu, "An equivariant Hochster's formula for $\mathfrak{S}_n$-invariant monomial ideals", *J. Lond. Math. Soc.* (2) **105**:3 (2022), 1974–2010. MR

[Raicu 2021] C. Raicu, "Regularity of $\mathfrak{S}_n$-invariant monomial ideals", *J. Combin. Theory Ser. A* **177** (2021), art. id. 105307. MR Zbl

[Sam 2009] S. V. Sam, "Computing inclusions of Schur modules", *J. Softw. Algebra Geom.* **1** (2009), 5–10. MR Zbl

[Sam and Weyman 2011] S. V. Sam and J. Weyman, "Pieri resolutions for classical groups", *J. Algebra* **329** (2011), 222–259. MR Zbl

[Seceleanu 2015] A. Seceleanu, "A homological criterion for the containment between symbolic and ordinary powers of some ideals of points in $\mathbb{P}^2$", *J. Pure Appl. Algebra* **219**:11 (2015), 4857–4871. MR Zbl

[Serre 1977] J.-P. Serre, *Linear representations of finite groups*, Graduate Texts in Mathematics **42**, Springer, 1977. MR Zbl

[Shibata and Yanagawa 2023] K. Shibata and K. Yanagawa, "Elementary construction of minimal free resolutions of the Specht ideals of shapes $(n-2, 2)$ and $(d, d, 1)$", *J. Algebra Appl.* **22**:9 (2023), Paper No. 2350199, 26. MR

[SymbolicPowers] E. Grifo, "SymbolicPowers", Macaulay2 package, available at https://github.com/Macaulay2/M2/tree/master/M2/Macaulay2/packages.

[Zamaere et al. 2014] C. B. Zamaere, S. Griffeth, and S. V. Sam, "Jack polynomials as fractional quantum Hall states and the Betti numbers of the $(k+1)$-equals ideal", *Comm. Math. Phys.* **330**:1 (2014), 415–434. MR Zbl

RECEIVED: 30 Jun 2021 REVISED: 26 Feb 2023 ACCEPTED: 30 May 2023

FEDERICO GALETTO:

f.galetto@csuohio.edu

Department of Mathematics and Statistics, Cleveland State University, Cleveland, OH, United States

*JSAG* **13** (2023), 53–59
https://doi.org/10.2140/jsag.2023.13.53

The Journal of Software for
Algebra and Geometry

# Simplicial complexes in Macaulay2

BEN HERSEY, GREGORY G. SMITH AND ALEXANDRE ZOTINE

ABSTRACT: We highlight some features of the *SimplicialComplexes* package in *Macaulay*2.

This updated version of the *SimplicialComplexes* package in Macaulay2, originally developed by Sorin Popescu, Gregory G. Smith, and Mike Stillman, adds constructors for many classic examples, implements a new data type for simplicial maps, and incorporates many improvements to the methods and documentation. Emphasizing combinatorial and algebraic applications, the primary data type encodes an abstract simplicial complex—a family of subsets of a finite set that is closed under taking subsets. These simplicial complexes are the combinatorial counterpart to their geometric realizations formed from points, line segments, filled-in triangles, solid tetrahedra, and their higher-dimensional analogues in some Euclidean space. The subsets in a simplicial complex are called faces. The faces having cardinality 1 are its vertices and the maximal faces (ordered by inclusion) are its facets. The dimension of a simplicial complex is one less than the maximum cardinality of its faces. Following the combinatorial conventions, every nonempty simplicial complex has the empty set as a face.

In this package, a simplicial complex is represented by its Stanley–Reisner ideal. The vertices are identified with a subset of the variables in a polynomial ring and each face is identified with the product of the corresponding variables. A nonface is any subset of the vertices that does not belong to the simplicial complex and each nonface is again identified with a product of variables. The Stanley–Reisner ideal of a simplicial complex is generated by the monomials corresponding to its nonfaces; see Definition 5.1.2 in [Bruns and Herzog 1993], Definition 1.6 in [Miller and Sturmfels 2005], or Definition II.1.1 in [Stanley 1996]. Because computations in the associated polynomial ring are typically prohibitive, this package is not intended for simplicial complexes with a large number of vertices.

CONSTRUCTORS. The basic constructor for a simplicial complex accepts two different kinds of input. Given a list of monomials, it returns the smallest simplicial complex containing the corresponding faces. Given a radical monomial ideal $I$, it returns the simplicial complex whose Stanley–Reisner ideal is $I$. We illustrate both methods using the "bowtie" complex in Figure 1.

```
Macaulay2, version 1.20
with packages:  ConwayPolynomials, Elimination, IntegralClosure, InverseSystems,
Isomorphism, LLLBases, MinimalPrimes, OnlineLookup,
PrimaryDecomposition, ReesAlgebra, Saturation, TangentCone
```

*MSC2020:* 05E45, 13F55, 55U10.
*Keywords:* simplicial complexes, Stanley–Reisner ideals, monomial ideals, resolutions, Cohen–Macaulay complexes.
*SimplicialComplexes.m2* version 2.0

© 2023 The Authors, under license to MSP (Mathematical Sciences Publishers).

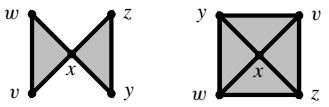


**Figure 1.** On the left is the bowtie complex ⋈ and on the right its Alexander dual ⋈*

```
i1 :  needsPackage "SimplicialComplexes"; S = QQ[v..z];
i3 :  ⋈ = simplicialComplex {v*w*x, x*y*z}
o3 =  simplicialComplex | xyz vwx |
o3 :  SimplicialComplex
i4 :  I = monomialIdeal ⋈
o4 =  monomialIdeal (v*y, w*y, v*z, w*z)
o4 :  MonomialIdeal of S
i5 :  ⋈◁ = simplicialComplex I
o5 =  simplicialComplex | xyz vwx |
o5 :  SimplicialComplex
i6 :  assert(⋈ === ⋈◁)
```

The package also has convenient constructors for some archetypal simplicial complexes. For example, we recognize the real projective plane and the Klein bottle from the reduced homology groups of some explicit triangulations; see Theorems 6.3–6.4 in [Munkres 1984].

```
i7 :  ℙ = realProjectiveSpaceComplex(2, R = ZZ[a..h])
o7 =  simplicialComplex | bef aef cdf adf bcf cde bde ace abd abc |
o7 :  SimplicialComplex
i8 :  for j from 0 to 2 list prune HH_j ℙ
o8 =  {0, cokernel | 2 |, 0}
o8 :  List
i9 :  for j from 0 to 2 list prune HH_j kleinBottleComplex R
o9 =  {0, cokernel | 2 |, 0}
                   | 0 |
o9 :  List
```

More comprehensively, Frank H. Lutz enumerates simplicial complexes having a small number of vertices; see [Lutz]. Using this list, the package creates a database of 43138 simplicial 2-manifolds having at most 10 vertices and 1343 simplicial 3-manifolds having at most 9 vertices. We demonstrate this feature by exhibiting the distribution of $f$-vectors among the 3-manifolds having 9 vertices. For all nonnegative integers $j$, the $j$-th entry in the $f$-vector is the number of faces having $j$ vertices.

```
i10 :  tally for j from 0 to 1296 list fVector smallManifold(3, 9, j, ZZ[vars(1..9)])
o10 =  Tally{{1, 9, 26, 34, 17} => 7  }
             {1, 9, 27, 36, 18} => 23
             {1, 9, 28, 38, 19} => 45
             {1, 9, 29, 40, 20} => 84
             {1, 9, 30, 42, 21} => 128
             {1, 9, 31, 44, 22} => 175
             {1, 9, 32, 46, 23} => 223
             {1, 9, 33, 48, 24} => 231
             {1, 9, 34, 50, 25} => 209
             {1, 9, 35, 52, 26} => 121
             {1, 9, 36, 54, 27} => 51
o10 :  Tally
```

Exploiting the same loop, we construct the simplicial maps from a minimal triangulation of a torus to the induced subcomplex on the first 7 vertices for each of these 3-manifolds.

```
i11 :  T = smallManifold(2, 7, 6, R = ZZ[a..i])
o11 =  simplicialComplex | cfg afg beg aeg cdg bdg def bef adf bcf cde ace abd abc |
o11 :  SimplicialComplex
i12 :  for j from 0 to 2 list prune HH_j T
              2   1
o12 =  {0, ZZ, ZZ }
o12 :  List
i13 :  for j from 0 to 1296 list (
           phi := map(smallManifold(3, 9, j, R), T, gens R);
           if not isWellDefined phi then continue else phi);
o13 :  {}
o13 :  List
```

COMBINATORIAL TOPOLOGY. We use the bowtie complex to showcase some of the key operations on simplicial complexes. Viewing a simplicial complex as a subcomplex of a simplex yields a duality theory. For any simplicial complex $\Delta$ whose vertices belong to a set $V$, the Alexander dual is the simplicial complex $\Delta^* := \{F \subseteq V \mid V \setminus F \notin \Delta\}$. Since each simplicial complex in this package has an underlying polynomial ring, the variables in this ring form a canonical superset of the vertices.

```
i14 :  dual ⧓
o14 =  simplicialComplex | wxz vxz wxy vxy |
o14 :  SimplicialComplex
i15 :  assert(dual dual ⧓ === ⧓ and dual monomialIdeal ⧓ === monomialIdeal dual ⧓)
```

Algebraically, Alexander duality switches the roles of the minimal generators and the irreducible components in the Stanley–Reisner ideal.

```
i16 :  monomialIdeal dual ⧓
o16 =  monomialIdeal (v*w, y*z)
o16 :  MonomialIdeal of S
i17 :  irreducibleDecomposition monomialIdeal ⧓
o17 =  {monomialIdeal (v, w), monomialIdeal (y, z) }
o17 :  List
```

The topological form of Alexander duality gives an isomorphism between the reduced homology of a simplicial complex and reduced cohomology of its dual; see Theorem 5.6 in [Miller and Sturmfels 2005]:

```
i18 :  n = numgens ring ⧓
o18 =  5
i19 :  assert all(-1..n-1, j -> prune HH^(n-j-3) dual ⧓ == prune HH_j ⧓)
```

A simplicial complex $\Delta$ is Cohen–Macaulay if the associated quotient ring $S/I$, where $I$ is the Stanley–Reisner ideal of $\Delta$ in the polynomial ring $S$, is Cohen–Macaulay. To characterize this attribute topologically, we introduce a family of subcomplexes. For any face $F$ in $\Delta$, the link is the subcomplex $\mathrm{link}_\Delta(F) := \{G \in \Delta \mid F \cup G \in \Delta \text{ and } F \cap G = \varnothing\}$. The link of the vertex $x$ in ⧓ has two disjoint facets.

```
i20 :  L = link(⧓, x)
o20 =  simplicialComplex | yz vw |
o20 :  SimplicialComplex
```

```
i21 :  prune HH_0 L
         1
o21 =  QQ
o21 :  QQ-module, free
```

As discovered by Gerald Reisner, the simplicial complex $\Delta$ is Cohen–Macaulay if and only if, for all faces $F$ in $\Delta$ and all integers $j$ less than the dimension of $\mathrm{link}_\Delta(F)$, the $j$-th reduced homology group of $\mathrm{link}_\Delta(F)$ vanishes; see Corollary 5.3.9 in [Bruns and Herzog 1993], Theorem 5.53 in [Miller and Sturmfels 2005], or Corollary II.4.2 in [Stanley 1996]. Using this criterion, the 0-th reduced homology certifies that ⋈ is not Cohen–Macaulay.

```
i22 :  assert(HH_0 L != 0)
i23 :  assert(dim(S^1/monomialIdeal ⋈) =!= n - pdim(^1/monomialIdeal ⋈))
```

However, the 1-skeleton of ⋈ is Cohen–Macaulay.

```
i24 :  ⋉ = skeleton(1, ⋈)
o24 =  simplicialComplex | yz xz xy wx vx vw |
o24 :  SimplicialComplex
i25 :  faceList = rsort flatten values faces ⋉
o25 =  {v*w, v*x, w*x, x*y, x*z, y*z, v, w, x, y, z, 1}
o25 :  List
i26 :  assert all(faceList, F -> (L := link(⋉, F); all(dim L, j -> HH_j L == 0)))
i27 :  assert(dim(S^1/monomialIdeal ⋉) === n - pdim(S^1/monomialIdeal ⋉))
```

Alternatively, we verify that ⋈ is not Cohen–Macaulay by showing that its $h$-vector has a negative entry; see Theorem 5.1.10 in [Bruns and Herzog 1993] or Corollary II.2.5 in [Stanley 1996]. By definition, the $h$-vector of a simplicial complex $\Delta$ is a binomial transform of its $f$-vector: for all $0 \leqslant j \leqslant d := \dim \Delta$, we have $h_j = \sum_{k=0}^{j} (-1)^{j-1} \binom{d+1-k}{j-k} f_{k-1}$. The $h$-vector encodes the numerator of the Hilbert series for $S/I$.

```
i28 :  d = dim ⋈
o28 =  2
i29 :  faces ⋈
o29 =  HashTable{-1 => {1}                              }
                 0 => {v, w, x, y, z}
                 1 => {v*w, v*x, w*x, x*y, x*z, y*z}
                 2 => {v*w*x, x*y*z}
o29 :  HashTable
i30 :  fVec = fVector ⋈
o30 =  {1, 5, 6, 2}
o30 :  List
i31 :  hVec = for j from 0 to d list
              sum(j+1, k -> (-1)^(j-k) * binomial(d+1-k, j-k) * fVec#k)
o31 =  {1, 2, -1}
o31 :  List
i32 :  hilbertSeries(S^1/monomialIdeal ⋈, Reduce => true)
                2
       1 + 2T - T
o32 =  ----------
                3
         (1 - T)
o32 :  Expression of class Divide
```

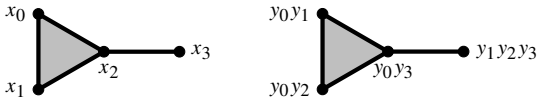


**Figure 2.** On the left is $\Gamma$ and on the right is the labeling of its vertices.

RESOLUTIONS OF MONOMIAL IDEALS. As David Bayer, Irena Peeva, and Bernd Sturmfels [Bayer et al. 1998] revealed, minimal free resolutions of monomial ideals are frequently encoded by a simplicial complex. Consider a monomial ideal $J$ in the polynomial ring $R := \mathbb{Q}[y_1, y_2, \ldots, y_m]$. Assume that $R$ is equipped with the $\mathbb{N}^m$-grading given by $\deg(y_i) = \boldsymbol{e}_i$, for each $1 \leqslant i \leqslant m$, where $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_m$ is the standard basis. Let $\Delta$ be a simplicial complex whose vertices are labeled by the generators of $J$. We label each face $F$ of $\Delta$ by the least common multiple $y^{\boldsymbol{a}_F} \in R$ of its vertices; the empty face is labeled by the monomial $1 = y^{\boldsymbol{a}_\varnothing}$. The chain complex $C(\Delta)$ supported on the labeled simplicial complex $\Delta$ is the chain complex of free $\mathbb{N}^m$-graded $R$-modules with basis corresponding to the faces of $\Delta$. More precisely, the chain complex $C(\Delta)$ is determined by the data

$$C_i(\Delta) := \bigoplus_{\dim(F)=i-1} R(-\boldsymbol{a}_F) \quad \text{and} \quad \partial(F) = \sum_{\dim(G)=\dim(F)-1} \operatorname{sign}(G, F)\, y^{\boldsymbol{a}_F - \boldsymbol{a}_G}\, G\,.$$

The symbols $F$ and $G$ represent both faces in $\Delta$ and basis vectors in the underlying free module of $C(\Delta)$. The sign of the pair $(G, F)$ belongs to $\{-1, 0, 1\}$ and is part of the data in the boundary map of the chain complex of $\Delta$. For more information, see Subsection 4.1 in [Miller and Sturmfels 2005] or Chapter 55 in [Peeva 2011].

We illustrate this construction with an explicit example. Consider the simplicial complex $\Gamma$ in Figure 2 and the monomial ideal $J = (y_0y_1, y_0y_2, y_0y_3, y_1y_2y_3)$ in $R = \mathbb{Q}[y_0, y_1, y_2, y_3]$. Label the vertices of $\Gamma$ by the generators of $J$:

$$x_0 \mapsto y_0y_1, \quad x_1 \mapsto y_0y_2, \quad x_2 \mapsto y_0y_3 \quad \text{and} \quad x_3 \mapsto y_1y_2y_3.$$

```
i33 :  x = getSymbol "x"; S = ZZ[x_0..x_3];
i35 :  Δ = simplicialComplex{x_0*x_1*x_2, x_2*x_3}
o35 =  simplicialComplex | x_2x_3 x_0x_1x_2 |
o35 :  SimplicialComplex
i36 :  chainComplex Δ
        1      4      4      1
o36 =  ZZ <- ZZ <- ZZ <- ZZ
       -1      0      1      2
o36 :  ChainComplex
i37 :  y = getSymbol "y"; R = QQ[y_0..y_3, DegreeRank => 4];
i39 :  J = ideal(y_0*y_1, y_0*y_2, y_0*y_3, y_1*y_2*y_3)
o39 =  ideal (y y , y y , y y , y y y )
              0 1   0 2   0 3   1 2 3
o39 :  Ideal of R
i40 :  C = chainComplex(Δ, Labels => J_*)
       1    4    4    1
o40 =  R <- R <- R <- R
       0    1    2    3
o40 :  ChainComplex
```

```
i41 :  C.dd
             1                                    4
o41 =  0 :  R <---------------------------------- R : 1
              | y_0y_1 y_0y_2 y_0y_3 y_1y_2y_3 |
             4                                     4
       1 :  R <-------------------------------------- R : 2
                  {1, 1, 0, 0} | -y_2 -y_3 0 0     |
                  {1, 0, 1, 0} | y_1 0 -y_3 0      |
                  {1, 0, 0, 1} | 0 y_1 y_2 -y_1y_2 |
                  {0, 1, 1, 1} | 0 0 0 y_0         |
             4                           1
       2 :  R <------------------------ R : 3
                  {1, 1, 1, 0} | y_3  |
                  {1, 1, 0, 1} | -y_2 |
                  {1, 0, 1, 1} | y_1  |
                  {1, 1, 1, 1} | 0    |
o41 :  ChainComplexMap
i42 :  assert(res(R^1/J) == C)
```

The chain complex $C(\Delta)$ depends on the labeling and is not always a resolution.

```
i43 :  C' = chainComplex(Δ, Labels => reverse J_*)
       1    4    4    1
o43 =  R <- R <- R <- R
       0    1    2    3
o43 :  ChainComplex
i44 :  prune homology C'
o44 =  0 :  cokernel | y_0y_3 y_0y_2 y_0y_1 y_1y_2y_3 |
       1 :  cokernel {1, 1, 0, 1} | y_2 |
       2 :  0
       3 :  0
o44 :  GradedModule
```

Given a monomial ideal $J$, there are several algorithms that return a labeled simplicial complex $\Delta$ such that chain complex $C(\Delta)$ is a free resolution of $R/J$. We exhibit a few.

```
i45 :  J' = monomialIdeal(y_1*y_3, y_2^2, y_0*y_2, y_1^2, y_0^2);
o45 :  MonomialIdeal of R
i46 :  T = taylorResolution J'
       1    5    10   10   5    1
o46 =  R <- R <- R <- R <- R <- R
       0    1    2    3    4    5
o46 :  ChainComplex
i47 :  gensJ' = first entries mingens J'
              2         2   2
o47 =  {y y , y , y y , y , y }
         1 3   2   0 2   1   0
o47 :  List
i48 :  S = ZZ[x_0..x_4];
i49 :  assert(T == chainComplex(simplexComplex(4, S), Labels => gensJ'))
i50 :  assert(lyubeznikSimplicialComplex(J', S) === simplexComplex(4, S))
i51 :  Γ = buchbergerSimplicialComplex(J',S)
o52 =  simplicialComplex | x_0x_2x_3x_4 x_0x_1x_2x_3 |
o52 :  SimplicialComplex
```

```
i53 :  B = buchbergerResolution J'
       1    5    9    7    2
o53 =  R <- R <- R <- R <- R
       0    1    2    3    4
o53 :  ChainComplex
i54 :  assert all(3, i -> HH_(i+1) B == 0)
i55 :  assert(betti B == betti res J')
i56 :  assert(B == chainComplex(Γ, Labels => first entries mingens J'))
i57 :  assert(Γ === lyubeznikSimplicialComplex(J', S, MonomialOrder => 2,1,0,3,4))
i59 :  assert(Γ === scarfSimplicialComplex(J', S))
```

For more information about the Taylor resolution, the Lyubeznik resolution, and the Scarf complex, see [Mermin 2012]. The Buchberger resolution is described in [Olteanu and Welker 2016].

ACKNOWLEDGEMENTS. All three authors were partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

SUPPLEMENT. The online supplement contains version 2.0 of `SimplicialComplexes.m2`.

REFERENCES.

[Bayer et al. 1998] D. Bayer, I. Peeva, and B. Sturmfels, "Monomial resolutions", *Math. Res. Lett.* **5**:1-2 (1998), 31–46. MR Zbl

[Bruns and Herzog 1993] W. Bruns and J. Herzog, *Cohen–Macaulay Rings*, Cambridge Studies in Advanced Mathematics **39**, Cambridge University Press, 1993. MR

[Lutz] F. H. Lutz, "The Manifold Page", website, available at http://page.math.tu-berlin.de/~lutz/stellar/.

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

[Mermin 2012] J. Mermin, "Three simplicial resolutions", pp. 127–141 in *Progress in commutative algebra* 1, de Gruyter, Berlin, 2012. MR Zbl

[Miller and Sturmfels 2005] E. Miller and B. Sturmfels, *Combinatorial commutative algebra*, Graduate Texts in Mathematics **227**, Springer, 2005. MR Zbl

[Munkres 1984] J. R. Munkres, *Elements of algebraic topology*, Addison-Wesley Publishing Company, Menlo Park, CA, 1984. MR Zbl

[Olteanu and Welker 2016] A. Olteanu and V. Welker, "The Buchberger resolution", *J. Commut. Algebra* **8**:4 (2016), 571–587. MR Zbl

[Peeva 2011] I. Peeva, *Graded syzygies*, Algebra and Applications **14**, Springer, 2011. MR Zbl

[Stanley 1996] R. P. Stanley, *Combinatorics and commutative algebra*, 2nd ed., Progress in Mathematics **41**, Birkhäuser, 1996. MR Zbl

RECEIVED: 24 May 2022 REVISED: 25 Oct 2022 ACCEPTED: 21 Mar 2023

BEN HERSEY:
benjamin.hersey@concordia.ca
Department of Mathematics and Statistics, Concordia University, Montréal QC, Canada

GREGORY G. SMITH:
ggsmith@mast.queensu.ca
Department of Mathematics and Statistics, Queen's University, Kingston ON, Canada

ALEXANDRE ZOTINE:
18az45@queensu.ca
Department of Mathematics and Statistics, Queen's University, Kingston ON, Canada

Journal of Software for Algebra and Geometry vol 13, no 1, 2023