Journal of Software for Algebra and Geometry

```
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
```

```
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 ) ]
```

```
  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
```

vol 11 2021

JSAG **11** (2021), 1–7
https://doi.org/10.2140/jsag.2021.11.1
The Journal of Software for
Algebra and Geometry

Phylogenetic trees

HECTOR BAÑOS, NATHANIEL BUSHEK, RUTH DAVIDSON,
ELIZABETH GROSS, PAMELA E. HARRIS, ROBERT KRONE,
COLBY LONG, ALLEN STEWART AND ROBERT WALKER

ABSTRACT: We introduce the package *PhylogeneticTrees* for *Macaulay*2, which allows users to compute phylogenetic invariants for group-based tree models. We provide some background information on phylogenetic algebraic geometry and show how the package *PhylogeneticTrees* can be used to calculate a generating set for a phylogenetic ideal as well as a lower bound for its dimension. Finally, we show how methods within the package can be used to compute a generating set for the join of any two ideals.

MOTIVATION. A central problem in phylogenetics is to describe the evolutionary history of n species from their aligned DNA sequences. One way to do this is through a model-based approach. A phylogenetic model is a statistical model, specified parametrically, of molecular evolution at a single DNA site. We can regard the aligned sequences as a collection of n-tuples of the four DNA bases, one from each site. Each choice of parameters results in a probability distribution on the 4^n possible n-tuples. The goal of model-based reconstruction is to find a choice of parameters that yields a distribution close to the empirical distribution. If we are able to do so then it is reasonable to assume that the model is an accurate reflection of the underlying evolutionary process. Most significantly, we can infer that the underlying tree parameter of the phylogenetic model is the evolutionary tree of the species under consideration.

MATHEMATICAL BACKGROUND. In phylogenetic algebraic geometry, the statistical models under consideration are tree-based Markov models. This means that we assume a Markov process proceeds along a tree with a transition matrix associated to each edge. A κ-state phylogenetic model on an n-leaf tree $\mathcal{T}$ induces a polynomial map from the parameter space $\Theta_{\mathcal{T}} \subseteq \mathbb{R}^m$ to the probability simplex Δ^{κ^n-1},

$$\psi_{\mathcal{T}} : \Theta_{\mathcal{T}} \to \Delta^{\kappa^n-1} \subset \mathbb{C}^{\kappa^n}.$$

The image of this map is the set of all probability distributions we obtain by varying the entries of the transition matrices; we refer to the image as the model $\mathcal{M}_{\mathcal{T}}$. For phylogenetic applications, usually $\kappa = 2$ or $\kappa = 4$.

The Zariski closure of the model $V_{\mathcal{T}} := \overline{\mathcal{M}_{\mathcal{T}}} \subseteq \mathbb{C}^{\kappa^n}$ is an affine algebraic variety. For the models we consider, the entries of $\psi_{\mathcal{T}}$ are homogeneous polynomials of uniform degree. Thus, $V_{\mathcal{T}}$ can be viewed

MSC2010: primary 13P25; secondary 05C05, 14M25, 92D15.
Keywords: phylogenetic trees, secant ideals, algebraic statistics, toric ideals.
`PhylogeneticTrees` version 2.0

as the affine cone of a projective variety in $\mathbb{P}^{\kappa^n-1}$. The ideal $\mathcal{I}_{\mathcal{T}} := I(V_{\mathcal{T}}) \subseteq \mathbb{C}[x_1, \ldots, x_{\kappa^n}]$ of all polynomials vanishing on $V_{\mathcal{T}}$ is a homogeneous ideal called the *ideal of phylogenetic invariants*; this ideal carries useful information about the model that can be used for determining model identifiability and performing model selection [E. S. Allman and Sullivant 2011; Allman et al. 2012; Cavender and Felsenstein 1987; Casanellas and Fernández-Sánchez 2006; A. 1987; Long and Sullivant 2015; Matsen et al. 2008; Matsen and Steel 2007; Rhodes and Sullivant 2012; Rusinko and Hipp 2012].

Given two models, $\mathcal{M}_{\mathcal{T}_1}$ and $\mathcal{M}_{\mathcal{T}_2}$, we define the *mixture model* $\mathcal{M}_{\mathcal{T}_1} * \mathcal{M}_{\mathcal{T}_2}$ to be the image of the map

$$\begin{aligned} \psi_{\mathcal{T}_1,\mathcal{T}_2} &: \Theta_{\mathcal{T}_1} \times \Theta_{\mathcal{T}_2} \times [0,1] \to \Delta^{\kappa^n-1} \subset \mathbb{C}^{\kappa^n}; \\ \psi_{\mathcal{T}_1,\mathcal{T}_2}(\theta_1,\theta_2,\pi) &= \pi\psi_{\mathcal{T}_1}(\theta_1) + (1-\pi)\psi_{\mathcal{T}_2}(\theta_2). \end{aligned} \tag{1}$$

As before, we take the Zariski closure $\overline{\mathcal{M}_{\mathcal{T}_1} * \mathcal{M}_{\mathcal{T}_2}} \subseteq \mathbb{C}^{\kappa^n}$, and now we obtain the algebraic variety $V_{\mathcal{T}_1 * \mathcal{T}_2} = V_{\mathcal{T}_1} * V_{\mathcal{T}_2}$, the join of $V_{\mathcal{T}_1}$ and $V_{\mathcal{T}_2}$. The *join* of two algebraic varieties V and W embedded in a common ambient space is the variety

$$V * W := \overline{\{\lambda v + (1-\lambda)w \mid \lambda \in \mathbb{C},\ v \in V,\ w \in W\}}.$$

In the special case when $V = W$, the join variety $V * V$ is called the *secant variety of* V. Similarly, given two ideals $I_1, I_2 \subset \mathbb{C}[x_1, \ldots, x_n]$, the ideal $I_1 * I_2 \subset \mathbb{C}[x_1, \ldots, x_n]$ is the *join ideal*. As for varieties, if $I_1 = I_2$, the ideal $I_1 * I_2$ is the *secant ideal*. We refer to [Sturmfels and Sullivant 2006, Section 2] for the definition of $I_1 * I_2$, but note the following important property:

$$\mathcal{I}_{\mathcal{T}_1} * \mathcal{I}_{\mathcal{T}_2} = I(V_{\mathcal{T}_1} * V_{\mathcal{T}_2}). \tag{2}$$

Our package provides a means of computing invariants for those working in phylogenetic algebraic geometry. We handle a class of commonly used models called *group-based models* that have special restrictions on the entries of the transition matrices. These entries are indexed by elements of a group and thus are subject to the Fourier–Hadamard coordinate transformation, which makes the parametrization monomial and the ideals toric [Evans and Speed 1993; Székely et al. 1993]. We will refer to the original coordinates, which represent leaf probabilities, as *probability coordinates* and the transformed coordinates as *Fourier coordinates*; furthermore, following the literature, we will use p for probability coordinates and q for Fourier coordinates. For these group-based models, we implement a theoretical construction for inductively determining the ideal of phylogenetic invariants for any tree from the invariants for claw trees [Sturmfels and Sullivant 2005]. We also handle the join and secant ideals formed from these ideals, which allows for computations involving mixture models.

FUNCTIONALITY FOR TORIC PHYLOGENETIC VARIETIES. As an example, let $\mathcal{T}$ be the four-leaf tree illustrated in Figure 1 and consider the Cavender–Farris–Neyman (CFN) model, a two-state group-based model, on $\mathcal{T}$. Then the toric ideal $\mathcal{I}_{\mathcal{T}}$ is generated in degree 2. Using our package, we can compute a generating set for $\mathcal{I}_{\mathcal{T}}$ using two different methods, `phyloToric42` and `phyloToricFP`.

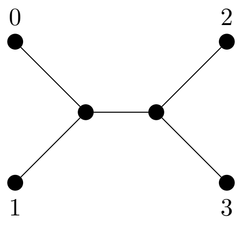

Figure 1. Four leaf tree

The first method phyloToric42 calls FourTiTwo.m2, the [Macaulay2] interface to [4ti2], a software package with functionality for computing generating sets of toric ideals. We input $\mathcal{T}$ by its set of nontrivial splits. In this instance, $01|23$ is the only nontrivial split of $\mathcal{T}$, which we can enter as either $\{0, 1\}$ or $\{2, 3\}$. The indices on the q's correspond to two-state labelings of the four leaves of $\mathcal{T}$:

```
Macaulay2, version 1.7

i1: load "PhylogeneticTrees.m2"

i2: n = 4; T = {{0,1}}; M = CFNmodel;

i5: toString phyloToric42(n,T,M)

o5: = ideal(-q_(0,1,1,0)*q_(1,0,0,1)+q_(0,1,0,1)*q_(1,0,1,0),
          -q_(0,0,1,1)*q_(1,1,0,0)+q_(0,0,0,0)*q_(1,1,1,1))
```

The second method phyloToricFP computes generators of $\mathcal{I}_{\mathcal{T}}$ using Theorem 24 in [Sturmfels and Sullivant 2005]; the "FP" in the method name stands for fiber product [Sullivant 2007]. For this example, this theorem allows us to explicitly construct a generating set of $\mathcal{I}_{\mathcal{T}}$ from generators of $\mathcal{I}_{K_{1,3}}$, the ideal associated to the CFN model on the claw tree $K_{1,3}$. While our example here is binary, we note that this method is implemented for all trees, binary or not.

```
i6: = toString phyloToricFP(n,T,M)

o6: = ideal(-q_(0,0,1,1)*q_(1,1,0,0)+q_(0,0,0,0)*q_(1,1,1,1),
             q_(0,0,1,1)*q_(1,1,0,0)-q_(0,0,0,0)*q_(1,1,1,1),
             q_(0,0,1,1)*q_(1,1,0,0)-q_(0,0,0,0)*q_(1,1,1,1),
            -q_(0,0,1,1)*q_(1,1,0,0)+q_(0,0,0,0)*q_(1,1,1,1),
            -q_(0,1,1,0)*q_(1,0,0,1)+q_(0,1,0,1)*q_(1,0,1,0),
             q_(0,1,1,0)*q_(1,0,0,1)-q_(0,1,0,1)*q_(1,0,1,0),
             q_(0,1,1,0)*q_(1,0,0,1)-q_(0,1,0,1)*q_(1,0,1,0),
            -q_(0,1,1,0)*q_(1,0,0,1)+q_(0,1,0,1)*q_(1,0,1,0))
```

The algorithm used by phyloToricFP returns more polynomials than are required to generate the ideal. If we wish to directly compare this ideal to that returned by phyloToric42 we must reconstruct both ideals in the same ring. Thus, we use the function qRing to define the ring of Fourier coordinates and use the option of specifying the ring for our ideals:

```
i7: R = qRing(n,M)

i8: = phyloToric42(n,T,M,QRing=>R) == phyloToricFP(n,T,M,QRing=>R)

o8: = true
```

In our experiments, for most cases, `phyloToric42` runs much faster than `phyloToricFP`. This is likely because we have implemented a naive version of the toric fiber product algorithm from [Sturmfels and Sullivant 2005] with no attempt to avoid producing redundant polynomials. It would be worth investigating if there is a faster implementation of this algorithm. Still, one advantage offered by the fiber product method is the ability to inductively construct a single invariant when computing the entire ideal is infeasible. The method `phyloToricRandom` returns such a randomly constructed invariant.

The polynomials that are returned by both methods are in Fourier coordinates, however, they can be converted to probability coordinates using the function `fourierToProbability`. To do so, we must first construct the ring of probability coordinates using `pRing`. Then the method `fourierToProbability` returns a ring map that converts polynomials in Fourier coordinates to probability coordinates:

```
i9:  = S = pRing(n,M);

i10: = phi = fourierToProbability(S,R,4,M);

i11: = f = (vars R)_(0,0)

o11: = q_(0,0,0,0)

i12: = phi(f)

o12: = (1/2)*p_(0,0,0,0)+(1/2)*p_(0,0,0,1)+(1/2)*p_(0,0,1,0)+(1/2)*p_(0,0,1,1)
       +(1/2)*p_(0,1,0,0)+(1/2)*p_(0,1,0,1)+(1/2)*p_(0,1,1,0)+(1/2)*p_(0,1,1,1)
       +(1/2)*p_(1,0,0,0)+(1/2)*p_(1,0,0,1)+(1/2)*p_(1,0,1,0)+(1/2)*p_(1,0,1,1)
       +(1/2)*p_(1,1,0,0)+(1/2)*p_(1,1,0,1)+(1/2)*p_(1,1,1,0)+(1/2)*p_(1,1,1,1)
```

FUNCTIONALITY FOR SECANT VARIETIES. Mixtures of group-based phylogenetic models correspond to secants and joins of toric ideals, objects that are of interest in combinatorial commutative algebra, but are notoriously hard to compute. In the methods `joinIdeal` and `secant`, we implement the elimination method described in [Sturmfels and Sullivant 2006, Section 2] for computing the join of two homogeneous ideals or the secant of one homogeneous ideal.

Consider now the Jukes–Cantor model on $\mathcal{T}$ from Figure 1. The phylogenetic ideal for the mixture of $\mathcal{M}_{\mathcal{T}}$ with itself is the second secant ideal of the homogeneous ideal $\mathcal{I}_{\mathcal{T}}$, denoted $\mathcal{I}_{\mathcal{T}} * \mathcal{I}_{\mathcal{T}}$. For secants, the method `secant` takes as input a homogenous ideal and an integer k and returns a generating set for the k-th secant ideal. The method also accepts the optional argument `DegreeLimit=>{l}`, which computes generators of the ideal only up to degree l. Thus, we can obtain generators of degree 3 or less of $\mathcal{I}_{\mathcal{T}} * \mathcal{I}_{\mathcal{T}}$ with the following commands. The minimal generating set of `SecI3` contains 49 linear invariants; we only print the generators with degree greater than one:

```
i13: = I = phyloToric42(n,T,JCmodel);

i14: SecI3 = secant(I,2,DegreeLimit={3});

i15: toString for i in flatten entries mingens SecI3
     list (if (degree i)#0 == 1 then continue; i)

o15 = {q_(0,3,3,0)*q_(3,0,2,1)*q_(3,2,0,1)-q_(0,3,2,1)*q_(3,0,3,0)*q_(3,2,0,1)
       +q_(0,3,2,1)*q_(3,0,0,3)*q_(3,2,1,0)-q_(0,3,0,3)*q_(3,0,2,1)*q_(3,2,1,0)
       -q_(0,3,3,0)*q_(3,0,0,3)*q_(3,2,3,2)+q_(0,3,0,3)*q_(3,0,3,0)*q_(3,2,3,2)}
```

The degree bound allows for the possibility of obtaining some invariants when computing a generating set for the secant ideal is infeasible. Let $(\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T})_l$ be the ideal generated by the elements of $\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T}$ of degree less than or equal to l. In some instances $(\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T})_l$ may be equal to $\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T}$. To prove this, we must verify that $\dim((\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T})_l) = \dim(\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T})$ and that $(\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T})_l$ is prime. Assuming we are able to compute $(\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T})_l$, we can compute its dimension and verify that it is prime. We then know that $\dim((\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T})_l) \geq \dim(\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T})$, leaving the inequality $\dim((\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T})_l) \leq \dim(\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T})$ to show. The method `toricSecantDim` enables us to do this using a probabilistic method based on Terracini's lemma [1911] to compute a lower bound on $\dim(\mathcal{I}_\mathcal{T} * \mathcal{I}_\mathcal{T})$.

Using this method, we can show that the secant of the ideal from the previous example is in fact generated in degree less than three:

```
i16: = dim(SecI3)

o16: = 12

i17: = isPrime(SecI3)

o17: = true

i18: = toricSecantDim(phyloToricAMatrix(4,{{0,1}},JCmodel),2))

o18: = 12
```

In the code above, we used `phyloToricAMatrix(n,T,JCmodel)` to construct the defining integral matrix of the toric ideal. For more details, see the documentation and [Sturmfels 1996]. In this instance, the method outlined is substantially faster than using the `secant` method without a degree bound.

ADDITIONAL FUNCTIONALITY. Although this package was developed with toric ideals from phylogenetics in mind, the methods `secant` and `joinIdeal` can be used for *any* homogeneous ideals. Thus, these can be employed for computations outside of phylogenetic algebraic geometry.

The following models are loaded with the package: the Cavender–Farris–Neyman model, the Jukes–Cantor model, the Kimura 2-parameter model, and the Kimura 3-parameter model. Additionally, some functionality for working with trees is available in this package, which includes the methods `edgeCut`, `vertexCut`, `edgeContract`, `internalEdges`, `internalVertices`.

ACKNOWLEDGEMENTS. This work began at the 2016 AMS Mathematics Research Community on "Algebraic Statistics," which was supported by the National Science Foundation under grant number DMS-1321794. Ruth Davidson was supported by NSF DMS-1401591. Elizabeth Gross was supported by NSF DMS-1620109. Robert Walker was primarily supported by an NSF GRF under grant number PGF-031543 and partially supported by the NSF RTG grant 0943832.

SUPPLEMENT. The online supplement contains version 2.0 of `PhylogeneticTrees`.

REFERENCES.

[4ti2] 4ti2 team, "4ti2: a software package for algebraic, geometric and combinatorial problems on linear spaces", available at www.4ti2.de.

[A. 1987] L. J. A., "A rate-independent technique for analysis of nucleic acid sequences: evolutionary parsimony", *Mol. Biol. Evol.* **4**:2 (1987), 167–191.

[Allman et al. 2012] E. S. Allman, J. A. Rhodes, and S. Sullivant, "When Do Phylogenetic Mixture Models Mimic Other Phylogenetic Models?", *Systematic Biology* **61**:6 (2012), 1049–1059.

[Casanellas and Fernández-Sánchez 2006] M. Casanellas and J. Fernández-Sánchez, "Performance of a New Invariants Method on Homogeneous and Nonhomogeneous Quartet Trees", *Molecular Biology and Evolution* **24**:1 (10 2006), 288–293.

[Cavender and Felsenstein 1987] J. A. Cavender and J. Felsenstein, "Invariants of phylogenies in a simple case with discrete states", *Journal of Classification* **4** (1987), 57–71.

[E. S. Allman and Sullivant 2011] J. A. R. E. S. Allman, S. Petrović and S. Sullivant, "Identifiability of Two-Tree Mixtures for Group-Based Models", *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **8**:3 (2011), 710–722.

[Evans and Speed 1993] S. N. Evans and T. P. Speed, "Invariants of some probability models used in phylogenetic inference", *Ann. Statist.* **21**:1 (1993), 355–377. MR

[Long and Sullivant 2015] C. Long and S. Sullivant, "Identifiability of 3-class Jukes–Cantor mixtures", *Adv. in Appl. Math.* **64** (2015), 89–110. MR

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

[Matsen and Steel 2007] F. A. Matsen and M. Steel, "Phylogenetic Mixtures on a Single Tree Can Mimic a Tree of Another Topology", *Systematic Biology* **56**:5 (10 2007), 767–775.

[Matsen et al. 2008] F. A. Matsen, E. Mossel, and M. Steel, "Mixed-up trees: the structure of phylogenetic mixtures", *Bull. Math. Biol.* **70**:4 (2008), 1115–1139. MR

[Rhodes and Sullivant 2012] J. A. Rhodes and S. Sullivant, "Identifiability of large phylogenetic mixture models", *Bull. Math. Biol.* **74**:1 (2012), 212–231. MR

[Rusinko and Hipp 2012] J. P. Rusinko and B. Hipp, "Invariant based quartet puzzling", *Algorithms for Molecular Biology* **7**:1 (2012).

[Sturmfels 1996] B. Sturmfels, *Gröbner bases and convex polytopes*, University Lecture Series **8**, American Mathematical Society, Providence, RI, 1996. MR

[Sturmfels and Sullivant 2005] B. Sturmfels and S. Sullivant, "Toric ideals of phylogenetic invariants", *Journal of Computational Biology* **12**:2 (2005), 204–228.

[Sturmfels and Sullivant 2006] B. Sturmfels and S. Sullivant, "Combinatorial secant varieties", *Pure Appl. Math. Q.* **2**:3, Special Issue: In honor of Robert D. MacPherson. Part 1 (2006), 867–891. MR

[Sullivant 2007] S. Sullivant, "Toric fiber products", *J. Algebra* **316**:2 (2007), 560–577. MR

[Székely et al. 1993] L. A. Székely, P. L. Erdős, M. A. Steel, and D. Penny, "A Fourier inversion formula for evolutionary trees", *Appl. Math. Lett.* **6**:2 (1993), 13–16. MR

[Terracini 1911] A. Terracini, "Sulle V_k per cui la varieta degli $S_h(h+1)$-seganti ha dimensione minore dell'ordinario", *Rend. Circ. Mat. Palermo* (1911).

RECEIVED: 16 Nov 2016 REVISED: 4 Jan 2020 ACCEPTED: 8 Aug 2020

HECTOR BAÑOS:
hbanos@gatech.edu
School of Mathematics, Georgia Institute of Technology, Atlanta, GA, United States

NATHANIEL BUSHEK:
nbushek@css.edu
Department of Mathematics and Physics, The College of St. Scholastica, Duluth, MN, United States

RUTH DAVIDSON:
redavid2@illinois.edu
Department of Mathematics, University of Illinois Urbana-Champaign, Urbana, IL, United States

ELIZABETH GROSS:

egross@hawaii.edu
Department of Mathematics, University of Hawaii‘i at Manoa, Honolulu, HI, United States

PAMELA E. HARRIS:

pamela.e.harris@williams.edu
Department of Mathematics and Statistics, Williams College, Williamstown, MA, United States

ROBERT KRONE:

rckrone@ucdavis.edu
Department of Mathematics, UC Davis, Davis, CA, United States

COLBY LONG:

clong@wooster.edu
Department of Mathematics and Computational Sciences, The College of Wooster, 308 E. University Street, Wooster, OH 44691, United States

ALLEN STEWART:

stewaral@seattleu.edu
Department of Mathematics, Seattle University, Seattle, WA, United States

ROBERT WALKER:

rwalker@math.wisc.edu
Department of Mathematics, University of Wisconsin–Madison, Madison, WI, United States

JSAG **11** (2021), 9–14
https://doi.org/10.2140/jsag.2021.11.9

The Journal of Software for
Algebra and Geometry

Software for doing computations in graded Lie algebras

CLAS LÖFWALL AND SAMUEL LUNDQVIST

ABSTRACT: We introduce the *Macaulay*2 package *GradedLieAlgebras* for doing computations in graded Lie algebras presented by generators and relations.

1. INTRODUCTION.

In order to support computer based research on graded Lie (super-)algebras, we have developed the package *GradedLieAlgebras* as part of [Macaulay2].

The package has basic routines for computing Hilbert series, for doing operations on ideals, subalgebras, derivations, and maps. It also has support for constructing holonomy Lie algebras of arrangements, computing homology and constructing minimal models. For a full list of features, we refer to the documentation of the package [GradedLieAlgebras].

The algorithmic idea used in the package goes back to [Löfwall and Roos 1997, Theorem 5.3], which was used to identify a periodic structure in a certain 1,2-presented Lie algebra. The first author then developed an algorithm and implemented that algorithm in *Mathematica*, under the name [Liedim]. That implementation has been cited or referred to in a number of papers; see for instance [Fröberg and Löfwall 2002; Löfwall et al. 2015; Peeva 2003; Roos 2008].

The aim of this paper is to describe the *Macaulay*2 implementation, which is a major extension of the implementation in *Mathematica*.

In the next two sections, we discuss implementation details and present the algorithmic theory used in the package. In the last section, we give a brief introduction to using the package.

2. REPRESENTING LIE ALGEBRAS IN *Macaulay2*.

In order to be able to use the built-in operations in *Macaulay*2, we decided to convert each computational step in the algorithm to a computation in a corresponding polynomial ring over the same field as the Lie algebra. That polynomial ring is referred to as `lieRing` in the code.

Let $\mathfrak{g}$ be a Lie algebra given by a finite set $\{x_i\}$ of generators and a set of relations. The generators have predefined degrees given by a function $\deg : \{x_i\} \to \mathbb{Z}_+$ and the relations are supposed to be homogeneous with respect to this degree function. This makes $\mathfrak{g}$ a *positively graded Lie algebra*. When $\mathfrak{g}$ is a Lie super-algebra, the generators have an additional $\mathbb{Z}/2\mathbb{Z}$-grading, and the relations are then supposed to be homogeneous also with respect to this grading.

MSC2010: primary 17-04, 17B70; secondary 13-04.
Keywords: graded Lie algebras, Hilbert series, holonomy Lie algebras, Koszul duality, differential graded Lie algebras.
GradedLieAlgebras version 3.0

An iterated Lie product in $\mathfrak{g}$ of the form $[x_{i_1}, [x_{i_2}, [x_{i_3}, \dots, [x_{i_{m-1}}, x_{i_m}], \dots]]]$ is called a *Lie monomial*. In the program, Lie monomials are identified with monomials in `lieRing`, and the Lie product of two elements is performed by a repeated series of normal form computations in `lieRing`, where the normal form computations are being performed with respect to a family of Gröbner bases in `lieRing`. It is important to understand that we use the Gröbner bases only as a way of doing Gaussian elimination, and that there is no connection to the Buchberger algorithm.

We now describe this correspondence in detail. To make the notation more easy to follow, we will use a slightly different way of naming the generators than in the program.

Each generator x_i in $\mathfrak{g}$ corresponds to n generators $x_{i1}, \dots, x_{in}$ in `lieRing`, where n is an upper bound on the degrees handled during our computations. A Lie monomial $[x_{i_1}, [x_{i_2}, [x_{i_3}, \dots, [x_{i_{m-1}}, x_{i_m}], \dots]]]$ in $\mathfrak{g}$ is represented as a monomial in `lieRing` in the following way. A generator x_i is represented by x_{i1}, and if e is a Lie monomial of degree $d-1$ represented by m, then $[x_i, e]$ is represented by $x_{id} \cdot m$, which is also denoted $x_i.m$.

Remark 1. We were informed by Jörgen Backelin that similar approaches to coding noncommutative monomials as commutative monomials have been considered independently in [Gerasimov 1976; La Scala and Levandovskyy 2009].

From the algorithm described in Section 3, it follows that the basis elements of degree d are of the form $[x_i, e_j]$, with e_j a basis element of degree $< d$.

If e_j is a basis element of degree $r-1$ represented by the monomial m in `lieRing`, and $[x_i, e_j]$ is of degree d but not a basis element, then $x_i.m = x_{ir} \cdot m$ will be the leading monomial of a polynomial in a reduced Gröbner basis associated to degree d. This polynomial then has the form $x_{ir} \cdot m - \sum c_i m_i$, where each m_i corresponds to a Lie monomial in $\mathfrak{g}$ that is a basis element in degree d, and where each c_i is an element in the underlying field.

The reduced Gröbner basis associated to degree d is the degree d part of the reduced Gröbner basis of the ideal generated by the set of polynomials of degree d corresponding to the elements that come from the expressions (2), (3), (5) and (6) in the next section.

3. Computing a vector space basis of a graded Lie algebra in a given degree.

A Lie (super-)algebra $\mathfrak{g}$ over a field k may be specified by giving a positively graded finite set X of generators and a finite set Y of homogenous relations,

$$y = \sum_{x \in X} \lambda_{x,y} x + \sum_i [g_{i,y}, h_{i,y}],$$

where $\lambda_{x,y} \in k$ and $g_{i,y}$ and $h_{i,y}$ are in $\mathcal{F}(X)$, the free Lie algebra on X over k. Throughout this section we will use the following example to illustrate the steps in the algorithm.

Let $k = \mathbb{Z}/3\mathbb{Z}$. The set of generators is $X = \{a, b, c\}$ where a is odd, and b, c are even, and a, b have degree 1, and c has degree 2. The set of relations is $Y = \{[a, a], [b, [b, a]] - [a, c]\}$. We want to compute a basis in degree 3, and we will use lieRing=$\mathbb{Z}/3\mathbb{Z}[a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3, c_3]$ with GRevLex, where a_i, b_i have degree 1, and c_i has degree 2 for $i = 1, 2, 3$.

The program constructs, degree by degree, starting with degree 1, a graded $\mathfrak{g}$-module M that is a subspace of lieRing. The $\mathfrak{g}$-module operation on M is denoted $g.m$, where $g \in \mathfrak{g}$ and $m \in M$. There is also a $\mathfrak{g}$-module map def : $M \to \mathfrak{g}$. Assume $n \geq 1$ and everything is done in degree $< n$. This means that def : $M_{<n} \to \mathfrak{g}/\mathfrak{g}_{\geq n}$ is an isomorphism of $\mathfrak{g}$-modules, with an inverse to be denoted by fed. If $\deg(g) + \deg(m) \geq n$, $g.m$ is defined to be zero.

In the example $\{a_1, b_1\}$ is a basis for M in degree 1 and $\{b_2a_1, c_1\}$ is a basis for M in degree 2. We also have the reduction rules $a_2a_1 \to 0$, $a_2b_1 \to -b_2a_1$, $b_2b_1 \to 0$. Moreover, $\mathrm{def}(a_1) = a$, $\mathrm{def}(b_1) = b$, $\mathrm{def}(b_2a_1) = [b, a]$, $\mathrm{def}(c_1) = c$.

To construct M_n, the first step is to construct a subspace $\hat{M}_n$ of lieRing, with basis

$$\{x.m \mid x \in X, m \in M, \deg(x) + \deg(m) = n\} \cup \{x_{i1} \mid x_i \in X, \deg(x_i) = n\}.$$

In a natural way we get an $\mathcal{F}(X)$-module $\hat{M} = M_{<n} \oplus \hat{M}_n$ by first defining the action of X, and then extending this action by the derivation rule. Also def is defined on $\hat{M}_n$ by $\mathrm{def}(x.m) = [x, \mathrm{def}(m)]$ and $\mathrm{def}(x_{i1}) = x_i$. It follows that def is surjective in degree n.

In the example, a basis for $\hat{M}_3$ is

$$\{a_3b_2a_1, a_3c_1, b_3b_2a_1, b_3c_1, c_2a_1, c_2b_1\}. \tag{1}$$

The next step is to divide out by a subspace R_n of $\hat{M}_n$ to obtain a $\mathfrak{g}$-module. For this reason we compute

$$R = Y.\hat{M}. \tag{2}$$

It is a subspace of $\hat{M}_n$, since $M_{<n}$ is a $\mathfrak{g}$-module. A generating set for R_n is obtained by computing $y.m$ for each relation y and basis element $m \in M$ such that $\deg(y) + \deg(m) = n$.

In the example, the space R_3 is spanned by $[a, a].a_1 = 2a.(a_2a_1) = 0$ and $[a, a].b_1 = 2a.(a_2b_1) = -2a_3b_2a_1$, yielding the reduction rule $a_3b_2a_1 \to 0$.

We apply [Löfwall and Roos 1997, Theorem 5.3] to obtain M_n as $\hat{M}_n$ modulo R_n and the expressions

$$\sum_{x \in X} \lambda_{x,y} m_x + \sum_i g_{i,y}.\mathrm{fed}(h_{i,y}) \quad \text{for all } y \in Y_n, \tag{3}$$

$$x.m + \epsilon(m, x)\mathrm{def}(m).m_x, \tag{4}$$

where m_x is the element in lieRing corresponding to $x \in X$, and where the last expression is computed for all basis elements $m \in M$ and all $x \in X$ such that $\deg(x) + \deg(m) = n$. Here $\epsilon(m, x)$ is the sign of interchanging the super-elements m and x.

In fact, using a linearization idea described in [Löfwall and Roos 1997], the "commutative" law (4) — the computationally most heavy part — does not need to be checked for all elements. Indeed, in characteristic zero, a basis is computed for the quotient space $\tilde{M}_n$ of $\hat{M}_n$ with respect to (2), (3), and the extra expressions

$$g_{i,y}.\mathrm{fed}(h_{i,y}) + \epsilon(g_{i,y}, h_{i,y})h_{i,y}.\mathrm{fed}(g_{i,y}) \quad \text{for all } y \in Y_n \text{ and for all } i. \tag{5}$$

If $\mathrm{char}(k) > 0$ then a basis for the quotient space $\tilde{M}_n$ of $\hat{M}_n$ is computed with respect to (2), (3), (5), and also the expressions

$$x.m + \epsilon(m, x)\mathrm{def}(m).m_x \tag{6}$$

coming from (4), for which $\deg x$ is a multiple of the characteristic.

Finally, M_n is obtained from $\tilde{M}_n$ by factoring out (4) applied to the basis elements $x.m$ of $\tilde{M}_n$.

In the example, (6) gives nothing, while (3) and (5) give

$$b.b_2a_1 - a.c_1 = b_3b_2a_1 - a_3c_1 \implies a_3c_1 \to b_3b_2a_1,$$
$$b.b_2a_1 + [b, a].b_1 = b_3b_2a_1 - b.(b_2a_1) - a.(b_2b_1) = b_3b_2a_1 - b_3b_2a_1 - 0 = 0,$$
$$a.c_1 + c.a_1 = a_3c_1 + c_2a_1 \implies c_2a_1 \to -a_3c_1 \to -b_3b_2a_1.$$

Hence, we have the reduction rules $a_3b_2a_1 \to 0$ (from (2)), $a_3c_1 \to -b_3b_2a_1$, $c_2a_1 \to b_3b_2a_1$ and hence $\tilde{M}_3$ has the basis $\{b_3b_2a_1, b_3c_1, c_2b_1\}$. Finally, (4) gives the reduction rule $c_2b_1 \to -b_3c_1$ yielding the basis $\{b_3b_2a_1, b_3c_1\}$ for M_3 and $\mathrm{def}(b_3b_2a_1) = [b, [b, a]]$, $\mathrm{def}(b_3c_1) = [b, c]$.

4. Using the package. The main introduction to using the package is by means of the tutorials that are part of the documentation [GradedLieAlgebras]. Here we give three small examples of possible computations.

The most common way to construct a Lie algebra is by means of the constructor `lieAlgebra`. In our first example, we construct the free Lie algebra on three even generators, all of degree 1:

```
i2 : L1=lieAlgebra({a,b,c})
o2 : LieAlgebra
i3 : dims(1,6,L1)
o3 = {3, 3, 8, 18, 48, 116}
i4 : basis(2,L1)
o4 =  {(b a), (c a), (c b)}
```

Here is the example from Section 3:

```
i5 : L2=lieAlgebra({a,b,c}, Field=>ZZ/3, Signs=>{1,0,0}, Weights=>{1,1,2})/{a a, b b a - a c}
o5 : LieAlgebra
i6 : dims(1,5,L2)
o6 = {2, 2, 2, 3, 5}
i7 :  b c c a
o7 = (b a b b c) + (b b a b c) + (b b b b b a)
o7 : L2
i8 :  basis(3,L2)
o8 = {(b b a), (b c)}
```

Let us now give a short example of computing the homology of a Lie algebra. The generators are odd, a and b have degree 1, and homological degree 0, c has degree 2, and homological degree 1. The differential is defined by $a, b \mapsto 0$, $c \mapsto [a, b]$. The homology can now be obtained using `lieHomology`, basis and dims (the columns refer to the first degree, and the rows refer to the homological degree). The Lie subalgebras consisting of the cycles and boundaries of the Lie algebra are obtained using cycles and boundaries. The underlying field is $\mathbb{Q}$ by default.

```
i9  : F3=lieAlgebra({a,b,c},Signs=>1,
         Weights=>{{1,0},{1,0},{2,1}},LastWeightHomological=>true)

o9  : LieAlgebra

i10 : L3 = differentialLieAlgebra{0_F3,0_F3,a b}/{a a, b b}

o10 : L3

o10 : LieAlgebra

i11 : H = lieHomology L3

o11 : H

o11 : VectorSpace

i12 : dims(4,H)

o12 = | 2 0 0 0 |
      | 0 0 2 1 |
      | 0 0 0 0 |
      | 0 0 0 0 |

i13 : basis(4,1,H)

o13 : {(b a c)}

i14 : B = boundaries L3

o14 : B

o14 : LieSubAlgebra

i15 : basis(4,1,B)

o15 : {(a b c) + b a c)}
```

ACKNOWLEDGEMENT. The authors want to thank Dan Grayson for his help with many of the implementation issues we faced during the development of the package.

SUPPLEMENT. The online supplement contains version 3.0 of *GradedLieAlgebras*.

REFERENCES.

[Fröberg and Löfwall 2002] R. Fröberg and C. Löfwall, "Koszul homology and Lie algebras with application to generic forms and points", *Homology Homotopy Appl.* **4**:2, part 2 (2002), 227–258. MR

[Gerasimov 1976] V. N. Gerasimov, "Distributive lattices of subspaces and the word problem for one-relator algebras", *Algebra i Logika* **15**:4 (1976), 384–435, 487. In Russian. MR

[GradedLieAlgebras] C. Löfwall and S. Lundqvist, "GradedLieAlgebras", Macaulay2 package, available at https://github.com/Macaulay2/M2/tree/master/M2/Macaulay2/packages.

[La Scala and Levandovskyy 2009] R. La Scala and V. Levandovskyy, "Letterplace ideals and non-commutative Gröbner bases", *J. Symbolic Comput.* **44**:10 (2009), 1374–1393. MR

[Liedim] C. Löfwall, "Liedim: a Mathematica program for Lie-calculations", available at http://www2.math.su.se/liedim/.

[Löfwall and Roos 1997] C. Löfwall and J.-E. Roos, "A nonnilpotent 1-2-presented graded Hopf algebra whose Hilbert series converges in the unit circle", *Adv. Math.* **130**:2 (1997), 161–200. MR

[Löfwall et al. 2015] C. Löfwall, S. Lundqvist, and J.-E. Roos, "A Gorenstein numerical semi-group ring having a transcendental series of Betti numbers", *J. Pure Appl. Algebra* **219**:3 (2015), 591–621. MR

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

[Peeva 2003] I. Peeva, "Hyperplane arrangements and linear strands in resolutions", *Trans. Amer. Math. Soc.* **355**:2 (2003), 609–618. MR

[Roos 2008] J.-E. Roos, "The homotopy Lie algebra of a complex hyperplane arrangement is not necessarily finitely presented", *Experiment. Math.* **17**:2 (2008), 129–143. MR

RECEIVED: 21 Aug 2017 REVISED: 22 Jun 2020 ACCEPTED: 28 Aug 2020

CLAS LÖFWALL:

clas.lofwall@gmail.com
Department of Mathematics, Stockholm University, Stockholm, Sweden

SAMUEL LUNDQVIST:

samuel@math.su.se
Department of Mathematics, Stockholm University, Stockholm, Sweden

JSAG **11** (2021), 15–24
https://doi.org/10.2140/jsag.2021.11.15

The Journal of Software for
Algebra and Geometry

The relative canonical resolution: Macaulay2-package, experiments and conjectures

CHRISTIAN BOPP AND MICHAEL HOFF

ABSTRACT: This short note provides a quick introduction to relative canonical resolutions of curves on rational normal scrolls. We present our *Macaulay*2 package that computes the relative canonical resolution associated to a curve and a pencil of divisors. We end with a list of conjectural shapes of relative canonical resolutions. In particular, for curves of genus $g = n \cdot k + 1$ and pencils of degree k for $n \geq 1$, we conjecture that the syzygy divisors on the Hurwitz scheme $\mathscr{H}_{g,k}$ constructed by Deopurkar and Patel (*Contemp. Math.* **703** *(2018) 209–222*) all have the same support.

1. RELATIVE CANONICAL RESOLUTIONS.

The *relative canonical resolution* is the minimal free resolution of a canonically embedded curve C inside a rational normal scroll. Every such scroll is swept out by linear spaces parametrized by pencils of divisors on C.

Studying divisors on moduli spaces reveals certain aspects of the global geometry of these spaces. A famous example for odd genus g is the *Koszul divisor* on the moduli space of curves $\mathscr{M}_g$ (see [Hirschowitz and Ramanan 1998; Voisin 2005; Farkas 2009]). It can be derived from the minimal free resolution of $C \subset \mathbb{P}^{g-1}$. Set-theoretically the Koszul divisor consists of curves such that the minimal free resolution of the canonical model has extra syzygies at a certain step. In [Bujokas and Patel 2015; Deopurkar and Patel 2015; 2018], the relative canonical resolution was used to define similar *syzygy divisors* on Hurwitz spaces $\mathscr{H}_{g,k}$, parametrizing pairs of curves of genus g and pencils of divisors of degree k (equivalently, covers of $\mathbb{P}^1$ of degree k by curves of genus g). We also refer to [Farkas 2018] for divisors on Hurwitz spaces.

We will briefly summarize the connection between pencils of divisors on canonical curves and rational normal scrolls in order to define the relative canonical resolution. The following definition and statements can be found in [Harris 1981, §3] and [Schreyer 1986, §1]. Let $C \subset \mathbb{P}^{g-1}$ be a canonically embedded curve of genus g, and let

$$g^1_k = \{D_\lambda\}_{\lambda \in \mathbb{P}^1} \subset |D|$$

MSC2010: 13D02, 14H51, 14Q05.
Keywords: Hurwitz space, syzygy modules, relative canonical resolution.
RelativeCanonicalResolution version 1.0

be a pencil of divisors of degree k. If we denote by $\overline{D_\lambda} \subset \mathbb{P}^{g-1}$ the linear span of the divisor, then

$$X := \bigcup_{\lambda \in \mathbb{P}^1} \overline{D_\lambda} \subset \mathbb{P}^{g-1}$$

is a $(k-1)$-dimensional rational normal scroll of degree $\deg(X) := f = g - k + 1$.

Definition 1.1. Let $e_1 \geq e_2 \geq \cdots \geq e_d \geq 0$ be integers, $\mathscr{E} = \mathscr{O}_{\mathbb{P}^1}(e_1) \oplus \cdots \oplus \mathscr{O}_{\mathbb{P}^1}(e_d)$, and let $\pi : \mathbb{P}(\mathscr{E}) \to \mathbb{P}^1$ be the corresponding $\mathbb{P}^{d-1}$-bundle. A *rational normal scroll* of type $(e_1, \ldots, e_d)$ is the image of

$$j : \mathbb{P}(\mathscr{E}) \to \mathbb{P}H^0(\mathbb{P}(\mathscr{E}), \mathscr{O}_{\mathbb{P}(\mathscr{E})}(1)) = \mathbb{P}^r.$$

Note that $r = f + d - 1$ with $f = e_1 + \cdots + e_d \geq 2$. Conversely if X is a rational normal scroll of degree f containing a canonical curve, then the ruling on X cuts out a pencil of divisors $\{D_\lambda\} \subset |D|$ such that $h^0(C, \omega_C \otimes \mathscr{O}_C(D)^{-1}) = f$.

Example 1.2. We consider a nonhyperelliptic canonically embedded curve $C \subset \mathbb{P}^3$ of genus 4. The curve C is a complete intersection of a quadric surface Q and a cubic surface S. If C admits exactly two pencils of degree 3 (which is also the maximal number), then the quadric Q is isomorphic to $\mathbb{P}^1 \times \mathbb{P}^1$. By Bézout's theorem, the two rulings of lines on Q cut out the two pencils of degree 3 on C, and conversely, the quadric is the scroll of type $(1, 1)$ swept out by either of these pencils. If C admits only one pencil of degree 3, then the quadric Q is isomorphic to a cone (i.e., a quadric of rank 3) and coincides with the scroll of type $(0, 2)$ swept out by the unique pencil.

In [Harris 1981] it is shown that the variety X defined above is a nondegenerate d-dimensional variety of minimal degree $\deg X = f = r - d + 1 = \operatorname{codim} X + 1$. If $e_1, \ldots, e_d > 0$, then

$$j : \mathbb{P}(\mathscr{E}) \to X \subset \mathbb{P}H^0(\mathbb{P}(\mathscr{E}), \mathscr{O}_{\mathbb{P}(\mathscr{E})}(1)) = \mathbb{P}^r$$

is an isomorphism. Otherwise, it is a resolution of singularities. Since $R^i j_* \mathscr{O}_{\mathbb{P}(\mathscr{E})} = 0$ for $i > 0$, it is convenient to consider $\mathbb{P}(\mathscr{E})$ instead of X for cohomological considerations.

It is known (see, e.g., [Eisenbud and Harris 1987b]) that the Picard group $\operatorname{Pic}(\mathbb{P}(\mathscr{E}))$ is generated by the class of the ruling $R = [\pi^* \mathscr{O}_{\mathbb{P}^1}(1)]$ and the hyperplane class $H = [j^* \mathscr{O}_{\mathbb{P}^r}(1)]$ with intersection products

$$H^d = f, \quad H^{d-1} \cdot R = 1, \quad R^2 = 0.$$

Hence, we will write a line bundle $\mathscr{O}_{\mathbb{P}(\mathscr{E})}(aH + bR)$ in the form

$$\mathscr{O}_{\mathbb{P}(\mathscr{E})}(aH + bR) = \pi^*(\mathscr{O}_{\mathbb{P}^1}(b))(aH).$$

Theorem 1.3 [Schreyer 1986, Corollary 4.4]. *Let C be a curve with a complete base point free g^1_k, and let $\mathbb{P}(\mathscr{E})$ be the projective bundle associated to the scroll X swept out by the g^1_k.*

(a) *$C \subset \mathbb{P}(\mathscr{E})$ has a resolution $F_\bullet$ of type*

$$0 \to \pi^* N_{k-2}(-kH) \to \pi^* N_{k-3}((-k+2)H) \to \cdots \to \pi^* N_1(-2H) \to \mathscr{O}_{\mathbb{P}(\mathscr{E})} \to \mathscr{O}_C \to 0$$

with

$$N_i = \bigoplus_{j=1}^{\beta_i} \mathscr{O}_{\mathbb{P}^1}(a_j^{(i)}) \quad \text{and} \quad \beta_i = \frac{i(k-2-i)}{k-1}\binom{k}{i+1}.$$

(b) *The complex $F_\bullet$ is self-dual, i.e., $\mathscr{H}om(F_\bullet, \mathscr{O}_{\mathbb{P}(\mathscr{E})}(-kH + (f-2)R)) \cong F_\bullet$.*

The resolution $F_\bullet$ above is called the *relative canonical resolution*. The degree of the bundles N_i is known.

Proposition 1.4 [Bopp and Hoff 2015, Proposition 2.9]. *The degree of the bundle N_i of rank $\beta_i = \frac{k}{i+1}(k-2-i)\binom{k-2}{i-1}$ in the relative canonical resolution $F_\bullet$ is*

$$\deg(N_i) = \sum_{j=1}^{\beta_i} a_j^{(i)} = (g-k-1)(k-2-i)\binom{k-2}{i-1}.$$

Since the rank and degree of the syzygy bundles N_i over $\mathbb{P}^1$ are known, the main object of investigation is the *splitting type*.

Remark 1.5. Casnati and Ekedahl [1996] generalized the relative canonical resolution to finite Gorenstein covers $\pi : X \to Y$ of degree k. They define a relative resolution of $X \subset \mathbb{P}(\mathscr{E}_T)$, where $\mathscr{E}_T$ is the Tschirnhausen bundle on Y defined by the short exact sequence

$$0 \to \mathscr{O}_Y \to \pi_*(\mathscr{O}_X) \to \mathscr{E}_T^\vee \to 0.$$

Note that for a cover $C \xrightarrow{k:1} \mathbb{P}^1$, $\mathscr{E}_T = \mathscr{E} \otimes \mathscr{O}_{\mathbb{P}^1}(2)$, where $\mathscr{E}$ is the bundle associated to (C, g_k^1) as in Theorem 1.3. The twists and hence the splitting types of the syzygy bundles in a resolution of $C \subset \mathbb{P}(\mathscr{E}_T)$ also differ from the ones in the relative canonical resolution of $C \subset \mathbb{P}(\mathscr{E})$. Indeed, following the proof of [Casnati and Ekedahl 1996, Step B, p. 445], for each i, the twist in the i-th syzygy bundle in a resolution of $C \subset \mathbb{P}(\mathscr{E}_T)$ differs by exactly $2 \cdot (i+1)$ from the ones given in our definition. Hence, we can deduce the degrees of the bundles in this relative resolution of $C \subset \mathbb{P}(\mathscr{E}_T)$ from Proposition 1.4. These degrees have also been computed directly in [Deopurkar and Patel 2018].

Definition 1.6. We say that a bundle on $\mathbb{P}^1$ of the form $N = \bigoplus_{j=1}^{\beta} \mathscr{O}_{\mathbb{P}^1}(a_j)$ is *balanced* if

$$\max_{i,j} |a_j - a_i| \leq 1.$$

Equivalently, the bundle N is balanced if $h^1(\mathbb{P}^1, \mathscr{E}nd(N)) = 0$. The relative canonical resolution is called balanced if all bundles N_i occurring in the resolution are balanced.

Remark 1.7. The locus of curves inside $\mathscr{H}_{g,k}$ that have a balanced relative canonical resolution forms an open subset of $\mathscr{H}_{g,k}$ which might be empty. Hence, to show that a generic relative canonical resolution is balanced for fixed values (g, k) it is sufficient to examine a single balanced example.

Remark 1.8. The scroll associated to a general element in $\mathscr{H}_{g,k}$ is always balanced by [Ballico 1989] and [Harris 1981]. The sublocus inside $\mathscr{H}_{g,k}$ parametrizing covers such that the associated scroll is

unbalanced defines a divisor on $\mathscr{H}_{g,k}$ precisely if g is a multiple of $(k-1)$. This divisor is called the *Maroni divisor* (for more details on the Maroni divisor see, e.g., [van der Geer and Kouvidakis 2017] and [Deopurkar and Patel 2015]).
On the other hand, knowing the splitting type of the syzygy bundles in the relative canonical resolution for generic elements in $\mathscr{H}_{g,k}$ one can study the sublocus inside $\mathscr{H}_{g,k}$ consisting set-theoretically of curves for which a certain syzygy bundle has nongeneric splitting type. This yields interesting subvarieties which also turn out to be divisors in some cases (see [Deopurkar and Patel 2018]). Similar to Koszul divisors on the moduli space $\mathscr{M}_g$, the study of the divisors obtained from the relative canonical resolution sheds light on the global geometry of the Hurwitz space.

2. Macaulay2 package.

The Macaulay2 package [RelativeCanonicalResolution] includes various useful functions to do experiments with k-gonal canonical curves and the relative canonical resolution of those curves. We will briefly explain how functions in this package construct g-nodal k-gonal canonical curves of genus g.

The main idea is that we start with a rational normalization of the desired curve and a degree k map from the normalization to $\mathbb{P}^1$. In the next step we choose g pairs of points $\{P_i, Q_i\}$ for $i = 1, \dots, g$ on the normalization, and we glue the points in each pair to each other. If $\mathscr{L}$ is a line bundle of degree k on a g-nodal curve C with rational normalization $\nu : \mathbb{P}^1 \to C$, then $\mathscr{L}$ is given as $\nu(\mathscr{L}) \cong \mathscr{O}_{\mathbb{P}^1}(k)$ together with gluing data between the residue class fields

$$\frac{a_i}{b_i} : \mathscr{O}_{\mathbb{P}^1}(k) \otimes \Bbbk(P_i) \to \mathscr{O}_{\mathbb{P}^1}(k) \otimes \Bbbk(Q_i),\ \ i = 1, \dots, g.$$

Let $S = \Bbbk[s, t]$ be the coordinate ring of $\mathbb{P}^1$. We start over by choosing two forms $f, h \in S_k$ of degree k and g points $R_i = (R_i^{(0)} : R_i^{(1)}) \in \mathbb{P}^1$ such that for all $i = 1, \dots, g$ the determinant

$$\det \begin{pmatrix} f & R_i^{(0)} \\ h & R_i^{(1)} \end{pmatrix} = l_i^0 \cdot l_i^{(1)} \cdot r_i$$

has at least two linear factors $l_i^{(0)}$ and $l_i^{(1)}$. Note that this step might be hard to perform over a field $\Bbbk$ of characteristic 0 and we therefore work over a finite field. We compute $2g$ points $P_i = V(l_i^{(0)})$ and $Q_i = V(l_i^{(1)})$ as the vanishing loci of these linear forms. We want to define multipliers $\{a_i, b_i\}_{i=1,\dots,g}$ such that

$$b_i \cdot f(P_i) = a_i \cdot h(Q_i) \text{ and } b_i \cdot h(P_i) = a_i \cdot f(Q_i) \quad \text{for } i = 1, \dots, g.$$

By construction, we can choose $\{a_i, b_i\}_{i=1,\dots,g}$ to be $b_i = 1$ and $a_i = \frac{f(P_i)}{f(Q_i)} = \frac{h(P_i)}{h(Q_i)}$. If we define

$$q_i := \det \begin{pmatrix} s & P_i^{(0)} \\ t & P_i^{(1)} \end{pmatrix} \cdot \det \begin{pmatrix} s & Q_i^{(0)} \\ t & Q_i^{(1)} \end{pmatrix} \quad \text{for } i = 1, \dots, g,$$

then a basis of $H^0(C, \omega_C)$ is given by

$$\left\{ s_j := \prod_{i=1, i \neq j}^{g} q_i \right\}_{j=1,\dots,g}.$$

This basis $\{s_j\}_{j=1,\dots,g}$ can furthermore be modified in such a way that the scroll defined by the line bundle of degree k will have a “normalized” form, i.e., the $2 \times (g-k+1)$ matrix defining the scroll will consist of blocks of the form

$$\begin{pmatrix} t_i & t_{i+2} \\ t_{i+1} & t_{i+3} \end{pmatrix},$$

where $T = \mathbb{k}[t_0, \dots, t_{g-1}]$ is the coordinate ring of $\mathbb{P}^{g-1}$.

In the package [RelativeCanonicalResolution] we also provide a function that describes the generators of the ideal of C in terms of elements of the Cox ring of the scroll $\mathbb{P}(\mathscr{E})$.

Remark 2.1. There is an explicit identification

$$H^0(\mathbb{P}(\mathscr{E}), \mathscr{O}_{\mathbb{P}(\mathscr{E})}(aH + bR)) \cong H^0(\mathbb{P}^1, (S_a\mathscr{E})(b)) \text{ for } a \geq 0,$$

where $S_a\mathscr{E}$ is the a-th symmetric power of the vector bundle $\mathscr{E}$ (see [Schreyer 1986, (1.3)]). This gives a description of the coordinate ring

$$R_{\mathbb{P}(\mathscr{E})} = \bigoplus_{a,b \in \mathbb{Z}} H^0(\mathbb{P}(\mathscr{E}), \mathscr{O}_{\mathbb{P}(\mathscr{E})}(aH + bR))$$

of $\mathbb{P}(\mathscr{E})$ as the Cox ring $\mathbb{k}[v, w, \varphi_0, \dots, \varphi_{d-1}]$ equipped with bigrading $\deg v = \deg w = (1,0)$ and $\deg \varphi_i = (e_1 - e_{i+1}, 1)$.

Finally the relative canonical resolution of $C \subset \mathbb{P}(\mathscr{E})$ can be computed by successively picking syzygies in correct degrees.

Example 2.2. We compute a nodal 6-gonal canonical curve of genus 9.

```
i1 : loadPackage("RelativeCanonicalResolution")
i2 : g=9; -- the genus
i3 : k=6; -- the degree of the pencil
i4 : n=10000; -- characteristic: next prime number after n
i5 : Ican=canCurveWithFixedScroll(g,k,n); -- the canonical curve
i6 : (dim Ican,genus Ican, degree Ican)
o6 = (2, 9, 16)
i7 : betti(res(Ican,DegreeLimit=>1))
            0  1  2  3
o7 = total: 1 15 35 21
         0: 1  .  .  .
         1: . 21 64 70
```

Next we compute the ideal of C inside the Cox ring of the scroll $\mathbb{P}(\mathscr{E})$.

```
i8 : Jcan=curveOnScroll(Ican,g,k); -- the curve inside the scroll
i9 : RX=ring Jcan; -- the bigraded Cox ring of the scroll
       ZZ
o9 = -----[pp , pp , pp , pp , pp , v,w]
     10007   0    1    2    3    4
```

We compute the relative canonical resolution:

```
i10 : T=ring Ican; -- the canonical ring
i11 : H=basis({1,1},RX); -- a basis of H^0(PE, OO_PE(H))
i12 : phi=map(RX,T,H)
i13 : Ican==preimage_phi(Jcan)
o13 = true
i14 : lengthRes=2; -- a lengthlimit for the resolution on the scroll
```

With respect to the total degree, the Betti table of the relative canonical resolution has the following form:

```
-- the relative canonical resolution:
i15 : betti(resX=resCurveOnScroll(Jcan,g,lengthRes))
             0 1  2 3 4
o15 = total: 1 9 16 9 1
          0: 1 .  . . .
          1: . .  . . .
          2: . 6  2 . .
          3: . 3 12 3 .
          4: . .  2 6 .
          5: . .  . . 1
```

The scroll cut out by the g^1_6 on C has the following normalized determinantal representation:

```
i16 : X=preimage_phi(ideal 0_RX); -- the ideal of the scroll
i17 : repX=matrix{{t_0,t_2,t_4,t_6},{t_1,t_3,t_5,t_7}}
o17 = | t_0 t_2 t_4 t_6 |
      | t_1 t_3 t_5 t_7 |
i18 : minors(2,repX)==X
o18 = true
```

Remark 2.3. By o15, we see that the second syzygy bundle N_2 is unbalanced in our example. Although this single example does not show that the generic relative canonical resolution is unbalanced for this case, one can show that this is indeed the generic form (see [Bopp and Hoff 2017]).

3. Experiments and conjectures.

Database of experiments. Using our Macaulay2 package [RelativeCanonicalResolution] we have computed the relative canonical resolution for various cases. For nonhyperelliptic, generic curves of genus $g \leq 23$ with a pencil of degree $3 \leq k \leq \min\{g-1, 14\}$, all expected Betti tables are listed on the webpage [Blug et al. 2018].

The web page was set up with the help of Sascha Blug. All the experiments that led to Betti tables in [Blug et al. 2018] were performed over a finite field. If the examples for certain values (g, k) yield a balanced relative canonical resolution, then by semi-continuity one can conclude that this is indeed the general behavior (even for complex algebraic curves).

Since changing the characteristic for the unbalanced cases did not change the shape of the Betti tables, we believe that the Betti tables in [Blug et al. 2018] reflect the generic behavior. In general, we do not have a proof of this statement. However, it has been determined in some cases whether the first bundle N_1 is balanced (see [Bopp and Hoff 2015] and [Bujokas and Patel 2015]). For several cases our examples lead to a conjecture that certain higher syzygy bundles in the relative canonical resolution are unbalanced. Most of these cases remain mysterious.

Syzygy divisors on Hurwitz spaces. Deopurkar and Patel used the relative canonical resolution to describe new effective divisors on the Hurwitz scheme $\mathscr{H}_{g,k}$. If the degree k divides $g-1$, it is shown in [Bujokas and Patel 2015] that the relative canonical resolution for a generic element in $\mathscr{H}_{g,k}$ is totally balanced, and hence, the locus μ_i, corresponding set-theoretically to covers in $\mathscr{H}_{g,k}$ for which the i-th syzygy bundle N_i is unbalanced, has expected codimension 1. Deopurkar and Patel [2018] give these syzygy divisors $\mu_1, \dots, \mu_{k-3}$ a scheme structure and compute their classes in a partial compactification of the Hurwitz scheme $\widetilde{\mathscr{H}}_{g,k}$. In their main theorem, they represent the divisor classes $[\mu_i]$ in terms of certain tautological classes κ, ζ and δ (see [Deopurkar and Patel 2018, §2] for the precise definition of those classes).

Theorem 3.1 [Deopurkar and Patel 2018, Theorem 1.1]. *Suppose k divides $g-1$. Let i be an integer with $1 \le i \le k-3$. The locus $\mu_i \subset \widetilde{\mathscr{H}}_{g,k}$ is an effective divisor whose class in* $\mathrm{Pic}_{\mathbb{Q}}(\widetilde{\mathscr{H}}_{g,k})$ *is given by*

$$[\mu_i] = A_i \cdot \bigl(6(gk-6g+k+6)\cdot\zeta - k(k-12)\cdot\kappa - k^2\cdot\delta\bigr),$$

where

$$A_i := \frac{(k-2)(k-3)}{6(i+1)(k-i-1)} \cdot \binom{k-4}{i-1}^2.$$

Note that all the classes $[\mu_i]$ are proportional. The same phenomenon appears for classes of divisorial Brill–Noether loci in the moduli space $\overline{\mathscr{M}}_g$ (see [Eisenbud and Harris 1987a] and [Harris and Mumford 1982]). For the divisorial Brill–Noether classes it is known that these classes are supported on different sets, and in [Deopurkar and Patel 2018] the authors conjecture that this also happens for the syzygy divisors on $\mathscr{H}_{g,k}$.

We come to a different conclusion. Computing various examples of curves and their relative canonical resolution for $(g,k) \in \{(6,13),(7,15),(8,17),(6,19)\}$ over a field of small characteristic $p \le 500$ we found the following pattern which we conjecture to be true in general. Note that the probability of a random computed example to end up in a certain codimension 1 locus is roughly $\frac{1}{p}$.

Conjecture 1. *Let n and k be integers, and $g-1 = n\cdot k$. Let i be an integer with $1 \le i \le k-3$. For a general element $(C, g^1_k) \in \mu_i \subset \widetilde{\mathscr{H}}_{g,k}$, let N_j be the j-th syzygy bundle in the relative canonical resolution of C with $1 \le j \le k-3$. Then N_j is unbalanced and the splitting type of N_j is*

$$N_j = \mathscr{O}_{\mathbb{P}^1}\bigl((n-1)(j+1)-1\bigr)^{\oplus\binom{k-4}{j-1}} \oplus \mathscr{O}_{\mathbb{P}^1}\bigl((n-1)(j+1)\bigr)^{\mathrm{rk}\,N_j - 2\cdot\binom{k-4}{j-1}} \oplus \mathscr{O}_{\mathbb{P}^1}\bigl((n-1)(j+1)+1\bigr)^{\oplus\binom{k-4}{j-1}}.$$

In particular, all the effective divisors μ_i are supported on the same set.

Remark 3.2. One can easily check that the number A_i in Theorem 3.1 is precisely

$$A_i = \frac{1}{6k}\cdot \mathrm{rk}\,N_i \cdot \binom{k-4}{i-1}.$$

Conjecture 1 predicts that the factor $\binom{k-4}{i-1}$ of A_i also measures how unbalanced the bundle N_i is.

Remark 3.3. If $(g-1) \neq n \cdot k$ then one can still consider the jumping loci set-theoretically defined as the subset of $\widetilde{\mathscr{H}}_{g,k}$ consisting of covers such that the i-th syzygy bundle in the relative canonical resolution does not have generic splitting type. As in the divisorial case, one could ask if all those loci are supported on the same set. Experiments using our package [RelativeCanonicalResolution] show that there are several examples where these jumping loci have different support.

Further conjectures. We state several conjectures concerning the shape of relative canonical resolutions. This has partly also been discussed in [Bopp and Hoff 2015]. We refer to [Arbarello et al. 1985] for basics about Brill–Noether theory. Recall that the Brill–Noether number is defined as $\rho(g,k,r) := g-(r+1)(g-k+r)$ for integers g, k and r.

Conjecture 2. *Let $C \subset \mathbb{P}^{g-1}$ be a general canonical curve, and let k be a positive integer such that $\rho := \rho(g,k,1) \geq 0$, and let g^1_k be a general pencil in $W^1_k(C)$. Then for bundles $N_i = \bigoplus \mathscr{O}_{\mathbb{P}^1}(a^{(i)}_j)$, $i = 2, \dots, \lceil \frac{k-3}{2} \rceil$, there is the bound*

$$\max_{j,l} |a^{(i)}_j - a^{(i)}_l| \leq \min\{g-k-1, i+1\}.$$

This bound is furthermore sharp in the following sense. Given two integers $k \geq 3$ and $2 \leq i \leq \lceil \frac{k-3}{2} \rceil$, there exists an integer g such that the general canonical curve C of genus g has an i-th syzygy bundle N_i in the relative canonical resolution, associated to a general pencil in $W^1_k(C)$, that satisfies $\max_{j,l}|a^{(i)}_j - a^{(i)}_l| = \min\{g-k-1, i+1\}$. In particular, if $g-k=2$, the relative canonical resolution is balanced.

Remark 3.4. Conjecture 2 in the case $g-k=2$ says that the bundles in the relative canonical resolution are of the form

$$N_i = \mathscr{O}_{\mathbb{P}^1}^{\oplus i \cdot \binom{g-4}{i+1}} \oplus \mathscr{O}_{\mathbb{P}^1}(1)^{\oplus (g-4-i) \cdot \binom{g-4}{g-3-i}}.$$

Note that the Betti numbers $i \cdot \binom{k-2}{i+1}$ appearing in the conjecture are the Betti numbers of a rational normal curve of degree $k-2$.

We also verified Conjecture 3 for $g \leq 23$.

Conjecture 3. *For a general cover $C \to \mathbb{P}^1$ in $\mathscr{H}_{g,k}$ with $\rho(g,k,1) \leq 0$, the bundle N_1 is balanced.*

ACKNOWLEDGEMENTS. We would like to thank Sascha Blug for setting up the web page [Blug et al. 2018] that displays all the experimental data. This work is a contribution to the Project 1.7 of the SFB-TRR 195 "Symbolic Tools in Mathematics and their Application" of the German Research Foundation (DFG). We also thank the referees for suggestions and comments improving the presentation of the article and the *Macaulay*2 package.

SUPPLEMENT. The online supplement contains version 1.0 of [RelativeCanonicalResolution].

References.

[Arbarello et al. 1985] E. Arbarello, M. Cornalba, P. A. Griffiths, and J. Harris, *Geometry of algebraic curves, I*, Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences] **267**, Springer, 1985. MR

[Ballico 1989] E. Ballico, "A remark on linear series on general k-gonal curves", *Boll. Un. Mat. Ital. A* (7) **3**:2 (1989), 195–197. MR

[Blug et al. 2018] S. Blug, C. Bopp, and M. Hoff, "Relative canonical resolutions", 2018, available at https://www.math.uni-sb.de/ag/schreyer/images/data/computeralgebra/relcanres/html/.

[Bopp and Hoff 2015] C. Bopp and M. Hoff, "Resolutions of general canonical curves on rational normal scrolls", *Arch. Math.* (*Basel*) **105**:3 (2015), 239–249. MR

[Bopp and Hoff 2017] C. Bopp and M. Hoff, "Moduli of lattice polarized K3 surfaces via relative canonical resolutions", preprint, 2017. arXiv

[Bujokas and Patel 2015] G. Bujokas and A. Patel, "Invariants of a general branched cover of $\mathbb{P}^1$", preprint, 2015. arXiv

[Casnati and Ekedahl 1996] G. Casnati and T. Ekedahl, "Covers of algebraic varieties, I: A general structure theorem, covers of degree 3, 4 and Enriques surfaces", *J. Algebraic Geom.* **5**:3 (1996), 439–460. MR

[Deopurkar and Patel 2015] A. Deopurkar and A. Patel, "The Picard rank conjecture for the Hurwitz spaces of degree up to five", *Algebra Number Theory* **9**:2 (2015), 459–492. MR

[Deopurkar and Patel 2018] A. Deopurkar and A. Patel, "Syzygy divisors on Hurwitz spaces", pp. 209–222 in *Higher genus curves in mathematical physics and arithmetic geometry*, edited by A. Malmendier and T. Shaska, Contemp. Math. **703**, Amer. Math. Soc., Providence, RI, 2018. MR

[Eisenbud and Harris 1987a] D. Eisenbud and J. Harris, "The Kodaira dimension of the moduli space of curves of genus ≥ 23", *Invent. Math.* **90**:2 (1987), 359–387. MR

[Eisenbud and Harris 1987b] D. Eisenbud and J. Harris, "On varieties of minimal degree (a centennial account)", pp. 3–13 in *Algebraic geometry* (Brunswick, Maine, 1985), edited by S. J. Bloch, Proc. Sympos. Pure Math. **46**, Amer. Math. Soc., Providence, RI, 1987. MR

[Farkas 2009] G. Farkas, "Koszul divisors on moduli spaces of curves", *Amer. J. Math.* **131**:3 (2009), 819–867. MR

[Farkas 2018] G. Farkas, "Effective divisors on Hurwitz spaces", preprint, 2018. arXiv

[van der Geer and Kouvidakis 2017] G. van der Geer and A. Kouvidakis, "The cycle classes of divisorial Maroni loci", *Int. Math. Res. Not.* **2017**:11 (2017), 3463–3509. MR

[Harris 1981] J. Harris, "A bound on the geometric genus of projective varieties", *Ann. Scuola Norm. Sup. Pisa Cl. Sci.* (4) **8**:1 (1981), 35–68. MR

[Harris and Mumford 1982] J. Harris and D. Mumford, "On the Kodaira dimension of the moduli space of curves", *Invent. Math.* **67**:1 (1982), 23–88. MR

[Hirschowitz and Ramanan 1998] A. Hirschowitz and S. Ramanan, "New evidence for Green's conjecture on syzygies of canonical curves", *Ann. Sci. École Norm. Sup.* (4) **31**:2 (1998), 145–152. MR

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

[RelativeCanonicalResolution] C. Bopp and M. Hoff, "RelativeCanonicalResolution - construction of relative canonical resolutions and Eagon–Northcott type complexes", Macaulay2 package, available at https://www.math.uni-sb.de/ag/schreyer/index.php/computeralgebra.

[Schreyer 1986] F.-O. Schreyer, "Syzygies of canonical curves and special linear series", *Math. Ann.* **275**:1 (1986), 105–137. MR

[Voisin 2005] C. Voisin, "Green's canonical syzygy conjecture for generic curves of odd genus", *Compos. Math.* **141**:5 (2005), 1163–1190. MR

Received: 31 Jul 2018 Revised: 22 Jun 2020 Accepted: 28 Aug 2020

Christian Bopp:

bopp@math.uni-sb.de
Universität des Saarlandes, Saarbrücken, Germany

Michael Hoff:

hahn@math.uni-sb.de
Universität des Saarlandes, Saarbrücken, Germany

JSAG **11** (2021), 25–39
https://doi.org/10.2140/jsag.2021.11.25

The Journal of Software for
Algebra and Geometry

The FrobeniusThresholds package for Macaulay2

DANIEL J. HERNÁNDEZ, KARL SCHWEDE, PEDRO TEIXEIRA AND EMILY E. WITT

ABSTRACT: This article describes the *Macaulay*2 package *FrobeniusThresholds*, designed to estimate and calculate F-pure thresholds, more general F-thresholds, and related numerical invariants arising in the study of singularities in prime characteristic commutative algebra.

1. INTRODUCTION. This paper describes the *Macaulay*2 package *FrobeniusThresholds* [Grayson and Stillman; Bruce et al.] (previously named *FThresholds*), which provides tools for computing or estimating certain fundamental invariants in positive characteristic commutative algebra, namely *F-pure thresholds*, *F-thresholds*, and *F-jumping exponents*. Recall that a ring of prime characteristic $p > 0$ is *F-pure* if the Frobenius map — that is, the ring endomorphism sending an element to its p-th power — is a pure morphism; under natural geometric hypotheses, this is equivalent to the condition that the Frobenius morphism is a split injection of rings. The concept of F-purity has proven to be important in commutative algebra, and has a rich history. It first appeared in [Hochster and Roberts 1976] to study local cohomology, was compared with rational singularities in [Fedder 1983], and was used to study global properties of Schubert varieties in [Mehta and Ramanathan 1985].

After the advent of tight closure [Hochster and Huneke 1990], the use of the Frobenius map to quantify singularity — that is, deviation from regularity — proliferated, and based on a connection discovered between F-pure and *log canonical* singularities [Hara and Watanabe 2002], the concept of F-purity was generalized to the context of *pairs*. Along these lines, the *F-pure threshold* was defined in analogy with the *log canonical threshold* [Takagi and Watanabe 2004], and *F-thresholds* were introduced as a natural extension [Mustaţă et al. 2005].

The connection between the F-pure threshold and the log canonical threshold, however, extends beyond mere analogy. For example, suppose h is a polynomial with integer coefficients, and that h_p is the polynomial obtained by reducing the coefficients of h modulo a prime p. Then, the F-pure thresholds of the reductions h_p converge to the log canonical threshold of h as p tends to infinity [Hara and Yoshida 2003]. A related result is [Zhu 2017, Corollary 4.2], which proves that the log canonical threshold of h is at least the F-pure threshold of any reduction h_p.

D. J. Hernández was partially supported by NSF DMS #1600702. K. Schwede was supported by NSF CAREER Grant DMS #1252860/1501102, NSF FRG Grant DMS #1265261/1501115, NSF grant #1801849 and a Sloan Fellowship. E. E. Witt was partially supported by NSF DMS #1623035.
MSC2010: 13A35.
Keywords: Macaulay2, F-singularity, Frobenius, F-threshold, F-pure threshold.
FrobeniusThresholds version 2.1

This latter result is interesting from a computational perspective, in that it provides lower bounds for log canonical thresholds. Though there is a general purpose implementation of an algorithm for computing log canonical thresholds in the *Dmodules* package [Leykin and Tsai],[1] the function for computing F-pure thresholds contained in the *FrobeniusThresholds* package is typically much faster, especially so in low characteristic.

In summary, the F-pure threshold is an interesting numerical invariant, related to many other measures of singularity across all characteristics, and has been the focus of intense study over the past fifteen years. Unfortunately, it is typically difficult to calculate. The package *FrobeniusThresholds* is centered on calculating and estimating the F-pure threshold and other F-thresholds, with the function `fpt` at its core. It builds heavily upon the *TestIdeals* package for *Macaulay*2 [Bela et al.; Boix et al. 2019], which provides a broad range of functionality for effective computation in prime characteristic commutative algebra.

1.1. ***Some background and notation.*** Though some functionality implemented in *FrobeniusThresholds* is not restricted to regular ambient rings (see Section 3), for the sake of concreteness, in this introduction we will work in a polynomial ring over a finite field of characteristic p. The ideal of this ring generated by its variables is denoted $\mathfrak{m}$.

Let us outline a way in which natural numerical invariants in prime characteristic commutative algebra are often constructed: For every natural number e, associate to some fixed data—often, a collection of polynomials or ideals—an integer describing something of relevance that depends on e (e.g., the dimension of some interesting vector space constructed in terms of the initial data). Normalize this integer by dividing by some power of p^e, and then take the limit as the integer e tends to infinity. The resulting limit, if it exists, should encode interesting information about the initial data.

For example, the *Hilbert–Kunz multiplicity* is realized in this way. Suppose that I is an ideal of a ring R of characteristic $p > 0$. Given an integer $e \geqslant 1$, $I^{[p^e]}$ denotes the p^e-th Frobenius power of I, that is, the ideal generated by the p^e-th powers of the elements of I. If R has dimension d and $\lambda(R/I^{[p^e]})$ denotes the length of $R/I^{[p^e]}$, then the limit of $\lambda(R/I^{[p^e]})/p^{ed}$ as e tends to infinity is the Hilbert–Kunz multiplicity of R with respect to I.

Consider a nonzero polynomial f and a natural number e. If f does not vanish at the origin, then set $\nu_f^{\mathfrak{m}}(p^e) := \infty$. Otherwise, $f \in \mathfrak{m}$, and we instead define

$$\nu_f^{\mathfrak{m}}(p^e) := \max\{n \in \mathbb{N} : f^n \notin \mathfrak{m}^{[p^e]}\}.$$

If $f^n \notin \mathfrak{m}^{[p^e]}$ for some n, then by the flatness of Frobenius [Kunz 1969], $f^{pn} \notin (\mathfrak{m}^{[p^e]})^{[p]} = \mathfrak{m}^{[p^{e+1}]}$. Hence the sequence $(\nu_f^{\mathfrak{m}}(p^e)/p^e)_{e=0}^{\infty}$ is nondecreasing, and since $f^{p^e} \in \mathfrak{m}^{[p^e]}$, the sequence is bounded above by 1. Following our outline, we define

$$\mathrm{c}^{\mathfrak{m}}(f) := \lim_{e\to\infty} \frac{\nu_f^{\mathfrak{m}}(p^e)}{p^e}.$$

[1]The *MultiplierIdeals* package [Teitler et al.; Teitler 2015] also computes log canonical thresholds in many special cases, including monomial ideals, hyperplane arrangements, generic determinantal ideals, and certain binomial ideals.

This limit exists by the above discussion, and is a rational number when $f \in \mathfrak{m}$, though the latter is far from obvious [Blickle et al. 2008, Theorem 3.1]. Inspired by its connections with the F-purity of pairs, this limit is called the *F-pure threshold* of f at the origin.

The F-pure threshold is closely related to many other fundamental concepts in prime characteristic commutative algebra. For instance,

$$\mathrm{c}^{\mathfrak{m}}(f) = \inf\{t > 0 : \tau(f^t) \subseteq \mathfrak{m}\} = \sup\{t > 0 : \sigma(f^t) \neq 0\},$$

where $\tau(f^t)$ and $\sigma(f^t)$ are the *test ideal* and *F-signature*, respectively, associated to f and the formal nonnegative real exponent t. The former is an ideal in the ambient ring of f, and the latter is a real number; both depend on the parameter t and the characteristic p in subtle ways [Blickle et al. 2008; 2012].

In the literature, the F-pure threshold $\mathrm{c}^{\mathfrak{m}}(f)$ is often denoted $\mathrm{fpt}(f)$, for obvious reasons. However, in this note we adopt the former notation to avoid any possible confusion with the function `fpt` described in Section 4, which sometimes does not output the number $\mathrm{c}^{\mathfrak{m}}(f) = \mathrm{fpt}(f)$, but returns, instead, lower and upper bounds for that number.

It turns out that the sequence $(\nu_f^{\mathfrak{m}}(p^e))_{e=0}^{\infty}$ itself, and not just its limit, encodes interesting information about f. For example, it is closely related to the *Bernstein–Sato polynomial* of f [Mustaţă et al. 2005]. Remarkably, one can recover the sequence $(\nu_f^{\mathfrak{m}}(p^e))_{e=0}^{\infty}$ from the limit $\mathrm{c}^{\mathfrak{m}}(f)$ [Mustaţă et al. 2005; Hernández 2012]: For each e, we have

$$\nu_f^{\mathfrak{m}}(p^e) = \lceil p^e \cdot \mathrm{c}^{\mathfrak{m}}(f) \rceil - 1.$$

We conclude this subsection by briefly reviewing some natural generalizations. Suppose that I and J are ideals. If I is contained in the radical of J, then we set

$$\nu_I^J(p^e) := \max\{n \in \mathbb{N} : I^n \not\subseteq J^{[p^e]}\},$$

or $\nu_I^J(p^e) := 0$, when the set on the right-hand side is empty. Otherwise, we set $\nu_I^J(p^e) := \infty$. This clearly generalizes the quantity $\nu_f^{\mathfrak{m}}(p^e)$ considered earlier, and we call

$$\mathrm{c}^J(I) := \lim_{e\to\infty} \frac{\nu_I^J(p^e)}{p^e}$$

the *F-threshold of I with respect to J*. This limit again exists, and the value $\mathrm{c}^{\mathfrak{m}}(I)$ is called the *F-pure threshold of I* at the origin, and if $I = \langle f \rangle$ is principal, $\mathrm{c}^J(f) := \mathrm{c}^J(I)$ is called the *F-threshold of f with respect to J*. Like F-pure thresholds, F-thresholds are rational (when finite), and can be characterized in terms of test ideals.

2. THE `frobeniusNu` FUNCTION. We first describe the `frobeniusNu` function, a fundamental component of the package *FrobeniusThresholds*. We adopt the setup established in the introduction: we work in a polynomial ring R over a finite field of characteristic $p > 0$, $\mathfrak{m}$ denotes the ideal generated by the variables, and e is a natural number. If I and J are ideals of R, the command `frobeniusNu(e,I,J)` outputs the extended integer $\nu_I^J(p^e)$ defined in the introduction; if f is an element of R, `frobeniusNu(e,f,J)` outputs $\nu_f^J(p^e) := \nu_{\langle f\rangle}^J(p^e)$. When the third argument is omitted from the function `frobeniusNu`, it is assumed to be the maximal ideal $\mathfrak{m}$.

```
i1 : R = ZZ/11[x,y];

i2 : I = ideal(x^2 + y^3, x*y);

o2 : Ideal of R

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

o3 : Ideal of R

i4 : frobeniusNu(2, I, J)

o4 = 281

i5 : f = x*y*(x^2 + y^2);

i6 : frobeniusNu(2, f, J)

o6 = 120
```

In general, the function `frobeniusNu` works by searching through a list of integers n, and checking containments of the n-th power of I in the specified Frobenius power of J. It is well known that, for any positive integer e,

$$\nu_I^J(p^e) = \nu_I^J(p^{e-1}) \cdot p + L,$$

where the error term L is nonnegative and can be explicitly bounded from above in terms of p and the number of generators of I and J. For instance, the error term L is at most $p-1$ when I is principal and J is arbitrary. This implies that when searching for the maximal exponent defining `frobeniusNu(e,I,J)` for positive e, it is safe to start at p times the output of `frobeniusNu(e-1,I,J)`, and one need not search too far past this number.

2.1. *Options for* `frobeniusNu`.

The user can specify how the search is approached through the option `Search`, which can take two values: `Binary` (the default value) and `Linear`. In the example below, the default search method, `Binary`, is used.

```
i7 : R = ZZ/5[x,y,z];

i8 : m = ideal(x, y, z);

o8 : Ideal of R

i9 : time frobeniusNu(2, m, m^2)
     -- used 1.82479 seconds

o9 = 97
```

However, a linear search is faster in this case.

```
i10 : time frobeniusNu(2, m, m^2, Search => Linear)
      -- used 0.597035 seconds

o10 = 97
```

If the option `ReturnList` is changed from its default value of `false` to `true`, `frobeniusNu` outputs a list of the values $\nu_I^J(p^s)$, for $s = 0, \ldots, e$, at no additional computational cost.

```
i11 : frobeniusNu(5, x^2*y^4 + y^2*z^7 + z^2*x^8, ReturnList => true)

o11 = {0, 1, 8, 44, 224, 1124}

o11 : List
```

The same information can be found by setting the option `Verbose` to `true`, to request that the values $\nu_I^J(p^s)$ be printed as they are iteratively computed (serving also as a way to monitor the progress of the computation).

As described in the introduction, the integer $\nu_I^J(p^e)$ is the maximal integer n such that the n-th power of I is not contained in the p^e-th Frobenius power of J. However,

$$I^n \subseteq J^{[p^e]} \iff (I^n)^{[1/p^e]} \subseteq J,$$

where $(I^n)^{[1/p^e]}$ denotes the p^e-th *Frobenius root* of I^n, as defined in [Blickle et al. 2008]. The option `ContainmentTest` for `frobeniusNu` allows the user to choose which of the two types of containment statements appearing above to use toward the calculation of $\nu_I^J(p^e)$.

If `ContainmentTest` is set to `StandardPower`, then `frobeniusNu(e,I,J)` is computed by testing the left-hand containment above, and when it is set to `FrobeniusRoot`, the right-hand containment is checked. For efficiency reasons, the default value for `ContainmentTest` is set to `FrobeniusRoot` if the second argument passed to `frobeniusNu` is a polynomial, and is set to `StandardPower` if the second argument is an ideal.

In this example, `ContainmentTest` is set to its default value for polynomials, namely, `FrobeniusRoot`:

```
i12 : R = ZZ/11[x,y,z];

i13 : f = x^3 + y^3 + z^3 + x*y*z;

i14 : time frobeniusNu(3, f)
     -- used 0.153691 seconds

o14 = 1209
```

If `ContainmentTest` is set to `StandardPower`, instead, the computation is significantly slower.

```
i15 : time frobeniusNu(3, f, ContainmentTest => StandardPower)
     -- used 10.1343 seconds

o15 = 1209
```

The option `ContainmentTest` has a third possible value, called `FrobeniusPower`, which allows `frobeniusNu` to compute a different but analogous invariant. In [Hernández et al. 2020], we introduced the notion of a (generalized) Frobenius power $I^{[n]}$ of an ideal I, when n is an arbitrary nonnegative integer. When `ContainmentTest` is set to `FrobeniusPower`, rather than computing $\nu_I^J(p^e)$, the function `frobeniusNu` computes the maximal integer n for which $I^{[n]}$ is not contained in $J^{[p^e]}$. We denoted this number by $\mu_I^J(p^e)$, and it equals $\nu_I^J(p^e)$ when I is a principal ideal. However, these numbers need not agree in general, as we see below:

```
i16 : R = ZZ/3[x,y];

i17 : m = ideal(x, y);

o17 : Ideal of R

i18 : frobeniusNu(4, m^5)

o18 = 32

i19 : frobeniusNu(4, m^5, ContainmentTest => FrobeniusPower)

o19 = 26
```

As pointed out in the introduction, if $f \in \mathfrak{m}$, the values $\nu_f^{\mathfrak{m}}(p^e)$ can be recovered from the F-pure threshold of f. This is used to speed up computations for certain polynomials whose F-pure thresholds can be computed quickly via specialized algorithms or formulas, namely diagonal polynomials, binomials, forms in two variables, and polynomials that define simple normal crossing divisors (see Section 4). This feature can be disabled by setting the option `UseSpecialAlgorithms` (default value `true`) to `false`.[2]

The following example shows, for a diagonal polynomial, how much faster the computation can be when special algorithms are enabled:

```
i20 : R = ZZ/17[x,y,z];
i21 : f = x^3 + y^4 + z^5;
i22 : time frobeniusNu(10, f)
     -- used 0.0161622 seconds
o22 = 1541642394460
i23 : time frobeniusNu(10, f, UseSpecialAlgorithms => false)
     -- used 2.06877 seconds
o23 = 1541642394460
```

The last option we describe for `frobeniusNu` is `AtOrigin`. Recall that $\nu_I^{\mathfrak{m}}(p^e)$ can be interpreted as the maximum integer n for which $(I^n)^{[1/p^e]}$ is not contained in $\mathfrak{m}$. When the option `AtOrigin` is set to `false` (from its default value `true`), the function `frobeniusNu` determines, instead, the maximum integer n for which $(I^n)^{[1/p^e]}$ is the unit ideal, which can also be characterized as the minimal integer $\nu_I^{\mathfrak{n}}(p^e)$ as $\mathfrak{n}$ varies among all maximal ideals of the ring.

```
i24 : R = ZZ/7[x,y];
i25 : f = (x - 1)^3 - (y - 2)^2;
i26 : frobeniusNu(3, f)
o26 = infinity
o26 : InfiniteNumber
i27 : frobeniusNu(3, f, AtOrigin => false)
o27 = 285
```

3. `isFPT`, `compareFPT` and `isFJumpingExponent`.

The *FrobeniusThresholds* package contains methods to test candidate values for F-pure thresholds and F-jumping numbers, even in some singular rings. Consider a $\mathbb{Q}$-Gorenstein ring R of characteristic $p > 0$, whose index is not divisible by p. Given a parameter $t \in \mathbb{Q}$ and an element f of R, the command `isFPT(t, f)` checks whether t is the F-pure threshold of f, while `compareFPT(t, f)` provides further information, returning `-1`, `0`, or `1` when t is, respectively, less than, equal to, or greater than the F-pure threshold of f. Setting the option `AtOrigin` to `true` tells the function to consider the F-pure threshold at the origin.

```
i1 : R = ZZ/11[x,y,z]/(x^2 - y*(z - 1));
i2 : compareFPT(5/11, z - 1)
o2 = -1
```

[2]In Section 4.1 we discuss a couple of situations in which this may be desirable.

```
i3 : isFPT(1/2, z - 1)
o3 = true
i4 : isFPT(1/2, z - 1, AtOrigin => true)
o4 = false
```

The general method applied calls upon functionality from the *TestIdeals* package. The test ideals $\tau(f^t)$ of f vary discretely with the parameter t; the function `FPureModule` in *TestIdeals* is used to compute the "last" test ideal of f with parameter in the interval $[0, t)$. Comparing this with the test ideal $\tau(f^t)$, computed by the function `testIdeal`, we can determine whether t is an F-jumping number of f, or more specifically, the F-pure threshold of f.

```
i5 : R = ZZ/13[x,y];
i6 : f = y*((y + 1) - (x - 1)^2)*(x - 2)*(x + y - 2);
i7 : isFJumpingExponent(3/4, f)
o7 = true
i8 : isFPT(3/4, f)
o8 = false
```

4. The `fpt` function.

The core function in the package *FrobeniusThresholds* is called `fpt`. Throughout this section, let f be a polynomial with coefficients in a finite field of characteristic p. When passed the polynomial f, the function `fpt` attempts to find the exact value for the F-pure threshold of f at the origin, and returns that value, if possible. Otherwise, it returns lower and upper bounds for the F-pure threshold, as demonstrated below.

```
i1 : R = ZZ/5[x,y,z];
i2 : fpt(x^3 + y^3 + z^3 + x*y*z)
     4
o2 = -
     5
o2 : QQ
i3 : fpt(x^5 + y^6 + z^7 + (x*y*z)^3)
       7   2
o3 = {--, -}
      25   5
o3 : List
```

4.1. *The option* `UseSpecialAlgorithms`.

The `fpt` function has an option `UseSpecialAlgorithms`, which, when set to `true` (its default value), tells `fpt` to first check whether f is a diagonal polynomial, a binomial, a form in two variables, or a polynomial that defines a simple normal crossing divisor, in that order. When f is a diagonal polynomial, a binomial, or a form in two variables, algorithms of Hernández [2015; 2014] or Hernández and Teixeira [2017] are executed to compute the F-pure threshold.

In the example below, we compute the F-pure threshold of a diagonal polynomial.

```
i4 : fpt(x^17 + y^20 + z^24)
      94
o4 = ---
     625
o4 : QQ
```

Next, we compute the F-pure threshold of a binomial.

```
i5 : fpt(x^2*y^6*z^10 + x^10*y^5*z^3)

      997
o5 = ----
     6250

o5 : QQ
```

Finally, we compute the F-pure threshold of a form in two variables.

```
i6 : R = ZZ/5[x,y];

i7 : fpt(x^2*y^6*(x + y)^9*(x + 3*y)^10)

      5787
o7 = -----
     78125

o7 : QQ
```

The algorithm for computing the F-pure threshold of a binary form f requires factoring f into linear forms, and that may be difficult or impossible when that factorization occurs in a Galois field of excessively large order. This is a situation when the user will want to set the option `UseSpecialAlgorithms` to `false`. However, when a factorization is already known, instead of passing f to `fpt`, the user can pass a list of all the pairwise coprime linear factors of f to `fpt`, and a list of their respective multiplicities.

```
i8 : fpt({x, y, x + y, x + 3*y}, {2, 6, 9, 10}) == oo

o8 = true
```

If `UseSpecialAlgorithms` is set to `true` and f does not fall into any of the aforementioned cases, then the function `fpt` next calls `isSimpleNormalCrossing(f)` (see Section 4.3) to check whether the polynomial f defines (locally, at the origin) a simple normal crossing divisor, in which case the F-pure threshold is simply the reciprocal of the largest multiplicity occurring in that factorization. Note that the function `factor` is called whenever `isSimpleNormalCrossing` is used, and that can sometimes make the verification slow. The user can avoid this by setting `UseSpecialAlgorithms` to `false`.

4.2. ***When no special algorithm applies.*** We now explain how the function `fpt` proceeds when no special algorithm is available, or when `UseSpecialAlgorithms` is set to `false`. In this case, `fpt` computes a sequence of lower and upper bounds for the F-pure threshold of f, and either finds its exact value in this process, or outputs the last of these sets of bounds, which will be the tightest among all computed. The value of the option `DepthOfSearch` determines the precision of the initial set of bounds, and the option `Attempts` determines, roughly, how many new, tighter sets of bounds are to be computed.

More specifically, let e denote the value of the option `DepthOfSearch`, which conservatively defaults to 1. The `fpt` function first computes $\nu = \nu_f(p^e)$, which agrees with the output of `frobeniusNu(e,f)`. It is well known that the F-pure threshold of f is greater than ν/p^e and at most $(\nu+1)/p^e$, and applying [Hernández 2012, Proposition 4.2] to the lower bound tells us that the F-pure threshold of f must be at least $\nu/(p^e-1)$. In summary, we know that the F-pure threshold of f must lie in the closed interval

$$\left[\frac{\nu}{p^e-1}, \frac{\nu+1}{p^e}\right]. \tag{†}$$

With these estimates in hand, the subroutine `guessFPT` is called to make some "educated guesses" in an attempt to identify the F-pure threshold within this interval, or at least narrow down this interval to produce improved estimates. The number of "guesses" is controlled by the option `Attempts`, which conservatively defaults to `3`. If `Attempts` is set to `0`, then `guessFPT` is bypassed. If `Attempts` is set to at least `1`, then a first check is run to verify whether the right-hand endpoint $(\nu+1)/p^e$ of the above interval (†) is the F-pure threshold.

To illustrate these options, first we obtain a rather crude estimate for the F-threshold of a polynomial.

```
i9 : f = x^2*(x + y)^3*(x + 3*y^2)^5;

i10 : fpt(f, Attempts => 0)

            1
o10 = {0, -}
            5

o10 : List
```

Increasing the depth of search, we obtain a better estimate.

```
i11 : fpt(f, Attempts => 0, DepthOfSearch => 3)

        21    22
o11 = {---, ---}
       124   125

o11 : List
```

Finally, increasing the number of attempts we find that the right-hand endpoint of the above interval is the desired F-pure threshold.

```
i12 : fpt(f, Attempts => 1, DepthOfSearch => 3)

       22
o12 = ---
      125

o12 : QQ
```

If `Attempts` is set to at least 2 and the right-hand endpoint $(\nu+1)/p^e$ of the interval (†) is not the F-pure threshold, then a second check is run to verify whether the left-hand endpoint $\nu/(p^e-1)$ of this interval is the F-pure threshold.

```
i13 : f = x^6*y^4 + x^4*y^9 + (x^2 + y^3)^3;

i14 : fpt(f, Attempts => 1, DepthOfSearch => 3)

       17   7
o14 = {--, --}
       62  25

o14 : List
```

With `Attempts` set to 2, we find that the left-hand endpoint of the above interval is the desired F-pure threshold.

```
i15 : fpt(f, Attempts => 2, DepthOfSearch => 3)

      17
o15 = --
      62

o15 : QQ
```

If neither endpoint is the F-pure threshold and `Attempts` is set to more than 2, then additional checks are performed at certain numbers within the interval. First, a number in the interval is selected, according to criteria specified by the value of the option `GuessStrategy`; we refer the reader to the documentation of this option for more details. Then the function `compareFPT` is used to test that number. If that “guess” is correct, its value is returned; otherwise, the information returned by `compareFPT` is used to narrow down the interval, and this process is repeated as many times as specified by `Attempts`.

```
i16 : f = x^3*y^11*(x + y)^8*(x^2 + y^3)^8;
i17 : fpt(f, DepthOfSearch => 3, Attempts => 4)
        1   4
o17 = {--, --}
       20  75
o17 : List
i18 : fpt(f, DepthOfSearch => 3, Attempts => 6)
        13   4
o18 = {---, --}
       250  75
o18 : List
i19 : fpt(f, DepthOfSearch => 3, Attempts => 8)
       1
o19 = --
      19
o19 : QQ
```

The option `Bounds` allows the user to specify known lower and upper bounds for the F-pure threshold of f, in order to speed up computations or to refine previously obtained estimates.

```
i20 : f = x^7*y^5*(x + y)^5*(x^2 + y^3)^4;
i21 : fpt(f, DepthOfSearch => 3, Attempts => 5)
        19   1
o21 = {---, --}
       250  13
o21 : List
i22 : fpt(f, DepthOfSearch => 3, Attempts => 5, Bounds => oo)
        45   1
o22 = {---, --}
       589  13
o22 : List
```

If `guessFPT` is unsuccessful and `FinalAttempt` is set to `true`, the `fpt` function proceeds to use the convexity of the F-signature function and a secant line argument to attempt to narrow down the interval bounding the F-pure threshold. If successful, the new lower bound may coincide with the upper bound, in which case we can conclude that it is the desired F-pure threshold. If this is not the case, a check is performed to verify if the new lower bound is the F-pure threshold.

```
i23 : f = 2*x^10*y^8 + x^4*y^7 - 2*x^3*y^8;
i24 : numeric fpt(f, DepthOfSearch => 3)
o24 = {.14, .144}
o24 : List
```

With `FinalAttempt` set to `true`, we can slightly improve this estimate.

```
i25 : numeric fpt(f, DepthOfSearch => 3, FinalAttempt => true)

o25 = {.142067, .144}

o25 : List
```

The computations performed when `FinalAttempt` is set to `true` are often slow, and often fail to improve the estimate, and for this reason, this option should be used sparingly. It is typically more effective to increase the values of `Attempts` or `DepthOfSearch`, instead.

```
i26 : time numeric fpt(f, DepthOfSearch => 3, FinalAttempt => true)
     -- used 0.72874 seconds

o26 = {.142067, .144}

o26 : List

i27 : time fpt(f, DepthOfSearch => 3, Attempts => 7)
     -- used 0.452872 seconds

      1
o27 = -
      7

o27 : QQ

i28 : time fpt(f, DepthOfSearch => 4)
     -- used 0.338834 seconds

      1
o28 = -
      7

o28 : QQ
```

As seen in several examples above, when the exact answer is not found, a list containing the endpoints of an interval containing the F-pure threshold of f is returned. Whether that interval is open, closed, or a mixed interval depends on the options passed (it will be *open* whenever `Attempts` is set to at least 3); if the option `Verbose` is set to true, the precise interval will be printed.

```
i29 : fpt(f, DepthOfSearch => 3, FinalAttempt => true, Verbose => true)
Starting fpt ...
fpt is not 1 ...
Verifying whether special algorithms apply...
Special fpt algorithms were not used ...
ν has been computed: ν = frobeniusNu(3,f) = 17 ...
fpt lies in the interval [ν/(p^e-1),(ν+1)/p^e] = [17/124,18/125] ...
Starting guessFPT ...
The right-hand endpoint is not the fpt ...
The left-hand endpoint is not the fpt ...
guessFPT narrowed the interval down to (7/50,18/125) ...
Beginning F-signature computation ...
First F-signature computed: s(f,(ν-1)/p^e) = 793/15625 ...
Second F-signature computed: s(f,ν/p^e) = 342/15625 ...
Computed F-signature secant line intercept: 8009/56375 ...
F-signature intercept is an improved lower bound;
Using F-regularity to check if it is the fpt ...
The new lower bound is not the fpt ...
fpt failed to find the exact answer; try increasing the value of
    DepthOfSearch or Attempts.
fpt lies in the interval (8009/56375,18/125).

        8009   18
o29 = -----, ---
       56375  125

o29 : List
```

Finally, we point out that one can set the option `AtOrigin` from its default value of `true` to `false`, to compute the F-pure threshold globally. In other words, it computes the minimum of the F-pure threshold at all maximal ideals.

```
i30 : R = ZZ/7[x,y];
i31 : f = x*(y - 1)^2 - y*(x - 1)^3;
i32 : fpt(f)
o32 = 1
i33 : fpt(f, AtOrigin => false)
      5
o33 = -
      6
o33 : QQ
```

In this case, most features enabled by `UseSpecialAlgorithms => true` are ignored, except for the check for simple normal crossings; `FinalAttempt => true` is also ignored.

4.3. *The function* `isSimpleNormalCrossing`.

As mentioned earlier, `isSimpleNormalCrossing` verifies whether a polynomial f defines a simple normal crossing divisor. Suppose that f has factorization $f_1^{a_i} f_2^{a_2} \cdots f_n^{a_n}$. Recall that f defines a simple normal crossing divisor at a point if, locally, its factors form part of a regular system of parameters. The function `isSimpleNormalCrossing` determines whether f defines a simple normal crossing divisor at the origin by computing the Jacobian matrix of each subset of $\{f_1, \dots, f_n\}$ (evaluated at the origin), and checking that these matrices have the expected rank, and that these subsets generate ideals of the appropriate height.

```
i34 : R = ZZ/7[x,y,z];
i35 : isSimpleNormalCrossing(x^2 - y^2)
o35 = true
i36 : isSimpleNormalCrossing(x^2 - y*z)
o36 = false
```

The function `isSimpleNormalCrossing` is exposed to the user, so can be used independently of any F-pure threshold calculation. If the user sets its option `AtOrigin` to `false` (its default value is `true`), then the function checks whether f defines a simple normal crossing divisor *everywhere*, which can be much slower, since Jacobian ideals are computed.

```
i37 : R = QQ[x,y,z];
i38 : f = (y - (x - 1)^2)*y^2;
i39 : isSimpleNormalCrossing(f)
o39 = true
i40 : isSimpleNormalCrossing(f, AtOrigin => false)
o40 = false
```

5. Possible future directions.

As a natural and simple way to extend the functionality of the *FrobeniusThresholds* package, we wish to implement a method analogous to `fpt` to compute F-thresholds of polynomials with respect to arbitrary ideals. Although most of our current specialized algorithms do not extend to such generality, our “guess-and-check” methods do, and will likely give us an effective tool in computing or estimating F-thresholds.

The lack of specialized algorithms for the computation of F-thresholds, noted above, has one exception: the algorithm for computing the F-pure threshold of a homogeneous polynomial in two variables. Results of [Hernández and Teixeira 2017] show that this algorithm can be easily modified to compute F-thresholds of such polynomials with respect to ideals generated by two relatively prime homogeneous polynomials. Once this is implemented, we will be able to compute F-thresholds of polynomials in two variables homogeneous under nonstandard grading, as those agree with F-thresholds of standard-homogeneous polynomials (with respect to different ideals). For instance, the F-pure threshold of the polynomial $x^6+x^2y^2+y^3$, homogeneous under the grading $\deg x = 1$, $\deg y = 2$, can be computed as the F-threshold of the standard-homogeneous polynomial $x^6+x^2y^4+y^6$ with respect to the ideal $\langle x, y^2\rangle$.

It would be desirable to develop and implement additional algorithms for computing F-pure thresholds and F-jumping numbers for additional classes of polynomials. Along with Josep Álvarez Montaner, Jack Jeffries, and Luis Núñez-Betancourt, we are currently working on developing such algorithms. The theoretical foundation of these algorithms lies in polyhedral geometry and integer programming, making them natural candidates for implementation in *Macaulay*2.

Finally, one natural direction of development would be to incorporate the test ideals $\tau(I^t)$ when computing F-thresholds in the case where the ideal I is nonprincipal. The theoretical foundation for computing such test ideals has already largely been worked out in [Schwede and Tucker 2014], but such an update to the *FrobeniusThresholds* package would require the *TestIdeals* package to be updated first.

ACKNOWLEDGEMENTS. The authors enthusiastically thank everyone who helped complete the *FrobeniusThresholds* package: the package coauthors Juliette Bruce and Daniel Smolkin, and contributors Erin Bela, Zhibek Kadyrsizova, Moty Katzman, Sara Malec, and Marcus Robinson. We also thank the authors of the *TestIdeals* package, which, beyond the authors of the present paper and those listed above, are Alberto Boix, Drew Ellingson, Matthew Mastroeni, and Maral Mostafazadehfard.

Thanks go to the organizers of the *Macaulay*2 workshops where much of the functionality described herein was developed, hosted by Wake Forest University in 2012, the University of California, Berkeley in 2014 and 2017, Boise State University in 2015, and the University of Utah in 2016.

Finally, the authors are grateful to the University of Utah for hosting a collaborative development visit in 2018, and to the Institute of Mathematics and its Applications for its generous support for our 2019 Coding Sprint. The current version of the package was finalized during these events.

SUPPLEMENT. The online supplement contains version 2.1 of *FrobeniusThresholds*.

REFERENCES.

[Bela et al.] E. Bela, A. F. Boix, J. Bruce, D. Ellingson, D. J. Hernández, Z. Kadyrsizova, M. Katzman, S. Malec, M. Mastroeni, M. Mostafazadehfard, M. Robinson, K. Schwede, D. Smolkin, P. Teixeira, and E. E. Witt, “TestIdeals: a package for calculations of singularities in positive characteristic, version 1.01”, available at https://github.com/Macaulay2/M2/tree/ba24e16/M2/Macaulay2/packages.

[Blickle et al. 2008] M. Blickle, M. Mustaţă, and K. E. Smith, "Discreteness and rationality of F-thresholds", pp. 43–61 , 2008. MR

[Blickle et al. 2012] M. Blickle, K. Schwede, and K. Tucker, "F-signature of pairs and the asymptotic behavior of Frobenius splittings", *Adv. Math.* **231**:6 (2012), 3232–3258. MR

[Boix et al. 2019] A. F. Boix, D. J. Hernández, Z. Kadyrsizova, M. Katzman, S. Malec, M. Robinson, K. Schwede, D. Smolkin, P. Teixeira, and E. E. Witt, "The TestIdeals package for Macaulay2", *J. Softw. Algebra Geom.* **9**:2 (2019), 89–110. MR

[Bruce et al.] J. Bruce, D. J. Hernández, K. Schwede, D. Smolkin, P. Teixeira, and E. E. Witt, "FrobeniusThresholds: a package for computing F-pure thresholds and related invariants, version 2.0", available at https://github.com/Macaulay2/M2/tree/5f330a2/M2/Macaulay2/packages.

[Fedder 1983] R. Fedder, "F-purity and rational singularity", *Trans. Amer. Math. Soc.* **278**:2 (1983), 461–480. MR

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

[Hara and Watanabe 2002] N. Hara and K.-I. Watanabe, "F-regular and F-pure rings vs. log terminal and log canonical singularities", *J. Algebraic Geom.* **11**:2 (2002), 363–392. MR

[Hara and Yoshida 2003] N. Hara and K.-I. Yoshida, "A generalization of tight closure and multiplier ideals", *Trans. Amer. Math. Soc.* **355**:8 (2003), 3143–3174. MR

[Hernández 2012] D. J. Hernández, "F-purity of hypersurfaces", *Math. Res. Lett.* **19**:2 (2012), 389–401. MR

[Hernández 2014] D. J. Hernández, "F-pure thresholds of binomial hypersurfaces", *Proc. Amer. Math. Soc.* **142**:7 (2014), 2227–2242. MR

[Hernández 2015] D. J. Hernández, "F-invariants of diagonal hypersurfaces", *Proc. Amer. Math. Soc.* **143**:1 (2015), 87–104. MR

[Hernández and Teixeira 2017] D. J. Hernández and P. Teixeira, "F-threshold functions: syzygy gap fractals and the two-variable homogeneous case", *J. Symbolic Comput.* **80**:part 2 (2017), 451–483. MR

[Hernández et al. 2020] D. J. Hernández, P. Teixeira, and E. E. Witt, "Frobenius powers", *Math. Z.* **296**:1-2 (2020), 541–572. MR

[Hochster and Huneke 1990] M. Hochster and C. Huneke, "Tight closure, invariant theory, and the Briançon–Skoda theorem", *J. Amer. Math. Soc.* **3**:1 (1990), 31–116. MR

[Hochster and Roberts 1976] M. Hochster and J. L. Roberts, "The purity of the Frobenius and local cohomology", *Advances in Math.* **21**:2 (1976), 117–172. MR

[Kunz 1969] E. Kunz, "Characterizations of regular local rings of characteristic p", *Amer. J. Math.* **91** (1969), 772–784. MR

[Leykin and Tsai] A. Leykin and H. Tsai, "Dmodules: functions for computations with D-modules, version 1.4.0.1", available at https://github.com/Macaulay2/M2/tree/master/M2/Macaulay2/packages.

[Mehta and Ramanathan 1985] V. B. Mehta and A. Ramanathan, "Frobenius splitting and cohomology vanishing for Schubert varieties", *Ann. of Math.* (2) **122**:1 (1985), 27–40. MR

[Mustaţă et al. 2005] M. Mustaţă, S. Takagi, and K.-i. Watanabe, "F-thresholds and Bernstein–Sato polynomials", pp. 341–364 in *European Congress of Mathematics*, edited by A. Laptev, Eur. Math. Soc., Zürich, 2005. MR

[Schwede and Tucker 2014] K. Schwede and K. Tucker, "Test ideals of non-principal ideals: computations, jumping numbers, alterations and division theorems", *J. Math. Pures Appl.* (9) **102**:5 (2014), 891–929. MR

[Takagi and Watanabe 2004] S. Takagi and K.-i. Watanabe, "On F-pure thresholds", *J. Algebra* **282**:1 (2004), 278–297. MR

[Teitler 2015] Z. Teitler, "Software for multiplier ideals", *J. Softw. Algebra Geom.* **7** (2015), 1–8. MR

[Teitler et al.] Z. Teitler, B. Snapp, and C. Raicu, "MultiplierIdeals: A *Macaulay2* package, version 1.1", available at https://github.com/Macaulay2/M2/tree/master/M2/Macaulay2/packages.

[Zhu 2017] Z. Zhu, "Log canonical thresholds in positive characteristic", *Math. Z.* **287**:3-4 (2017), 1235–1253. MR

RECEIVED: 24 Jun 2019 REVISED: 16 Jun 2020 ACCEPTED: 11 Sep 2020

DANIEL J. HERNÁNDEZ:
hernandez@ku.edu
Department of Mathematics, University of Kansas, Lawrence, KS, United States

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

PEDRO TEIXEIRA:
pteixeir@knox.edu
Department of Mathematics, Knox College, Galesburg, IL, United States

EMILY E. WITT:
witt@ku.edu
Department of Mathematics, University of Kansas, Lawrence, KS, United States

JSAG **11** (2021), 41–51
https://doi.org/10.2140/jsag.2021.11.41

The Journal of Software for
Algebra and Geometry

Computing theta functions with Julia

DANIELE AGOSTINI AND LYNN CHUA

ABSTRACT: We present a new package `Theta.jl` for computing the Riemann theta function. It is implemented in Julia and offers accurate numerical evaluation of theta functions with characteristics and their derivatives of arbitrary order. Our package is optimized for multiple evaluations of theta functions for the same Riemann matrix, in small dimensions. As an application, we report on experimental approaches to the Schottky problem in genus 5.

1. INTRODUCTION.

The *Riemann theta function* is the holomorphic function

$$\theta : \mathbb{C}^g \times \mathbb{H}_g \to \mathbb{C}, \qquad \theta(z,\tau) = \sum_{n\in\mathbb{Z}^g} \boldsymbol{e}\left(\tfrac{1}{2}n^t\tau n + n^t z\right) \tag{1}$$

where $\boldsymbol{e}(x) = e^{2\pi i x}$ and $\mathbb{H}_g$ is the *Siegel upper-half space*, which consists of all complex symmetric $g \times g$ matrices with positive definite imaginary part. Theta functions occupy a central role throughout mathematics, appearing in fields as diverse as algebraic geometry [Birkenhake and Lange 2004; Mumford 2007], number theory [Mumford 2007; Eichler and Zagier 1985], differential geometry [Agostini et al. 2020], integrable systems [Krichever and Shiota 2013; Segur 2008], discrete mathematics [Regev and Stephens-Davidowitz 2017], cryptography [Gaudry 2007] and statistics [Agostini and Améndola 2019].

We present a new package `Theta.jl` for numerical computations of theta functions, programmed in Julia [Bezanson et al. 2017]. Our package is specialized for multiple evaluations of theta functions for the same Riemann matrix $\tau \in \mathbb{H}_g$ and different z, for small values of the genus g. Our implementation is based on the algorithm from [Deconinck et al. 2004], which we extend to support computations of theta functions with characteristics and derivatives of arbitrary order. Our package is designed as an alternative to existing packages such as `algcurves` [Deconinck et al. 2004] in Maple, `abelfunctions` [Swierczewski and Deconinck 2016] in Sage and [Frauendiener et al. 2019] in Matlab, with additional functionalities and optimizations.

As an application, we study numerical approaches to the Schottky problem in genus 5. The Schottky problem seeks to recognize Jacobians of curves amongst principally polarized abelian varieties, and has been one of the central questions in algebraic geometry since the 19th century [Grushevsky 2012]. The first nontrivial case of the Schottky problem is in genus 4, which is completely solved [Igusa 1981]. For a recent approach linking computations and tropical geometry, see [Chua et al. 2019]. In this paper, we

MSC2010: 14-04, 14H42, 14K25, 32-04, 65E99.
Keywords: theta function, numerical, Julia, abelian varieties, Riemann surfaces, Schottky problem, Jacobian.
`Theta.jl` version 1.0

describe computational approaches for studying the Schottky problem in genus 5, using our new package. In particular, we use `Theta.jl` to compute the equations in [Farkas et al. 2021; Accola 1983] which give a weak solution to the Schottky problem in genus 5. We also use our package for computations on the genus 5 Schottky problem for Jacobians with a vanishing theta null, which is described in our companion paper [Agostini and Chua 2019].

2. Theta functions.

We recall here the basic definitions about theta functions with characteristics. For a more detailed account we refer to [Birkenhake and Lange 2004; Mumford 2007; Igusa 1981]. A *characteristic* is an element $m \in (\mathbb{Z}/2\mathbb{Z})^{2g}$, which we represent as a vector $m = \left[\begin{smallmatrix}\varepsilon\\ \delta\end{smallmatrix}\right]$ where $\varepsilon, \delta \in \{0, 1\}^g$. The *Riemann theta function with characteristic m* is defined as

$$\theta[m](z,\tau) = \theta\begin{bmatrix}\varepsilon\\ \delta\end{bmatrix}(z,\tau) = \sum_{n\in\mathbb{Z}^g} \boldsymbol{e}\left(\frac{1}{2}\left(n+\frac{\varepsilon}{2}\right)^t \tau\left(n+\frac{\varepsilon}{2}\right) + \left(n+\frac{\varepsilon}{2}\right)^t\left(z+\frac{\delta}{2}\right)\right) \tag{2}$$

and it is a holomorphic function $\theta[m] : \mathbb{C}^g \times \mathbb{H}_g \to \mathbb{C}$. The Riemann theta function in (1) is a special case of (2), where the characteristic is the all-zero vector. The *sign* of a characteristic m is defined as $e(m) = (-1)^{\varepsilon^t\delta}$, and we call a characteristic *even* or *odd* if the sign is 1 or -1, respectively. As a function of z, $\theta[m](z,\tau)$ is even (respectively, odd) if and only if the characteristic m is even (respectively, odd). There are $2^{g-1}(2^g+1)$ even theta characteristics and $2^{g-1}(2^g-1)$ odd theta characteristics.

The *theta constants* are the functions on $\mathbb{H}_g$ obtained by evaluating the theta functions with characteristics at $z = 0$,

$$\theta[m](\tau) = \theta[m](0,\tau). \tag{3}$$

Theta constants corresponding to odd characteristics vanish identically.

3. Numerically approximating theta functions.

We describe in this section the algorithm that we use to compute theta functions in `Theta.jl`. In our implementation, we modify the algorithm from [Deconinck et al. 2004], generalizing it for theta functions with characteristics and derivatives of arbitrary order.

In this section, we separate $z \in \mathbb{C}^g$ and $\tau \in \mathbb{H}_g$ into real and imaginary parts, by writing $z = x + iy$, $\tau = X + iY$, where $x, y \in \mathbb{R}^g$ and X, Y are real symmetric $g \times g$ matrices. We also denote by $Y = T^tT$ the Cholesky decomposition of Y, where T is upper-triangular. For any real vector $V \in \mathbb{R}^g$, we use $[V]$ to denote the vector whose entries are the entries of V rounded to the closest integers, and we denote $[\![V]\!] = V - [V]$.

We set $v(n) = \sqrt{\pi}T(n + [\![Y^{-1}y]\!])$ and we define the lattice $\Lambda = \{v(n) \mid n \in \mathbb{Z}^g\}$, letting ρ be the length of the shortest nonzero vector in Λ. We denote by $\Gamma(z,x) = \int_x^\infty t^{z-1}e^{-t}dt$ the incomplete Gamma function.

3A. *Theta functions with characteristics.*

Deconick et al. [2004] derive numerical approximations of the theta function and its first and second derivatives. We extend their results for computing theta functions with characteristics and derivatives of arbitrary order.

We denote the N-th order derivative of the theta function along the vectors $k^{(1)}, \dots, k^{(N)}$ as

$$D(k^{(1)}, \dots, k^{(N)})\theta(z, \tau) = \sum_{i_1,\dots,i_N=1}^{g} k_{i_1}^{(1)} \cdots k_{i_N}^{(N)} \frac{\partial^N \theta(z, \tau)}{\partial z_{i_1} \cdots \partial z_{i_N}}. \tag{4}$$

By the quasiperiodicity of the theta function, it suffices to consider inputs z of the form $z = a + \tau b$, for $a, b \in [0, 1)^g$.

Theorem 3.1. *Fix $\tau \in \mathbb{H}_g$, $\epsilon > 0$. Let $k^{(1)}, \dots, k^{(N)} \in \mathbb{C}^g$ be unit vectors, and let R be the greater of $\frac{1}{2}\sqrt{g + 2N + \sqrt{g^2 + 8N}} + \frac{\rho}{2}$ and the real positive solution of R in*

$$\epsilon = (2\pi)^N \frac{g}{2}\left(\frac{2}{\rho}\right)^g \sum_{j=0}^{N} \binom{N}{j} \frac{1}{\pi^{j/2}} \|T^{-1}\|^j \sqrt{g}^{N-j} \Gamma\left(\frac{g+j}{2}, \left(R - \frac{\rho}{2}\right)^2\right). \tag{5}$$

For z of the form $z = a + \tau b$, for $a, b \in [0, 1)^g$, and $\left[\begin{smallmatrix}\varepsilon\\ \delta\end{smallmatrix}\right] \in \{0, 1\}^{2g}$, the N-th derivative

$$D(k^{(1)}, \dots, k^{(N)})\theta\begin{bmatrix}\varepsilon\\ \delta\end{bmatrix}(z, \tau)$$

of the theta function with characteristic is approximated by

$$e^{\pi y^t Y^{-1} y}(2\pi i)^N \sum_{n\in C_R} (k^{(1)}\cdot(n-\eta)) \cdots (k^{(N)}\cdot(n-\eta)) \\ \times \boldsymbol{e}\left(\frac{1}{2}(n-\eta)^t X (n-\eta) + (n-\eta)^t\left(x + \frac{\delta}{2}\right)\right) e^{-\|v(n+\frac{\varepsilon}{2})\|^2}, \tag{6}$$

with an absolute error ϵ on the product of $(2\pi i)^N$ with the sum, where $\eta = [Y^{-1}y] - \frac{\varepsilon}{2}$ and

$$C_R = \{n \in \mathbb{Z}^g \mid \pi(n-c)^t Y (n-c) < R^2, |c_j| < 1, \text{ for all } j = 1, \dots, g\}. \tag{7}$$

Proof. We first consider the case $N = 0$ without derivatives. Then the result for characteristics $\varepsilon = \delta = 0$ is proven in [Deconinck et al. 2004, Theorem 2], where they replace the deformed ellipsoid C_R in (7) with the ellipsoid

$$S_R = \{n \in \mathbb{Z}^g \mid \|v(n)\| < R\}. \tag{8}$$

For arbitrary characteristics ε, δ, we see from (2) that we can compute the corresponding theta function in a similar way to the usual theta function, by translating z to $z + \frac{\delta}{2}$, and translating the lattice points in the sum from n to $n + \frac{\varepsilon}{2}$. Note that this only changes the real part of z, while the imaginary part stays the same. Hence the approximation in Theorem 3.1 holds for theta functions with characteristics, if we take the sum over the ellipsoid

$$S_{R,\varepsilon} = \left\{n \in \mathbb{Z}^g \;\middle|\; \left\|v\left(n + \frac{\varepsilon}{2}\right)\right\| < R\right\}. \tag{9}$$

To obtain a uniform approximation for any $z \in \mathbb{C}^g$ and any characteristic, we take the union of the ellipsoids $S_{R,\varepsilon}$ from (9) as z and ε vary. Since $v\left(n + \frac{\varepsilon}{2}\right) = \sqrt{\pi}T\left(n + [\![Y^{-1}y]\!] + \frac{\varepsilon}{2}\right)$, and the entries of $[\![Y^{-1}y]\!]+\frac{\varepsilon}{2}$ have absolute value at most 1, it follows that the deformed ellipsoid C_R from (7) is the union of the ellipsoids $S_{R,\varepsilon}$.

To prove the result in the case of derivatives of order N, it will be enough to prove it for the case of zero characteristics, and then follow the same strategy as above. More precisely, we are going to prove the same statement as in Theorem 3.1, where $\varepsilon=\delta=0$ and C_R is replaced by

$$U_R=\{n\in\mathbb{Z}^g \mid \pi(n-c)^tY(n-c)<R^2,\ |c_j|<1/2,\ \text{for all } j=1,\dots,g\}. \tag{10}$$

To do so, we write the derivative as

$$D(k^{(1)},\dots,k^{(N)})\theta(z,\tau)=(2\pi i)^N\sum_{n\in\mathbb{Z}^g}(k^{(1)}\cdot n)\cdots(k^{(N)}\cdot n)\boldsymbol{e}\big(\tfrac{1}{2}n^t\tau n+n^tz\big)$$

and then the error in the approximation is

$$\epsilon=\Bigg|(2\pi i)^N\sum_{n\in\mathbb{Z}^g\setminus U_R}(k^{(1)}\cdot(n-[Y^{-1}y]))\cdots(k^{(N)}\cdot(n-[Y^{-1}y])) \\ \times\boldsymbol{e}\big(\tfrac{1}{2}(n-[Y^{-1}y])^tX(n-[Y^{-1}y])+(n-[Y^{-1}y])^tx\big)e^{-\|v(n)\|^2}\Bigg|$$

Since the $k^{(i)}$ have norm one, using the triangle inequality and the Cauchy–Schwarz inequality we can bound this by

$$\epsilon\le(2\pi)^N\sum_{n\in\mathbb{Z}^g\setminus U_R}\|n-[Y^{-1}y]\|^Ne^{-\|v(n)\|^2}=(2\pi)^N\sum_{n\in\mathbb{Z}^g\setminus U_R}\left\|\frac{1}{\sqrt{\pi}}T^{-1}v(n)-Y^{-1}y\right\|^Ne^{-\|v(n)\|^2}.$$

Using again the triangle inequality and the binomial expansion, we get to the bound

$$\epsilon\le(2\pi)^N\sum_{j=0}^N\binom{N}{j}\frac{1}{\pi^{j/2}}\|T^{-1}\|^j\|Y^{-1}y\|^{N-j}\sum_{n\in\mathbb{Z}^g\setminus U_R}\|v(n)\|^je^{-\|v(n)\|^2}.$$

We then apply [Deconinck et al. 2004, Lemma 2] to get the bound

$$\begin{aligned}\epsilon&\le(2\pi)^N\sum_{j=0}^N\binom{N}{j}\frac{1}{\pi^{j/2}}\|T^{-1}\|^j\|Y^{-1}y\|^{N-j}\frac{g}{2}\left(\frac{2}{\rho}\right)^g\Gamma\left(\frac{g+j}{2},\left(R-\frac{\rho}{2}\right)^2\right),\\&\le(2\pi)^N\frac{g}{2}\left(\frac{2}{\rho}\right)^g\sum_{j=0}^N\binom{N}{j}\frac{1}{\pi^{j/2}}\|T^{-1}\|^j\|Y^{-1}y\|^{N-j}\Gamma\left(\frac{g+j}{2},\left(R-\frac{\rho}{2}\right)^2\right).\end{aligned}$$

For inputs z of the form $z=a+\tau b$, we can write z as $z=a+(X+iY)b=(a+Xb)+iYb=x+iy$. Then $\|Y^{-1}y\|=\|b\|\le\sqrt{g}$. Substituting this into the expression for ϵ, the result follows. □

Remark 3.2. *The R appearing in Theorem 3.1 is computed numerically.*

4. Computing theta functions in Julia.

4A. *Interface.*

Our Julia package `Theta.jl` is available at the following website, which has instructions and a link to more detailed documentation:

https://github.com/chualynn/Theta.jl

We describe the basic interface of the package here. Starting with a matrix $\tau \in \mathbb{H}_g$, we first construct a `RiemannMatrix` from it. This is a type in `Theta.jl` which contains information needed to compute the theta function with input τ. As an example, we start with a genus 5 curve defined by the singular model

$$x^6y^2-4x^4y^2-2x^3y^3-2x^4y+2x^3y+4x^2y^2+3xy^3+y^4+4x^2y+2xy^2+x^2-4xy-2y^2-2x+1. \quad (11)$$

We compute the Riemann matrix τ of the curve using the package [Bruin et al. 2019] in [SageMath], and we type it as an input in Julia. We then construct a `RiemannMatrix` in `Theta.jl`, where we specify in the input the options to compute a Siegel transformation, an error of 10^{-12}, and to compute derivatives up to the fourth order.

```
julia> R = RiemannMatrix($\tau$, siegel=true, $\epsilon$=1.0e-12, nderivs=4);
```

We pick some input z and compute the theta function $\theta(z, \tau)$ as follows:

```
julia> z = [1.041+0.996im; 1.254+0.669im; 0.591+0.509im; -0.301+0.599im; 0.388+0.051im];
julia> theta(z, R)
-854877.6514446283 + 2.3935081163150463e6im
```

We can compute derivatives of theta functions by specifying the directions using the optional argument `derivs`. For instance, to compute $\frac{\partial^3\theta}{\partial z_3 \partial z_4}(z, \tau)$, we use

```
julia> theta(z, R, derivs=[[0,0,1,0,0], [0,0,0,1,0]])
1.0478325534969474e8 - 3.369999441122761e8im
```

We can also compute derivatives of theta functions with characteristics, where we specify the characteristic using the optional argument `char`.

```
julia> theta(z, R, derivs=[[1,0,0,0,0]], char=[[0,1,0,0,1],[1,1,0,0,1]])
-2.448093122926732e7 + 3.582557740667034e7im
```

4B. *Algorithms.*

We describe here some details of the algorithms and the design choices that we made in our implementation.

Choice of ellipsoid. We optimize our package for multiple evaluations of theta functions at the same Riemann matrix τ, and with different inputs z, characteristics and derivatives. We do this using the approximation in Theorem 3.1, which allows us to compute derivatives of theta functions with characteristics, for inputs z of the form $z = a + \tau b$, for $a, b \in [0, 1)^g$. In this approximation, we take the sum over the deformed ellipsoid C_R of (7), which depends only on the order N of the derivative for a fixed τ. Hence it suffices to compute the ellipsoids C_R once for each order of the derivative that we are interested in, after which we can compute theta functions for any N-th order derivatives. These ellipsoids are stored in the `RiemannMatrix` type.

Lattice reductions. In [Deconinck et al. 2004], the authors approximate the length ρ of the shortest vector of the lattice generated by T using the LLL algorithm by Lenstra, Lenstra and Lovász [Lenstra et al. 1982].

This is a reasonable choice if g is large, since the LLL algorithm gives a polynomial time approximation, but with an error that grows exponentially with g. In our implementation, since we focus on lattices with small dimensions, we compute the shortest vector exactly using the enumeration algorithm in [Schnorr and Euchner 1994]. Moreover, by computing ρ exactly, we obtain a smaller ellipsoid (7) than if we use the LLL algorithm.

If we are interested in computing the theta function for a fixed τ at many values of z, it may be more efficient if we transform τ such that the ellipsoids in (7) contain fewer lattice points. For this purpose, we use Siegel's algorithm, which iteratively finds a new matrix where the corresponding ellipsoid has a smaller eccentricity. In our implementation, we compute the Siegel transformation once for each Riemann matrix, and work with the Siegel-transformed matrix for all computations. We use the algorithm for Siegel reduction described in [Deconinck et al. 2004; Frauendiener et al. 2019], where we use the algorithm for HKZ reduction in [Zhang et al. 2012] as a subroutine.

4C. ***Comparisons with other packages.*** The main advantage of `Theta.jl` over other packages [Deconinck et al. 2004; Frauendiener et al. 2019; Swierczewski and Deconinck 2016] is that we support computations of theta functions with characteristics, as well as their derivatives, which to our knowledge is not implemented elsewhere. Moreover, we make optimizations described in Section 4B for faster computations with a fixed Riemann matrix of low genus.

We compare the performance of `Theta.jl` with the Sage package `abelfunctions` [Swierczewski and Deconinck 2016], by comparing the average time taken to compute the genus 5 FGSM relations of Section 5A, as well as to compute the Hessian matrix of Section 5C. For our experiments, we sample matrices in the Siegel upper-half space as follows. First we sample 5×5 matrices M_X, M_Y such that the entries are random floating point numbers between -1 and 1, using the random number generators in Julia and NumPy. Then we sample $\tau \in \mathbb{H}_5$ as $\tau = \frac{1}{2}(M_X + M_X^t) + M_Y^t M_Y i$. This is implemented in `Theta.jl` for general dimensions g, in the function `random_siegel(g)`. In each experiment, we randomly sample 1000 such matrices, then we compute the FGSM relations and the Hessian matrix using both packages on a standard laptop. We list in the table below the average time and standard deviation.

experiment	package	average time (s)	standard deviation (s)
FGSM	`Theta.jl`	2.5	0.6
	`abelfunctions`	114.2	290.5
Hessian	`Theta.jl`	0.7	0.2
	`abelfunctions`	20.3	58.0

One major reason for the faster runtime on `Theta.jl` is the use of the Siegel transformation on the Riemann matrix, which is not implemented in `abelfunctions`. This also leads to the higher standard deviation in the computations for the latter.

5. Applications to the Schottky problem in genus 5.

Here we describe the main application that we had in mind when designing our package: experiments around the Schottky problem in genus 5. We start with a brief account of the background of the problem; see [Grushevsky 2012] for more details.

An abelian variety is a projective variety that has the structure of an algebraic group, and it is a fundamental object in algebraic geometry. Especially important are principally polarized abelian varieties, which can all be described in terms of Riemann matrices. For every $\tau \in \mathbb{H}_g$, we define the corresponding *principally polarized abelian variety* (*ppav*) as the quotient $A_\tau = \mathbb{C}^g/\Lambda_\tau$, where $\Lambda_\tau = \mathbb{Z}^g \oplus \tau\mathbb{Z}^g$ is a sublattice of $\mathbb{C}^g$. The polarization on A_τ is given by the *theta divisor*

$$\Theta_\tau = \{z \in A_\tau \mid \theta(z,\tau) = 0\}. \tag{12}$$

Two ppavs A_τ and $A_{\tau'}$ are isomorphic if and only if the corresponding Riemann matrices are related via an action of the symplectic group $\Gamma_g = \mathrm{Sp}(2g,\mathbb{Z})$. Hence, the quotient $\mathcal{A}_g = \mathbb{H}_g/\mathrm{Sp}(2g,\mathbb{Z})$ is the *moduli space of principally polarized abelian varieties of dimension* g. This is a quasiprojective variety of dimension $\dim \mathcal{A}_g = \dim \mathbb{H}_g = \frac{1}{2}g(g+1)$, and we can look at the theta constants $\theta[m](0,\tau)$ as homogeneous coordinates on (a finite cover of) $\mathcal{A}_g$.

Perhaps the most important examples of abelian varieties are Jacobians of Riemann surfaces. The *Jacobian* of a Riemann surface C of genus g is defined as the quotient

$$J(C) = H^0(C,\omega_C)^\vee/H_1(C,\mathbb{Z}), \tag{13}$$

where the lattice $H^1(C,\mathbb{Z})$ is embedded in $H^0(C,\omega_C)^\vee$ via the integration pairing

$$H^0(C,\omega_C) \times H^1(C,\mathbb{Z}) \to \mathbb{C}, \qquad (\omega,\alpha) \mapsto \int_\alpha \omega. \tag{14}$$

The Jacobian is a principally polarized abelian variety, and the corresponding Riemann matrix $\tau \in \mathcal{A}_g$ can be obtained by computing bases of $H^0(C,\omega_C)$ and $H^1(C,\mathbb{Z})$, as well as the integration pairing. This is implemented numerically in the packages `abelfunctions` [Swierczewski and Deconinck 2016] and `RiemannSurfaces` [Bruin et al. 2019] in Sage, and `algcurves` [Deconinck et al. 2004] in Maple.

The *Schottky locus* $\mathcal{J}_g$ is the closure of the set of Jacobian varieties in $\mathcal{A}_g$, and the *Schottky problem* asks for a characterization of $\mathcal{J}_g$ inside $\mathcal{A}_g$. It is one of the most celebrated questions in algebraic geometry, dating from the 19th century. There are many possible interpretations of and solutions to the Schottky problem. Here we focus on the most classical one, which asks for equations in the theta constants $\theta[m](0,\tau)$ that vanish exactly on the Schottky locus. In this form, the Schottky problem is completely solved only in genus 4, with an explicit equation given by Schottky and Igusa [Igusa 1981]. A computational implementation and analysis of this solution was presented in [Chua et al. 2019].

The *weak Schottky problem* asks for explicit equations that characterize Jacobians up to extra irreducible components. A solution to this problem was given in genus 5 by Accola [1983], and in a recent breakthrough, by Farkas, Grushevsky and Salvati Manni in all genera [Farkas et al. 2021]. In the rest of this section, we discuss briefly these two solutions, together with related algorithms that we implemented in `Theta.jl`. We also present a computational solution of a weak Schottky problem for genus 5 Jacobians with a theta null, from our companion paper [Agostini and Chua 2019].

5A. ***Farkas, Grushevsky and Salvati Manni's solution.*** In [Farkas et al. 2021], H. Farkas, Grushevsky and Salvati Manni give a solution to the weak Schottky problem in arbitrary genus. More precisely, for every genus $g \geq 4$ they give $\binom{g-2}{2} = \frac{1}{2}(g-2)(g-3)$ explicit homogeneous equations of degree $2^{3\cdot 2^{g-4}+1}$ in the theta constants, such that their zero locus contains the Schottky locus as an irreducible component.

In the case of genus 5, this gives three equations of degree 128. We implement them in the function `fgsm()` in `Theta.jl`. Using the same example matrix τ from Section 4A, the function `fgsm(τ)` gives us the output 7.850462293418876e-16. This is expected since τ is the Jacobian of a genus 5 curve.

5B. ***Accola's equations in genus 5.*** A solution to the weak Schottky problem in genus 5 was given in [Accola 1983], in the form of eight equations of degree 32 in the theta constants whose zero locus contains the Schottky locus as an irreducible component. We implement these equations in the function `accola()` in `Theta.jl`. Again using the example τ from Section 4A, the function `accola(τ)` gives us the output 3.062334813867916e-9, which is expected since τ is in the Schottky locus.

5C. ***The Schottky problem for Jacobians with a vanishing theta null.*** We describe here a variant of the Schottky problem focusing on two-torsion points on Jacobians, referring to our companion article [Agostini and Chua 2019] for a more complete account. A *two-torsion point* on an abelian variety A_τ is a point $z \in A_\tau$ such that $2z = 0$. These can be written as

$$z = \frac{\varepsilon}{2} + \tau\frac{\delta}{2}, \qquad \text{for } m = \begin{bmatrix}\varepsilon\\ \delta\end{bmatrix} \in (\mathbb{Z}/2\mathbb{Z})^{2g}. \tag{15}$$

Hence two-torsion points correspond to characteristics, and we say that such a point is *even* or *odd* if the corresponding characteristic is. Observe that

$$\theta\Big(\frac{\varepsilon}{2} + \tau\frac{\delta}{2}, \tau\Big) = 0 \quad \text{if and only if} \quad \theta\begin{bmatrix}\varepsilon\\ \delta\end{bmatrix}(0, \tau) = 0. \tag{16}$$

Thus the two-torsion points in Θ_τ correspond to the characteristics m such that the theta constants $\theta[m](0, \tau)$ vanish. For this reason, we say that A_τ has a *vanishing theta null* if it has an even two-torsion point in the theta divisor. The abelian varieties with this property have been intensely studied and they form a divisor θ_{null} in $\mathcal{A}_g$. The Jacobians with a vanishing theta null lie in the locus $\mathcal{J}_g \cap \theta_{\text{null}}$ and they correspond to Riemann surfaces with an effective even theta characteristic. The Schottky problem in this case becomes that of recognizing $\mathcal{J}_g \cap \theta_{\text{null}}$ inside θ_{null}.

The first observation is that a vanishing theta null is automatically a singular point of the theta divisor, because the partial derivatives $\partial\theta[m]/\partial z_i$ are odd. Hence one is led to study the local structure of Θ_τ around the singular point, and the first natural invariant is the *rank of the quadric tangent cone*, which corresponds to the rank of the Hessian matrix of θ at the theta null. In particular, if a Jacobian has a vanishing theta null, then the quadric tangent cone has rank at most 3. Hence

$$\mathcal{J}_g \cap \theta_{\text{null}} \subseteq \theta^3_{\text{null}}, \tag{17}$$

where we denote by θ^3_{null} the locus of abelian varieties with a vanishing theta null whose quadric tangent cone has rank at most 3. Conversely, Grushevsky and Salvati Manni [2008] proved that this inclusion is actually an equality in genus 4, confirming a conjecture of Farkas. In the same paper, they ask whether $\mathcal{J}_g \cap \theta_{\text{null}}$ is an irreducible component of θ^3_{null} in higher genera, which would imply a solution to the weak Schottky problem for Jacobians with a vanishing theta null. The main result of our companion paper [Agostini and Chua 2019] is an affirmative answer in genus 5.

Theorem 5.1 [Agostini and Chua 2019]. *In genus* 5, *the locus* $\mathcal{J}_5 \cap \theta_{\text{null}}$ *is an irreducible component of* θ^3_{null}.

We observe that the containment $\tau \in \theta^3_{\text{null}}$ can be checked explicitly. Indeed, the condition of having an even two-torsion point in the theta divisor can be checked by evaluating the finitely many theta constants $\theta[m](0, \tau)$, and then numerically computing the rank of the Hessian matrix. We present such a computation here, which is also in [Agostini and Chua 2019]. From the example in Section 4A, we use the function `schottky_null(τ)` in `Theta.jl`. The output gives the even characteristic

$$m = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}, \tag{18}$$

where the theta constant vanishes. The output also gives the corresponding Hessian matrix

$$\begin{pmatrix} -2.79665+5.29764i & -9.57825-9.04671i & 7.36305+2.28697i & 7.58338+5.34729i & 6.15667-1.90199i \\ -9.57825-9.04671i & 18.9738+8.34582i & -23.1027-3.10545i & -9.31944-0.822821i & 0.524289-3.64991i \\ 7.36305+2.28697i & -23.1027-3.10545i & 16.8441-1.15986i & 13.9363-4.56541i & -3.32248+4.10698i \\ 7.58338+5.34729i & -9.31944-0.822821i & 13.9363-4.56541i & 2.89309+1.21773i & 3.86617-0.546202i \\ 6.15667-1.90199i & 0.524289-3.64991i & -3.32248+4.10698i & 3.86617-0.546202i & -12.9726-1.928i \end{pmatrix}.$$

The Hessian has the eigenvalues

$$47.946229109152995+9.491932144035298i,$$
$$-15.491689246713147+3.3401255907497958i,$$
$$-9.512858919129267-1.0587349322052013i,$$
$$-2.7271385943272036\times10^{-15}-1.1117459994936022i\times10^{-14},$$
$$-5.698014266322794\times10^{-15}+6.342925068807627i\times10^{-15},$$

so it has numerical rank 3 as expected.

ACKNOWLEDGMENTS. We are grateful to Bernd Sturmfels for the suggestion to study the Schottky problem in genus 5, and for his continuous encouragement. We thank Paul Breiding, Gavril Farkas, Jörg Frauendiener, Sam Grushevsky, Christian Klein, Riccardo Salvati Manni, Andrey Soldatenkov, Sasha Timme and Sandro Verra for useful comments and discussions. We thank the anonymous referees for their suggestions, which improved the quality of the paper and of the code. This project was initated at the Max Planck Institute for Mathematics in the Sciences in Leipzig, which both authors would like to thank for the hospitality and support at various stages of this work.

SUPPLEMENT. The online supplement contains version 1.0 of `Theta.jl`.

REFERENCES.

[Accola 1983] R. D. M. Accola, "On defining equations for the Jacobian locus in genus five", *Proc. Amer. Math. Soc.* **89**:3 (1983), 445–448. MR

[Agostini and Améndola 2019] D. Agostini and C. Améndola, "Discrete Gaussian distributions via theta functions", *SIAM J. Appl. Algebra Geom.* **3**:1 (2019), 1–30. MR

[Agostini and Chua 2019] D. Agostini and L. Chua, "On the Schottky problem for genus five Jacobians with a vanishing theta null", 2019. To appear in *Ann. Scuola Norm. Sci.* arXiv

[Agostini et al. 2020] D. Agostini, T. Çelik, J. Struwe, and B. Sturmfels, "Theta surfaces", *Vietnam J. Math.* (2020).

[Bezanson et al. 2017] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: a fresh approach to numerical computing", *SIAM Rev.* **59**:1 (2017), 65–98. MR

[Birkenhake and Lange 2004] C. Birkenhake and H. Lange, *Complex abelian varieties*, 2nd ed., Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences] **302**, Springer, 2004. MR

[Bruin et al. 2019] N. Bruin, J. Sijsling, and A. Zotine, "Riemann matrices and endomorphism rings of algebraic Riemann surfaces", 2019, http://doc.sagemath.org/html/en/reference/curves/sage/schemes/riemann_surfaces/riemann_surface.html.

[Chua et al. 2019] L. Chua, M. Kummer, and B. Sturmfels, "Schottky algorithms: classical meets tropical", *Math. Comp.* **88**:319 (2019), 2541–2558. MR

[Deconinck et al. 2004] B. Deconinck, M. Heil, A. Bobenko, M. van Hoeij, and M. Schmies, "Computing Riemann theta functions", *Math. Comp.* **73**:247 (2004), 1417–1442. MR

[Eichler and Zagier 1985] M. Eichler and D. Zagier, *The theory of Jacobi forms*, Progress in Mathematics **55**, Birkhäuser, Boston, 1985. MR

[Farkas et al. 2021] H. M. Farkas, S. Grushevsky, and R. S. Manni, "An explicit solution to the weak Schottky problem", *Algebr. Geom.* **8**:3 (2021), 358–373. MR

[Frauendiener et al. 2019] J. Frauendiener, C. Jaber, and C. Klein, "Efficient computation of multidimensional theta functions", *J. Geom. Phys.* **141** (2019), 147–158. MR

[Gaudry 2007] P. Gaudry, "Fast genus 2 arithmetic based on theta functions", *J. Math. Cryptol.* **1**:3 (2007), 243–265. MR

[Grushevsky 2012] S. Grushevsky, "The Schottky problem", pp. 129–164 in *Current developments in algebraic geometry*, edited by M. M. Lucia Caporaso, James McKernan and M. Popa, Math. Sci. Res. Inst. Publ. **59**, Cambridge Univ. Press, 2012. MR

[Grushevsky and Manni 2008] S. Grushevsky and R. S. Manni, "Jacobians with a vanishing theta-null in genus 4", *Israel J. Math.* **164** (2008), 303–315. MR

[Igusa 1981] J.-i. Igusa, "On the irreducibility of Schottky's divisor", *J. Fac. Sci. Univ. Tokyo Sect. IA Math.* **28**:3 (1981), 531–545. MR

[Krichever and Shiota 2013] I. Krichever and T. Shiota, "Soliton equations and the Riemann–Schottky problem", pp. 205–258 in *Handbook of moduli*, *II*, edited by G. Farkas and I. Morrison, Adv. Lect. Math. (ALM) **25**, International Press, Somerville, MA, 2013. MR

[Lenstra et al. 1982] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, "Factoring polynomials with rational coefficients", *Math. Ann.* **261**:4 (1982), 515–534. MR

[Mumford 2007] D. Mumford, *Tata lectures on theta, I*, Birkhäuser, Boston, 2007. MR

[Regev and Stephens-Davidowitz 2017] O. Regev and N. Stephens-Davidowitz, "An inequality for Gaussians on lattices", *SIAM J. Discrete Math.* **31**:2 (2017), 749–757. MR

[SageMath] The Sage Developers, "SageMath, the Sage Mathematics Software System (Version 8.6)", https://www.sagemath.org.

[Schnorr and Euchner 1994] C.-P. Schnorr and M. Euchner, "Lattice basis reduction: improved practical algorithms and solving subset sum problems", *Math. Programming* **66**:2, Ser. A (1994), 181–199. MR

[Segur 2008] H. Segur, "Integrable models of waves in shallow water", pp. 345–371 in *Probability, geometry and integrable systems*, edited by M. Pinsky and B. Birnir, Math. Sci. Res. Inst. Publ. **55**, Cambridge Univ. Press, 2008. MR

[Swierczewski and Deconinck 2016] C. Swierczewski and B. Deconinck, "Computing Riemann theta functions in Sage with applications", *Math. Comput. Simulation* **127** (2016), 263–272. MR

[Zhang et al. 2012] W. Zhang, S. Qiao, and Y. Wei, "HKZ and Minkowski reduction algorithms for lattice-reduction-aided MIMO detection", *IEEE Trans. Signal Process.* **60**:11 (2012), 5963–5976. MR

RECEIVED: 21 Feb 2020 REVISED: 21 Oct 2020 ACCEPTED: 9 Nov 2020

DANIELE AGOSTINI:
daniele.agostini@mis.mpg.de
Max-Planck-Institut für Mathematik in den Naturwissenschaften, Leipzig, Germany

LYNN CHUA:
lchua@caltech.edu
Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA, United States

JSAG **11** (2021), 53–59
https://doi.org/10.2140/jsag.2021.11.53

The Journal of Software for
Algebra and Geometry

Decomposable sparse polynomial systems

TAYLOR BRYSIEWICZ, JOSE ISRAEL RODRIGUEZ, FRANK SOTTILE AND THOMAS YAHL

ABSTRACT: The *Macaulay*2 package `DecomposableSparseSystems` implements methods for studying and numerically solving decomposable sparse polynomial systems. We describe the structure of decomposable sparse systems and explain how the methods in this package may be used to exploit this structure, with examples.

1. INTRODUCTION. Améndola and Rodriguez [2016] gave numerical methods to efficiently solve systems of sparse polynomial equations in a family, when that family is decomposable (Definition 1). A consequence of Esterov's study of Galois groups of systems of sparse polynomial equations [2019] is that for sparse systems, recognizing and computing a decomposition is algorithmic. Solving a decomposable sparse system reduces to solving two smaller sparse polynomial systems. In [Brysiewicz et al. 2021], we presented algorithms to detect and compute such decompositions, and a recursive algorithm exploiting decomposability for solving a decomposable sparse polynomial system using numerical homotopy continuation.

The *Macaulay*2 package `DecomposableSparseSystems` implements methods for decomposable sparse polynomial systems. These include methods to detect decomposability, to compute a decomposition, and a recursive procedure to compute numerical solutions to a given decomposable sparse system. Detection and computation of decompositions uses integer linear algebra, including computing a Smith normal form and the corresponding monomial changes of variables. Numerical homotopy continuation is used to compute solutions. When no further decompositions are possible, the algorithm solves multivariate systems using numerical software chosen by the user (default: `PHCpack` [Verschelde 1999]), and solves univariate polynomials using companion matrices.

Using the methods in `DecomposableSparseSystems` to solve a decomposable system allows for quicker solving and more accurate solution counts than calling other solvers. One reason is that after each decomposition, the child systems always involve either fewer variables, or polynomials of smaller degree. The cost of the methods in `DecomposableSparseSystems` is low as they rely only on linear algebra and numerical homotopy algorithms.

Research of Sottile and Yahl was supported by grant 636314 from the Simons Foundation.
MSC2020: 14M25, 65H10, 65H20.
Keywords: sparse, decomposable, polynomial, solving.
`DecomposableSparseSystems` version 1.0.1

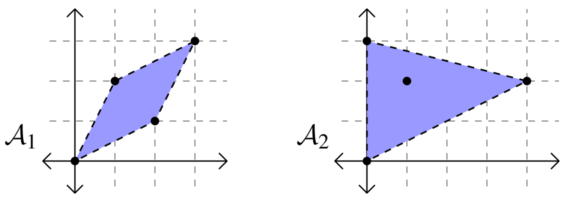

Figure 1. A pair of supports.

2. DECOMPOSABLE SPARSE POLYNOMIAL SYSTEMS. A *branched cover* is a dominant map $\pi : X \to Y$ of irreducible varieties X and Y of the same dimension. There is a number d (the *degree* of π) and an open dense subset V of Y such that $\pi^{-1}(v)$ consists of d points for $v \in V$. When $d > 1$, the branched cover is *nontrivial*.

Definition 1. A branched cover $\pi : X \to Y$ is *decomposable* if it is a composition of nontrivial branched covers. That is, if there is a dense open subset $U \subset Y$ and a variety Z such that $\pi^{-1}(U) \to U$ factors as

$$\pi^{-1}(U) \to Z \to U,$$

with each map a nontrivial branched cover.

In general it is not easy to determine if a branched cover is decomposable, or even to compute a decomposition for a decomposable branched cover. (See [Améndola et al. 2016, Section 5.4] and [Brysiewicz et al. 2021, Section 1.2] for examples and a discussion.)

An integer vector $\alpha \in \mathbb{Z}^n$ is the exponent of a (Laurent) monomial $x^\alpha := x_1^{\alpha_1} \cdots x_n^{\alpha_n}$. A (complex) linear combination of monomials $\sum c_\alpha x^\alpha$ is a (Laurent) polynomial. Monomials are multiplicative maps $(\mathbb{C}^\times)^n \to \mathbb{C}^\times$ and polynomials are maps $(\mathbb{C}^\times)^n \to \mathbb{C}$. For a finite set $\mathcal{A} \subset \mathbb{Z}^n$ of exponents, the set of all polynomials whose monomials have exponents contained in $\mathcal{A}$ (have *support* $\mathcal{A}$) forms the vector space $\mathbb{C}^{\mathcal{A}}$. Given a list $\mathcal{A}_\bullet = (\mathcal{A}_1, \dots, \mathcal{A}_n)$ of finite subsets of $\mathbb{Z}^n$, write $\mathbb{C}^{\mathcal{A}_\bullet}$ for the vector space $\mathbb{C}^{\mathcal{A}_1} \oplus \cdots \oplus \mathbb{C}^{\mathcal{A}_n}$ of lists $F = (f_1, \dots, f_n)$ of polynomials with f_i having support $\mathcal{A}_i$. Such a list $F \in \mathbb{C}^{\mathcal{A}_\bullet}$ is a function $F : (\mathbb{C}^\times)^n \to \mathbb{C}^n$, and $F = 0$ is a system of sparse polynomials with support $\mathcal{A}_\bullet$ whose solutions are $F^{-1}(0)$.

Example 2. Let $\mathcal{A}_\bullet = (\mathcal{A}_1, \mathcal{A}_2)$ be the pair of supports in $\mathbb{Z}^2$ illustrated in Figure 1. The corresponding vector spaces of polynomials are

$$\mathbb{C}^{\mathcal{A}_1} = \{a_1 + a_2xy^2 + a_3x^2y + a_4x^3y^3 \mid a_i \in \mathbb{C}\},$$
$$\mathbb{C}^{\mathcal{A}_2} = \{b_1 + b_2y^3 + b_3xy^2 + b_4x^4y^2 \mid b_j \in \mathbb{C}\},$$

and $\mathbb{C}^{\mathcal{A}_\bullet}$ is the space of systems of the form

$$F = \begin{pmatrix} a_1 + a_2xy^2 + a_3x^2y + a_4x^3y^3 \\ b_1 + b_2y^3 + b_3xy^2 + b_4x^4y^2 \end{pmatrix}, \quad a_i, b_j \in \mathbb{C}.$$

In `DecomposableSparseSystems`, the family $\mathbb{C}^{\mathcal{A}_\bullet}$ is encoded by a list of matrices whose column vectors are the exponent vectors of each polynomial. Given a system $F \in \mathbb{C}^{\mathcal{A}_\bullet}$, these data can be extracted from a given system via the *Macaulay*2 function `exponents`.

The Bernstein–Kushnirenko theorem [Bernstein 1975] provides a sharp upper bound on the number of solutions to a system of sparse polynomials. Denote the convex hull of a set $\mathcal{A} \subseteq \mathbb{R}^n$ by $\operatorname{conv}(\mathcal{A})$. Given a list of supports $\mathcal{A}_\bullet = (\mathcal{A}_1, \dots, \mathcal{A}_n)$, let $\operatorname{MV}(\mathcal{A}_\bullet)$ be the mixed volume (see [Ewald 1996, Section IV.3]) of the list $(\operatorname{conv}(\mathcal{A}_1), \dots, \operatorname{conv}(\mathcal{A}_n))$.

Theorem 3 (Bernstein–Kushnirenko). *Let $\mathcal{A}_\bullet$ be a list of n finite subsets of $\mathbb{Z}^n$. For $F \in \mathbb{C}^{\mathcal{A}_\bullet}$, the number of isolated solutions in $(\mathbb{C}^\times)^n$ to the system $F = 0$ is bounded above by $\operatorname{MV}(\mathcal{A}_\bullet)$ and this bound is achieved for F lying in a dense, open subset of $\mathbb{C}^{\mathcal{A}_\bullet}$.*

Define $X_{\mathcal{A}_\bullet} \subset (\mathbb{C}^\times)^n \times \mathbb{C}^{\mathcal{A}_\bullet}$ to be the set of pairs (x, F) such that $F(x) = 0$. For $F \in \mathbb{C}^{\mathcal{A}_\bullet}$, the fiber $\pi^{-1}(F)$ of the map $\pi : X_{\mathcal{A}_\bullet} \to \mathbb{C}^{\mathcal{A}_\bullet}$ consists of solutions to $F = 0$. By the Bernstein–Kushnirenko theorem, the map π has degree $\operatorname{MV}(\mathcal{A}_\bullet)$. When $\operatorname{MV}(\mathcal{A}_\bullet) \geq 1$, it is a branched cover. When the branched cover $\pi : X_{\mathcal{A}_\bullet} \to \mathbb{C}^{\mathcal{A}_\bullet}$ is decomposable, we say the sparse system $F \in \mathbb{C}^{\mathcal{A}_\bullet}$ is decomposable. Decomposability depends only on the support $\mathcal{A}_\bullet$ of a system.

There are two transparent ways for a sparse system to decompose.

Lacunary. A system $F \in \mathbb{C}^{\mathcal{A}_\bullet}$ is *lacunary* if there is a surjective monomial map $\Phi : (\mathbb{C}^\times)^n \to (\mathbb{C}^\times)^n$ such that $F = G \circ \Phi$ for some sparse polynomial system G. We require that Φ be nontrivial in that its kernel is not the identity subgroup. A lacunary system $F = G \circ \Phi = 0$ can be solved by computing solutions, $z_1, \dots, z_d$, to the system $G = 0$ and then computing the fibers $\Phi^{-1}(z_1), \dots, \Phi^{-1}(z_d)$. In appropriate coordinates, Φ is diagonal, and $\Phi^{-1}(z)$ is obtained by extracting roots of the components of z.

Example 4. Consider the following system with support from Example 2:

$$F(x, y) = \begin{pmatrix} 1 - 2xy^2 + 3x^2y - 4x^3y^3 \\ 2 + 3y^3 + 5xy^2 + 7x^4y^2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

It is lacunary as it is the composition of the following maps:

$$G(s, t) = \begin{pmatrix} 1 - 2st^2 + 3st - 4s^2t^3 \\ 2 + 3st^3 + 5st^2 + 7s^2t^2 \end{pmatrix}, \qquad \Phi(x, y) = (x^3, x^{-1}y).$$

This can be detected via the methods in `DecomposableSparseSystems`:

```
i1 : R = CC[x,y];

i2 : F = {1-2*x*y^2+3*x^2*y-4*x^3*y^3,2+3*y^3+5*x*y^2+7*x^4*y^2};

i3 : isLacunary F
o3 = true
```

The method `isLacunary` extracts the set of supports of the system and computes the Smith normal form of a matrix associated to these supports to determine whether the system is lacunary.

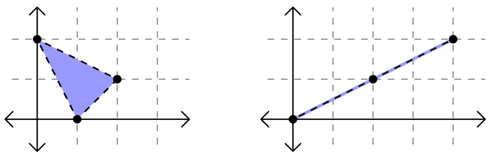

Figure 2. Triangular support.

Triangular. A system $F \in \mathbb{C}^{\mathcal{A}_\bullet}$ is *triangular* if there exists $k < n$ so that after a monomial change of variables, the system F has the form

$$F = (F_1(x_1, \dots, x_k), \dots, F_k(x_1, \dots, x_k), F_{k+1}(x_1, \dots, x_n), \dots, F_n(x_1, \dots, x_n)).$$

Solutions to triangular systems are computed by first computing the solutions $z_1, \dots, z_d$ of the square subsystem $(F_1, \dots, F_k) = 0$. A residual system is obtained by substituting z_1 into the original system for the first k variables, $F_2(z_1, x_{k+1}, \dots, x_n)$. Solutions to the original system are obtained by solving the residual system and then applying a homotopy algorithm as described in [Brysiewicz et al. 2021].

Example 5. Consider the system

$$F(x, y, z) = \begin{pmatrix} y^2 - 2x + 3x^2y \\ 2 + 3x^2y + 5x^4y^2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Figure 2 shows the supports. This system is triangular as the second polynomial is quadratic in the monomial x^2y. The method `isTriangular` detects this subsystem.

```
i4 : F = {y^2-2*x+3*x^2*y,2+3*x^2*y+5*x^4*y^2};

i5 : isTriangular F
o5 = true
```

A consequence of Esterov's study of Galois groups of sparse polynomial systems [2019] and Pirola and Schlesinger's result that a branched cover is decomposable if and only if its Galois group is imprimitive [2005] is that a sparse polynomial system is decomposable if and only if it is either lacunary or triangular. In each case, the solutions to the original system are computed via solutions to simpler systems. The methods in `DecomposableSparseSystems` iteratively decompose these sparse polynomial systems to efficiently solve them.

3. Main method: solveDecomposableSystem.

The main method implemented in the package `DecomposableSparseSystems` is named `solveDecomposableSystem` and this implements Algorithm 9 in [Brysiewicz et al. 2021]. It takes as input a sparse polynomial system $F \in \mathbb{C}^{\mathcal{A}_\bullet}$ and outputs all solutions to $F = 0$ in the algebraic torus. It recursively checks whether or not the input sparse polynomial system is decomposable, computes the decomposition, and then calls itself on each portion of the decomposition. When the input is not decomposable it solves multivariate polynomial systems with the numerical solver given by the option `Software` (default: `PHCpack`) and it solves univariate polynomial systems using companion matrices. For complete details, see [Brysiewicz et al. 2021, Section 3.1].

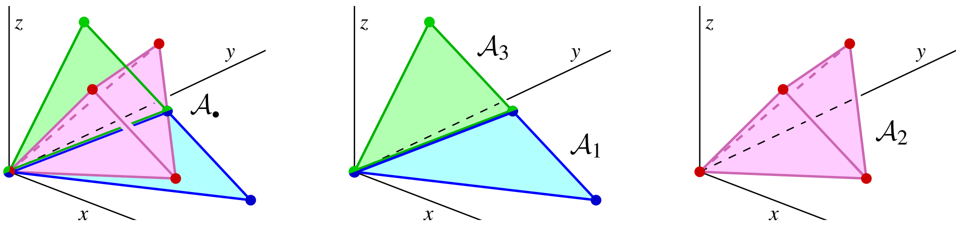

Figure 3. Support of F.

3.1. ***Using the main method.*** Consider the system

$$F = \begin{pmatrix} 2+xyz-x^2y \\ 4-y^2z+2xz^2-3x^2z \\ 1-yz^2-3xyz \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

This system is supported on the triple $\mathcal{A}_\bullet = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ shown in Figure 3.

The method `isDecomposable` determines that this system is decomposable. In particular, it is triangular with a subsystem indexed by the first and third polynomials. This can be observed in the figure as the span of the supports $\mathcal{A}_1$ and $\mathcal{A}_3$ are coplanar. It is also lacunary, as the exponent vectors lie in the sublattice of $\mathbb{Z}^3$ of index 3 generated by the columns of

$$\begin{pmatrix} 1 & 1 & 2 \\ 1 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}.$$

The solutions to $F = 0$ are found via the main method, `solveDecomposableSystem`.

```
i6 : R = CC[x,y,z];

i7 : F = {2+x*y*z-x^2*y,4-y^2*z+2*x*z^2-3*x^2*z,1-y*z^2-3*x*y*z};

  -- True if and only if the sparse system F is decomposable.

i8 : isDecomposable F
o8 : true

  -- A list of numerical solutions to F=0.

i9 : S = solveDecomposableSystem F;

  -- Evaluates F at the first numerical solution.

i10 : F/(f-> sub(f, matrix {S_0}))
o10 = {1.77636e-15, 4.44089e-16+1.4623e-16*ii, 4.66294e-15}
```

Our main method also accepts a two-argument input `(A,C)` where `A` is a list of matrices whose columns support a system of (Laurent) polynomial equations, and `C` is a list, whose i-th entry is the list of coefficients for the i-th polynomial equation. We demonstrate some of the other types of inputs here, and leave details to the documentation.

```
i11 : (A,C) = (F/exponents/matrix/transpose,
               F/coefficients/last/entries/flatten);

i12 : S = solveDecomposableSystem (A,C);
```

```
  -- Expected timing for solving a specific system.

i13 : benchmark "solveDecomposableSystem(A,C)";
o13 = .0605920270512821

  -- Expected timing for solving a random system with support A.

i14 : benchmark "solveDecomposableSystem(A, )";
o14 = .0558867168108108
```

3.2. ***Options for the main method.*** Numerical in nature, the function solveDecomposableSystem features a variety of options for the user. The option Software (default: PHCpack) dictates which numerical solver is used to solve multivariate sparse systems which are not decomposable. The method solveDecomposableSystem removes solutions having any coordinate which is numerically zero up to Tolerance (default: 10^{-5}) throughout the computation. Having this tolerance is necessary, as our methods are for Laurent polynomials with solutions in the complex torus $(\mathbb{C}^\times)^n$, while the solvers we call may return solutions in $\mathbb{C}^n$ that are not in the torus.

Setting the option Verify (default: 0) to have the value 1 significantly increases the probability that solveDecomposableSystem computes the correct number of solutions. It does this by checking that the algorithm specified by the Software option computes $\mathrm{MV}(\mathcal{A}_\bullet)$ solutions to any system F with support $\mathcal{A}_\bullet$, where $\mathrm{MV}(\mathcal{A}_\bullet)$ is probabilistically determined using mixedVolume in the package Polyhedra [Birkner 2009]. If the mixed volume according to Polyhedra and the number of solutions do not agree, then the missing solutions are searched for using techniques related to those in MonodromySolver [Duff et al. 2019]. Lastly, we allow the user to compute the solutions to F by first solving an internally generated random instance and then using that in a parameter homotopy [Li et al. 1989] to solve F by setting Strategy to FromGeneric. We conclude by using the options Verify and Strategy on an example with 6000 solutions.

```
i15 : A = <<< omitted, see example from Section 4 in [4] with
              i_1=(2,0,0,2,0)
              i_2=(4,4,2,2,2)
              j_1=(0,2,0,1,3)
              j_2=(0,0,1,0,2)
              >>;

  -- A has five supports, print the first one

i16 : print(length A,  A_0)
(5, | 0 2 4 4 6 |)
    | 0 0 0 4 4 |
    | 0 0 0 2 2 |
    | 0 2 4 2 4 |
    | 0 0 0 2 2 |

i17 : elapsedTime (F,S) = solveDecomposableSystem(A,,Verify=>1);
     -- 8.93938 seconds elapsed

i18 : elapsedTime S' = solveDecomposableSystem(F,Strategy=>FromGeneric);
     -- 29.0802 seconds elapsed

i19 : print(#S,#S')
o19 = (6000, 6000)
```

SUPPLEMENT. The online supplement contains version 1.0.1 of `DecomposableSparseSystems`.

REFERENCES.

[Améndola et al. 2016] C. Améndola, J. Lindberg, and J. I. Rodriguez, "Solving Parameterized Polynomial Systems with Decomposable Projections", 2016. arXiv

[Bernstein 1975] D. N. Bernstein, "The number of roots of a system of equations", *Funkcional. Anal. i Priložen.* **9**:3 (1975), 1–4. In Russian; translated in *Functional Analysis and Its Applications*, **9**:3, (1975), 183–185. MR

[Birkner 2009] R. Birkner, "Polyhedra: a package for computations with convex polyhedral objects", *J. Softw. Algebra Geom.* **1** (2009), 11–15. MR Zbl

[Brysiewicz et al. 2021] T. Brysiewicz, J. I. Rodriguez, F. Sottile, and T. Yahl, "Solving decomposable sparse systems", *Numerical Algorithms* (2021).

[Duff et al. 2019] T. Duff, C. Hill, A. Jensen, K. Lee, A. Leykin, and J. Sommars, "Solving polynomial systems via homotopy continuation and monodromy", *IMA J. Numer. Anal.* **39**:3 (2019), 1421–1446. MR Zbl

[Esterov 2019] A. Esterov, "Galois theory for general systems of polynomial equations", *Compos. Math.* **155**:2 (2019), 229–245. MR Zbl

[Ewald 1996] G. Ewald, *Combinatorial convexity and algebraic geometry*, Graduate Texts in Mathematics **168**, Springer, 1996. MR Zbl

[Li et al. 1989] T. Y. Li, T. Sauer, and J. A. Yorke, "The cheater's homotopy: an efficient procedure for solving systems of polynomial equations", *SIAM J. Numer. Anal.* **26**:5 (1989), 1241–1251. MR Zbl

[Pirola and Schlesinger 2005] G. P. Pirola and E. Schlesinger, "Monodromy of projective curves", *J. Algebraic Geom.* **14**:4 (2005), 623–642. MR Zbl

[Verschelde 1999] J. Verschelde, "Algorithm 795: PHCpack: A General-Purpose Solver for Polynomial Systems by Homotopy Continuation", *ACM Trans. Math. Softw.* **25**:2 (1999), 251–276. Version containing reference manual available at http://homepages.math.uic.edu/ jan/PHCpack/phcpack.html. Zbl

RECEIVED: 8 Jun 2020 REVISED: 24 Jan 2021 ACCEPTED: 25 Mar 2021

TAYLOR BRYSIEWICZ:
taylorbrysiewicz@gmail.com
Max-Planck Institut fur Mathematik, Leipzig, Germany

JOSE ISRAEL RODRIGUEZ:
jose@math.wisc.edu
Department of Mathematics, University of Wisconsin, Madison, WI, United States

FRANK SOTTILE:
sottile@math.tamu.edu
Department of Mathematics, Texas A&M University, College Station, TX, United States

THOMAS YAHL:
thomasjyahl@math.tamu.edu
Department of Mathematics, Texas A&M University, College Station, TX, United States

JSAG **11** (2021), 61–69
https://doi.org/10.2140/jsag.2021.11.61

The Journal of Software for
Algebra and Geometry

A package for computations with sparse resultants

GIOVANNI STAGLIANÒ

ABSTRACT: We introduce the *Macaulay*2 package *SparseResultants*, which provides general tools for computing sparse resultants, sparse discriminants, and hyperdeterminants. We give some background on the theory and briefly show how the package works.

INTRODUCTION. The classical Macaulay resultant [1903] (also called the dense resultant) of a system of $n+1$ polynomial equations in n variables characterizes the solvability of the system, and therefore it is a fundamental tool in computer algebra. However, it is a large polynomial, since it depends on all coefficients of the equations. If we restrict attention to *sparse* polynomial equations, that is, to polynomials which involve only monomials lying in a small set, then we can replace the *dense* resultant with the *sparse* resultant.

The sparse resultant generalizes not only the dense resultant but, for specific choices of the set of monomials, we can obtain other types of classical resultants, such as for instance the *Dixon resultant* [1909] and the *hyperdeterminant* [Cayley 1845; Gelfand et al. 1992]. In the last decades, sparse resultants have received a lot of interest, both from a theoretical point of view (see, e.g., [Gelfand et al. 2008; Sturmfels 1994; Cattani et al. 1998; D'Andrea and Sombra 2015]) and from more computational and applied aspects (see, e.g., [Emiris and Mourrain 1999; Canny and Emiris 2000; Sturmfels 2002; D'Andrea 2002; Jeronimo et al. 2004; Cox et al. 2005; Jeronimo et al. 2009]).

Using the computer program Macaulay2, dense resultants can be calculated using the package *Resultants* [Staglianò 2018], while sparse resultants can be calculated using the new package *SparseResultants*. We point out that in the latter most of the algorithms implemented are based on elimination via Gröbner basis methods. The main defect of this approach is that even when the input polynomials have numerical coefficients, in the calculation all the coefficients are replaced by variables. However, this approach suffices for a number of applications, as we try to show in the following.

This short paper is organized as follows. In Section 1, we review the general theory of sparse resultants (Sections 1A and 1B) and related topics such as the sparse discriminants (Section 1C) and the hyperdeterminants (Section 1D). We focus on the computational aspects used in the package *SparseResultants*. In Section 2, we illustrate how this package works with the help of some examples.

MSC2020: 13P15, 68W30.
Keywords: sparse resultant, sparse discriminant, hyperdeterminant.
`SparseResultants` version 1.1

1. An overview of sparse elimination.

In this section we give some background on the theory of sparse resultants, sparse discriminants, and hyperdeterminants. For details and proofs we refer mainly to [Gelfand et al. 2008, Chapters 8, 9, 13, and 14] and [Cox et al. 2005, Chapter 7]; other references are [Sturmfels 1993; Ottaviani 2013].

1A. ***Sparse mixed resultant.*** Let $R = \mathbb{C}[x_1^{\pm 1}, \dots, x_n^{\pm 1}]$ be the ring of complex Laurent polynomials in n variables. The set of monomials in R is identified with $\mathbb{Z}^n$ by associating to $x^\omega = x_1^{\omega_1} \cdots x_n^{\omega_n} \in R$ the exponent vector $\omega = (\omega_1, \dots, \omega_n) \in \mathbb{Z}^n$. If $\mathcal{A}$ is a finite subset of $\mathbb{Z}^n$, we denote by $\mathbb{C}^{\mathcal{A}}$ the space of polynomials in R involving only monomials from $\mathcal{A}$, that is, of polynomials of the form $\sum_{\omega\in\mathcal{A}} a_\omega x^\omega$.

Let $\mathcal{A}_0, \dots, \mathcal{A}_n$ be $n+1$ finite subsets of $\mathbb{Z}^n$ satisfying the following conditions:

(1) Each $\mathcal{A}_i$ generates $\mathbb{R}^n$ as an affine space.

(2) The union of the sets $\mathcal{A}_i$ generates $\mathbb{Z}^n$ as a $\mathbb{Z}$-module.

Let $\mathcal{Z}_{\mathcal{A}_0,\dots,\mathcal{A}_n} \subset \prod_{i=0}^n \mathbb{C}^{\mathcal{A}_i}$ be the Zariski closure in the product $\prod_{i=0}^n \mathbb{C}^{\mathcal{A}_i}$ of the set

$$\left\{ (f_0, \dots, f_n) \in \prod_{i=0}^{n} \mathbb{C}^{\mathcal{A}_i} : \text{there exists } x \in (\mathbb{C}^*)^n \text{ such that } f_0(x) = \cdots = f_n(x) = 0 \right\}, \tag{1-1}$$

where $\mathbb{C}^* = \mathbb{C} \setminus \{0\}$ and $f_i(x) = \sum_{\omega\in\mathcal{A}_i} a_{i,\omega} x^\omega$, for $i = 0, \dots, n$.

Proposition-Definition 1.1 [Gelfand et al. 2008, Chapter 8, §1]. Under the above assumptions, the variety $\mathcal{Z}_{\mathcal{A}_0,\dots,\mathcal{A}_n}$ is an irreducible hypersurface in $\prod_{i=0}^n \mathbb{C}^{\mathcal{A}_i}$ that can be defined by an integral irreducible polynomial $\mathrm{Res}_{\mathcal{A}_0,\dots,\mathcal{A}_n} \in \mathbb{Z}[(a_{i,\omega}), i = 0, \dots, n]$ in the coefficients $a_{i,\omega}$ of f_i, for $i = 0, \dots, n$. Such a polynomial $\mathrm{Res}_{\mathcal{A}_0,\dots,\mathcal{A}_n}$ is unique up to sign and is called the $(\mathcal{A}_0, \dots, \mathcal{A}_n)$-resultant (also known as the *sparse* (*mixed*) *resultant*).

The polynomial $\mathrm{Res}_{\mathcal{A}_0,\dots,\mathcal{A}_n}$ is homogeneous with respect to each group of variables $(a_{i,\omega})$, for $i = 0, \dots, n$. Moreover, $\mathrm{Res}_{\mathcal{A}_0,\dots,\mathcal{A}_n}(f_0, \dots, f_n) = 0$ if the $(n+1)$-tuple $(f_0, \dots, f_n)$ belongs to (1-1).

Example 1.2. Let $d_0, \dots, d_n$ be positive integers. For $i = 0, \dots, n$, let

$$\mathcal{A}_i = \left\{ \omega = (\omega_1, \dots, \omega_n) \in \mathbb{Z}^n_{\geq 0} : \sum_{j=1}^{n} \omega_j \leq d_i \right\}.$$

Then the $(\mathcal{A}_0, \dots, \mathcal{A}_n)$-resultant coincides with the classical (affine) resultant $\mathrm{Res}_{d_0,\dots,d_n}$, also called the *dense resultant*. Therefore, if $F_i \in \mathbb{C}[x_0, x_1, \dots, x_n]$ denotes the polynomial obtained by homogenizing $f_i \in \mathbb{C}^{\mathcal{A}_i}$ with respect to a new variable x_0, then $\mathrm{Res}_{\mathcal{A}_0,\dots,\mathcal{A}_n}(f_0, \dots, f_n) = 0$ if and only if $F_0, \dots, F_n$ have a common nontrivial root.

1B. ***Sparse unmixed resultant.*** Keep the notation and assumptions as above. If all the sets $\mathcal{A}_i$ coincide with each other, that is, $\mathcal{A}_0 = \cdots = \mathcal{A}_n = \mathcal{A}$, then the $(\mathcal{A}_0, \dots, \mathcal{A}_n)$-resultant is called the $\mathcal{A}$-resultant (also known as the *sparse* (*unmixed*) *resultant*). In this case, we have a useful geometric interpretation that allows us to write down the $\mathcal{A}$-resultant in a compact form. By choosing a numbering $\omega^{(0)}, \dots, \omega^{(k)}$

of the elements of $\mathcal{A}$, we get a map $\phi_{\mathcal{A}} : (\mathbb{C}^*)^n \to \mathbb{P}^k$ defined by $\phi_{\mathcal{A}}(x) = (\omega^{(0)}(x) : \cdots : \omega^{(k)}(x))$. Let $X_{\mathcal{A}} \subset \mathbb{P}^k$ be the closure of the image of $\phi_{\mathcal{A}}$, which is an irreducible toric variety of dimension n. Then, by taking pull-backs we get an identification between the space of polynomials in $\mathbb{C}^{\mathcal{A}}$ with the space of linear forms on $\mathbb{P}^k$. Moreover, if $f_0, \dots, f_n \in \mathbb{C}^{\mathcal{A}}$ have a common root in $(\mathbb{C}^*)^n$ then the corresponding linear forms $l_0, \dots, l_n$ on $\mathbb{P}^k$ define a linear subspace that intersects $X_{\mathcal{A}}$. From this, the following proposition follows directly.

Proposition 1.3 [Gelfand et al. 2008, Chapter 8, §2]. *The polynomial* $\mathrm{Res}_{\mathcal{A}} \in \mathbb{Z}[a_0^{(i)}, \dots, a_k^{(i)}, i = 0, \dots, n]$ *coincides with the X-resultant of $X_{\mathcal{A}} \subset \mathbb{P}^k$. More precisely, let $W_{\mathcal{A}} \subset \mathbb{G}(k-n-1, \mathbb{P}^k)$ be the Chow hypersurface of the variety $X_{\mathcal{A}}$, and let*

$$\psi : \mathbb{P}(\mathbb{C}^{(n+1)\times(k+1)}) \dashrightarrow \mathbb{G}(n,k) \simeq \mathbb{G}(k-n-1,k)$$

be the natural projection from the projectivization of the space of complex matrices which have the shape $(n+1)\times(k+1)$ to $\mathbb{G}(n,k)$. Then we have that $\mathrm{Res}_{\mathcal{A}}$ *is the polynomial defining the pull-back $\overline{\psi^{-1}(W_{\mathcal{A}})}$.*

Remark 1.4. With the notation of the proposition above, in coordinates, the map ψ is defined by the $(n+1)\times(n+1)$ minors of the generic $(n+1)\times(k+1)$ matrix of variables

$$\begin{pmatrix} a_0^{(0)} & a_1^{(0)} & \cdots & a_k^{(0)} \\ \vdots & \vdots & \ddots & \vdots \\ a_0^{(n)} & a_1^{(n)} & \cdots & a_k^{(n)} \end{pmatrix}. \tag{1-2}$$

Notably, $\mathrm{Res}_{\mathcal{A}}$ can be expressed as a homogeneous polynomial of degree $\deg(X_{\mathcal{A}})$ in the $(n+1)\times(n+1)$ minors of the matrix (1-2).

Example 1.5. Let $\mathcal{A} = \{(\omega_1, \omega_2) \in \mathbb{Z}^2 : \omega_1 + \omega_2 \leq 2\}$, so that $X_{\mathcal{A}} \subset \mathbb{P}^5$ is the Veronese surface. The $\mathcal{A}$-resultant is a polynomial of degree 12 in 18 variables with 21894 terms. It can be expressed as a polynomial of degree 4 in the Plücker coordinates of $\mathbb{G}(2,5)$ with 74 terms.

1C. ***Sparse discriminant.*** We continue by letting $\mathcal{A} \subset \mathbb{Z}^n$ be a finite set of $k+1$ elements that generate $\mathbb{Z}^n$ as a $\mathbb{Z}$-module, and let $\phi_{\mathcal{A}} : (\mathbb{C}^*)^n \to \mathbb{P}^k$ and $X_{\mathcal{A}} \subset \mathbb{P}^k$ be defined as above. Let $\nabla_{\mathcal{A}} \subset \mathbb{C}^{\mathcal{A}}$ be the Zariski closure of the set

$$\left\{ f \in \mathbb{C}^{\mathcal{A}} : \text{there exists } x \in (\mathbb{C}^*)^n \text{ such that } f(x) = \frac{\partial f}{\partial x_1}(x) = \cdots = \frac{\partial f}{\partial x_n}(x) = 0 \right\}. \tag{1-3}$$

Proposition-Definition 1.6 [Gelfand et al. 2008, Chapter 9, §1]. The projectivization $\mathbb{P}(\nabla_{\mathcal{A}}) \subset \mathbb{P}^k$ of the variety $\nabla_{\mathcal{A}}$ coincides with the dual variety $X_{\mathcal{A}}^{\vee}$ of $X_{\mathcal{A}}$. In the case where $X_{\mathcal{A}}^{\vee}$ is a hypersurface, an integral irreducible polynomial $\mathrm{Disc}_{\mathcal{A}}$ defining it (which is unique up to sign) is called the $\mathcal{A}$-discriminant (also known as the *sparse discriminant*).

Thus the $\mathcal{A}$-discriminant (when it exists) is a homogeneous polynomial $\mathrm{Disc}_{\mathcal{A}} \in \mathbb{Z}[a_\omega, \omega \in \mathcal{A}]$, and $\mathrm{Disc}_{\mathcal{A}}(f) = 0$ for each polynomial f belonging to (1-3).

Example 1.7. Let $d \geq 1$ and let $\mathcal{A} = \left\{(\omega_1, \dots, \omega_n) \in \mathbb{Z}^n_{\geq 0} : \sum_{j=1}^n \omega_j \leq d\right\}$. Then the $\mathcal{A}$-discriminant coincides with the classical (affine) discriminant Disc_d, also called the *dense discriminant*. Therefore, if $F \in \mathbb{C}[x_0, x_1, \dots, x_n]$ denotes the polynomial obtained by homogenizing $f \in \mathbb{C}^{\mathcal{A}}$ with respect to a new variable x_0, then $\mathrm{Disc}_{\mathcal{A}}(f) = 0$ if and only if the hypersurface $\{F = 0\} \subset \mathbb{P}^n$ is not smooth.

Remark 1.8 ("Cayley trick", [Gelfand et al. 2008, Chapter 9, Proposition 1.7]). Let $\mathcal{A}_0, \dots, \mathcal{A}_n \subset \mathbb{Z}^n$ be finite subsets satisfying the assumptions in Section 1A. Let $\mathcal{A} \subset \mathbb{Z}^n \times \mathbb{Z}^n$ be defined by

$$\mathcal{A} = (\mathcal{A}_0 \times \{0\}) \cup (\mathcal{A}_1 \times \{e_1\}) \cup \cdots \cup (\mathcal{A}_n \times \{e_n\}),$$

where the e_i are the standard basis vectors of $\mathbb{Z}^n$. Thus a polynomial $f \in \mathbb{C}^{\mathcal{A}}$ has the form

$$f_0(x) + \sum_{i=1}^n y_i\, f_i(x) \in \mathbb{C}[x_1, \dots, x_n, y_1, \dots, y_n],$$

where $f_i \in \mathbb{C}^{\mathcal{A}_i}$. We have the following relation (up to sign), known as the "Cayley trick":

$$\mathrm{Res}_{\mathcal{A}_0,\dots,\mathcal{A}_n}(f_0, \dots, f_n) = \mathrm{Disc}_{\mathcal{A}}\left(f_0(x) + \sum_{i=1}^n y_i\, f_i(x)\right). \tag{1-4}$$

1D. *Hyperdeterminant.* An important special type of sparse discriminant is the determinant (or hyperdeterminant) of multidimensional matrices, which was introduced by Cayley [1845] (see also [Gelfand et al. 2008, Chapter 14] and [Ottaviani 2013]). Let f be a multilinear form in r groups of variables $x_0^{(1)}, \dots, x_{k_1}^{(1)}; \dots; x_0^{(r)}, \dots, x_{k_r}^{(r)}$, that is

$$f = \sum_{0 \leq i_\iota \leq k_\iota} a_{i_1,\dots,i_r} x_{i_1}^{(1)} \cdots x_{i_r}^{(r)}.$$

Let $\mathcal{A} \subset \mathbb{Z}^{(k_1+1)+\cdots+(k_r+1)}$ denote the set of exponent vectors that can occur in such a form f. Notice that to give f is equivalent to giving an r-dimensional matrix

$$M_f = (a_{i_1,\dots,i_r})_{0 \leq i_\iota \leq k_\iota}$$

of shape $(k_1+1) \times \cdots \times (k_r+1)$. The determinant of shape $(k_1+1) \times \cdots \times (k_r+1)$ is defined to be the $\mathcal{A}$-discriminant, that is, for a form f as above, we have

$$\det(M_f) = \mathrm{Disc}_{\mathcal{A}}(f).$$

One sees that the variety $X_{\mathcal{A}}$ is the image of the Segre embedding of $\mathbb{P}^{k_1} \times \cdots \times \mathbb{P}^{k_r}$. Therefore, the hypersurface in $\mathbb{P}(\mathbb{C}^{(k_1+1)\times\cdots\times(k_r+1)})$ defined by the determinant of shape $(k_1+1) \times \cdots \times (k_r+1)$ is the dual variety of $\mathbb{P}^{k_1} \times \cdots \times \mathbb{P}^{k_r}$. Notice also that we have $\det(M_f) = 0$ if and only if the hypersurface

$$\{f = 0\} \subset \mathbb{P}^{k_1} \times \cdots \times \mathbb{P}^{k_r}$$

is not smooth.

The next two basic results have been proved in [Gelfand et al. 2008, Chapter 14, Theorems 1.3 and 2.4].

Theorem 1.9 [Gelfand et al. 2008]. *The determinant of shape $(k_1+1)\times\cdots\times(k_r+1)$ exists (that is the dual variety of $\mathbb{P}^{k_1}\times\cdots\times\mathbb{P}^{k_r}$ is a hypersurface) if and only if*

$$2\max_{1\le j\le r}(k_j)\le\sum_{j=1}^{r}k_j. \tag{1-5}$$

Theorem 1.10 [Gelfand et al. 2008]. *Denote by $N(k_1,\dots,k_r)$ the degree of the determinant of shape $(k_1+1)\times\cdots\times(k_r+1)$ when (1-5) is satisfied, and let $N(k_1,\dots,k_r)=0$ otherwise. We have*

$$\sum_{k_1,\dots,k_r\ge 0}N(k_1,\dots,k_r)z_1^{k_1}\cdots z_r^{k_r}=\frac{1}{\big(1-\sum_{i=2}^{r}(i-2)e_i(z_1,\dots,z_r)\big)^2},$$

where $e_i(z_1,\dots,z_r)$ is the i-th elementary symmetric polynomial.

Remark 1.11 [Gelfand et al. 2008, Chapter 4, Propositions 1.4 and 1.8]. The determinant of shape $(k_1+1)\times\cdots\times(k_r+1)$ is invariant under the action of $\mathrm{SL}(k_1+1)\times\cdots\times\mathrm{SL}(k_r+1)$ on the space of matrices of shape $(k_1+1)\times\cdots\times(k_r+1)$. It is also invariant under permutations of the dimensions, that is, if $M=(a_{i_1,\dots,i_r})$ is a matrix of shape $(k_1+1)\times\cdots\times(k_r+1)$ and σ is a permutation of indices $1,\dots,r$, denoting by $\sigma(M)$ the matrix of shape $(k_{\sigma^{-1}(1)}+1)\times\cdots\times(k_{\sigma^{-1}(r)}+1)$, whose $(i_1,\dots,i_r)$-th entry is equal to $a_{i_{\sigma(1)},\dots,i_{\sigma(r)}}$, we have $\det(\sigma(M))=\det(M)$.

There are at least two important cases where determinants can be computed without resorting to elimination. We briefly recall them in 1D1 and 1D2.

1D1. *Schläfli's method.* Let M be an r-dimensional matrix of shape $(k_1+1)\times\cdots\times(k_r+1)$ corresponding to a multilinear form $f\in\mathbb{C}[x_0^{(1)},\dots,x_{k_1}^{(1)};\dots;x_0^{(r)},\dots,x_{k_r}^{(r)}]$. Assume that there exist both the determinants of shapes $(k_1+1)\times\cdots\times(k_r+1)$ and $(k_1+1)\times\cdots\times(k_{r-1}+1)$. We can interpret the r-dimensional matrix M as an $(r-1)$-dimensional matrix $\tilde{M}(x_0^{(r)},\dots,x_{k_r}^{(r)})$ of shape $(k_1+1)\times\cdots\times(k_{r-1}+1)$ whose entries are linear forms in the variables $x_0^{(r)},\dots,x_{k_r}^{(r)}$; in other words, we can see f as a polynomial $\tilde{f}\in(\mathbb{C}[x_0^{(r)},\dots,x_{k_r}^{(r)}])[x_0^{(1)},\dots,x_{k_1}^{(1)};\dots;x_0^{(r-1)},\dots,x_{k_{r-1}}^{(r-1)}]$. Let

$$F_M=F_M(x_0^{(r)},\dots,x_{k_r}^{(r)})=\det(\tilde{M}(x_0^{(r)},\dots,x_{k_r}^{(r)})),$$

which is a homogeneous polynomial in $x_0^{(r)},\dots,x_{k_r}^{(r)}$, and let $\mathrm{Disc}(F_M)$ be the (classical) discriminant of F_M. Then we have the following:

Theorem 1.12 [Gelfand et al. 2008; Schläfli 1852]. *The polynomial* $\mathrm{Disc}(F_M)$ *is divisible by the determinant* $\det(M)$. *Moreover if the shape of M is one of*

$$m\times m\times 2,\qquad m\times m\times 3,\qquad 2\times 2\times 2\times 2,\qquad \textit{with } m\ge 2, \tag{1-6}$$

then we have $\mathrm{Disc}(F_M)=\det(M)$.

The method above turns out to be very effective; however it was conjectured in [Gelfand et al. 2008, p. 479], and later proved in [Weyman and Zelevinsky 1996], that the shapes in (1-6) are the only ones for which the method gives the determinant exactly.

1D2. *Determinants of boundary shape.* For an $(r+1)$-dimensional matrix M of shape $(k_0+1)\times(k_1+1)\times\cdots\times(k_r+1)$, we say that it is of *boundary shape* if the inequality (1-5) is an equality. Without loss of generality, we can assume that $k_0 = \max_{0\le j\le r}(k_j)$, so that $k_0 = k_1+\cdots+k_r$. Let $f\in\mathbb{C}[x_0^{(0)},\dots,x_{k_0}^{(0)};\dots;x_0^{(r)},\dots,x_{k_r}^{(r)}]$ be the corresponding multilinear form of such a matrix M. Thinking of f as a linear polynomial in

$$(\mathbb{C}[x_0^{(1)},\dots,x_{k_1}^{(1)};\dots;x_0^{(r)},\dots,x_{k_r}^{(r)}])[x_0^{(0)},\dots,x_{k_0}^{(0)}],$$

we can interpret M as a list of k_0+1 multilinear forms $f_0,\dots,f_{k_0}$ in the r groups of variables

$$x_0^{(1)},\dots,x_{k_1}^{(1)};\dots;x_0^{(r)},\dots,x_{k_r}^{(r)}.$$

A simple consequence of the "Cayley trick" (see [Gelfand et al. 2008, Chapter 3, Corollary 2.8]) gives the following:

Proposition 1.13 [Gelfand et al. 2008]. *The determinant of an* $(r+1)$*-dimensional matrix* M *of boundary shape* $(k_0+1)\times\cdots\times(k_r+1)$ *coincides with the resultant of the multilinear forms* $f_0,\dots,f_{k_0}$*, that is,* $\det(M)=0$ *if and only if the system of multilinear equations* $f_0(x)=\cdots=f_{k_0}(x)=0$ *has a nontrivial solution on* $\mathbb{P}^{k_1}\times\cdots\times\mathbb{P}^{k_r}$*. In other words, the determinant of shape* $(k_0+1)\times\cdots\times(k_r+1)$ *coincides with the* X*-resultant of the Segre embedding of* $\mathbb{P}^{k_1}\times\cdots\times\mathbb{P}^{k_r}$.

Remark 1.14. The determinant of a matrix M of boundary shape $(k_0+1)\times\cdots\times(k_r+1)$ can be explicitly expressed as the determinant of an ordinary square matrix of order $(k_0+1)!/(k_1!\cdots k_r!)$ whose entries are linear forms in the entries of M; see [Gelfand et al. 2008, Chapter 14, Theorem 3.3].

2. Sparse resultants in *Macaulay2*.

In this section, we describe some of the functions implemented in the package *SparseResultants*. For more details and examples, we refer to its documentation.

One of the main functions is `sparseResultant`, which via elimination techniques calculates sparse mixed resultants $\mathrm{Res}_{\mathcal{A}_0,\dots,\mathcal{A}_n}$ (see Section 1A) and sparse unmixed resultants $\mathrm{Res}_{\mathcal{A}}$ (see Section 1B). This function can be called in two ways. The first one is to pass a list of $n+1$ matrices $A_0,\dots,A_n$ over $\mathbb{Z}$ and with n rows to represent the sets $\mathcal{A}_0,\dots,\mathcal{A}_n\subset\mathbb{Z}^n$ (it is enough to pass just one matrix A in the unmixed case). Then the output will be another function that takes as input $n+1$ polynomials $f_i=\sum_{\omega\in\mathcal{A}_i}a_{i,\omega}x^\omega$, for $i=0,\dots,n$, and returns their sparse resultant. An error is thrown if the polynomials f_i do not have the correct form. Roughly, this returned function is a container for the general expression of the sparse resultant (possibly written out in a compact form as in Proposition 1.3) and for the rule to evaluate it at the $n+1$ polynomials f_i. The second way to call `sparseResultant` is to pass directly the polynomials f_i. This is equivalent to forming the matrices A_i whose columns are given by $\{\omega\in\mathbb{Z}^n:$ the coefficient in f_i of x^ω is $\neq 0\}$ (see the function `exponentsMatrix`) and then proceeding as described above.

As an example we now calculate a particular type of sparse unmixed resultant, known as the *Dixon resultant* (see [Sturmfels 1993, Section 2.4] and [Cox et al. 2005, Chapter 7, §2, Exercise 10]; see also the classical reference [Dixon 1909]).

Example 2.1. Consider the following system of three bihomogeneous polynomials of bidegree (2, 1) in the two groups of variables (x_0, x_1), (y_0, y_1):

$$\begin{aligned}
c_{1,1}x_1^2y_1 + c_{1,2}x_1x_2y_1 + c_{1,3}x_2^2y_1 + c_{1,4}x_1^2y_2 + c_{1,5}x_1x_2y_2 + c_{1,6}x_2^2y_2 = 0,\\
c_{2,1}x_1^2y_1 + c_{2,2}x_1x_2y_1 + c_{2,3}x_2^2y_1 + c_{2,4}x_1^2y_2 + c_{2,5}x_1x_2y_2 + c_{2,6}x_2^2y_2 = 0,\\
c_{3,1}x_1^2y_1 + c_{3,2}x_1x_2y_1 + c_{3,3}x_2^2y_1 + c_{3,4}x_1^2y_2 + c_{3,5}x_1x_2y_2 + c_{3,6}x_2^2y_2 = 0.
\end{aligned} \tag{2-1}$$

Putting $x_2 = y_2 = 1$ we get a system of three nonhomogeneous polynomials in two variables $(x, y) = (x_1, y_1)$, of which we can calculate the sparse (unmixed) resultant. This polynomial is homogeneous of degree 12 in the 18 variables $c_{1,1}, \dots, c_{3,6}$ with 20791 terms, which vanishes if and only if (2-1) has a nontrivial solution. The time for this computation is less than one second (on a standard laptop).

```
$ M2 --no-preload
Macaulay2, version 1.17
i1 : needsPackage "SparseResultants";

i2 : R = ZZ[c_(1,1)..c_(3,6)][x,y];

i3 : f = (c_(1,1)*x^2*y+c_(1,2)*x*y+c_(1,3)*y+c_(1,4)*x^2+c_(1,5)*x+c_(1,6),
          c_(2,1)*x^2*y+c_(2,2)*x*y+c_(2,3)*y+c_(2,4)*x^2+c_(2,5)*x+c_(2,6),
          c_(3,1)*x^2*y+c_(3,2)*x*y+c_(3,3)*y+c_(3,4)*x^2+c_(3,5)*x+c_(3,6));

i4 : A = exponentsMatrix f
o4 = | 0 0 1 1 2 2 |
     | 0 1 0 1 0 1 |

              2        6
o4 : Matrix ZZ  <--- ZZ

i5 : time Res = sparseResultant A;
     -- used 0.241391 seconds
o5 : SparseResultant (sparse unmixed resultant associated to | 0 0 1 1 2 2 |)
                                                             | 0 1 0 1 0 1 |

i6 : time U = Res f;
     -- used 0.574002 seconds

i7 : (first degree U, # terms U)
o7 = (12, 20791)
```

Another function, sparseDiscriminant, calculates sparse discriminants $\mathrm{Disc}_{\mathcal{A}}$ (see Section 1C). This function works similarly to the previous one. In particular, it accepts as input either a matrix representing the exponent vectors of a (Laurent) polynomial or the polynomial directly.

Example 2.2. Using the Cayley trick (1-4), we express the dense resultant of three generic ternary forms of degrees 1, 1, 2 (which is a special type of sparse mixed resultant) as a sparse discriminant. The calculation time is less than one second.

```
i8 : clearAll;

i9 : K = ZZ[a_0..a_2,b_0..b_2,c_0..c_5], Rx = K[x_1,x_2];

i10 : f = (a_0+a_1*x_1+a_2*x_2,
           b_0+b_1*x_1+b_2*x_2,
           c_0+c_1*x_1+c_2*x_2+c_3*x_1^2+c_4*x_1*x_2+c_5*x_2^2);

i11 : Rxy = K[x_1,x_2,y_1,y_2], f' = (sub(f_0,Rxy), sub(f_1,Rxy), sub(f_2,Rxy));

i12 : time sparseResultant(f_0,f_1,f_2) ==
           -sparseDiscriminant(f'_0 + y_1*f'_1 + y_2*f'_2)
      -- used 0.746274 seconds
o12 = true
```

A derived function of sparseDiscriminant is determinant (or simply det), which calculates determinants of multidimensional matrices (see Section 1D). However for this last one, more specialized algorithms are also available and automatically applied.

Example 2.3. We calculate the determinant of a generic four-dimensional matrix of shape $2 \times 2 \times 2 \times 2$ (see also [Huggins et al. 2008]). This polynomial is homogeneous of degree 24 in the 16 variable entries of the matrix and it has 2894276 terms. The approach for this calculation is to apply (1-6) recursively. The calculation time is about 10 minutes, but it takes much less time if we specialize the entries of the matrix to be random numbers.

```
i13 : M = genericMultidimensionalMatrix {2,2,2,2}
o13 = {{{{a        , a        }, {a        , a        }}, {{a        , a        }, ...
           0,0,0,0    0,0,0,1     0,0,1,0    0,0,1,1       0,1,0,0    0,1,0,1     ...

o13 : 4-dimensional matrix of shape 2 x 2 x 2 x 2 over ZZ[a        , a        , ...
                                                           0,0,0,0    0,0,0,1    ...

i14 : time D = det M;
      -- used 634.773 seconds

i15 : (first degree D, # terms D)
o15 = (24, 2894276)
```

Example 2.4. Here we take A and B to be random matrices of shapes $2 \times 2 \times 2 \times 4$ and $4 \times 2 \times 5$, respectively. We calculate the convolution $A * B$ (see [Gelfand et al. 2008, p. 449]), which is a matrix of shape $2 \times 2 \times 2 \times 2 \times 5$. Then we verify a formula proved in [Dionisi and Ottaviani 2003] for $\det(A * B)$, which generalizes the Cauchy–Binet formula in the multidimensional case. The approach for the calculation of the determinant of shape $4 \times 2 \times 5$ is using Proposition 1.13, while the determinants of shapes $2 \times 2 \times 2 \times 4$ and $2 \times 2 \times 2 \times 2 \times 5$ are calculated using Remark 1.14. The calculation time is less than one second.

```
i16 : K = ZZ/33331;

i17 : A = randomMultidimensionalMatrix({2,2,2,4},CoefficientRing=>K);
o17 : 4-dimensional matrix of shape 2 x 2 x 2 x 4 over K

i18 : B = randomMultidimensionalMatrix({4,2,5},CoefficientRing=>K);
o18 : 3-dimensional matrix of shape 4 x 2 x 5 over K

i19 : time det(A * B) == (det A)^5 * (det B)^6
      -- used 0.535271 seconds
o19 = true
```

SUPPLEMENT. The online supplement contains version 1.1 of SparseResultants.

REFERENCES.

[Canny and Emiris 2000] J. F. Canny and I. Z. Emiris, "A subdivision-based algorithm for the sparse resultant", *J. ACM* **47**:3 (2000), 417–451. MR Zbl

[Cattani et al. 1998] E. Cattani, A. Dickenstein, and B. Sturmfels, "Residues and resultants", *J. Math. Sci. Univ. Tokyo* **5**:1 (1998), 119–148. MR Zbl

[Cayley 1845] A. Cayley, "On the theory of linear transformations", *Cambridge Math. J.* **4** (1845), 193–209.

[Cox et al. 2005] D. A. Cox, J. Little, and D. O'Shea, *Using algebraic geometry*, 2nd ed., Graduate Texts in Mathematics **185**, Springer, 2005. MR Zbl

[D'Andrea 2002] C. D'Andrea, "Macaulay style formulas for sparse resultants", *Trans. Amer. Math. Soc.* **354**:7 (2002), 2595–2629. MR Zbl

[D'Andrea and Sombra 2015] C. D'Andrea and M. Sombra, "A Poisson formula for the sparse resultant", *Proc. Lond. Math. Soc.* (3) **110**:4 (2015), 932–964. MR

[Dionisi and Ottaviani 2003] C. Dionisi and G. Ottaviani, "The Binet–Cauchy theorem for the hyperdeterminant of boundary format multi-dimensional matrices", *J. Algebra* **259**:1 (2003), 87–94. MR Zbl

[Dixon 1909] A. L. Dixon, "The Eliminant of Three Quantics in two Independent Variables", *Proc. London Math. Soc.* (2) **7** (1909), 49–69. MR Zbl

[Emiris and Mourrain 1999] I. Z. Emiris and B. Mourrain, "Matrices in elimination theory", pp. 3–44 Polynomial elimination—algorithms and applications **1**, 1999. MR Zbl

[Gelfand et al. 1992] I. M. Gelfand, M. M. Kapranov, and A. V. Zelevinsky, "Hyperdeterminants", *Adv. Math.* **96**:2 (1992), 226–263. MR Zbl

[Gelfand et al. 2008] I. M. Gelfand, M. M. Kapranov, and A. V. Zelevinsky, *Discriminants*, *resultants and multidimensional determinants*, Birkhäuser, Boston, 2008. MR

[Huggins et al. 2008] P. Huggins, B. Sturmfels, J. Yu, and D. S. Yuster, "The hyperdeterminant and triangulations of the 4-cube", *Math. Comp.* **77**:263 (2008), 1653–1679. MR Zbl

[Jeronimo et al. 2004] G. Jeronimo, T. Krick, J. Sabia, and M. Sombra, "The computational complexity of the Chow form", *Found. Comput. Math.* **4**:1 (2004), 41–117. MR

[Jeronimo et al. 2009] G. Jeronimo, G. Matera, P. Solernó, and A. Waissbein, "Deformation techniques for sparse systems", *Found. Comput. Math.* **9**:1 (2009), 1–50. MR Zbl

[MacAulay 1903] F. S. MacAulay, "Some Formulae in Elimination", *Proc. Lond. Math. Soc.* **35** (1903), 3–27. MR Zbl

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

[Ottaviani 2013] G. Ottaviani, "Introduction to the hyperdeterminant and to the rank of multidimensional matrices", pp. 609–638 in *Commutative algebra*, edited by I. Peeva, Springer, 2013. MR Zbl

[Schläfli 1852] L. Schläfli, "Über die Resultante eines Systemes mehrerer algebraischer Gleichungen: Ein Beitrag zur Theorie der Elimination", *Denkschr. der Kaiserlichen Akad. der Wiss*, *Math-Naturwiss. Classe* **4** (1852), 1–74.

[Staglianò 2018] G. Staglianò, "A package for computations with classical resultants", *J. Softw. Algebra Geom.* **8** (2018), 21–30. MR Zbl

[Sturmfels 1993] B. Sturmfels, "Sparse elimination theory", pp. 264–298 in *Computational algebraic geometry and commutative algebra* (Cortona, 1991), edited by D. Eisenbud and L. Robbiano, Sympos. Math. **34**, Cambridge Univ. Press,, 1993. MR Zbl

[Sturmfels 1994] B. Sturmfels, "On the Newton polytope of the resultant", *J. Algebraic Combin.* **3**:2 (1994), 207–236. MR Zbl

[Sturmfels 2002] B. Sturmfels, *Solving systems of polynomial equations*, CBMS Regional Conference Series in Mathematics **97**, American Mathematical Society, Providence, RI, 2002. MR Zbl

[Weyman and Zelevinsky 1996] J. Weyman and A. Zelevinsky, "Singularities of hyperdeterminants", *Ann. Inst. Fourier* **46**:3 (1996), 591–644. MR Zbl

RECEIVED: 23 Jul 2020 REVISED: 25 Jan 2021 ACCEPTED: 5 May 2021

GIOVANNI STAGLIANÒ:
giovannistagliano@gmail.com
Dipartimento di Matematica e Informatica, Università degli Studi di Catania, Catania, Italy

JSAG **11** (2021), 71–81
https://doi.org/10.2140/jsag.2021.11.71

The Journal of Software for
Algebra and Geometry

ExteriorModules: a package for computing monomial modules over an exterior algebra

LUCA AMATA AND MARILENA CRUPI

ABSTRACT: Let K be a field, E the exterior algebra of a finite-dimensional K-vector space, and F a finitely generated graded free E-module with homogeneous basis $g_1, \dots, g_r$ such that $\deg(g_1) \leq \deg(g_2) \leq \cdots \leq \deg(g_r)$. We present a *Macaulay*2 package to manage some classes of monomial submodules of F. The package is an extension of our `ExteriorIdeals` package on monomial ideals (*J. of Software for Alg. and Geom.* **8**:7 (2018), 71–79), and contains some algorithms for computing stable, strongly stable and lexicograhic E-submodules of F. This package also includes some methods to check whether a sequence of nonnegative integers is the Hilbert function of a graded E-module of the form F/M, with M a graded submodule of F. Moreover, if $H_{F/M}$ is the Hilbert function of a graded E-module F/M, some routines are able to compute the unique lexicograhic submodule L of F such that $H_{F/M} = H_{F/L}$.

1. INTRODUCTION. Monomial modules generalize the notion of monomial ideals which are ideals generated by monomials. Therefore, many tools in monomial ideal theory are available to deal with such a class of modules. Indeed, many statements on monomial modules can be deduced from the ones on monomial ideals. Many authors have been interested in the study of such classes of modules in different contexts and in the computation of certain invariants associated to them (see, for instance [Aramova et al. 1997; Aramova and Herzog 2000; Crupi and Ferrò 2016; Eisenbud 1995; Gasharov 1997; Herzog and Hibi 2011; Kämpf 2010; Pardue 1996]).

In this paper, we introduce a new Macaulay2 package, `ExteriorModules`, for manipulating special classes of monomial modules over an exterior algebra of a finite-dimensional vector space over a field. This package extends the one on monomial ideals, `ExteriorIdeals`, and needs it to work. An updated version of such a package is available on the *Macaulay*2 repository. In more detail, the package `ExteriorModules` provides functions to check whether a monomial module is (strongly) stable, or lexicographic, and to compute the smallest (strongly) stable module containing a given monomial module. Moreover, given F a finitely generated graded free module over an exterior algebra E, the package allows us to characterize the Hilbert sequences of the E-modules of the type F/M, with M a graded submodule of F.

Some service methods are inherited from `ExteriorIdeals` or overloaded to extend them to modules in order to optimize the implementation of the main algorithms.

MSC2020: 13A02, 15A75, 68W30.
Keywords: exterior algebra, monomial modules, Hilbert functions, algorithms.
`ExteriorModules` version 1.0

2. MATHEMATICAL BACKGROUND. Let K be a field and let $E = K\langle e_1, \dots, e_n\rangle$ be the exterior algebra of a K-vector space V with basis $e_1, \dots, e_n$. For any subset $\sigma = \{i_1, \dots, i_d\}$ of $\{1, \dots, n\}$ with $i_1 < i_2 < \cdots < i_d$ we write $e_\sigma = e_{i_1} \wedge \cdots \wedge e_{i_d} = e_{i_1} \cdots e_{i_d}$, in order to simplify the notation, and call e_σ a monomial of degree d. We set $e_\sigma = 1$, if $\sigma = \emptyset$. The set of monomials in E forms a K-basis of E of cardinality 2^n.

Let $e_\sigma = e_{i_1} \cdots e_{i_d} \neq 1$ be a monomial in E. We define

$$\mathrm{supp}(e_\sigma) = \sigma = \{i : e_i \text{ divides } e_\sigma\},$$

and we write

$$\mathrm{m}(e_\sigma) = \max\{i : i \in \mathrm{supp}(e_\sigma)\}.$$

Moreover, we set $\mathrm{m}(e_\sigma) = 0$, if $e_\sigma = 1$.

From now on, we write $fg = f \wedge g$ for any two elements f and g in E. An element $f \in E$ is called homogeneous of degree j if $f \in E_j$, where $E_j = \bigwedge^j V$. An ideal I is called graded if I is generated by homogeneous elements. If I is graded, then $I = \bigoplus_{j \geq 0} I_j$, where I_j is the K-vector space of all homogeneous elements $f \in I$ of degree j. Moreover, we use $\mathrm{indeg}(I)$ to denote the initial degree of I, i.e., the minimum s such that $I_s \neq 0$.

Let $\mathcal{M}$ be the category of finitely generated $\mathbb{Z}$-graded left and right E-modules M such that $am = (-1)^{\deg a \deg m} ma$ for all homogeneous elements $a \in E,\ m \in M$.

If $M \in \mathcal{M}$, the function $H_M : \mathbb{Z} \to \mathbb{Z}$ given by $H_M(d) = \dim_K M_d$ is called the Hilbert function of M [Bruns and Herzog 1993; Eisenbud 1995].

Let $F \in \mathcal{M}$ be a free module with homogeneous basis $g_1, \dots, g_r$, where $\deg(g_i) = f_i$ for each $i = 1, \dots, r$, with $f_1 \leq f_2 \leq \cdots \leq f_r$. We write $F = \bigoplus_{i=1}^r Eg_i$. The elements $e_\sigma g_i$, with e_σ a monomial of E, are called *monomials* of F, and $\deg(e_\sigma g_i) = \deg(e_\sigma) + \deg(g_i)$. Furthermore, when we write $F \simeq E^r$, we mean that $F = \bigoplus_{i=1}^r Eg_i$ is the free E-module with trivial homogeneous basis $g_1, \dots, g_r$, where g_i $(i = 1, \dots, r)$ is the r-tuple having the only nonzero entry equal to 1 in the i-th position and such that $\deg(g_i) = 0$, for all i.

Definition 2.1. A graded submodule M of F is a *monomial submodule* if M is a submodule generated by monomials of F, i.e.,

$$M = I_1 g_1 \oplus \cdots \oplus I_r g_r,$$

where I_i is a monomial ideal of E, for each i.

We observe that, if $r = 1$ and $f_1 = 0$ then a monomial submodule of F is a monomial ideal of E.

Definition 2.2. A monomial ideal I of E is called *stable* if for each monomial $e_\sigma \in I$ and each $j < \mathrm{m}(e_\sigma)$ one has $e_j e_{\sigma \setminus \{\mathrm{m}(e_\sigma)\}} \in I$. I is called *strongly stable* if for each monomial $e_\sigma \in I$ and each $j \in \sigma$ one has $e_i e_{\sigma \setminus \{j\}} \in I$, for all $i < j$.

Remark 2.3. One can observe that the defining property of a strongly stable ideal needs to be checked only for the set of monomial generators of a monomial ideal [Amata and Crupi 2018b, Remark 2.2.].

Definition 2.4. A monomial submodule $M = \bigoplus_{i=1}^{r} I_i g_i$ of F is called *almost* (*strongly*) *stable* if I_i is a (strongly) stable ideal of E, for each i.

Definition 2.5. A monomial submodule $M = \bigoplus_{i=1}^{r} I_i g_i$ of F is called (*strongly*) *stable* if I_i is a (strongly) stable ideal of E, for each i, and $(e_1, \dots, e_n)^{f_{i+1}-f_i} I_{i+1} \subseteq I_i$, for $i = 1, \dots, r-1$.

Given a monomial ideal I of E, we denote by $G(I)$ the unique minimal set of monomial generators of I, and by $G(I)_d$ the set of all monomials $u \in G(I)$ such that $\deg(u) = d,\ d > 0$. Similarly, for every monomial submodule $M = \bigoplus_{i=1}^{r} I_i g_i$ of F, we write

$$G(M) = \{u g_i \,:\, u \in G(I_i),\ i = 1, \dots, r\},$$
$$G(M)_d = \{u g_i \,:\, u \in G(I_i)_{d-f_i},\ i = 1, \dots, r\}.$$

For the classification of the Hilbert functions of graded E-modules, the class of lexicographic modules plays a crucial role [Amata and Crupi 2020a; 2020b]. Moreover, such a class of monomial modules is essential if one wants to determine certain upper bounds for the graded Betti numbers of graded E-modules [Amata and Crupi 2018a; 2019].

Given a nonempty subset S of E (respectively, of F), we denote by $\mathrm{Mon}(S)$ the set of all monomials in S (respectively, in F). Moreover, we denote by $\mathrm{Mon}_d(S)$ the set of all monomials of degree d in S.

We denote by $>_{\mathrm{lex}}$ the lexicographic order (lex order, for short) on $\mathrm{Mon}_d(E)$, i.e., if $e_\sigma = e_{i_1} e_{i_2} \cdots e_{i_d}$ and $e_\tau = e_{j_1} e_{j_2} \cdots e_{j_d}$ are monomials belonging to $\mathrm{Mon}_d(E)$ with $1 \le i_1 < i_2 < \cdots < i_d \le n$ and $1 \le j_1 < j_2 < \cdots < j_d \le n$, then $e_\sigma >_{\mathrm{lex}} e_\tau$ if $i_1 = j_1, \dots, i_{s-1} = j_{s-1}$ and $i_s < j_s$ for some $1 \le s \le d$.

Definition 2.6. Let L be a nonempty subset of $\mathrm{Mon}_d(E)$. L is called a *lexicographic segment* (lex segment, for short) of degree d if for all $v \in L$ and all $u \in \mathrm{Mon}_d(E)$ such that $u >_{\mathrm{lex}} v$, we have that $u \in L$.

Definition 2.7. Let I be a monomial ideal of E. I is called a *lexicographic ideal* (lex ideal, for short) if for all monomials $v \in I$ and all monomials $u \in E$ with $\deg v = \deg u$ and $u >_{\mathrm{lex}} v$, then $u \in I$, i.e., $\mathrm{Mon}_d(I)$ is a lex segment, for all d.

Remark 2.8. The trivial ideals of E, i.e., (0) and E itself, are considered monomial lex ideals.

Now, we extend the previous definitions to monomial submodules of F. To do this, we order the set of monomials $\mathrm{Mon}(F)$ by using the ordering $>_{\mathrm{lex}_F}$ defined as follows: if $u g_i$ and $v g_j$ are monomials of F such that $\deg(u g_i) = \deg(v g_j)$, then $u g_i >_{\mathrm{lex}_F} v g_j$ if $i < j$ or $i = j$ and $u >_{\mathrm{lex}} v$.

Definition 2.9. Let N be a nonempty subset of $\mathrm{Mon}_d(F)$. N is called a *lexicographic segment of F* (lex_F segment, for short) of degree d if for all $v \in N$ and all $u \in \mathrm{Mon}_d(F)$ such that $u >_{\mathrm{lex}_F} v$, then $u \in N$.

Definition 2.10. Let L be a monomial submodule of F. L is a *lex submodule* if for all $u, v \in \mathrm{Mon}_d(F)$ with $v \in L$ and $u >_{\mathrm{lex}_F} v$, one has $u \in L$, for every d, i.e., $\mathrm{Mon}_d(L)$ is a lex_F segment of degree d, for each degree d.

An equivalent definition of a lex submodule is the following one [Amata and Crupi 2018a, Proposition 3.12] (see also [Crupi and Ferrò 2016, Proposition 3.8]).

Definition 2.11. Let L be a graded submodule of F. L is a *lex submodule of* F if $L = \bigoplus_{i=1}^{r} I_i g_i$, with I_i lex ideals of E $(i = 1, \dots, r)$, and $(e_1, \dots, e_n)^{\rho_i + f_i - f_{i-1}} \subseteq I_{i-1}$, for $i = 2, \dots, r$, with $\rho_i = \operatorname{indeg} I_i$.

Remark 2.12. The class of lex submodules of F is obviously contained in the class of strongly stable submodules, and consequently in the class of the stable ones.

A particular overclass of the class of lex submodules, introduced in [Amata and Crupi 2018a] for bounding the graded Betti numbers of a graded E-module, is the following one.

Definition 2.13. Let L be a graded submodule of F. L is an *almost lexicographic submodule* if $L = \bigoplus_{i=1}^{r} I_i g_i$, with I_i lex ideals of E $(i = 1, \dots, r)$.

In order to discuss the Hilbert functions of quotients of free E-modules, we need some notation and remarks. For more details on the subject see [Amata and Crupi 2020a; 2020b].

Firstly, we set

$$\binom{m}{k} = 0 \quad \text{if } m < k \text{ or } k < 0.$$

One can observe that if $F = \bigoplus_{i=1}^{r} E g_i$, $\deg g_i = f_i$, for $i = 1, \dots, r$ and $f_1 \le \dots \le f_r$, we have that

$$H_F(d) = \sum_{i=1}^{r} H_{Eg_i}(d) = \sum_{i=1}^{r} \binom{n}{d - f_i}.$$

Hence, if M is a graded submodule of F, one has

$$H_{F/M}(d) + H_M(d) = \sum_{i=1}^{r} \binom{n}{d - f_i},$$

where $\binom{n}{d-f_i}$ is the number of monomials of degree $d - f_i$ in E.

As a consequence, we have that

$$H_F(d) = \dim_K F_d = 0, \qquad \text{for } d < f_1 \text{ and } d > f_r + n. \tag{1}$$

If M is a monomial submodule of F, from (1), it follows that

$$H_{F/M}(t) = \sum_{i=f_1}^{f_r+n} H_{F/M}(i) t^i,$$

and we can associate to F/M the sequence

$$(H_{F/M}(f_1), H_{F/M}(f_1 + 1), \dots, H_{F/M}(f_r + n)) \in \mathbb{N}_0^{f_r + n - f_1 + 1}. \tag{2}$$

Such a sequence is called the Hilbert sequence of F/M, and we denote it by $Hs_{F/M}$. The integers $f_1, f_1 + 1, \dots, f_r + n$ are called the $Hs_{F/M}$-degrees.

Let us define

$$\mathrm{indeg} Hs_{F/M} = \min\{d : H_{F/M}(d) \neq 0\}, \quad \text{for } d = f_1, \dots, f_r + n.$$

We use the notation $[p]$ for the set $\{1, 2, \dots, p\}$.

The entries $H_{F/M}(f_i)$ ($i = 1, \dots, r$) are called the *critical values* of $Hs_{F/M}$. Moreover, the integers

$$\mu_{f_i} = |\{s \in [r] : f_s = f_i\}|, \quad \text{for } i = 1, 2, \dots, r$$

are called the *multiplicity* of $H_{F/M}(f_i)$.

We can observe that some critical values can be zero, and this implies that all the entries of the Hilbert sequence are zero until the next nonzero critical value. So it makes sense to investigate the minimum critical value of a Hilbert sequence to get important information about the behavior of a Hilbert function.

Let k be the minimum integer such that $H_{F/M}(f_k) \neq 0$, i.e., $\mathrm{indeg} Hs_{F/M} = f_k$. The integer $H_{F/M}(f_k)$ is called the *initial critical value* (of F/M) and f_k the *initial critical degree* (of F/M). Moreover, we have that

$$H_{F/M}(f_k) \leq \mu_{f_k},$$

and

$$H_{F/M}(f_k + 1) \leq n\mu_{f_k} + \mu_{f_k+1}.$$

Now we have all the necessary ingredients to quote the main result on the classification of Hilbert functions of quotients of graded free E-modules [Amata and Crupi 2020a, Theorem 4.2]. The pivotal idea of such a classification is that if M is a graded submodule of F, then there exists a unique lex submodule of F with the same Hilbert function as M.

Let a and i be two positive integers. Then a has the unique i-th Macaulay expansion [Herzog and Hibi 2011, Lemma 6.3.4]

$$a = \binom{a_i}{i} + \binom{a_{i-1}}{i-1} + \cdots + \binom{a_j}{j}$$

with $a_i > a_{i-1} > \cdots > a_j \geq j \geq 1$. We define

$$a^{(i)} = \binom{a_i}{i+1} + \binom{a_{i-1}}{i} + \cdots + \binom{a_j}{j+1}.$$

We also set $0^{(i)} = 0$ for all $i \geq 1$.

For $p, q \in \mathbb{Z}$ with $p < q$, let us define the set

$$[p, q] = \{j \in \mathbb{Z} : p \leq j \leq q\}.$$

Theorem 2.14 [Amata and Crupi 2020a, Theorem 4.2]. *Let $(f_1, f_2, \dots, f_r) \in \mathbb{Z}^r$ be an r-tuple such that $f_1 \leq f_2 \leq \cdots \leq f_r$, and let $(h_{f_1}, h_{f_1+1}, \dots, h_{f_r+n})$ be a sequence of nonnegative integers. Set*

$$s = \min\{k \in [f_1, f_r + n] : h_k \neq 0\}$$

and

$$\tilde{r}_j = |\{p \in [r] : f_p = s + j\}|, \quad \textit{for } j = 0, 1.$$

Then the following conditions are equivalent:

(a) $\sum_{i=s}^{f_r+n} h_i t^i$ *is the Hilbert series of a graded* E*-module* F/M, *with* $F = \bigoplus_{i=1}^r Eg_i$ *a finitely generated graded free* E*-module with the basis elements* g_i *of degrees* f_i.

(b) $h_s \le \tilde{r}_0$, $h_{s+1} \le n\tilde{r}_0 + \tilde{r}_1$, $h_i = \sum_{j=N+1}^r \binom{n}{i-f_j} + a$, *where* a *is a positive integer less than* $\binom{n}{i-f_N}$, $0 < N \le r$, *and* $h_{i+1} \le \sum_{j=N+1}^r \binom{n}{i-f_j+1} + a^{(i-f_N)}$, $i = s+1, \dots, f_r + n$.

(c) *There exists a unique lex submodule* L *of a finitely generated graded free* E*-module* $F = \bigoplus_{i=1}^r Eg_i$ *with the basis elements* g_i *of degrees* f_i, *and such that* $\sum_{i=s}^{f_r+n} h_i t^i$ *is the Hilbert series of* F/L.

From now on, if M is a monomial submodule of the finitely generated graded free E-module $F = \bigoplus_{i=1}^r Eg_i$, we denote by M^{lex} the unique lex submodule of F with the same Hilbert function as M. M^{lex} is called the *lex submodule associated* to M.

Remark 2.15. Theorem 2.14 generalizes the well-known Kruskal–Katona's theorem in [Aramova et al. 1997], and can be also obtained via results on ideals in an exterior algebra [Amata and Crupi 2020b, Criterion 3.3]. The underlying algorithm is implemented as the `lexModuleBySequences` method in the `ExteriorModules` package.

3. Examples.

In this section, we collect some examples in order to describe the algorithms.

In what follows, let $F = \bigoplus_{i=1}^r Eg_i$ be a finitely generated graded free E-module such that

$$\deg(g_1) \le \deg(g_2) \le \cdots \le \deg(g_r).$$

Example 3.1. Let M be a monomial submodule of the graded free module F, we illustrate functions from the `ExteriorModules` package (analogous to those for ideals [Amata and Crupi 2018b]) in order to check whether M is (strongly) stable or (almost) lex, and to produce the smallest (strongly) stable submodule containing M.

```
i1 : loadPackage "ExteriorModules";
i2 : E=QQ[e_1..e_5,SkewCommutative=>true];
i3 : F=E^2;
i4 : I_1=ideal {e_1*e_2, e_1*e_3, e_1*e_4*e_5};
i5 : I_2=ideal {e_1*e_2, e_2*e_3*e_4};
i6 : M=createModule({I_1, I_2},F)
o6 = image|e_1e_3 e_1e_2 e_1e_4e_5 0      0        |
          |0      0      0         e_1e_2 e_2e_3e_4|
o6 : E-module, submodule of E^2
i7 : isStableModule M
o7 = false
```

The submodule M is almost stable but not stable. In fact, the monomial $e_2e_3e_4$ does not belong to I_1

(Definition 2.5). We can compute the smallest stable submodule of F containing M by the function `stableModule(module)`.

```
i8 : Ms=stableModule M

o8 = image|e_1e_2 e_1e_3 e_1e_4e_5 e_2e_3e_4 0      0        |
           |0      0      0         0         e_1e_2 e_2e_3e_4|

o8 : E-module, submodule of E^2

i9 : isStronglyStableModule Ms

o9 = false
```

The submodule Ms is stable, and but neither almost strongly stable nor strongly stable. In fact, the ideal $(e_1e_2, e_2e_3e_4)$ is not strongly stable. We compute the smallest strongly stable submodule of F containing Ms by using the function `stronglyStableModule(module)`:

```
i10 : Mss=stronglyStableModule Ms

o10 = image|e_1e_2 e_1e_3 e_1e_4e_5 e_2e_3e_4 0      0         0        |
            |0      0      0         0         e_1e_2 e_1e_3e_4 e_2e_3e_4|

o10 : E-module, submodule of E^2

i11 : isStronglyStableModule Mss

o11 = true

i12 : Mss==stronglyStableModule M

o12 = true
```

The submodule Mss is not an almost lex submodule of F. Indeed, the ideal $(e_1e_3e_4, e_2e_3e_4)$ is not lex.

```
i13 : isLexIdeal (getIdeals Mss)_1

o13 = false

i14 : isAlmostLexModule Mss

o14 = false
```

Remark 3.2. The functions `stableModule(module)` and `stronglyStableModule(module)` allow the construction of (strongly) stable submodules of a finitely generated graded free module F. The methods to compute the smallest stable and strongly stable submodule containing a given submodule are useful, although they do not preserve invariants. In fact, the computation by hand of a stable or a strongly stable submodule implies some tedious calculations overall in the case when the elements of the homogeneous basis of F have different degrees. Furthermore, it is worth pointing out that such methods are analogous to the *Macaulay*2 function `borel` that computes the smallest borel ideal containing a given ideal.

Example 3.3. Let h be a sequence of nonnegative integers. We describe how one can check whether h is a Hilbert sequence of a graded E-module of the type F/M, with M graded submodule of F. The key tools are the functions `lexModuleBySequences(list,F)` (Remark 2.15), `isHilbertSequence(list,F)`, and `lexModule(list,F)`. The first function verifies if a list of nonnegative integers of a given length is a Hilbert function; the other ones return a lex submodule of F if and only if the *list* is a Hilbert sequence. In more detail, if hs is a given Hilbert sequence, the lex submodule of F produced by both the functions `lexModule(`hs`, F)` and `lexModuleBySequences(`hs`, F)` is the unique lex submodule L of F with $H_{F/L} = hs$. These functions work also in the case when the basis elements of the free module F have different degrees.

```
i1 : loadPackage "ExteriorModules";

i2 : E=QQ[e_1..e_4,SkewCommutative=>true];

i3 : F=E^3;

i4 : hs={3, 12, 16, 6, 0};

i5 : lexModule(hs,F)

o5 = image|e_1e_2 e_1e_3 e_2e_3e_4 0         0         0           |
          |0      0      0         e_1e_2e_3 e_1e_2e_4 0           |
          |0      0      0         0         0         e_1e_2e_3e_4|

o5 : E-module, submodule of E^3

i6 : F=E^{2,0,-2};

i7 : hs={1, 4, 5, 4, 6, 5, 6, 3, 0};

i8 : lexModuleBySequences(hs,F)

o8 = image {-2}|e_1e_3 e_1e_2 e_2e_3e_4 0      0         0        |
           {0} |0      0      0         e_1e_2 e_1e_3e_4 0        |
           {2} |0      0      0         0      0         e_1e_2e_3|

o8 : E-module, submodule of E^3

i9 : F=E^{3,1,-2};

i10 : hs={1, 2, 2, 4, 3, 3, 4, 5, 2, 0};

i11 : isHilbertSequence(hs,F)

o11 = false
```

Example 3.4. Given a graded submodule M of F, we illustrate another way for computing the unique lex submodule associated to M. Given M, we compute M^{lex} by the function `lexModule(module)`. The procedure for the computation of the required lex submodule is based on the constructive proof of Theorem 2.14, (b) $\Rightarrow$ (c).

```
i1 : loadPackage "ExteriorModules";

i2 : E=QQ[e_1..e_4,SkewCommutative=>true];

i3 : F=E^3;

i4 : I_1=ideal {e_1, e_2*e_3*e_4};

i5 : I_2=ideal {e_1*e_2, e_1*e_3*e_4};

i6 : I_3=ideal {e_1*e_2*e_3};

i7 : M=createModule({I_1, I_2, I_3},F)

o7 = image|e_1 e_2e_3e_4 0      0         0        |
          |0   0         e_1e_2 e_1e_3e_4 0        |
          |0   0         0      0         e_1e_2e_3|

o7 : E-module, submodule of E^3

i8 : isAlmostLexModule M

o8 = true

i9 : isLexModule M

o9 = false

i10 : L=lexModule M

o10 = image|e_1 e_2e_3 0         0         0         0         0           |
           |0   0      e_1e_2e_3 e_1e_2e_4 e_1e_3e_4 e_2e_3e_4 0           |
           |0   0      0         0         0         0         e_1e_2e_3e_4|

o10 : E-module, submodule of E^3
```

```
i11 : hilbertSequence M
o11 = {3, 11, 14, 4, 0}
i12 : hilbertSequence M==hilbertSequence L
o12 = true
```

The function `lexModule(module)` also works in the case when the basis elements of the free module F have different degrees.

```
i1 : loadPackage "ExteriorModules";
i2 : E=QQ[e_1..e_4,SkewCommutative=>true];
i3 : F=E^{2,0,-1};
i4 : I_1=ideal {e_1*e_2, e_3*e_4};
i5 : I_2=ideal {e_1*e_2, e_2*e_3*e_4};
i6 : I_3=ideal {e_2*e_3*e_4};
i7 : M=createModule({I_1, I_2, I_3},F)
o7 = image {-2}|e_1e_2 e_3e_4 0      0          0         |
           {0} |0      0      e_1e_2 e_2e_3e_4 0         |
           {1} |0      0      0      0          e_2e_3e_4|
o7 : E-module, submodule of E^3
i8 : L=lexModule M
o8 = image {-2}|e_1e_2 e_1e_3 e_2e_3e_4 0      0          0         |
           {0} |0      0      0          e_1e_2 e_1e_3e_4 0         |
           {1} |0      0      0          0      0          e_1e_2e_3|
o8 : E-module, submodule of E^3
i9 : hilbertSequence M
o9 = {1, 4, 5, 5, 9, 7, 3, 0}
09 : List
i10 : hilbertSequence M==hilbertSequence L
o10 = true
```

4. Conclusions and perspectives.

The procedures described in this paper are part of the *Macaulay*2 package `ExteriorModules` (which uses the `ExteriorIdeals` package [Amata and Crupi 2018b]), and tested with Macaulay 1.14 as well as all the examples in this paper.

As far as we know, specific packages for manipulating classes of monomial modules over an exterior algebra have not been implemented yet. Many characterizations and algorithmic methods presented in the package are due to the authors of this paper. We believe that these packages may reveal useful further applications. Indeed, it would be nice to create functions for the computation of the generic initial module of a graded E-module M in the category $\mathcal{M}$, which is a strongly stable module with the same Hilbert function as M.

Moreover, it would be interesting to manage the dual module of a graded E-module $M \in \mathcal{M}$ in a general case, i.e., when M is a submodule of a finitely generated graded free module whose basis elements have different degrees. The case when the basis elements have the same degree was faced and solved in [Amata and Crupi 2019].

These problems are currently under investigation by the authors.

APPENDIX: LIST OF FUNCTIONS PROVIDED.

`createModule(list)`	Give the monomial module from a list of ideals
`getIdeals(M)`	Get ideals from a monomial module `M`
`hilbertSequence(M)`	Give the Hilbert function sequence of `M`
`isMonomialModule(M)`	Check whether a module `M` is monomial
`isAlmostLexModule(M)`	Check whether a module `M` is almost lex
`almostLexModule(M)`	Give the almost lex module associated to `M`
`isLexModule(M)`	Check whether a module `M` is lex
`isHilbertSequence(l,F)`	Check whether the Kruskal–Katona theorem is satisfied for `l`
`lexModule(hs,F)`	Give the lex module with the given Hilbert sequence `hs`
`lexModule(M)`	Give the lex module associated to `M`
`lexModuleBySequences(hs,F)`	Give the lex module with the given Hilbert function 2.15
`isAlmostStronglyStableModule(M)`	Check whether a module `M` is almost strongly stable
`almostStronglyStableModule(M)`	Give the minimal almost strongly stable module containing `M`
`isAlmostStableModule(M)`	Check whether a module `M` is almost stable
`almostStableModule(M)`	Give the minimal almost stable module containing `M`
`isStronglyStableModule(M)`	Check whether a module `M` is strongly stable
`stronglyStableModule(M)`	Give the minimal strongly stable module containing `M`
`isStableModule(M)`	Check whether a module `M` is stable
`stableModule(M)`	Give the minimal stable module containing `M`
`minimalBettiNumbers(M)`	Give the (minimal) Betti numbers of `M`
`initialModule(M)`	Give the initial module of `M`

SUPPLEMENT. The online supplement contains version 1.0 of `ExteriorModules`.

REFERENCES.

[Amata and Crupi 2018a] L. Amata and M. Crupi, "Bounds for the Betti numbers of graded modules with given Hilbert function in an exterior algebra via lexicographic modules", *Bull. Math. Soc. Sci. Math. Roumanie* (*N.S.*) **61(109)**:3 (2018), 237–253. MR Zbl

[Amata and Crupi 2018b] L. Amata and M. Crupi, "ExteriorIdeals: A package for computing monomial ideals in an exterior algebra", *J. of Software for Alg. and Geom.* **8**:7 (2018), 71–79. Zbl

[Amata and Crupi 2019] L. Amata and M. Crupi, "Minimal resolutions of graded modules over an exterior algebra", *Atti Accad. Peloritana Pericolanti Cl. Sci. Fis. Mat. Natur.* **97**:1 (2019), art. id. A5. MR Zbl

[Amata and Crupi 2020a] L. Amata and M. Crupi, "A generalization of Kruskal–Katona's theorem", *An. Ştiinţ. Univ. "Ovidius" Constanţa Ser. Mat.* **28**:2 (2020), 35–52. MR

[Amata and Crupi 2020b] L. Amata and M. Crupi, "Hilbert functions of graded modules over an exterior algebra: an algorithmic approach", *Int. Electron. J. Algebra* **27** (2020), 271–287. MR Zbl

[Aramova and Herzog 2000] A. Aramova and J. Herzog, "Almost regular sequences and Betti numbers", *Amer. J. Math.* **122**:4 (2000), 689–719. MR Zbl

[Aramova et al. 1997] A. Aramova, J. Herzog, and T. Hibi, "Gotzmann theorems for exterior algebras and combinatorics", *J. Algebra* **191**:1 (1997), 174–211. 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

[Crupi and Ferrò 2016] M. Crupi and C. Ferrò, "Squarefree monomial modules and extremal Betti numbers", *Algebra Colloq.* **23**:3 (2016), 519–530. MR Zbl

[Eisenbud 1995] D. Eisenbud, *Commutative algebra: with a view toward algebraic geometry*, Graduate Texts in Mathematics **150**, Springer, 1995. MR Zbl

[Gasharov 1997] V. Gasharov, "Extremal properties of Hilbert functions", *Illinois J. Math.* **41**:4 (1997), 612–629. MR Zbl

[Herzog and Hibi 2011] J. Herzog and T. Hibi, *Monomial ideals*, Graduate Texts in Mathematics **260**, Springer, 2011. MR Zbl

[Kämpf 2010] G. Kämpf, *Module theory over the exterior algebra with applications to combinatorics*, Ph.D. thesis, Osnabrück, 2010.

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

[Pardue 1996] K. Pardue, "Deformation classes of graded modules and maximal Betti numbers", *Illinois J. Math.* **40**:4 (1996), 564–585. MR Zbl

RECEIVED: 16 Jun 2020 REVISED: 5 May 2021 ACCEPTED: 3 Jun 2021

LUCA AMATA:
lamata@unime.it
Department of Mathematics and Computer Sciences, Physical and Earth Sciences, University of Messina, Messina, Italy

MARILENA CRUPI:
mcrupi@unime.it
Department of Mathematics and Computer Sciences, Physical and Earth Sciences, University of Messina, Messina, Italy

JSAG **11** (2021), 83–87
https://doi.org/10.2140/jsag.2021.11.83

The Journal of Software for
Algebra and Geometry

The Schur–Veronese package in Macaulay2

JULIETTE BRUCE, DANIEL ERMAN, STEVE GOLDSTEIN AND JAY YANG

ABSTRACT: This note introduces the *Macaulay*2 package *SchurVeronese*, which gathers together data about Veronese syzygies and makes it readily accessible in *Macaulay*2. In addition to standard Betti tables, the package includes information about the Schur decompositions of the various spaces of syzygies. The package also includes a number of functions useful for manipulating and studying this data.

In [Bruce et al. 2020] the authors used a combination of high-throughput and high-performance computation and numerical techniques to compute the Betti tables of $\mathbb{P}^2$ under the d-fold Veronese embedding, as well as the Betti tables of the pushforwards of line bundles $\mathcal{O}_{\mathbb{P}^2}(b)$ under that embedding, for a number of values of b and d. These computations resulted in new data, such as Betti tables, multigraded Betti numbers, and Schur Betti numbers. (For $b = 0$, most the cases had been previously computed in [Castryck et al.].) This note introduces the *SchurVeronese* package for *Macaulay*2, which makes this data readily accessible via *Macaulay*2 for further experimentation and study.

1. VERONESE SYZYGIES. Throughout this section we fix $n \in \mathbb{N}$ and let $S = \mathbb{C}[x_0, x_1, \dots, x_n]$ be the polynomial ring with the standard grading. The d-th Veronese module of S twisted by b is

$$S(b; d) := \bigoplus_{i \in \mathbb{Z}} S_{di+b}.$$

If $b = 0$, then $S(0; d)$ is the Veronese subring of S, and if $b \neq 0$ then $S(b; d)$ is an $S(0; d)$-module. Moreover, if we set $R = \mathrm{Sym}(S_d)$ to be the symmetric algebra on S_d, then we may consider $S(b; d)$ as a graded R-module. Geometrically, if $b = 0$ this corresponds to the homogenous coordinate ring of $\mathbb{P}^n$ under the d-fold embedding $\mathbb{P}^n \to \mathbb{P}^{\binom{n+d}{d}-1}$, and for other b it corresponds to the pushforward of $\mathcal{O}_{\mathbb{P}^n}(b)$ under the d-fold embedding.

Our interest is in studying the syzygies of $S(b; d)$. See the introduction of [Bruce et al. 2020] for background on Veronese syzygies including a summary of known results. Throughout this paper, we set $K_{p,q}(\mathbb{P}^n, b; d) := \mathrm{Tor}_p^R(S(b; d), \mathbb{C})_{p+q}$, which is isomorphic to the vector space of degree $p+q$ syzygies of $S(b; d)$ of homological degree p. Using the standard conventions for graded Betti numbers, the rank

Bruce received support from the NSF GRFP under grant DGE-1256259, NSF grant DMS-1502553, NSF MSPRF DMS-2002239, and from the Graduate School and the Office of the Vice Chancellor for Research and Graduate Education at the University of Wisconsin–Madison with funding from the Wisconsin Alumni Research Foundation. She is grateful for the support of the Mathematical Sciences Research Institute in Berkeley, California, where she was in residence for the Fall 2020 semester. Erman received support from NSF grant DMS-1601619. Yang received support from NSF grant DMS-1502553.
MSC2010: 13D02.
Keywords: syzygies, free resolutions, Veronese syzygies.
SchurVeronese version 1.1

of the vector space $K_{p,q}$ corresponds to the Betti number $\beta_{p,p+q}$, and we write $\beta_{p,p+q}(S(b;d)) := \dim \operatorname{Tor}_p^R(S(b;d),\mathbb{C})_{p+q} = \dim K_{p,q}(\mathbb{P}^n, b; d)$. Following the usual *Macaulay*2 notation, the Betti table of $S(b;d)$ will be the table where $\beta_{p,p+q}(S(b;d))$ is placed in the (p,q)-spot.

Outside of the case $n=1$, the Betti tables of $S(b;d)$ are unknown even for modest values of d. There is not even a conjecture about what the Betti table of $S(b;d)$ should be for $n=2$ and $d \geq 7$.

This package provides an array of computed data about $S(b;d)$ in the case $n=2$ and for $0 \leq b < d \leq 8$ (though the data are incomplete for some of the larger values of d). While computing this data, including the Schur functor decompositions, took substantial time, the resulting data are concise and easy to work with in *Macaulay*2. The bulk of this package thus consists of these output data, which are included as auxiliary files. The functions provided in this package make this data accessible in a user-friendly way. Our hope is that this will allow those interested in Veronese syzygies to make headway on formulating conjectures and proving results in this area. Moreover, as new cases of Veronese syzygies are computed, these can easily be incorporated into future versions of the package.

2. An overview of the data.

When computing data for $S(b;d)$ we always work under the hypothesis that $0 \leq b < d$, as the Betti table of $S(b;d)$ and $S(b+d;d)$ differ only by a vertical shift. We have included data for the cases $n=1$ and $d \leq 10$, although this can also easily be computed using the Eagon–Northcott complex. The main data are for the cases $n=2$ and $0 \leq b < d \leq 8$. In [Bruce et al. 2020], we obtained full computations for $d \leq 6$; moreover since those algorithms worked in parallel with respect to multidegrees, we obtained incomplete data for some cases where $d = 7, 8$, and we have included these partial data in this package as well.

The algorithms in [Bruce et al. 2020] are a mix of symbolic and numeric algebra. Thus some entries in the data are not provably correct, while others are. One can determine precisely when $K_{p,q} \neq 0$ by combining [Ein and Lazarsfeld 2012, Remark 6.5], [Green 1984b, Theorem 2.2], and [Green 1984a, Theorem 2.c.6]. Our computation of a $K_{p,q}$-group (and all related data such as the Schur functor decomposition) will be provably correct if and only if $K_{p+1,q-1}$ and $K_{p-1,q+1}$ both vanish; in cases where this does not occur, the data for $K_{p,q}$ may have been computed numerically, and thus may not be provably correct. For a longer discussion of potential numerical error issues, see [Bruce et al. 2020, §5.2].

3. Total Betti tables.

The Betti table for $S(b;d)$ can be called up using the `totalBettiTally` command. For example, the Betti table of $S(2;4)$ when $n=2$ is produced below.

```
i6 : totalBettiTally(4,2,0)

            0  1   2    3    4    5    6    7    8    9  10 11 12
o6 = total: 1 75 536 1947 4488 7095 7920 6237 3344 1089 175 24  3
         0: 1  .   .    .    .    .    .    .    .    .   .  .  .
         1: . 75 536 1947 4488 7095 7920 6237 3344 1089 120  .  .
         2: .  .   .    .    .    .    .    .    .    .  55 24  3

o6 : BettiTally
```

Note that this is purely numeric: the package does not produce a minimal free resolution; the function simply returns the Betti numbers obtained by a previous computation. The command `totalBetti` is similar, but expresses the Betti numbers simply as a hash table.

There is also a distinction between the indexing conventions. When working with hash tables, we follow the more concise $K_{p,q}$ indexing conventions, instead of the $\beta_{p,p+q}$ indexing conventions used for Betti tallies. Thus, for instance, in the above example, the Betti number $\beta_{2,3}$ would correspond to key (2, {3}, 3) in the Betti tally, but in the hash table `totalBetti` it corresponds to key (2, 1):

```
i4 :E =  totalBetti(4,2,0);

i5 : E#(2,1)
o5 = 536
```

If one tries to call a Betti table outside of the acceptable range of n, b, d, we return an error message.

```
i10 : totalBettiTally(4,3,0)
o10 = Need n = 1 or 2
```

As noted above, there were instances where we were able to partially compute Betti tables, for instance in the case of the 7-uple embedding of $\mathbb{P}^2$. In those cases, we have recorded the entries that we know, and we mark the unknown entries with “infinity”. For example:

```
i14 : B = totalBetti(7,2,0);

i15 : B#(4,1)
o15 = 1031184

i16 : B#(20,1)
o16 = infinity
o16 : InfiniteNumber
```

Thus, in this case, we see $\dim K_{4,1}(\mathbb{P}^2, 2; 7) = 1031184$, but were unable to compute $\dim K_{20,1}(\mathbb{P}^2, 2; 7)$.

4. Schur decomposition.

When $n = 2$ and $d \geq 5$, the Betti tables of $S(b; d)$ are often unwieldy to work with, as they and their entries tend to be quite large. For example, the Betti table of $S(0; 6)$ has 26 columns and many of the entries are on the order of 10^7.

A more concise way of recording the syzygies would be to take into account the symmetries coming from representation theory. The natural linear action of $\boldsymbol{GL}_{n+1}(\mathbb{C})$ on S induces an action on each vector space $K_{p,q}(\mathbb{P}^n, b; d)$. We can thus decompose this as a direct sum of Schur functors of total weight $d(p+q)+b$, i.e.,

$$K_{p,q}(\mathbb{P}^n, b; d) = \bigoplus_{|\lambda|=d(p+q)+b} \boldsymbol{S}_\lambda(\mathbb{C}^{n+1})^{\oplus m_{p,\lambda}(\mathbb{P}^n,b;d)},$$

with $m_{p,\lambda}(\mathbb{P}^n, b; d)$ being the Schur Betti numbers and $\boldsymbol{S}_\lambda$ being the Schur functor corresponding to the partition λ [Fulton and Harris 1991, p. 76]. The Schur Betti numbers can be accessed via the `schurBetti` command, which returns a hash table whose keys correspond to pairs (p, q) for which $K_{p,q}(\mathbb{P}^n, b; d) \neq 0$, and whose values are lists corresponding to the Schur decomposition of this syzygy module.

For example, let us consider $K_{2,1}(\mathbb{P}^2, 0; 4)$, which is a vector space of dimension 536. As a representation of $\boldsymbol{GL}_3(\mathbb{C})$, it turns out to be the sum of 9 distinct Schur functors, each appearing with multiplicity 1:

$$K_{2,1}(\mathbb{P}^2, 0; 4) = \boldsymbol{S}_{(9,2,1)} \oplus \boldsymbol{S}_{(8,4,0)} \oplus \boldsymbol{S}_{(8,3,1)} \oplus \boldsymbol{S}_{(7,5,0)} \oplus \boldsymbol{S}_{(7,4,1)} \oplus \boldsymbol{S}_{(7,3,2)} \oplus \boldsymbol{S}_{(6,5,1)} \oplus \boldsymbol{S}_{(6,4,2)} \oplus \boldsymbol{S}_{(5,4,1)}.$$

```
i26 : (schurBetti(4,2,0))#(2,1)

o26 = {({9, 2, 1}, 1), ({8, 4, 0}, 1), ({8, 3, 1}, 1), ({7, 5, 0}, 1),
      ------------------------------------------------------------------------
       ({7, 4, 1}, 1), ({7, 3, 2}, 1), ({6, 5, 1}, 1), ({6, 4, 2}, 1), ({5, 4, 3}, 1)}

o8 : List
```

From this, it is easy to compute statistics such as the number of representations and the number of distinct representations appearing in the Schur decomposition of $K_{p,q}(n, b; d)$. The *SchurVeronese* package provides commands for these. For instance, in our example above we see that:

```
i11 : (numDistinctRepsBetti(4,2,0))#(2,1)
o11 = 9
```

We can also display the number of representations appearing in each entry of the Betti table. In the following example, the first table counts distinct Schur functors and the second counts the number of Schur functors with multiplicity:

```
i29 : makeBettiTally numDistinctRepsBetti(4,2,0)

             0 1 2  3  4  5  6  7  8  9 10 11 12
o29 = total: 1 2 9 17 23 23 26 25 21 13  3  1  1
          0: 1 . .  .  .  .  .  .  .  .  .  .  .
          1: . 2 9 17 23 23 26 25 21 13  1  .  .
          2: . . .  .  .  .  .  .  .  .  2  1  1

i30 : makeBettiTally numRepsBetti(4,2,0)

             0 1 2  3  4  5  6  7  8  9 10 11 12
o30 = total: 1 2 9 28 55 79 86 69 38 14  3  1  1
          0: 1 . .  .  .  .  .  .  .  .  .  .  .
          1: . 2 9 28 55 79 86 69 38 14  1  .  .
          2: . . .  .  .  .  .  .  .  .  2  1  1
```

Thus, $K_{4,1}(\mathbb{P}^2, 0; 4)$ is the sum of 55 irreducible representations, 23 of which are distinct.

5. Multigraded Betti numbers.

One can also specialize the action of $\boldsymbol{GL}_{n+1}(\mathbb{C})$ to the torus action via $(\mathbb{C}^*)^{n+1}$. This gives a decomposition of $K_{p,q}(\mathbb{P}^n, b; d)$ into a sum of $\mathbb{Z}^{n+1}$-graded vector spaces of total weight $d(p+q)+b$. Specifically, writing $\mathbb{C}(-\boldsymbol{a})$ for the vector space $\mathbb{C}$ together with the $(\mathbb{C}^*)^{n+1}$-action given by $(\lambda_0, \lambda_1, \ldots, \lambda_n)\cdot\mu = \lambda_0^{a_0}\lambda_1^{a_1}\cdots\lambda_n^{a_n}\mu$, we have

$$K_{p,q}(\mathbb{P}^n, b; d) = \bigoplus_{\substack{\boldsymbol{a}\in\mathbb{Z}^{n+1}\\ |\boldsymbol{a}|=d(p+q)+b}} \mathbb{C}(-\boldsymbol{a})^{\oplus\beta_{p,\boldsymbol{a}}(\mathbb{P}^n,b;d)}$$

as $\mathbb{Z}^{n+1}$-graded vector spaces, or equivalently as $(\mathbb{C}^*)^{n+1}$ representations.

The *SchurVeronese* package produces these multigraded Betti numbers for a number of examples via the `multiBetti` command. As `schurBetti` does, this command returns a hash table whose keys correspond to pairs (p, q) for which $K_{p,q}(\mathbb{P}^n, b; d) \neq 0$, and whose values are multigraded Hilbert polynomials encoding the multigraded decomposition of $K_{p,q}(n, b; d)$. More specifically, the value of `(multiBetti(d,n,b))#(p,q)` is the polynomial

$$\sum_{\substack{\boldsymbol{a}\in\mathbb{Z}^{n+1}\\ |\boldsymbol{a}|=d(p+q)+b}} \beta_{p,\boldsymbol{a}}(n, b; d)\boldsymbol{t}^{\boldsymbol{a}}$$

where $\boldsymbol{t}^{\boldsymbol{a}}$ denotes $t_0^{a_0}t_1^{a_1}\cdots t_n^{a_n}$.

For example, $K_{12,2}(2, 0; 4)$ is the 3-dimensional $\mathbb{Z}^3$-graded vector space

$$K_{12,2}(2, 0; 4) \cong \mathbb{C}(-(19, 19, 18)) \oplus \mathbb{C}(-(19, 18, 19)) \oplus \mathbb{C}(-(18, 19, 19)).$$

The following code computes this, illustrating that the multigraded Hilbert function for $K_{12,2}(2,0;4)$ is $t_0^{19}t_1^{19}t_2^{18}+t_0^{19}t_1^{18}t_2^{19}+t_0^{18}t_1^{19}t_2^{19}$.

```
i4 : (multiBetti(4,2,0))#(12,2)

       19 19 18    19 18 19    18 19 19
o4 = t  t  t   + t  t  t   + t  t  t
      0  1  2     0  1  2     0  1  2

o4 : QQ[t , t , t ]
         0   1   2
```

SUPPLEMENT. The online supplement contains version 1.1 of *SchurVeronese*.

ACKNOWLEDGMENTS. We thank Claudiu Raicu and Gregory G. Smith for useful conversations.

REFERENCES.

[Bruce et al. 2020] J. Bruce, D. Erman, S. Goldstein, and J. Yang, "Conjectures and Computations about Veronese Syzygies", *Experimental Mathematics* **29**:4 (2020), 398–413.

[Castryck et al.] W. Castryck, F. Cools, J. Demeyer, and A. Lemmens, "Computing graded Betti tables of toric surfaces". arXiv

[Ein and Lazarsfeld 2012] L. Ein and R. Lazarsfeld, "Asymptotic syzygies of algebraic varieties", *Inventiones Mathematicae* **190** (2012), 603–646.

[Fulton and Harris 1991] W. Fulton and J. Harris, *Representation theory*, vol. 129, Graduate Texts in Mathematics, Springer-Verlag, New York, 1991. MR

[Green 1984a] M. L. Green, "Koszul cohomology and the geometry of projective varieties, I", *J. Differential Geom.* **19**:1 (1984), 125–171.

[Green 1984b] M. L. Green, "Koszul cohomology and the geometry of projective varieties, II", *J. Differential Geom.* **20**:1 (1984), 279–289.

RECEIVED: 28 May 2019 REVISED: 18 Feb 2021 ACCEPTED: 5 May 2021

JULIETTE BRUCE:
juliette.bruce@berkeley.edu
Department of Mathematics, University of California, Berkeley, Berkeley, CA, United States

DANIEL ERMAN:
derman@math.wisc.edu
Department of Mathematics, University of Wisconsin, Van Vleck Hall, Madison, WI, United States

STEVE GOLDSTEIN:
sgoldstein@wisc.edu
Botany Department and Department of Biostatistics and Medical Informatics, University of Wisconsin, Madison, WI, United States

JAY YANG:
jkyang@umn.edu
School of Mathematics, University of Minnesota, Minneapolis, MN, United States

JSAG **11** (2021), 89–112
https://doi.org/10.2140/jsag.2021.11.89

The Journal of Software for
Algebra and Geometry

admcycles - a Sage package for calculations in the tautological ring of the moduli space of stable curves

VINCENT DELECROIX, JOHANNES SCHMITT AND JASON VAN ZELM

ABSTRACT: The tautological ring of the moduli space of stable curves has been studied extensively in the last decades. We present a SageMath implementation of many core features of this ring. This includes lists of generators and their products, intersection numbers and verification of tautological relations. Maps between tautological rings induced by functoriality, that is pushforwards and pullbacks under gluing and forgetful maps, are implemented. Furthermore, many interesting cycle classes, such as the double ramification cycles, strata of k-differentials and hyperelliptic or bielliptic cycles are available. We show how to apply the package, including concrete example computations.

1. INTRODUCTION. A crucial tool in the study of the singular cohomology of the moduli space $\overline{\mathcal{M}}_{g,n}$ of stable curves is the tautological ring

$$\mathrm{RH}^*(\overline{\mathcal{M}}_{g,n}) \subset H^*(\overline{\mathcal{M}}_{g,n}) = H^*(\overline{\mathcal{M}}_{g,n}, \mathbb{Q}).$$

It is a $\mathbb{Q}$-subalgebra of the singular cohomology of $\overline{\mathcal{M}}_{g,n}$ with an explicit, finite set of generators (indexed by decorated graphs $[\Gamma, \alpha]$) admitting combinatorial descriptions of operations like cup products and intersection numbers. For a detailed introduction to the tautological ring, see, e.g., [Faber and Pandharipande 2000; Arbarello et al. 2011; Pandharipande 2018].

Since computations with the generators $[\Gamma, \alpha]$ quickly become untractable by hand, it is natural to implement them in a computer program. With `admcycles` we present such an implementation using the open source mathematical software [SageMath]. It is based on an earlier implementation by Aaron Pixton. It features intersection products and numbers between the classes $[\Gamma, \alpha]$ and verification of linear relations between these generators using the known generalized Faber–Zagier relations [Pixton 2012; Pandharipande et al. 2015; Janda 2017]. For the gluing and forgetful morphisms between (products of) the moduli spaces $\overline{\mathcal{M}}_{g,n}$ it implements pullbacks and pushforwards of the generators $[\Gamma, \alpha]$ of the tautological ring.

Many geometric constructions of cohomology classes on $\overline{\mathcal{M}}_{g,n}$ (such as the Chern classes λ_d of the Hodge bundle $\mathbb{E}$ over $\overline{\mathcal{M}}_{g,n}$) give classes contained in the tautological ring and can thus be written as linear combinations of classes $[\Gamma, \alpha]$. For many examples of such classes, the package `admcycles`

MSC2010: 14H10, 97N80.
Keywords: moduli of curves, tautological ring, intersection theory, double ramification cycle.
`admcycles` version 1.3.1

implements known formulas or algorithms to calculate them and thus allows further computations, such as intersections or comparisons to other cohomology classes. In particular, `admcycles` contains

- a formula for double ramification cycles $\mathrm{DR}_g(A)$ from [Janda et al. 2017] ,
- a conjectural formula for the strata $\overline{\mathcal{H}}_g^k(\boldsymbol{m})$ of k-differentials from [Farkas and Pandharipande 2018; Schmitt 2018],
- (generalized) lambda classes, the Chern classes of derived pushforwards $R^\bullet\pi_*\mathcal{O}(D)$ of divisors D on the universal curve $\pi : \mathcal{C}_{g,n} \to \overline{\mathcal{M}}_{g,n}$, as discussed in [Pagani et al. 2020],
- admissible cover cycles,[1] such as the fundamental classes of loci of hyperelliptic or bielliptic curves with marked ramification points, as discussed in [Schmitt and van Zelm 2020]

Instead of discussing the details of the algorithms in `admcycles`, this document serves as a user manual for the package, with an emphasis on concrete example computations. These computations are also available in an interactive online format on CoCalc (without need for registration) here.

One way to explore `admcycles` is to go through these examples and refer back to the text below for additional explanations and background. While the code in the examples is mostly self-explanatory, some basic familiarity with SageMath and the Python programming language (e.g., as explained in the official SageMath tutorial) is helpful.

***Applications of* `admcycles`.** By now the package `admcycles` has been used in a variety of contexts. Its original purpose was computing new examples of admissible cover cycles in [Schmitt and van Zelm 2020], e.g., computing the class of the hyperelliptic locus in $\overline{\mathcal{M}}_5$ and $\overline{\mathcal{M}}_6$ and the locus of bielliptic cycles in $\overline{\mathcal{M}}_4$. It was also used to verify results about Hodge integrals on bielliptic cycles in [Pandharipande and Tseng 2019] and on loci of cyclic triple covers of rational curves in [Owens and Somerstep 2019].

Buryak and Rossi [2021] used `admcycles` to explore formulas for intersection numbers involving double ramification cycles and lambda classes. The implementation of generalized lambda classes led to the discovery of previously missing terms in the computations of [Pagani et al. 2020] when doing comparisons with double ramification cycles. The package was also used in [Chen et al. 2019] to verify computations of Masur–Veech volumes in terms of intersection numbers on $\overline{\mathcal{M}}_{g,n}$. It was used to check a new recursion for intersection numbers of ψ-classes presented in [Grosse et al. 2019] and formulas for double Hurwitz numbers in terms of intersection numbers in [Borot et al. 2020] and [Do and Lewański 2020]. In [Castorena and Gendron 2020], which computes a fundamental class of a stratum of meromorphic differentials in genus 3, some errors have been found and corrected after comparing the result with the output of `admcycles`. More recently, in [Bae and Schmitt 2020] some code based on `admcycles` was used to compute ranks of Chow groups of moduli stacks $\mathfrak{M}_{0,n}$ of prestable curves. Molcho et al. [2021] applied the package to verify the completeness of the generalized Faber–Zagier relations in two new cases on $\overline{\mathcal{M}}_{4,1}$ and $\overline{\mathcal{M}}_{5,1}$ and used this to show that for $g \geq 7$ the class λ_g is not contained in the subring of the

[1]Computing these cycles was the original purpose of `admcycles`, hence the name of the package.

cohomology of $\overline{\mathcal{M}}_g$ generated by classes of cohomological degree at most 4. Very recently, Canning and Larson [2021] used admcycles for computing the rational Chow rings of the spaces $\mathcal{M}_g$ for $g = 7, 8, 9$.

Other implementations. Apart from admcycles (and the code of Pixton on which it is based) there have been several other implementations of the tautological ring, starting with Faber's program [1999] for computing intersection numbers of divisors and Chern classes of the Hodge bundle. Yang [2008] presents a program computing intersection pairings of tautological classes on various open subsets of $\overline{\mathcal{M}}_{g,n}$. The package [mgn] by Johnson implements general intersections of the $[\Gamma, \alpha]$ and also verification of linear relations between these generators against the known generalized Faber–Zagier relations.

Based on admcycles there is the new SageMath-package diffstrata (included in admcycles since version 1.1) by Costantini, Möller and Zachhuber. It implements the tautological ring and intersection products on the smooth compactification of the strata of differentials presented in [Bainbridge et al. 2019a]. Computations with diffstrata are used in [Costantini et al. 2020a] to evaluate formulas for Euler characteristics of strata of differentials in examples. Similar to the present paper, a detailed description of the package diffstrata is given in [Costantini et al. 2020b].

1.1. ***Conventions.*** Let $\overline{\mathcal{M}}_{g,n}$ be the moduli space of stable curves and $\pi : \overline{\mathcal{M}}_{g,n+1} \to \overline{\mathcal{M}}_{g,n}$ be the forgetful morphism of the marking $n+1$, which can be seen as the universal curve over $\overline{\mathcal{M}}_{g,n}$. Let $\sigma_i : \overline{\mathcal{M}}_{g,n} \to \overline{\mathcal{M}}_{g,n+1}$ be the section of π corresponding to the i-th marked point ($i = 1, \dots, n$). For ω_π the relative dualizing line bundle of π on the space $\overline{\mathcal{M}}_{g,n+1}$ and $i = 1, \dots, n$ we define the ψ-class

$$\psi_i = c_1(\sigma_i^* \omega_\pi) \in H^2(\overline{\mathcal{M}}_{g,n}).$$

For $a = 0, 1, 2, \dots$ we define the (Arbarello–Cornalba) κ-class

$$\kappa_a = \pi_*((\psi_{n+1})^{a+1}) \in H^{2a}(\overline{\mathcal{M}}_{g,n}).$$

Finally, given a stable graph Γ of genus g with n legs, let

$$\xi_\Gamma : \overline{\mathcal{M}}_\Gamma = \prod_{v \in V(\Gamma)} \overline{\mathcal{M}}_{g(v),n(v)} \to \overline{\mathcal{M}}_{g,n}$$

be the gluing map associated to Γ. For a class $\alpha \in H^*(\overline{\mathcal{M}}_\Gamma)$ given as a product of κ and ψ-classes on the factors $\overline{\mathcal{M}}_{g(v),n(v)}$, define

$$[\Gamma, \alpha] = (\xi_\Gamma)_* \alpha \in H^*(\overline{\mathcal{M}}_{g,n}).$$

Such decorated boundary strata form a generating set (as a $\mathbb{Q}$-vector space) of the tautological ring $\mathrm{RH}^*(\overline{\mathcal{M}}_{g,n})$.

Note: The degree of the gluing map ξ_Γ to its image is given by the size $|\mathrm{Aut}(\Gamma)|$ of the automorphism group of Γ. Therefore many authors prefer to define $[\Gamma, \alpha]$ as $1/|\mathrm{Aut}(\Gamma)| \cdot (\xi_\Gamma)_* \alpha$ (so that $[\Gamma, 1]$ equals the class of the boundary stratum of $\overline{\mathcal{M}}_{g,n}$ associated to Γ). However, throughout this paper and in the package admcycles, we take the convention of *not* dividing by the size $|\mathrm{Aut}(\Gamma)|$ of the automorphism group of Γ.

2. Getting started. The `admcycles` package works on top of SageMath which is an open source software for mathematical computations. We describe how to install SageMath and `admcycles` on a computer and how to use the available online services.

2.1. **`admcycles` *in the cloud.*** The simplest way to play with `admcycles` without installing anything beyond a web browser is to use one of [SageMathCell] or the website [CoCalc]. The former provides a basic interface to SageMath. The latter requires registration and allows one to create worksheets that can easily be saved and shared. As mentioned before, it is possible to explore the computations presented below on share.cocalc.com without the need to register.

2.2. ***Obtaining SageMath.*** SageMath is available on most operating systems. Depending on the situation one can find it in the list of softwares available from the package manager of the operating system. Alternatively, there are binaries available from the SageMath website . Lastly, one can compile it from the source code. More information on the installation process can be found here.

2.3. ***Installation of the* `admcycles` *package.*** The package `admcycles` is available from the Python Package Index (PyPI) where detailed installation instructions are available for a range of systems. Note that the best performance (in particular for functions like `DR_cycle`) is obtained using version 9.0 of SageMath or newer.

The package `admcycles` is being developed on GitLab where one can find the latest development version and a link to report bugs. This is also the place to look at to suggest features or improvements.

2.4. ***First step with* `admcycles`*.*** Once successfully installed, to use `admcycles` one should start a SageMath-session and type

```
sage: from admcycles import *
```

In the sample code, we reproduce the behavior of the SageMath console that provides the `sage:` prompt on each input line. When using the online SageMathCell or a Jupyter worksheet, there is no need to write `sage:`. In all our examples, this `sage:` prompt allows one to distinguish between the input (command) and the output (result). *All other examples below assume that the line*

```
from admycles import *
```

has been executed before.

In addition to this manual, the package has an internal documentation with more information concerning the various functions. To access additional information about some function or object `foo`, type `foo?` during the SageMath session; e.g.,

```
sage: TautologicalRing?
```

3. Tautological ring and classes. The main objects in admcycles to manipulate tautological classes are `TautologicalRing` and `TautologicalClass`.

3.1. ***Creating tautological rings.*** A convenient way to start a computation in the tautological ring of $\overline{\mathcal{M}}_{g,n}$ is to construct the appropriate ring itself by calling the function `TautologicalRing(g, n)`.

```
sage: R = TautologicalRing(1, 1); R
TautologicalRing(g=1, n=1, moduli='st') over Rational Field
```

As we explain in Section 3.2, the object `R` above then allows easy access to many of the standard tautological classes on $\overline{\mathcal{M}}_{g,n}$. As an example, we show how to compute the integral

$$\int_{\overline{\mathcal{M}}_{1,1}} \psi_1 = \tfrac{1}{24}$$

using the ring `R` we created above (see Section 3.3 for more details):

```
sage: R.psi(1).evaluate()
1/24
```

Instead of working with the tautological ring of all of $\overline{\mathcal{M}}_{g,n}$, it is also possible to work on open subsets of the moduli space, such as the locus of compact type curves. This can be specified with the parameter `moduli`:

```
sage: Rct = TautologicalRing(3, 1, moduli='ct')
```

The available moduli types are:

- `'st'`: all stable curves (default).
- `'tl'`: treelike curves (all cycles in the stable graph have length 1).
- `'ct'`: compact type (stable graph is a tree).
- `'rt'`: rational tails (there exists a vertex of genus g).
- `'sm'`: smooth curves.

As an example of how this affects the behavior of the tautological ring, we can compute the so-called *socle degree*, i.e., the highest nonvanishing (complex) degree of the tautological ring of the corresponding subset of $\overline{\mathcal{M}}_{g,n}$.

```
sage: Rst = TautologicalRing(3, 1, moduli='st')
sage: Rst.socle_degree()
7
sage: Rsm = TautologicalRing(3, 1, moduli='sm')
sage: Rsm.socle_degree()
2
```

We will see in more detail in Section 3.4 how specifying the moduli affects computations.

3.2. ***Creating tautological classes.*** Each tautological class in admcycles has type `TautologicalClass`. We list in this section the different ways to enter tautological classes in the program. Depending on the example, some are more convenient than others.

As explained in Section 3.1 all computations happen in a given tautological ring (with a fixed base ring and fixed moduli). Once a tautological ring `R` for $\overline{\mathcal{M}}_{g,n}$ has been created as explained in Section 3.1, the fundamental class, boundary divisors as well as ψ, κ and λ-classes are predefined methods of the ring `R`.

- `R.fundamental_class()` returns the fundamental class of $\overline{\mathcal{M}}_{g,n}$.
- `R.separable_boundary_divisor(h,A)` gives the pushforward $\xi_*[\overline{\mathcal{M}}_\Gamma]$ of the boundary gluing map
$$\xi : \overline{\mathcal{M}}_\Gamma = \overline{\mathcal{M}}_{h,A\cup\{p\}} \times \overline{\mathcal{M}}_{g-h,(\{1,\dots,n\}\setminus A)\cup\{p'\}} \to \overline{\mathcal{M}}_{g,n},$$
where `A` can be a list, set or tuple[2] of numbers from 1 to n.
- `R.irreducible_boundary_divisor()` gives the pushforward $(\xi')_*[\overline{\mathcal{M}}_{g-1,n+2}]$ of the boundary gluing map
$$\xi' : \overline{\mathcal{M}}_{g-1,n+2} \to \overline{\mathcal{M}}_{g,n}$$
identifying the last two markings to a node. Note that, since ξ' has degree 2 onto its image, this gives *twice* the fundamental class of the boundary divisor of irreducible nodal curves.
- `R.psi(i)` gives the ψ-class ψ_i of marking i on $\overline{\mathcal{M}}_{g,n}$.
- `R.kappa(a)` gives the (Arbarello–Cornalba) κ-class κ_a on $\overline{\mathcal{M}}_{g,n}$.
- `R.lambdaclass(d)` gives the class λ_d on $\overline{\mathcal{M}}_{g,n}$, defined as the d-th Chern class $\lambda_d = c_d(\mathbb{E})$ of the Hodge bundle $\mathbb{E}$, the vector bundle on $\overline{\mathcal{M}}_{g,n}$ with fiber $H^0(C, \omega_C)$ over the point $(C, p_1, \dots, p_n) \in \overline{\mathcal{M}}_{g,n}$.

These tautological classes can be combined in the usual way by operations `+`, `-`, `*` and raising to an integral power `^`.

```
sage: R1 = TautologicalRing(3, 4)
sage: t1 = 3*R1.separable_boundary_divisor(1,(1,2)) - R1.psi(4)^2
sage: R2 = TautologicalRing(2, 1)
sage: t2 = -1/3*R2.irreducible_boundary_divisor() * R2.lambdaclass(1)
```

For user convenience, alternative functions are available to create the basic tautological classes (over the rationals and for the full moduli of stable curves), without having to create the tautological ring before. Each of these functions require extra arguments `g` and `n` to specify the genus and the number of marked points.

- `fundclass(g, n)`
- `sepbdiv(g1, A, g, n)`
- `irrbdiv(g, n)`
- `psiclass(i, g, n)`
- `kappaclass(a, g, n)`
- `lambdaclass(d, g, n)`

[2]Be careful that tuples of length 1 must be entered as `(a,)` in Python, instead of `(a)`.

```
sage: tt1 = 3 * sepbdiv(1, (1,2), 3, 4) - psiclass(4, 3, 4)^2

sage: t1 == tt1
True

sage: tt2 = -1/3*irrbdiv(2, 1) * lambdaclass(1, 2, 1)

sage: t2 == tt2
True
```

To enter more complicated classes coming from decorated boundary strata, it is often convenient to first list all such decorated strata forming the generating set of $\mathrm{RH}^{2r}(\overline{\mathcal{M}}_{g,n})$ in a specified degree r using `R.list_generators(r)` and then select the desired ones from the list (see below for an explanation of the notation). As a shortcut one can also directly use the function `tautgens(g,n,r)` to produce this list without having to create the ring `R` before.

```
sage: R = TautologicalRing(2, 0)

sage: R.list_generators(2)
[0] : Graph :      [2] [[]] []
Polynomial : (kappa_2)_0
[1] : Graph :      [2] [[]] []
Polynomial : (kappa_1^2)_0
[2] : Graph :      [1, 1] [[2], [3]] [(2, 3)]
Polynomial : (kappa_1)_0
[3] : Graph :      [1, 1] [[2], [3]] [(2, 3)]
Polynomial : psi_2
[4] : Graph :      [1] [[2, 3]] [(2, 3)]
Polynomial : (kappa_1)_0
[5] : Graph :      [1] [[2, 3]] [(2, 3)]
Polynomial : psi_2
[6] : Graph :      [0, 1] [[3, 4, 5], [6]] [(3, 4), (5, 6)]
Polynomial : 1
[7] : Graph :      [0] [[3, 4, 5, 6]] [(3, 4), (5, 6)]
Polynomial : 1
```

The list itself is created by `R.generators(r)`, from which one can then select the classes:

```
sage: L = R.generators(2)

sage: t3 = 2*L[3]+L[4]

sage: t3
Graph :      [1] [[2, 3]] [(2, 3)]
Polynomial : (kappa_1)_0
Graph :      [1, 1] [[2], [3]] [(2, 3)]
Polynomial : 2*psi_2
```

The output above should be interpreted as follows: each `TautologicalClass` consists of a sum of decorated boundary strata (represented by data type `decstratum`), which consist of a graph (datatype `StableGraph`) and a polynomial in κ and ψ-classes (datatype `KappaPsiPolynomial`).

To explain the notation above, let us look at the example of generator `L[3]`.

```
Graph :      [1, 1] [[2], [3]] [(2, 3)]
Polynomial : 1*psi_2^1
```

Its stable graph is represented by three lists.

(1) The first list `[1, 1]` are the genera of the vertices, so there are two vertices, both of genus 1. Note that vertices are numbered by $0, 1, 2, \ldots$, so in the above case, the vertices are numbers 0 and 1.

(2) The second list gives the legs (that is markings or half-edges) attached to the vertices, so vertex 0 carries the half-edge 2 and vertex 1 the half-edge 3.

(3) The third list gives the edges, that is half-edge pairs that are connected; in the above case, the two half-edges 2 and 3 form an edge, connecting the two vertices.

If we wanted to enter this `StableGraph` manually, we could use its constructor as follows:

```
sage: G = StableGraph([1,1],[[2],[3]],[(2,3)]); G
[1, 1] [[2], [3]] [(2, 3)]
```

The polynomial in κ and ψ is `1*psi_2^1` in this case, so the half-edge 2 on the first vertex carries a ψ-class. For the generator `L[4]` the polynomial looks like `1*(kappa_1^1)_0`, meaning that vertex 0 carries a class $\kappa_1^1 = \kappa_1$.

Finally, it is possible to manually enter tautological classes by constructing a stable graph `gamma` and calling the main constructor `R(gamma,kappa,psi)` of the tautological ring.

```
sage: R = TautologicalRing(3,2)

sage: g = StableGraph([2,0], [[1,3],[2,4,5,6]], [(3,4),(5,6)])

sage: R(g, kappa=[[],[1]], psi={1:2})
Graph :      [2, 0] [[1, 3], [2, 4, 5, 6]] [(3, 4), (5, 6)]
Polynomial : (kappa_1)_1*psi_1^2

sage: R(g, kappa=[[1,1],[]])
Graph :      [2, 0] [[1, 3], [2, 4, 5, 6]] [(3, 4), (5, 6)]
Polynomial : (kappa_1*kappa_2)_0
```

In the above call, the arguments `kappa` and `psi` are both optionals and specify the κ and ψ decorations on the stable graph `gamma`. We refer to the documentation of admcycles for more details.

3.3. *Basic operations.* Apart from the usual arithmetic operations, we can take forgetful pushforwards and pullbacks of tautological classes and also compute the degree of tautological zero-cycles. In particular, we can compute intersection numbers. Below, for the forgetful map $\pi : \overline{\mathcal{M}}_{1,3} \to \overline{\mathcal{M}}_{1,2}$ forgetting the marking 3 we verify the relations

$$\pi_* \psi_3^2 = \kappa_1 \quad \text{and} \quad \pi^* \psi_2 = \psi_2 - D_{0,\{2,3\}},$$

where $D_{0,\{2,3\}}$ is the class of the boundary divisor in $\overline{\mathcal{M}}_{1,3}$ where generically the curve splits into two components of genera 0, 1 connected at a node with the component of genus 0 carrying markings 2, 3.

```
sage: s1 = TautologicalRing(1, 3).psi(3)^2

sage: s1.forgetful_pushforward([3])
Graph :      [1] [[1, 2]] []
Polynomial : (kappa_1)_0

sage: s2 = TautologicalRing(1, 2).psi(2)

sage: s2.forgetful_pullback([3])
Graph :      [1] [[1, 2, 3]] []
Polynomial : psi_2
Graph :      [1, 0] [[1, 4], [2, 3, 5]] [(4, 5)]
Polynomial : -1
```

Using the method `evaluate` of `TautologicalClass`, we also compute intersection numbers of ψ-classes on $\overline{\mathcal{M}}_{g,n}$, the so-called *correlators* or *descendent integrals*. Here, given numbers $k_1, \ldots, k_n$ summing to $3g-3+n$ one can define these correlators $\langle \tau_{k_1} \cdots \tau_{k_n} \rangle_{g,n}$ as

$$\langle \tau_{k_1} \cdots \tau_{k_n} \rangle_{g,n} = \int_{\overline{\mathcal{M}}_{g,n}} \psi_1^{k_1} \cdots \psi_n^{k_n} . \tag{1}$$

Below we compute the intersection number

$$\langle \tau_0 \tau_1 \tau_2 \rangle_{1,3} = \int_{\overline{\mathcal{M}}_{1,3}} \psi_1^0 \psi_2 \psi_3^2 = \tfrac{1}{12}$$

and check that it agrees with the prediction $\langle \tau_0 \tau_1 \tau_2 \rangle_{1,3} = \langle \tau_0 \tau_2 \rangle_{1,2} + \langle \tau_1^2 \rangle_{1,2}$ by the string equation.

```
sage: R1 = TautologicalRing(1, 3)
sage: s3 = R1.psi(2) * R1.psi(3)^2
sage: s3.evaluate()
1/12
sage: R2 = TautologicalRing(1, 2)
sage: s4 = R2.psi(2)^2 + R2.psi(1) * R2.psi(2)
sage: s4.evaluate()
1/12
```

Instead of multiplying ψ-classes and evaluating by hand, we can also use the function `psi_correlator`, which takes as input the numbers $k_1, \ldots, k_n$ and outputs the correlator (1).

```
sage: psi_correlator(0,1,2)
1/12
```

Note that in the current version of `admcycles`, the list of tautological generators $[\Gamma_i, \alpha_i]$ in a tautological class is *not* automatically simplified by combining equivalent terms (since in general this requires testing graph isomorphisms between the Γ_i). When performing arithmetic operations with complicated tautological classes, this simplification can be manually triggered using the function `simplify`, as demonstrated below. For this toy example, we create two different but isomorphic stable graphs, convert them to tautological classes and form their sum `s`. After applying the method `simplify` they are recognized as equal, so that we obtain a shorter sum.

```
sage: gamma1 = StableGraph([1,2],[[3],[4]],[(3,4)]).to_tautological_class()
sage: gamma2 = StableGraph([2,1],[[5],[6]],[(5,6)]).to_tautological_class()
sage: s = gamma1 + gamma2; s
Graph :      [1, 2] [[3], [4]] [(3, 4)]
Polynomial : 1
Graph :      [2, 1] [[5], [6]] [(5, 6)]
Polynomial : 1
sage: s_simple = s.simplify(); s_simple
Graph :      [1, 2] [[2], [3]] [(2, 3)]
Polynomial : 2
```

In a future version of `admcycles` (after improving our algorithms for graph isomorphisms), we plan to automate this process.

3.4. ***A basis of the tautological ring and tautological relations.*** One can compute, using the function `generating_indices(g,n,r)`, the indices (for the list `tautgens(g,n,r)`) of a basis of $\mathrm{RH}^{2r}(\overline{\mathcal{M}}_{g,n})$, assuming that the generalized Faber–Zagier relations (see [Pixton 2012; Pandharipande et al. 2015; Janda 2017]) between the additive generators $[\Gamma, \alpha]$ give a complete set of relations between them. For many concrete examples of (g, n, r), this conjecture can be checked using `admcycles` via the function `FZ_conjecture_holds(g,n,r)` (see [Molcho et al. 2021, Appendix B] and the documentation of the function for more details). For the computation we show below, let us verify that the generalized Faber–Zagier relations for $\mathrm{RH}^{2\cdot 2}(\overline{\mathcal{M}}_{2,0})$ are complete:

```
sage: FZ_conjecture_holds(2,0,2)
True
```

If the relations are complete as discussed above, `Tautvecttobasis` converts a vector with respect to the whole generating set into a vector in this basis. The function `TautologicalClass.basis_vector(r)` converts a `TautologicalClass` into such a vector.

Continuing the example from Section 3.2 we see:

```
sage: generating_indices(2,0,2)
[0, 1]

sage: t3.basis_vector(2)
(-48, 22)
```

This means that the generators `L[0]` and `L[1]` form a basis of $\mathrm{RH}^4(\overline{\mathcal{M}}_2)$ and the `TautologicalClass` `t3=2*L[3]+L[4]` is equivalent to `-48*L[0]+22*L[1]`.

It is also possible to directly verify tautological relations using the built-in function `is_zero` of `TautologicalClass`. It checks if the tautological class is contained in the ideal generated by the 3-spin relations [Pandharipande et al. 2015] (what we call the generalized Faber–Zagier relations above). Below we verify the known relation $\kappa = \psi - \delta_0 \in R^1(\overline{\mathcal{M}}_{1,n})$ for $n = 4$. Here ψ is the sum of all ψ_i and δ_0 is the sum of all separating boundary divisors, i.e., those having a genus 0 component. For this, we list all stable graphs with one edge via `list_strata(g,n,1)`. We exclude the graph `gamma` with a self-loop by requiring that the number of vertices `gamma.numvert()` is at least 2. Then we can convert these graphs `bd` to tautological classes by using `to_tautological_class`.

```
sage: R = TautologicalRing(1, 4)

sage: bgraphs = [bd for bd in list_strata(1,4,1) if bd.num_verts() > 1]

sage: del0 = sum(bd.to_tautological_class() for bd in bgraphs)

sage: psisum = sum(R.psi(i) for i in range(1,5))

sage: rel = R.kappa(1) - psisum + del0

sage: rel.is_zero()
True
```

As a shorthand for `is_zero` one can also simply compare to the integer `0` as follows:

```
sage: rel == 0
True
```

It is also possible to express tautological classes in a basis of the tautological ring of suitable open subsets of $\overline{\mathcal{M}}_{g,n}$, e.g., to verify that some relation holds on the locus of compact type curves. This works with the optional argument `moduli` of `TautologicalRing` that was described in Section 3.1. We recall that moduli can be one of `'st'` (stable), `'tl'` (treelike), `'ct'` (compact type), `'rt'` (rational tails) or `'sm'` (smooth). The functions `basis_vector` and `is_zero` depend very much on the underlying moduli. For instance, we can verify the relation

$$\lambda_1 = \tfrac{B_2}{2}\kappa_1 = \tfrac{1}{12}\kappa_1 \in H^2(\mathcal{M}_g)$$

following from Mumford's computation [1983] in the case $g = 3$:

```
sage: R = TautologicalRing(3, 0, moduli='sm')

sage: R.kappa(1).basis_vector()
(1)

sage: R.lambdaclass(1).basis_vector()
(1/12)
```

It is also possible to start with a class on a bigger moduli (e.g., the default locus `'st'` of all stable curves) and check whether it vanishes on a smaller subset using the optional parameter `moduli` of the functions `is_zero` or `basis_vector`:

```
sage: R = TautologicalRing(2, 0)

sage: u = R.lambdaclass(2)

sage: u.is_zero()
False

sage: u.is_zero(moduli='ct')
True

sage: u.basis_vector()
(-3/2, 1/2)

sage: u.basis_vector(moduli='ct')
()
```

The vanishing here was expected as on $\mathcal{M}^{\mathrm{ct}}_{2,0}$ the tautological ring in degree 2 vanishes:

```
sage: R = TautologicalRing(2, 0, moduli='ct')

sage: R.socle_degree()
1
```

In practice, much of the time in some computations is spent on calculating generalized Faber–Zagier relations between tautological cycles on $\overline{\mathcal{M}}_{g,n}$. However, once computed, the relations can be saved to a file and reloaded in a later session using the functions `save_FZrels()` and `load_FZrels()`. Careful: the function `save_FZrels()` creates (and overwrites previous version of) a file `new_geninddb.pkl` which, depending on the previous computations, can be quite large.

3.5. ***Pulling back tautological classes to the boundary.*** Recall that for a stable graph Γ we have a gluing map

$$\xi_\Gamma : \overline{\mathcal{M}}_\Gamma = \prod_{i=1}^{m} \overline{\mathcal{M}}_{g(v_i),n(v_i)} \to \overline{\mathcal{M}}_{g,n} \tag{2}$$

taking one stable curve for each of the vertices $v_1, \dots, v_m$ of Γ and gluing them together according to the edges of Γ. By [Graber and Pandharipande 2003, Appendix A], the pullback of a tautological class under ξ_Γ is contained in the tensor product of the tautological rings of the factors $\overline{\mathcal{M}}_{g(v_i),n(v_i)}$ above, and this operation is implemented in `admcycles`.

Below we pull back a generator of $\mathrm{RH}^4(\overline{\mathcal{M}}_4)$ to the boundary divisor with genus partition $4 = 2+2$. This produces an element of type `prodtautclass`, a tautological class on a product of moduli spaces, in this case $\overline{\mathcal{M}}_{2,1} \times \overline{\mathcal{M}}_{2,1}$. Two elements on the same product of spaces can be added and multiplied and further operations like pushforwards under (partial) gluing maps are supported. More details are given in the documentation of the class `prodtautclass`.

Below, we want to express the pullback to $\overline{\mathcal{M}}_{2,1} \times \overline{\mathcal{M}}_{2,1}$ in terms of a basis of $H^2(\overline{\mathcal{M}}_{2,1} \times \overline{\mathcal{M}}_{2,1})$ obtained from the preferred bases of the factors $H^*(\overline{\mathcal{M}}_{2,1})$ given by `generating_indices`. We can either represent the result as a list of matrices (giving the coefficients in the tensor product bases) or as a combined vector (using the option `vecout=true`).

```
sage: bdry=StableGraph([2,2],[[1],[2]],[(1,2)])

sage: generator=tautgens(4,0,2)[3]; generator
Graph :      [1, 3] [[2], [3]] [(2, 3)]
Polynomial : psi_3

sage: pullback=bdry.boundary_pullback(generator)

sage: pullback.totensorTautbasis(2)
[
                               [-3]
                               [ 1]
                     [0 0 0]   [-3]
                     [0 0 0]   [ 7]
[-3  1 -3  7  1], [0 0 0], [ 1]
]

sage: pullback.totensorTautbasis(2,vecout=true)
(-3, 1, -3, 7, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -3, 1, -3, 7, 1)
```

3.6. ***Pushing forward classes from the boundary.*** The pushforward under the map ξ_Γ in (2) sends a product of tautological classes on the factors $\overline{\mathcal{M}}_{g(v_i),n(v_i)}$ to a tautological class of $\overline{\mathcal{M}}_{g,n}$. This operation is implemented by the function `boundary_pushforward` of `StableGraph`.

That is to say, if `Gamma` is a `StableGraph` and if `[c1, ..., cm]` is a list whose i-th element `ci` is a `TautologicalClass` on the i-th factor $\overline{\mathcal{M}}_{g(v_i),n(v_i)}$ of $\overline{\mathcal{M}}_\Gamma$, then

```
Gamma.boundary_pushforward([c1, ..., cm])
```

is the pushforward of the product of the `ci`. Here, the markings for the class `ci` are supposed to go from 1 to $n(v_i)$, where the j-th marking corresponds to leg number j on the i-th vertex of `Gamma`.

As an illustration, we verify that the package correctly computes the excess intersection formula proved in [Graber and Pandharipande 2003] for the self-intersection of a boundary divisor in $\overline{\mathcal{M}}_{3,3}$.

```
sage: B=StableGraph([2,1],[[4,1,2],[3,5]],[(4,5)])

sage: Bclass = B.boundary_pushforward() # class of undecorated boundary divisor

sage: si1 = B.boundary_pushforward([fundclass(2,3),-psiclass(2,1,2)])

sage: si1
Graph :      [2, 1] [[4, 1, 2], [3, 5]] [(4, 5)]
Polynomial : -psi_5

sage: si2 = B.boundary_pushforward([-psiclass(1,2,3),fundclass(1,2)])
```

```
sage: si2
Graph :      [2, 1] [[4, 1, 2], [3, 5]] [(4, 5)]
Polynomial : -psi_4

sage: (Bclass*Bclass-si1-si2).is_zero()
True
```

Note that, e.g., for the term `si2` we needed to hand the function the term `-psiclass(1,2,3)` in the first vertex, since in the graph `B` the half-edge `4` is leg number `1` in the list of legs at the first vertex (and we have $(g(v_1), n(v_1)) = (2, 3)$ for this vertex).

4. Special cycle classes.

Beyond the already mentioned standard tautological classes ψ_i, κ_a and λ_d and boundaries from Section 3.2, `admcycles` provides more advanced constructions that we describe now. The corresponding functions are summarized here:

`TautologicalRing` method	standalone function	manual section
`double_ramification_cycle`	`DR_cycle`	Section 4.1
`theta_class`	`ThetaClass`	Section 4.1
`differential_stratum`	`Strataclass`	Section 4.2
`generalized_lambda`	`generalized_lambda`	Section 4.3
`hyperelliptic_cycle`	`Hyperell`	Section 4.4
`bielliptic_cycle`	`Biell`	Section 4.4

A convenient way to find out about tautological class constructions is to use the *tab completion* feature of SageMath. When you enter a part of a name and press the tab key (denoted `<TAB>` below) the program will show you all available completions. It can be used to discover the names in the admcycles module.

```
sage: import admcycles

sage: admcycles.<TAB>
admcycles.Biell        admcycles.DR_phi              ...
admcycles.DR           admcycles.DRpoly              ...
admcycles.DR_cycle     admcycles.FZ_conjecture_holds ...
admcycles.DR_cycle_old admcycles.GRRcomp             ...
```

Similarly one can discover the methods of `TautologicalRing` starting with the letter `d`:

```
sage: R = TautologicalRing(2, 2)

sage: R.d<TAB>
R.differential_stratum      R.dump
R.dimension                 R.dumps
R.double_ramification_cycle
```

4.1. *Double ramification cycles.*

A particularly interesting family of cycles on $\overline{\mathcal{M}}_{g,n}$ is given by the double ramification cycles. Fixing g, n they are indexed by nonnegative integers $k, d \geq 0$ and a tuple $A = (a_1, a_2, \dots, a_n)$ of integers summing to $k(2g-2+n)$.

The classical double ramification cycle (for $k = 0, d = g$)

$$\mathrm{DR}_g(A) \in H^{2g}(\overline{\mathcal{M}}_{g,n})$$

has been defined as the pushforward of the virtual fundamental class of a space of maps to rubber $\mathbb{P}^1$

relative to 0, ∞ with tangency conditions at 0, ∞ specified by the vector A (see [Li and Ruan 2001; Li 2002; 2001; Graber and Vakil 2005]). In [Janda et al. 2017] it is shown that this cycle is tautological and an explicit formula in terms of tautological generators is provided.

More precisely, for g, n, k, d and A with A a partition of $k(2g-2+n)$, the paper constructs an explicit tautological class

$$P_g^{d,r,k}(A) \in \mathbb{Q}[r] \otimes_{\mathbb{Q}} \mathrm{RH}^{2d}(\overline{\mathcal{M}}_{g,n})$$

with coefficients being polynomials in a formal variable r. We obtain a usual tautological class $P_g^{d,k}(A) \in \mathrm{RH}^{2d}(\overline{\mathcal{M}}_{g,n})$ by setting $r = 0$ in these polynomial coefficients. Then it is shown ([Janda et al. 2017, Theorem 1]) that in the special case $k = 0,\ d = g$, this gives a formula for the double ramification cycle

$$\mathrm{DR}_g(A) = 2^{-g} P_g^{g,k}(A).$$

While this demonstrates that the cycle $P_g^{d,k}(A)$ is useful for $k = 0$, $d = g$, it has many interesting properties for other values of k, d:

- For k arbitrary and $d = 1$, the restriction of $2^{-1} P_g^{1,k}(A)$ to the compact-type locus $\mathcal{M}_{g,n}^{\mathrm{ct}}$ gives the pullback of the theta divisor on the universal Jacobian $\mathcal{J}$ over $\mathcal{M}_{g,n}^{\mathrm{ct}}$ under the extension of the Abel–Jacobi section
$$\begin{aligned} \mathcal{M}_{g,n} &\to \mathcal{J}, \\ (C, p_1, \dots, p_n) &\mapsto (\omega_C^{\log})^{\otimes k}\Big(-\sum_{i=1}^{n} a_i p_i\Big); \end{aligned}$$
see [Hain 2013; Grushevsky and Zakharov 2014].
- For k arbitrary and $d = g$, various geometric definitions of a double ramification cycle have been put forward and an equality with $2^{-g} P_g^{g,k}(A)$ was conjectured in [Farkas and Pandharipande 2018; Schmitt 2018] (see [Holmes and Schmitt 2019, Section 1.6] for an overview of the various definitions). Recently, this conjecture was proven in [Bae et al. 2020] based on earlier results of [Holmes and Schmitt 2019].
- For k arbitrary and $d > g$, the class $P_g^{d,k}(A)$ vanishes by [Clader and Janda 2018].

In `admcycles`, the formula for $P_g^{d,k}(A)$ has been implemented. The function `DR_cycle(g,A,d,k)` returns the cycle $2^{-d} P_g^{d,k}(A)$. The factor 2^{-d} was chosen such that `DR_cycle(g,A)` indeed gives the cycle $\mathrm{DR}_g(A)$. With the option `rpoly=True`, it is even possible to compute the cycle $2^{-d} P_g^{d,r,k}(A)$ whose coefficients are polynomials in the variable r.

As an application, we can verify the result from [Holmes et al. 2019] that DR cycles satisfy the multiplicativity property

$$\mathrm{DR}_g(A) \cdot \mathrm{DR}_g(B) = \mathrm{DR}_g(A) \cdot \mathrm{DR}_g(A+B) \in H^{4g}(\mathcal{M}_{g,n}^{tl})$$

on the locus $\mathcal{M}_{g,n}^{tl}$ of treelike curves but *not* on the locus of all stable curves, in the example given in [Holmes et al. 2019, Section 8].

```
sage: A=vector((2,4,-6)); B=vector((-3,-1,4))
sage: diff = DR_cycle(1,A)*DR_cycle(1,B)-DR_cycle(1,A)*DR_cycle(1,A+B)
sage: diff.is_zero(moduli='tl')
True
sage: diff.is_zero(moduli='st')
False
```

In fact, using that the cycle $\mathrm{DR}_g(A)$ is polynomial in the entries of the vector A (i.e., a tautological class with polynomial coefficients), we can check multiplicativity for all vectors A, B in the case $g = 1, n = 3$. To gain access to the polynomial-valued DR cycle, we define a polynomial ring and call `DR_cycle` with a vector A having as coefficients the generators of this ring:

```
sage: R.<a1,a2,a3,b1,b2,b3> = PolynomialRing(QQ,6)
sage: A = vector((a1,a2,a3)); B = vector((b1,b2,b3))
sage: diff = DR_cycle(1,A)*DR_cycle(1,B)-DR_cycle(1,A)*DR_cycle(1,A+B)
sage: diff.is_zero(moduli='tl')
True
```

As a second application, we can verify the formula from [Buryak and Rossi 2021, Theorem 2.1] for intersection numbers of two DR cycles with λ_g on $\overline{\mathcal{M}}_{g,3}$ in the case $g = 1$:

```
sage: intersect = DR_cycle(1,A)*DR_cycle(1,B)*lambdaclass(1,1,3)
sage: f = intersect.evaluate(); factor(f)
(1/216) * (a2*b1 - a3*b1 - a1*b2 + a3*b2 + a1*b3 - a2*b3)^2
sage: g = f.subs({a3:-a1-a2,b3:-b1-b2}); factor(g)
(1/24) * (a2*b1 - a1*b2)^2
```

The formula of the cycle $P_g^{d,r,k}(A)$ in [Janda et al. 2017] is obtained as a simplification (modulo r) of a cycle

$$r^{2d-2g+1}\epsilon_* c_d(-R^*\pi_*\mathcal{L}) \tag{3}$$

appearing in [Janda et al. 2017, Corollary 4, Proposition 5] (see there for the notation). The cycle (3) is often called a *Chiodo class* and it is relevant for certain computations (see [Borot et al. 2020; Do and Lewański 2020]). Since the latest version of `admcycles`, the cycle (3) can be obtained using the optional parameters `chiodo_coeff = True` and `r_coeff` of `DR_cycle`, which evaluates the expression (3) at the value `r_coeff` of r.

```
sage: g=2; A=(5,-1); d=2; k=1
sage: Chiodo = DR_cycle(g,A,d,k,chiodo_coeff=True,r_coeff=7)
```

As a special case of this formula, we can obtain the cycle class $\theta_{g,n} \in R^*(\overline{\mathcal{M}}_{g,n})$ described in [Norbury 2017], which is accessible via the function `ThetaClass`.

```
sage: T = ThetaClass(1,1)
sage: T == 3*psiclass(1,1,1)
True
```

codim	$\boldsymbol{m}=0$	$\boldsymbol{m}=k\cdot\boldsymbol{m}'$ for $\boldsymbol{m}'\in\mathbb{Z}^n_{\geq 0}$	$\boldsymbol{m}\neq k\cdot\boldsymbol{m}'$ for $\boldsymbol{m}'\in\mathbb{Z}^n_{\geq 0}$
$k=0$	0	0	g
$k=1$	0	$g-1$	g
$k>1$	0	$g-1$ and g	g

Table 1. Dimension theory of $\overline{\mathcal{H}}^k_g(\boldsymbol{m})$. Note that for $k>1$ and $\boldsymbol{m}=k\cdot\boldsymbol{m}'$ with $\boldsymbol{m}'\in\mathbb{Z}^n_{\geq 0}$, the set $\overline{\mathcal{H}}^1_g(\boldsymbol{m}')\subset\overline{\mathcal{H}}^k_g(\boldsymbol{m})$ is a union of components of codimension $g-1$ in $\overline{\mathcal{M}}_{g,n}$, with all other components of $\overline{\mathcal{H}}^k_g(\boldsymbol{m})$ having pure codimension g.

4.2. ***Strata of k-differentials.*** Let $g,n,k\geq 0$ with $2g-2+n>0$ and let $\boldsymbol{m}=(m_1,\dots,m_n)\in\mathbb{Z}^n$ with $\sum_i m_i=k(2g-2)$. Consider the subset

$$\mathcal{H}^k_g(\boldsymbol{m})=\left\{(C,p_1,\dots,p_n)\in\mathcal{M}_{g,n}:\omega_C^{\otimes k}\left(\sum_{i=1}^n m_ip_i\right)\cong\mathcal{O}_C\right\}\subset\mathcal{M}_{g,n}.$$

Denote by $\overline{\mathcal{H}}^k_g(\boldsymbol{m})$ the closure of $\mathcal{H}^k_g(\boldsymbol{m})$ inside $\overline{\mathcal{M}}_{g,n}$. Since the above equality of line bundles is equivalent to the existence of a meromorphic k-differential η on C with zeros and poles exactly at the points p_i with multiplicities m_i, the subsets $\overline{\mathcal{H}}^k_g(\boldsymbol{m})$ are called *strata of k-differentials.*

These strata are of interest in algebraic geometry, the theory of flat surfaces and Teichmüller dynamics and have been studied intensely in the past. Elements appearing in the boundary have been classified in [Bainbridge et al. 2018; 2019b] and a smooth, modular compactification has been constructed in [Bainbridge et al. 2019a]. The dimension of $\overline{\mathcal{H}}^k_g(\boldsymbol{m})$ depends on k, $\boldsymbol{m}$ as in Table 1 (see, e.g., [Farkas and Pandharipande 2018; Schmitt 2018]).

For $k\geq 1$, [Farkas and Pandharipande 2018; Schmitt 2018] present conjectural relations between the fundamental classes $[\overline{\mathcal{H}}^k_g(\boldsymbol{m})]$ and the formulas for the double ramification cycles proposed by Pixton (see Section 4.1). The conjectures were recently proven in [Bae et al. 2020] based on results from [Holmes and Schmitt 2019]. As explained in the papers, these conjectures can be used to recursively determine all cycles

- $[\overline{\mathcal{H}}^k_g(\boldsymbol{m})]\in\mathrm{RH}^{2g}(\overline{\mathcal{M}}_{g,n})$ for $k\geq 1$ and $\boldsymbol{m}\neq k\boldsymbol{m}'$ for some $\boldsymbol{m}'\in\mathbb{Z}^n_{\geq 0}$,
- $[\overline{\mathcal{H}}^1_g(\boldsymbol{m})]\in\mathrm{RH}^{2g-2}(\overline{\mathcal{M}}_{g,n})$ for $k=1$ and $\boldsymbol{m}\in\mathbb{Z}^n_{\geq 0}$.

These recursive algorithms have been implemented in the function `Strataclass(g,k,m)`, where as above `m` is a tuple of n integers summing to $k(2g-2)$.

As a small application, we can check that the stratum class $[\overline{\mathcal{H}}^1_2((3,-1))]$ vanishes (the stratum is empty since by the residue theorem there can be no meromorphic differential with a single, simple pole). Also, the stratum $\overline{\mathcal{H}}^1_2((2))$ exactly equals the class of the locus of genus 2 curves with a marked Weierstrass point, which can be computed by the function `Hyperell` (see below for details).

```
sage: L=Strataclass(2,1,(3,-1)); L.is_zero()
True
```

```
sage: L=Strataclass(2,1,(2,)); (L-Hyperell(2,1)).is_zero()
True
```

4.3. ***Generalized lambda classes.*** Let $\pi : \mathcal{C}_{g,n} \to \overline{\mathcal{M}}_{g,n}$ be the universal curve and assume $n \geq 1$. Every divisor of $\mathcal{C}_{g,n}$, up to pullback of divisors on $\overline{\mathcal{M}}_{g,n}$, takes the form

$$D = l\tilde{K} + \sum_{p=1}^{n} d_p \sigma_p + \sum_{\substack{h \leq g, \\ 1 \in S \subset [n]}} a_{h,S} C_{h,S}$$

for some integers l, d_p, $a_{h,S}$. Here $\tilde{K} = c_1(\omega_\pi)$ is the first Chern class of the relative dualizing sheaf, σ_p is the class of the p-th section and

$$C_{h,S} = \xi_*[\overline{\mathcal{M}}_{h,S\cup\{\bullet\}} \times \overline{\mathcal{M}}_{g-h,[n]\setminus S\cup\{\star,x\}}] \in \mathrm{CH}^1(\overline{\mathcal{M}}_{g,[n]\cup\{x\}}) = \mathrm{CH}^1(\mathcal{C}_{g,n}).$$

The fact that every divisor D on $\mathcal{C}_{g,n}$ can be written in this form up to pullbacks from $\overline{\mathcal{M}}_{g,n}$ follows from the identification $\mathcal{C}_{g,n} \cong \overline{\mathcal{M}}_{g,n+1}$ and the computation of the Picard group of the moduli spaces of stable curves due to Harer [1983] and Arbarello and Cornalba [1987]. In [Pagani et al. 2020] a formula is given for the Chern character $\mathrm{ch}(R^\bullet\pi_*\mathcal{O}(D))$. This Chern character can be computed up to degree `dmax` using `generalized_chern_hodge(l,d,a,dmax,g,n)`. It takes as input an integer `l`, a list `d=[d1,...,dn]` of the integers d_i and a list of triples `a=[[h1,S1,ahS1],...,[hn,Sn,ahSn]]` where the `ahSi` are the integers $a_{h,S}$ above (given in any order). It is enough to just include the triples `[h,S,ahS]` for which $a_{h,S}$ is nonzero.

Using `generalized_lambda(i,l,d,a,g,n)` the Chern class $c_i(-R^\bullet\pi_*\mathcal{O}(D))$ can be computed directly. In particular when $l = 1$ and the d_p and $a_{h,S}$ are zero, this equals the normal λ class.

```
sage: g=3;n=1
sage: l=1;d=[0];a=[]
sage: s=lambdaclass(2,g,n)
sage: t=generalized_lambda(2,l,d,a,g,n)
sage: (s-t).is_zero()
True
```

Let $d_1, ..., d_n$ be integers such that $\sum_{i=1}^n d_i$ is divisible by $2g-2$ and let $\phi \in V^0_{g,n}$ be an element of the stability space $V^0_{g,n}$ defined in [Kass and Pagani 2019, Definition 3.2]. This ϕ is an assignment which given a stable curve $(C, p_1, \ldots, p_n) \in \overline{\mathcal{M}}_{g,n}$ associates a real number $\phi(C, p_1, \ldots, p_n)_{C'}$ to every irreducible component C' of C. These numbers must sum to zero as C' runs through the components of C; they only depend on the stable graph of C and must be compatible with degenerations of curves. Given this data, Kass and Pagani construct a compactification $\overline{\mathcal{J}}_{g,n}(\phi)$ of the universal Jacobian over $\overline{\mathcal{M}}_{g,n}$.

Let now $l = \sum_{i=1}^n d_i/(2g-2)$ and let $a_{h,S}$ be integers such that

$$D(\phi) = l\tilde{K} + \sum d_i\sigma_i + \sum a_{h,S}(\phi)C_{h,S}$$

is ϕ-stable on the locus of stable curves with one node (for definitions, see [Kass and Pagani 2019] or [Pagani et al. 2020]). For the shifted[3] vector $A = (d_1 + l, \ldots, d_n + l)$, [Holmes et al. 2018] proves

[3]This shift is due to the fact that the literature on double ramification cycles uses the "log-convention", i.e., the entries of the input sum to $l(2g-2+n)$.

an equality

$$\mathrm{DR}_g(A)|_{U(\phi)} = c_g(-R^\bullet \pi_* \mathcal{O}(D(\phi)))|_{U(\phi)} \tag{4}$$

on the largest open locus $U(\phi) \subset \overline{\mathcal{M}}_{g,n}$ where the Abel–Jacobi section

$$s_{l,d}(\phi) : \overline{\mathcal{M}}_{g,n} \dashrightarrow \overline{\mathcal{J}}_{g,n}(\phi),\ (C, p_1, \ldots, p_n) \mapsto \omega_C^{\otimes l}\left(-\sum_{i=1}^{n} d_i p_i\right)$$

extends to a morphism. In particular, $U(\phi)$ always includes $\mathcal{M}^{\mathrm{ct}}_{g,n}$ and equals $\overline{\mathcal{M}}_{g,n}$ if and only if l, d are trivial or $l(2g-2) = 0$ and $d = [0, ..., \pm 1, ..., \mp 1, ..., 0]$. See [Pagani et al. 2020, Section 4.3] for more details.

The function DR_phi(g,d) computes $c_g(-R^\bullet \pi_* \mathcal{O}(D(\phi)))$. We can verify equality (4).

```
sage: g=2;d=[1,-1]
sage: (DR_cycle(g,d)-DR_phi(g,d)).is_zero()
True
```

We also see that equality does not always hold over all of $\overline{\mathcal{M}}_{g,n}$ but it does hold over $\mathcal{M}^{\mathrm{ct}}_{g,n}$.

```
sage: g=2;d=[2,-2]
sage: (DR_cycle(g,d)-DR_phi(g,d)).basis_vector()
(12, -4, 14, 7, -40, -10, -14, -12, 28, -4, 6, -1, 4, 0)
sage: (DR_cycle(g,d)-DR_phi(g,d)).basis_vector(moduli='ct')
(0, 0, 0, 0, 0)
```

4.4. *Admissible cover cycles.*

Hyperelliptic and bielliptic cycles. Before we go into details of how to specify general admissible cover cycles, let us mention the important cases of hyperelliptic and bielliptic cycles.

Recall that a smooth curve C is called *hyperelliptic* if C admits a double cover $C \to \mathbb{P}^1$ and is called *bielliptic* if it admits a double cover $C \to E$ of *some* smooth genus 1 curve E. In both cases we have an involution $C \to C$ that exchanges the two sheets of the cover. Given $g, n, m \geq 0$ with $n \leq 2g+2$ and $2g-2+n+2m > 0$, we have the locus $\overline{H}_{g,n,2m} \subset \overline{\mathcal{M}}_{g,n+2m}$ which is the closure of the locus of smooth curves $(C, p_1, \ldots, p_n, q_1, q_1', \ldots, q_m, q_m')$ such that C is hyperelliptic with $p_1, \ldots, p_n$ fixed points of the hyperelliptic involution and the pairs q_i, q_i' being exchanged by this involution. An analogous definition gives the locus $\overline{B}_{g,n,2m} \subset \overline{\mathcal{M}}_{g,n+2m}$ as the closure of the set of bielliptic curves with $n \leq 2g-2$ fixed points and m pairs of points forming orbits under the bielliptic involution.

Then the fundamental class of the (reduced) loci $\overline{H}_{g,n,2m}$ and $\overline{B}_{g,n,2m}$ can (in many cases) be computed by the functions Hyperell(g,n,m) and Biell(g,n,m) of our program.

As an example, we compute the class $[\overline{H}_3] \in \mathrm{RH}^2(\overline{\mathcal{M}}_3)$ and verify that we obtain the known result

$$[\overline{H}_3] = 9\lambda - \delta_0 - 3\delta_1,$$

where δ_0 is the class of the divisor of irreducible nodal curves and δ_1 is the divisor of curves with a separating node between a genus 1 and a genus 2 component.

```
sage: H = Hyperell(3,0,0)
sage: H.basis_vector()
(3/4, -9/4, -1/8)
sage: R = TautologicalRing(3, 0)
sage: H2 = 9*R.lambdaclass(1)-(1/2)*R.irreducible_boundary_divisor()
           -3*R.separable_boundary_divisor(1,())
sage: H2.basis_vector()
(3/4, -9/4, -1/8)
```

Here we need to divide `irrbdiv()` by two, the degree of the corresponding gluing map.

Creating and identifying general admissible cover cycles. Generalizing the case of hyperelliptic and bielliptic cycles, we can consider loci of curves C admitting a cover $C \to D$ to a second curve D such that the cover is Galois with respect to a fixed finite group G. Cycles defined via such covers were studied in [Schmitt and van Zelm 2020]. In general, such an admissible cover cycle is specified by the genus g of the curve C, the finite group G, and monodromy data (we refer the reader to [Schmitt and van Zelm 2020, Section 1.3] for the precise definitions). Currently, intersections are only implemented for cyclic groups. Below we will study bielliptic curves in genus 2, which are double covers of elliptic curves branched over two points. As a first step we enter the monodromy data.

```
sage: G=PermutationGroup([(1,2)])
sage: list(G)
[(), (1,2)]
sage: H=HurData(G,[G[1],G[1]])
```

The function `HurData` takes the group G as the first argument and as the second a list of group elements $\alpha \in G$, each of which corresponds to the G-orbit of some marking $p \in C$. Here α is a generator of the stabilizer of p under the group action $G \curvearrowright C$, which gives the monodromy around p. In other words, the natural action of the stabilizer $G_p = \langle \alpha \rangle$ on a tangent vector $v \in T_pC$ is given by

$$\alpha.v = \exp(2\pi i/\mathrm{ord}(h))v.$$

Thus in the example above, we have two markings, both with stabilizer generated by `G[1]=(1,2)` which acts by multiplication by -1 on the tangent space.

To identify the admissible cover cycle (inside the moduli space $\overline{\mathcal{M}}_{g,n}$ with n the total number of marked points from the monodromy data) in terms of tautological classes, one can use the function `Hidentify`. It pulls back the admissible cover cycle to all boundary divisors and (recursively) identifies the pullback itself in terms of tautological classes. It compares this pullback to the pullback of a basis of the tautological ring. Often this pullback map is injective in cohomology so that one can then write the admissible cover cycle in terms of the basis using linear algebra. Sometimes, it is necessary to additionally intersect with some monomials in κ and ψ-classes.

To apply `Hidentify` one gives the genus and the monodromy data as arguments. The standard output format is an instance of the class `TautologicalClass`. For users familiar with Pixton's implementation of the tautological ring, there is the option `vecout=true` which returns instead a vector with respect to the generating set of the tautological ring provided by this program.

```
sage: vbeta = Hidentify(2, H, vecout=true)

sage: vector(vbeta)
(517/4, -33, 11/4, 243/4, -125/4, 15/2, 41/4, 125, 99/4, -41, -1137/4, -285/4, 0, 0, 0, 0, 0,
 0, -57/8, -3/8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

This output means, specifically, that inside $\overline{\mathcal{M}}_{2,2}$ the locus of bielliptic curves with the two points fixed by the involution being the marked points is given (in the generating set `gens=all_strata(2,3,(1,2))` produced by Pixton's program) as $517/4 \cdot$ `gens[0]` $- 33 \cdot$ `gens[1]` $+ \dots$.

If we instead wanted to have bielliptic curves with two marked fixed-points of the involution and one pair of markings that are exchanged by the involution, we would need to use the monodromy data

```
sage: H2=HurData(G,[G[1],G[1],G[0]])
```

in which case `Hidentify(2,H2)` would live inside $\mathrm{RH}^8(\overline{\mathcal{M}}_{2,4})$.

If we only want to remember a subset of the markings, we can use the optional parameter `marking` to give this subset. For instance, the command `Hidentify(2,H,markings=[])` would give the pushforward of `Hidentify(2,H)` in $\overline{\mathcal{M}}_{2,2}$ to the space $\overline{\mathcal{M}}_2$ under the forgetful morphism (see also Section 4.4).

Example: specifying and identifying $[\bar{B}_2]$ *by hand.* The locus $\bar{B}_2 \subset \overline{\mathcal{M}}_2$ of bielliptic curves is a divisor. A bielliptic genus 2 curve is ramified over two points. In the following we use the methods of the previous section to identify its cycle class.

Now when treating admissible cover cycles in general, our program a priori handles the cycle where all possible ramification points are marked. In this case, this is the cycle $[\bar{B}_{2,2,0}] \in \mathrm{RH}^6(\overline{\mathcal{M}}_{2,2})$ of bielliptic curves C with the two ramification points p_1, p_2 marked. By specifying `markings=[]` when calling `Hidentify`, we tell it to remember none of the markings, in other words to push forward under the map $\pi : \overline{\mathcal{M}}_{2,2} \to \overline{\mathcal{M}}_2$ forgetting the markings.

```
sage: G = PermutationGroup([(1,2)])

sage: H = HurData(G, [G[1],G[1]])

sage: B22 = Hidentify(2, H, markings=[])

sage: B22.basis_vector(1)
(30, -9)
```

We compare the result with the known formulas for $[\bar{B}_2]$. For δ_0 the class of the irreducible boundary of $\overline{\mathcal{M}}_2$ and δ_1 the class of the boundary divisor with genus-splitting $(1, 1)$, it is known that $[\bar{B}_2] = \frac{3}{2}\delta_0 + 6\delta_1$ (see [Faber 1996]). If we want to enter this combination of δ_0 and δ_1, we have to be careful about conventions, though: the corresponding gluing maps $\xi : \overline{\mathcal{M}}_{1,2} \to \overline{\mathcal{M}}_2$ and $\xi' : \overline{\mathcal{M}}_{1,1} \times \overline{\mathcal{M}}_{1,1} \to \overline{\mathcal{M}}_2$ both have degree 2. This corresponds to the fact that the associated stable graphs both have an automorphism group of order 2. Hence we have to divide by a factor of two and obtain

```
sage: B22_formula = 3/4*irrbdiv(2,0) + 3*sepbdiv(1,(),2,0)

sage: B22_formula.basis_vector(1)
(15/2, -9/4)
```

We see that up to a factor of 4 the two vectors `(30, -9)` and `(15/2, -9/4)` agree. Where does this factor come from?

For this, recall that the cycle `B22` above is equal to $\pi_*[\bar{B}_{2,2,0}]$. Since for the generic bielliptic curve C there are two choices of orderings for marking p_1, p_2, this explains a factor of 2. On the other hand, the hyperelliptic involution $\sigma : C \to C$ on C exchanges p_1 and p_2. Thus $\sigma \in \mathrm{Aut}(C)$, but $\sigma \notin \mathrm{Aut}(C, p_1, p_2)$. This missing automorphism factor explains another factor of 2 in the pushforward under π, so in fact $[\bar{B}_2] = \frac{1}{4}\pi_*[\bar{B}_{2,2,0}]$.

Note that since the cycles of bielliptic loci are implemented via the function `Biell`, we could have taken a shortcut above.

```
sage: B = Biell(2,0,0)
sage: B.basis_vector()
(15/2, -9/4)
```

As an application, we can check the Hurwitz–Hodge integral

$$\int_{[\bar{B}_{2,2,0}]} \lambda_2\lambda_0 = \int_{\pi_*[\bar{B}_{2,2,0}]} \lambda_2 = \tfrac{1}{48}$$

predicted by [Pandharipande and Tseng 2019].

```
sage: (B22 * lambdaclass(2,2,0)).evaluate()
1/48
```

The corresponding integrals for $g = 3, 4$ have also been verified like this, but the amount of time and memory needed grows drastically.

We can also check the Hurwitz–Hodge integral

$$\int_{[\overline{\mathcal{H}}_{2,\mathbb{Z}/3\mathbb{Z},((1,2,3)^2,(1,3,2)^2)}]} \lambda_1 = \tfrac{2}{9}$$

of λ_1 against the locus of genus 2 curves admitting a cyclic triple cover of a genus 0 curve with two points of ramification $(1, 2, 3) \in \mathbb{Z}/3\mathbb{Z}$ and two points of ramification $(1, 3, 2) \in \mathbb{Z}/3\mathbb{Z}$, computed in [Owens and Somerstep 2019, Section 5].

```
sage: G = PermutationGroup([(1,2,3)]); sorted(list(G))
[(), (1,2,3), (1,3,2)]
sage: H = HurData(G,[G[1],G[1],G[2],G[2]]) #n=2, m=2
sage: t = Hidentify(2,H,markings=[])
sage: (t*lambdaclass(1,2,0)).evaluate()
2/9
```

Note that while originally the cycle $[\overline{\mathcal{H}}_{2,\mathbb{Z}/3\mathbb{Z},((1,2,3)^2,(1,3,2)^2)}]$ lives in $\overline{\mathcal{M}}_{2,4}$, since we intersect with λ_1 which is a pullback from $\overline{\mathcal{M}}_2$ we can specify `markings=[]` above to compute the pushforward `t` of this cycle to $\overline{\mathcal{M}}_2$ before intersecting. This significantly reduces the necessary computation time.

ACKNOWLEDGEMENTS. We are indebted to Aaron Pixton for letting us use and modify a previous implementation of operations in the tautological ring by him as well as pointing out several issues in `admcycles` which have now been fixed. We thank Frédéric Chapoton, Samuel Lelièvre and Jonathan Zachhuber for many valuable contributions. We are very grateful to the anonymous referees for many helpful comments, greatly improving the clarity of the paper and suggesting many improvements of the code.

We thank Harald Schilly and the team of CoCalc as well as Andrey Novoseltsev and the team of SageMathCell for making `admcycles` available on these platforms.

Delecroix was a guest of the Max Planck Institute and then of the Hausdorff Institut for Mathematics during the development of the project.

Schmitt was supported by the grant SNF-200020162928 and has received funding from the European Research Council (ERC) under the European Union Horizon 2020 research and innovation programme (grant agreement No 786580). During the last phase of the project, he profited from the SNF Early Postdoc Mobility grant 184245 and also wants to thank the Max Planck Institute for Mathematics in Bonn for its hospitality.

Van Zelm was supported by the Einstein Foundation Berlin during the course of this work.

SUPPLEMENT. The online supplement contains version 1.3.1 of `admcycles`.

REFERENCES.

[Arbarello and Cornalba 1987] E. Arbarello and M. Cornalba, "The Picard groups of the moduli spaces of curves", *Topology* **26**:2 (1987), 153–171. MR Zbl

[Arbarello et al. 2011] E. Arbarello, M. Cornalba, and P. A. Griffiths, *Geometry of algebraic curves, II*, Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences] **268**, Springer, 2011. MR Zbl

[Bae and Schmitt 2020] Y. Bae and J. Schmitt, "Chow rings of stacks of prestable curves II", 2020. arXiv

[Bae et al. 2020] Y. Bae, D. Holmes, R. Pandharipande, J. Schmitt, and R. Schwarz, "Pixton's formula and Abel–Jacobi theory on the Picard stack", 2020. arXiv

[Bainbridge et al. 2018] M. Bainbridge, D. Chen, Q. Gendron, S. Grushevsky, and M. Möller, "Compactification of strata of Abelian differentials", *Duke Math. J.* **167**:12 (2018), 2347–2416. MR Zbl

[Bainbridge et al. 2019a] M. Bainbridge, D. Chen, Q. Gendron, S. Grushevsky, and M. Möller, "The moduli space of multi-scale differentials", 2019. arXiv

[Bainbridge et al. 2019b] M. Bainbridge, D. Chen, Q. Gendron, S. Grushevsky, and M. Möller, "Strata of k-differentials", *Algebr. Geom.* **6**:2 (2019), 196–233. MR

[Borot et al. 2020] G. Borot, N. Do, M. Karev, D. Lewański, and E. Moskovsky, "Double Hurwitz numbers: polynomiality, topological recursion and intersection theory", 2020. arXiv

[Buryak and Rossi 2021] A. Buryak and P. Rossi, "Quadratic double ramification integrals and the noncommutative KdV hierarchy", *Bull. Lond. Math. Soc.* **53**:3 (2021), 843–854. MR Zbl

[Canning and Larson 2021] S. Canning and H. Larson, "The Chow rings of the moduli spaces of curves of genus 7, 8, and 9", 2021. arXiv

[Castorena and Gendron 2020] A. Castorena and Q. Gendron, "On the locus of genus 3 curves that admit meromorphic differentials with a zero of order 6 and a pole of order 2", 2020. arXiv

[Chen et al. 2019] D. Chen, M. Möller, and A. Sauvaget, "Masur–Veech volumes and intersection theory: the principal strata of quadratic differentials", 2019. With an appendix by G. Borot, A. Giacchetto, and D. Lewanski. arXiv

[Clader and Janda 2018] E. Clader and F. Janda, "Pixton's double ramification cycle relations", *Geom. Topol.* **22**:2 (2018), 1069–1108. MR Zbl

[CoCalc] SageMath, "CoCalc Collaborative Computation Online", available at https://cocalc.com/.

[Costantini et al. 2020a] M. Costantini, M. Möller, and J. Zachhuber, "The Chern classes and the Euler characteristic of the moduli spaces of abelian differentials", 2020. arXiv

[Costantini et al. 2020b] M. Costantini, M. Möller, and J. Zachhuber, "diffstrata – a Sage package for calculations in the tautological ring of the moduli space of Abelian differentials", 2020. arXiv

[Do and Lewański 2020] N. Do and D. Lewański, "On the Goulden–Jackson–Vakil conjecture for double Hurwitz numbers", 2020. arXiv

[Faber 1996] C. Faber, "Intersection-theoretical computations on $\overline{M}_g$", pp. 71–81 in *Parameter spaces* ((Warsaw, 1994)), edited by P. Pragacz, Banach Center Publ. **36**, Polish Acad. Sci. Inst. Math., Warsaw, 1996. MR Zbl

[Faber 1999] C. Faber, "Algorithms for computing intersection numbers on moduli spaces of curves, with an application to the class of the locus of Jacobians", pp. 93–109 in *New trends in algebraic geometry* (Warwick, 1996), edited by C. P. Klaus Hulek, Fabrizio Catanese and M. Reid, London Math. Soc. Lecture Note Ser. **264**, Cambridge Univ. Press, 1999. MR Zbl

[Faber and Pandharipande 2000] C. Faber and R. Pandharipande, "Logarithmic series and Hodge integrals in the tautological ring", pp. 215–252 , 2000. MR

[Farkas and Pandharipande 2018] G. Farkas and R. Pandharipande, "The moduli space of twisted canonical divisors", *J. Inst. Math. Jussieu* **17**:3 (2018), 615–672. MR

[Graber and Pandharipande 2003] T. Graber and R. Pandharipande, "Constructions of nontautological classes on moduli spaces of curves", *Michigan Math. J.* **51**:1 (2003), 93–109. MR

[Graber and Vakil 2005] T. Graber and R. Vakil, "Relative virtual localization and vanishing of tautological classes on moduli spaces of curves", *Duke Math. J.* **130**:1 (2005), 1–37. MR

[Grosse et al. 2019] H. Grosse, A. Hock, and R. Wulkenhaar, "A Laplacian to compute intersection numbers on $\overline{\mathcal{M}}_{g,n}$ and correlation functions in NCQFT", 2019. arXiv

[Grushevsky and Zakharov 2014] S. Grushevsky and D. Zakharov, "The zero section of the universal semiabelian variety, and the double ramification cycle", *Duke Math J.* **163**:5 (2014), 889–1070. arXiv

[Hain 2013] R. Hain, "Normal functions and the geometry of moduli spaces of curves", pp. 527–578 in *Handbook of moduli, I*, edited by G. Farkas and I. Morrison, Adv. Lect. Math. (ALM) **24**, International Press, Somerville, MA, 2013. MR Zbl

[Harer 1983] J. Harer, "The second homology group of the mapping class group of an orientable surface", *Invent. Math.* **72**:2 (1983), 221–239. MR Zbl

[Holmes and Schmitt 2019] D. Holmes and J. Schmitt, "Infinitesimal structure of the pluricanonical double ramification locus", 2019. arXiv

[Holmes et al. 2018] D. Holmes, J. L. Kass, and N. Pagani, "Extending the double ramification cycle using Jacobians", *Eur. J. Math.* **4**:3 (2018), 1087–1099. MR Zbl

[Holmes et al. 2019] D. Holmes, A. Pixton, and J. Schmitt, "Multiplicativity of the double ramification cycle", *Doc. Math.* **24** (2019), 545–562. MR Zbl

[Janda 2017] F. Janda, "Relations on $\overline{M}_{g,n}$ via equivariant Gromov–Witten theory of $\mathbb{P}^1$", *Algebr. Geom.* **4**:3 (2017), 311–336. MR

[Janda et al. 2017] F. Janda, R. Pandharipande, A. Pixton, and D. Zvonkine, "Double ramification cycles on the moduli spaces of curves", *Publ. Math. Inst. Hautes Études Sci.* **125** (2017), 221–266. MR

[Kass and Pagani 2019] J. L. Kass and N. Pagani, "The stability space of compactified universal Jacobians", *Trans. Amer. Math. Soc.* **372**:7 (2019), 4851–4887. MR Zbl

[Li 2001] J. Li, "Stable morphisms to singular schemes and relative stable morphisms", *J. Differential Geom.* **57**:3 (2001), 509–578. MR

[Li 2002] J. Li, "A degeneration formula of GW-invariants", *J. Differential Geom.* **60**:2 (2002), 199–293.

[Li and Ruan 2001] A.-M. Li and Y. Ruan, "Symplectic surgery and Gromov–Witten invariants of Calabi–Yau 3-folds", *Invent. Math.* **145**:1 (2001), 151–218. MR Zbl

[mgn] D. Johnson, "`mgn` – a Sage program for computing products, Faber-Zagier relations, and top intersections on the moduli space of stable curves", available at https://pypi.org/project/mgn/.

[Molcho et al. 2021] S. Molcho, R. Pandharipande, and J. Schmitt, "The Hodge bundle, the universal 0-section, and the log Chow ring of the moduli space of curves", 2021. arXiv

[Mumford 1983] D. Mumford, "Towards an enumerative geometry of the moduli space of curves", pp. 271–328 in *Arithmetic and geometry*, *II*, edited by M. Artin and J. Tate, Progr. Math. **36**, Birkhäuser, Boston, 1983. MR Zbl

[Norbury 2017] P. Norbury, "A new cohomology class on the moduli space of curves", 2017. arXiv

[Owens and Somerstep 2019] B. Owens and S. Somerstep, "Boundary Expression for Chern Classes of the Hodge Bundle on Spaces of Cyclic Covers", 2019. arXiv

[Pagani et al. 2020] N. Pagani, A. T. Ricolfi, and J. van Zelm, "Pullbacks of universal Brill–Noether classes via Abel–Jacobi morphisms", *Math. Nachr.* **293**:11 (2020), 2187–2207. MR

[Pandharipande 2018] R. Pandharipande, "A calculus for the moduli space of curves", pp. 459–487 in *Algebraic geometry*: *Salt Lake City* 2015, edited by T. de Fernex et al., Proc. Sympos. Pure Math. **97**, Amer. Math. Soc., Providence, RI, 2018. MR

[Pandharipande and Tseng 2019] R. Pandharipande and H.-H. Tseng, "Higher genus Gromov–Witten theory of $\mathrm{Hilb}^n(\mathbb{C}^2)$ and CohFTs associated to local curves", *Forum Math. Pi* **7** (2019), e4, 63. MR

[Pandharipande et al. 2015] R. Pandharipande, A. Pixton, and D. Zvonkine, "Relations on $\overline{M}_{g,n}$ via 3-spin structures", *J. Amer. Math. Soc.* **28**:1 (2015), 279–309. MR

[Pixton 2012] A. Pixton, "Conjectural relations in the tautological ring of $\overline{M}_{g,n}$", 2012. arXiv

[SageMath] The Sage Developers, "Sage Mathematics Software (version 9.0)", available at http://www.sagemath.org.

[SageMathCell] The Sage Developers, "SageMathCell, embeddable web interface for the Sage Mathematics Software System", available at https://sagecell.sagemath.org. (2020).

[Schmitt 2018] J. Schmitt, "Dimension theory of the moduli space of twisted k-differentials", *Doc. Math.* **23** (2018), 871–894. MR Zbl

[Schmitt and van Zelm 2020] J. Schmitt and J. van Zelm, "Intersections of loci of admissible covers with tautological classes", *Selecta Math.* (*N.S.*) **26**:5 (2020), Paper No. 79, 69. MR Zbl

[Yang 2008] S. Yang, "Calculating intersection numbers on moduli spaces of pointed curves", 2008. arXiv

RECEIVED: 16 Mar 2020 REVISED: 30 Jun 2021 ACCEPTED: 15 Jul 2021

VINCENT DELECROIX:
vincent.delecroix@u-bordeaux.fr
Laboratoire Bordelais de Recherche en Informatique, CNRS - Université de Bordeaux, Talence, France

JOHANNES SCHMITT:
schmitt@math.uni-bonn.de
Mathematical Institute, University of Bonn, Bonn, Germany

JASON VAN ZELM:
jasonvanzelm@outlook.com
Humboldt Universität zu Berlin, Berlin, Germany

msp

JSAG **11** (2021), 113–122
https://doi.org/10.2140/jsag.2021.11.113

The Journal of Software for
Algebra and Geometry

Coding theory package for Macaulay2

TAYLOR BALL, EDUARDO CAMPS, HENRY CHIMAL-DZUL,
DELIO JARAMILLO-VELEZ, HIRAM LÓPEZ, NATHAN NICHOLS, MATTHEW PERKINS,
IVAN SOPRUNOV, GERMAN VERA-MARTÍNEZ AND GWYN WHIELDON

ABSTRACT: In this *Macaulay*2 package we implement a type of object called a `LinearCode`. We implement functions that compute basic parameters and objects associated with a linear code, such as generator and parity check matrices, the dual code, length, dimension, and minimum distance, among others. We implement a type of object called an `EvaluationCode`, a construction which allows users to study linear codes using tools of algebraic geometry and commutative algebra. We implement functions to generate important families of linear codes, such as Hamming codes, cyclic codes, Reed–Solomon codes, Reed–Muller codes, Cartesian codes, monomial–Cartesian codes, and toric codes. In addition, we implement functions for the syndrome decoding algorithm and locally recoverable code construction, which are important tools in applications of linear codes.

1. INTRODUCTION.

Coding theory has been extensively studied since 1948, when Claude Shannon [1948] proved in his seminal paper that linear codes can be used to reliably transmit information from a single source to a single receiver through a noisy channel. Since then, coding theory has found many important engineering applications. For example, coding theory has been used in designing reliable data storage systems, radio communication protocols, and in the emerging field of quantum computers. Coding theory has close ties with many areas in mathematics including linear algebra, commutative algebra, algebraic geometry, and combinatorics.

In this note we introduce the new [Macaulay2] package called `CodingTheory`. The goal of this package is to provide a range of functions for constructing linear and evaluation codes over finite fields, and for computing some of their main properties. To this aim, we implement two types of objects, `LinearCode` and `EvaluationCode`. The package also includes implementations of functions for generating important families of linear codes like Hamming codes, cyclic codes, Reed–Solomon codes, Reed–Muller codes, Cartesian codes, monomial-Cartesian codes and toric codes. It also has functions for the syndrome decoding algorithm and locally recoverable codes.

The organization of this note is as follows. In Section 2 we describe various ways to construct a linear code over a finite field using the *CodingTheory* package. In Section 3 we show how to compute the main parameters of a linear code: length, dimension, and minimum distance. We also illustrate how to

MSC2020: primary 13P25, 94B05; secondary 11T71, 14G50.
Keywords: linear codes, locally recoverable codes, Cartesian codes, evaluation codes, Hamming codes.
CodingTheory version 1.0

compute some of the main algebraic objects associated with linear codes, such as generator and parity check matrices, dual codes, etc. In Section 4 we give a brief introduction to evaluation codes and describe some functions implemented to study these objects. In Section 5 we explain how to create some of the most studied families of linear codes, including Hamming codes, cyclic codes, Reed–Solomon codes, and Reed–Muller codes. Finally, we give instructions on how to create locally recoverable codes.

In this paper we do not attempt to fully explain every function distributed in this package. For a detailed explanation of all functions in the package, we refer to the *Macaulay*2 help page which can be accessed by running

```
i1: viewHelp CodingTheory
```

More information about basics of coding theory can be found in [Huffman and Pless 2003; MacWilliams and Sloane 1977; van Lint 1999]. Constructions of codes using commutative algebra as evaluation codes can be seen in [Carvalho et al. 2017; Gold et al. 2005; Hansen 2000; Little and Schenck 2006; Martínez-Bernal et al. 2017; 2018; Rentería-Márquez et al. 2011; Rentería and Tapia-Recillas 1997; Ruano 2007; Soprunov and Soprunova 2009; Soprunov 2013]. Excellent references for the theory of vanishing ideals and their properties are [Cox et al. 1992; Villarreal 2015].

2. Constructing linear codes.

Let $\mathbb{F}_q$ be a finite field with q elements. Mathematically, a *linear code* is defined as a vector subspace $C \subseteq \mathbb{F}_q^n$ and it is often specified by a *generator matrix*, which is a $k \times n$ matrix G with entries in $\mathbb{F}_q$ whose k rows form a basis for C. In *Macaulay*2, a linear code is defined as an $\mathbb{F}_q$-submodule of $\mathbb{F}_q^n$ using the constructor `linearCode`. This constructor is an instance of the `LinearCode` type. There are various ways to use the command `linearCode`. For example, one can use this command to construct a linear code $C \subseteq \mathbb{F}_q^n$ by specifying a generator matrix G of C (Example 2.1). Alternatively, one can use the command `linearCode` to construct a linear code $C \subseteq \mathbb{F}_q^n$ by indicating the finite field $\mathbb{F}_q$ and a list L of elements of $\mathbb{F}_q^n$ that span C (Example 2.2). More details and equivalent ways to use the constructor `linearCode` are given next.

Inputs:	Usage:
• `F = GF(q)`, a finite field with q elements	• `linearCode(G)`
• `n, r, p`, positive integers with p prime	• `linearCode(F,L)`
• `G`, a matrix with entries in `GF(q)`	• `linearCode(F,n,L)`
• `L`, a list of elements of `GF(q)`$^{\mathtt{n}}$	• `linearCode(p,r,n,L)`

In the next examples we construct a simple binary linear code $C \subseteq \mathbb{F}_2^4$ with generator matrix $G = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$ using equivalent versions of the command `linearCode`.

Example 2.1.

```
i1 : F = GF(2);
i2 : L = {{1,1,0,0},{0,0,1,1}};
i3 : G = matrix apply(L,codeword->apply(codeword,entry->sub(entry,F)));
```

```
i4 : C = linearCode(G);

i5 : C.GeneratorMatrix
o5 = | 1 1 0 0 |
     | 0 0 1 1 |
```

Note that in Example 2.1 it was necessary to coerce the entries of each vector in the list `L` into elements of `F = GF(2)`. An equivalent way to do this is to pass the field `GF(q)` to the `matrix` constructor or to use the constructor `linearCode(GF(q),L)`.

Example 2.2.

```
i1 : L = {{1,1,0,0},{0,0,1,1}};

i2 : C = linearCode(GF(2),L);

i3 : C.GeneratorMatrix
o3 = | 1 1 0 0 |
     | 0 0 1 1 |
```

The set $\mathbb{F}_q^*$ of nonzero elements of a finite field $\mathbb{F}_q$ is a multiplicative cyclic group ([Huffman and Pless 2003, Theorem 3.3.1]). A generator of $\mathbb{F}_q^*$ is called a *primitive element* of $\mathbb{F}_q$. One way to refer to a primitive element of a finite field is by specifying a symbol using the `Variable` option of the constructor `GF`. In the next example we illustrate how to define a linear code $C \subseteq \mathbb{F}_{11}^{10}$ with generator matrix

$$G = \begin{pmatrix} 1 & a^1 & a^2 & \cdots & a^9 \\ 1 & a^2 & a^4 & \cdots & (a^2)^9 \\ 1 & a^3 & (a^3)^2 & \cdots & (a^3)^9 \\ 1 & a^4 & (a^4)^2 & \cdots & (a^4)^9 \end{pmatrix},$$

where a is a primitive element of $\mathbb{F}_{11}$. In *Macaulay*2, $a = 2$.

Example 2.3.

```
i1 : F = GF(11,Variable => a);

i2 : G = matrix table({1,2,3,4},
                      {1,a,a^2,a^3,a^4,a^5,a^6,a^7,a^8,a^9},(i,j)->j^i);

i3 : C = linearCode(G);

i4 : C.GeneratorMatrix
o4 = | 1 2  4  -3 5  -1 -2 -4 3  -5 |
     | 1 4  5  -2 3  1  4  5  -2 3  |
     | 1 -3 -2 -5 4  -1 3  2  5  -4 |
     | 1 5  3  4  -2 1  5  3  4  -2 |
```

In $\mathbb{F}_q^n$ there is a standard inner product defined for all $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ in $\mathbb{F}_q^n$ as $x \cdot y = x_1 y_1 + \cdots + x_n y_n$. Given a linear code $C \subseteq \mathbb{F}_q^n$, the *dual* or *orthogonal code* of C with respect to this inner product is the linear code $C^\perp$ defined as

$$C^\perp = \{x \in \mathbb{F}_q^n : x \cdot c = 0 \text{ for all } c \in C\}.$$

A generator matrix of $C^\perp$ is called a *parity check matrix* for C. Since $(C^\perp)^\perp = C$, another common way to mathematically define C is by specifying one of its parity check matrices. By using the command `linearCode` and the `ParityCheck` option of this command, we can define a linear code $C \subseteq \mathbb{F}_q^n$ by either specifying a parity check matrix H of C or a list L of elements of $\mathbb{F}_q^n$ that span $C^\perp$.

In the next example, we define a linear code $C \subseteq \mathbb{F}_9^5$ whose dual code $C^\perp$ is generated by the set $\{(1, 0, a, 0, 0), (0, 1, a+1, 1, 0), (1, 1, 1, a, 0)\} \subseteq \mathbb{F}_9^5$, where a is a primitive element of $\mathbb{F}_9$. In *Macaulay*2, $a \in \mathbb{F}_9$ satisfies $a^2 = a + 1$.

Example 2.4.

```
i1 : F = GF(9,Variable => a);

i2 : L = {{1,0,a,0,0},{0,a,a+1,1,0},{1,1,1,a,0}};

i3 : C = linearCode(F,L,ParityCheck => true);

i4 : C.GeneratorMatrix
o4 = | a-1 0 a+1 1 0 |
     | 0   0 0   0 1 |

i5 : C.ParityCheckMatrix
o5 = | 1 0 a   0 0 |
     | 0 a a+1 1 0 |
     | 1 1 1   a 0 |
```

Although the dual code of a linear code can be constructed using the command `dualCode` implemented in the *CodingTheory* package, the `ParityCheck` option of the command `linearCode` also allows us to construct the dual of a linear code. In the next example we construct the dual of the linear code in Example 2.3.

Example 2.5.

```
i1 : F = GF(11,Variable => a);

i2 : G = matrix table({1,2,3,4},
                       {1,a,a^2,a^3,a^4,a^5,a^6,a^7,a^8,a^9},(i,j)->j^i);

i3 : D = linearCode(G,ParityCheck => true);

i4 : D.GeneratorMatrix
o4 = | 1  -3 5  3  1 0 0 0 0 0 |
     | -3 -1 4  -4 0 1 0 0 0 0 |
     | 4  -4 -3 5  0 0 1 0 0 0 |
     | -5 -3 4  4  0 0 0 1 0 0 |
     | -4 -4 -1 3  0 0 0 0 1 0 |
     | -3 5  3  1  0 0 0 0 0 1 |
```

3. Basic parameters of linear codes.

The dimension k and length n are basic parameters of a linear code $C \subseteq \mathbb{F}_q^n$. The *information rate* of a linear code is defined as k/n. Another parameter of a linear code $C \subseteq \mathbb{F}_q^n$ is the *minimum distance*, which is defined as

$$w_H(C) := \min\{\|c\| : c \in C,\ c \neq 0\},$$

where $\|c\|$ denotes the Hamming weight of $c \in \mathbb{F}_q^n$, that is, $\|c\|$ is the number of nonzero entries in c. This parameter is important in determining the error-correcting capability of C; the higher the minimum distance, the more errors the code can detect and correct (see [Huffman and Pless 2003]). While the dimension and the length of linear code are computationally easy to determine, it is known that computing the minimum distance of an arbitrary linear code is an NP-hard problem [Vardy 1997]. For this task, the function `minimumWeight` is provided.

In *Macaulay*2, the space $\mathbb{F}_q^n$ has been called the *ambient module* or *ambient space* of C. The field $\mathbb{F}_q$ is called the *alphabet* or *field* of C. The elements of C are referred to as *codewords*. The input in all the following commands implemented in the `CodingTheory` package is always a linear code `C`:

- `dim C`
- `length C`
- `alphabet C`
- `ambientSpace C`
- `field C`
- `codewords C`
- `C.Generators`
- `C.GeneratorMatrix`
- `C.AmbientModule`
- `minimumWeight C`
- `informationRate C`
- `C.ParityCheckMatrix`

Example 3.1.

```
i1 : L = {{1,1,0,0},{0,0,1,1}};

i2 : C = linearCode(GF(4),L);

i3 : dim C
o3 : 2

i4 : length C
o4 = 4

i5 : alphabet C
o5 = {0, a, a + 1, 1}

i6 : ambientSpace C
             4
o6 = (GF 4)

i7 : field C
o7 = GF 4

i8 : minimumWeight C
o8 = 2

i9 : codewords C
o9 = {{1, 1, 1, 1}, {1, 1, a, a}, {a, a, 1, 1}, {a, a, a, a},
     -----------------------------------------------------------------
      {a + 1, a + 1, a, a}, {a + 1, a + 1, 1, 1}, {1, 1, a + 1, a + 1},
     -----------------------------------------------------------------
      {a, a, a + 1, a + 1}, {a + 1, a + 1, a + 1, a + 1}, {1, 1, 0, 0},
     -----------------------------------------------------------------
      {0, 0, 1, 1}, {0, 0, a, a}, {a, a, 0, 0}, {a + 1, a + 1, 0, 0},
     -----------------------------------------------------------------
      {0, 0, a + 1, a + 1}, {0, 0, 0, 0}}
```

4. EVALUATION CODES. Let $P = \{\boldsymbol{a}_1, \dots, \boldsymbol{a}_n\}$ be a subset of $\mathbb{F}_q^m$. Consider a finite-dimensional subspace $\mathcal{S} \subset \mathbb{F}_q[X_1, \dots, X_m]$ of the ring of polynomials over $\mathbb{F}_q$ in m variables. The *evaluation map*

$$\mathrm{ev}_{\mathcal{S}} : \mathcal{S} \to {\mathbb{F}_q}^{|P|}, \quad f \mapsto (f(\boldsymbol{a}_1), \dots, f(\boldsymbol{a}_n)),$$

defines a linear map of $\mathbb{F}_q$-vector spaces. The image of $\mathrm{ev}_{\mathcal{S}}$ in ${\mathbb{F}_q}^{|P|}$, denoted by $C_P(\mathcal{S})$, is the *evaluation code* on the set P corresponding to $\mathcal{S}$. The *vanishing ideal* of P, denoted by $I(P)$, is the ideal in $\mathbb{F}_q[X_1, \dots, X_n]$ of all polynomials that vanish on P. A key observation that allows the use of commutative algebra in studying evaluation codes is that the kernel of the evaluation map $\mathrm{ev}_{\mathcal{S}}$ is precisely $\mathcal{S} \cap I(P)$.

An evaluation code $C_P(\mathcal{S})$ is implemented in *Macaulay*2 as an object of type `EvaluationCode`. However, the object `C.LinearCode` is a linear code in *Macaulay*2. This has been done in this way because there are more objects associated with an evaluation code than with a linear code. For instance, the vanishing ideal associated to the set P plays an important role when finding and estimating parameters of the code, so it is convenient to be able to access it.

There are many constructions of evaluation codes for specific choices of the set P and the subspace $\mathcal{S}$. These include Reed–Muller codes, Cartesian, monomial Cartesian codes, toric codes, and evaluation codes from graphs. We refer to [Carvalho et al. 2017; Hansen 2000; Little and Schenck 2006; López et al. 2014; Martínez-Bernal et al. 2017; Rentería-Márquez et al. 2011; Ruano 2007; Soprunov and Soprunova 2009] for details on how these codes are defined and what properties they have from coding theory, commutative algebra, and algebraic geometry perspectives.

Some functions implemented in the *CodingTheory* package for various constructions of evaluation codes and associated algebraic objects are the following:

Inputs:

- `I`, an ideal
- `d`,`r`, positive integers
- `F=GF(q)`, a finite field with `q` elements
- `P`, a list of points in `F`$^{\mathtt{m}}$
- `S`, a list of polynomials in `m` variables
- `M`, an integer matrix
- `L`, a list of subsets of `F`
- `v`, a list of `m` positive integers
- `MI`, an incident matrix of a graph

Usage:

- `evaluationCode(F,P,S)`
- `toricCode(F,M)`
- `cartesianCode(F,L,d)`
- `orderCode(F,P,v,d)`
- `evCodeGraph(F,MI,S)`
- `vNumber(I)`
- `footPrint(d,r,I)`
- `hYp(d,r,I)`
- `genMinDisIdeal(d,r,I)`
- `vasconcelosDegree(d,r,I)`

The input `S` above is a list of polynomials that span the subspace

$$\mathcal{S} \subset \mathbb{F}_q[X_1, \ldots, X_m].$$

The mathematical definitions of the functions in the second column of the previous list can be found in [Cooper et al. 2020]. The following example shows how to construct an evaluation code in *Macaulay2* using the *CodingTheory* package.

Example 4.1.

```
i1 : F=GF(4,Variable=>a); R=F[x,y,z];

i3 : P={{0,0,0},{1,0,0},{0,1,0},{0,0,1},{1,1,1},{a,a,a}};

i4 : S={x+y+z,a+y*z^2,z^2,x+y+z+z^2};

i5 : C=evaluationCode(F,P,S);

i6 : (C.LinearCode).GeneratorMatrix
o6 = | 0 1 1 1 1   a   |
     | a a a a a+1 a+1 |
     | 0 0 0 1 1   a+1 |
     | 0 1 1 0 0   1   |
i7 : length C.LinearCode
o7 = 6

i8 : dim C.LinearCode
o8 = 3

i9 : C.Points
o9 = {{0, 0, 0}, {1, 0, 0}, {0, 1, 0}, {0, 0, 1}, {1, 1, 1}, {a, a, a}}

i10 : C.VanishingIdeal
                               2    2                      2    2
o10 = ideal (x*z + y*z, y  + z  + y + z, x*y + y*z, x  + z  + x + z,
      --------------------------------------------------------------
       3          2
      z  + (a +1)z  + a*z)
```

5. Families of linear codes. Various important families of linear codes have been defined through the development of coding theory. These include Hamming codes, cyclic codes, Reed–Solomon codes, Reed–Muller codes, etc. The mathematical definitions of these and many other families of codes can be found in [Huffman and Pless 2003; MacWilliams and Sloane 1977; van Lint 1999].

The following functions have been implemented in the *CodingTheory* package to construct some of these important families of codes.

Inputs:

- `n, k, d, m, s`, positive integers
- `F = GF(q)`, a finite field with q elements
- `g(x)`, a polynomial in `F[x]`
- E, a list of elements of `F`
- L, a list of vectors in F^m

Usage:

- `hammingCode(q,s)`
- `cyclicCode(F,g(x),n)`
- `zeroCode(F,n)`
- `reedSolomonCode(F,E,s)`
- `repetitionCode(F,n)`
- `universeCode(F,n)`
- `reedMullerCode(q,m,d)`
- `randomCode(F,n,k)`
- `zeroSumCode(F,n)`

Example 5.1.

```
i1 : C = hammingCode(2,3);

i2 : C.GeneratorMatrix
o2 = | 1 1 1 1 0 0 0 |
     | 1 1 0 0 1 0 0 |
     | 0 1 1 0 0 1 0 |
     | 1 0 1 0 0 0 1 |

i3 : F = GF(5); R = F[x]; g = x-1; C = cyclicCode(F,g,6);

i7 : C.GeneratorMatrix
o7 = | -1 1  0  0  0  0 |
     | 0  -1 1  0  0  0 |
     | 0  0  -1 1  0  0 |
     | 0  0  0  -1 1  0 |
     | 0  0  0  0  -1 1 |

i8 : C = reedSolomonCode(GF(5),{1,2,3},3);

i9 : (C.LinearCode).GeneratorMatrix
o9 = | 1 1  1  |
     | 1 2  -2 |
     | 1 -1 -1 |
```

6. APPLICATIONS OF LINEAR CODES. An important aspect in coding theory is *decoding*, which is used when information is transmitted trough a noisy channel. In a few words the idea is the following. Take a vector $c \in C$. Change the value of some of the entries of c to obtain a new vector v. Decoding the vector v means to recover the vector c when only v and C are given. Detailed treatment of decoding algorithms can be found in [Huffman and Pless 2003]. Another, more recent application of coding theory is found in distributed and cloud storage systems. The idea is to use *locally recoverable codes*, which are linear codes with the property that every entry can be recovered from a few other entries. For more information on locally recoverable codes, see [Tamo and Barg 2014].

Some of the most important functions implemented in the *CodingTheory* package that can be used for applications of coding theory are the following:

Inputs:

- `C`, a linear code over `GF(q)`
- `v`, a vector in the ambient space of `C`
- `{q,n,k,r}`, where `q` is a prime power, and `n`, `k`, and `r` are positive integers
- `L`, a list of pairwise disjoint subsets of `GF(q)`

Usage:

- `syndromeDecode(C,`v`,minimumWeight(C))`
- `LocallyRecoverableCode({q,n,k,r},L,a polynomial)`

Example 6.1.

```
i1 : C = hammingCode(2,3);

i2 : msg = matrix {{1,0,1,0}};

i3 : v = msg*(C.GeneratorMatrix)
o3 = | 0 1 0 1 0 1 0 |

i4 : err = matrix take(random entries basis source v, 1)
o4 = | 0 0 0 0 1 0 0 |

i5 : received = transpose(transpose (v+err))
o5 = | 0 1 0 1 1 1 0 |

i6 : transpose syndromeDecode(C, transpose recieved, 3)
o6 = | 0 1 0 1 0 1 0 |
```

ACKNOWLEDGMENTS. We thank Federico Galetto, Courtney Gibbons, Hiram López, and Branden Stone for organizing the Macualay2 workshop at Cleveland State University, where this collaboration started. We want to give special thanks to Branden Stone for helping us to develop the package during the workshop. Finally, we are grateful to the anonymous referees for their valuable comments and suggestions. This work was partially supported by the NSF grant DMS-2003883.

SUPPLEMENT. The online supplement contains version 1.0 of *CodingTheory*.

REFERENCES.

[Carvalho et al. 2017] C. Carvalho, V. G. L. Neumann, and H. H. López, "Projective nested cartesian codes", *Bull. Braz. Math. Soc. (N.S.)* **48**:2 (2017), 283–302. MR Zbl

[Cooper et al. 2020] S. M. Cooper, A. Seceleanu, c. O. Tohăneanu, M. V. Pinto, and R. H. Villarreal, "Generalized minimum distance functions and algebraic invariants of Geramita ideals", *Adv. in Appl. Math.* **112** (2020), art. id. 101940. MR

[Cox et al. 1992] D. Cox, J. Little, and D. O'Shea, *Ideals, varieties, and algorithms*, Springer, 1992. MR Zbl

[Gold et al. 2005] L. Gold, J. Little, and H. Schenck, "Cayley–Bacharach and evaluation codes on complete intersections", *J. Pure Appl. Algebra* **196**:1 (2005), 91–99. MR Zbl

[Hansen 2000] J. P. Hansen, "Toric surfaces and error-correcting codes", pp. 132–142 in *Coding theory, cryptography and related areas* (Guanajuato (1998)), edited by J. Buchmann et al., Springer, 2000. MR Zbl

[Huffman and Pless 2003] W. C. Huffman and V. Pless, *Fundamentals of error-correcting codes*, Cambridge University Press, 2003. MR Zbl

[van Lint 1999] J. H. van Lint, *Introduction to coding theory*, 3rd ed., Graduate Texts in Mathematics **86**, Springer, 1999. MR Zbl

[Little and Schenck 2006] J. Little and H. Schenck, "Toric surface codes and Minkowski sums", *SIAM J. Discrete Math.* **20**:4 (2006), 999–1014. MR Zbl

[López et al. 2014] H. H. López, C. Rentería-Márquez, and R. H. Villarreal, "Affine Cartesian codes", *Des. Codes Cryptogr.* **71**:1 (2014), 5–19. MR Zbl

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

[MacWilliams and Sloane 1977] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*, *I*, vol. 16, North-Holland Mathematical Library, North-Holland Publishing Co., Amsterdam, 1977. MR

[Martínez-Bernal et al. 2017] J. Martínez-Bernal, Y. Pitones, and R. H. Villarreal, "Minimum distance functions of graded ideals and Reed–Muller-type codes", *J. Pure Appl. Algebra* **221**:2 (2017), 251–275. MR Zbl

[Martínez-Bernal et al. 2018] J. Martínez-Bernal, Y. Pitones, and R. H. Villarreal, "Minimum distance functions of complete intersections", *J. Algebra Appl.* **17**:11 (2018), art. id. 1850204. MR Zbl

[Rentería and Tapia-Recillas 1997] C. Rentería and H. Tapia-Recillas, "Reed–Muller codes: an ideal theory approach", *Comm. Algebra* **25**:2 (1997), 401–413. MR Zbl

[Rentería-Márquez et al. 2011] C. Rentería-Márquez, A. Simis, and R. H. Villarreal, "Algebraic methods for parameterized codes and invariants of vanishing ideals over finite fields", *Finite Fields Appl.* **17**:1 (2011), 81–104. MR Zbl

[Ruano 2007] D. Ruano, "On the parameters of r-dimensional toric codes", *Finite Fields Appl.* **13**:4 (2007), 962–976. MR Zbl

[Shannon 1948] C. E. Shannon, "A mathematical theory of communication", *Bell System Tech. J.* **27** (1948), 379–423. MR Zbl

[Soprunov 2013] I. Soprunov, "Toric complete intersection codes", *J. Symbolic Comput.* **50** (2013), 374–385. MR Zbl

[Soprunov and Soprunova 2009] I. Soprunov and J. Soprunova, "Toric surface codes and Minkowski length of polygons", *SIAM J. Discrete Math.* **23**:1 (2009), 384–400. MR Zbl

[Tamo and Barg 2014] I. Tamo and A. Barg, "A family of optimal locally recoverable codes", *IEEE Trans. Inform. Theory* **60**:8 (2014), 4661–4676. MR

[Vardy 1997] A. Vardy, "The intractability of computing the minimum distance of a code", *IEEE Trans. Inform. Theory* **43**:6 (1997), 1757–1766. MR Zbl

[Villarreal 2015] R. H. Villarreal, *Monomial algebras*, 2nd ed., CRC Press, Boca Raton, FL, 2015. MR

RECEIVED: 13 Jul 2020 REVISED: 16 Jul 2021 ACCEPTED: 10 Aug 2021

TAYLOR BALL:
trball13@gmail.com
University of Notre Dame, Notre Dame, IN, United States

EDUARDO CAMPS:
camps@esfm.ipn.mx
Escuela Superior de Física y Matemáticas, Instituto Politecnico Nacional, Zacatenco, Mexico City, Mexico

HENRY CHIMAL-DZUL:
hc118813@ohio.edu
Department of Mathematics and Center of Ring Theory and its Applications, Ohio University, Athens, OH, United States

DELIO JARAMILLO-VELEZ:
djaramillo@math.cinvestav.mx
Departamento de Matemáticas, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Mexico City, Mexico

HIRAM LÓPEZ:
h.lopezvaldez@csuohio.edu
Department of Mathematics and Statistics, Cleveland State University, Cleveland, OH, United States

NATHAN NICHOLS:
nathannichols454@gmail.com
School of Mathematics, University of Minnesota Twin Cities, Minneapolis, MN, United States

MATTHEW PERKINS:
m.r.perkins73@vikes.csuohio.edu
Department of Mathematics and Statistics, Cleveland State University, Cleveland, OH, United States

IVAN SOPRUNOV:
i.soprunov@csuohio.edu
Department of Mathematics and Statistics, Cleveland State University, Cleveland, OH, United States

GERMAN VERA-MARTÍNEZ:
gveram1100@alumno.ipn.mx
Escuela Superior de Física y Matemáticas, Instituto Politecnico Nacional, Zacatenco, Mexico City, Mexico

GWYN WHIELDON:
gwyn.whieldon@gmail.com
Frederick, MD, United States

JSAG **11** (2021), 123–127
https://doi.org/10.2140/jsag.2021.11.123

The Journal of Software for
Algebra and Geometry

Threaded Gröbner bases: a Macaulay2 package

SONJA PETROVIĆ AND SHAHRZAD ZELENBERG

ABSTRACT: The complexity of Gröbner computations has inspired many improvements to Buchberger's algorithm over the years. Looking for further insights into the algorithm's performance, we offer a threaded implementation of the classical Buchberger algorithm in Macaulay2. The output of the main function of the package includes information about *lineages* of nonzero remainders that are added to the basis during the computation. This information can be used for further algorithm improvements and optimization.

1. INTRODUCTION. The importance in computational algebra of Gröbner bases and therefore of Buchberger's algorithm, as well as its many variants, is indisputable. Yet it is still a challenge to apply brute force algorithms to larger problems primarily due to considerations in computer science. That is, the current computing paradigm favors clusters of CPUs, or nodes, rather than one massive CPU. As a result, there is a need to distribute this algorithm that is automated for the user (in that it does not require a user to know how it should be distributed).

Past work in this area has focused on synchronized methods as detailed in [Mityunin and Pankrat'ev 2005]. One method spreads a key step in Buchberger's algorithm — the reduction of S-pairs by division — across nodes; another sends tasks to individual nodes while a central, coordinating node waits for all threads to complete. Each of these still requires some central node and synchronization, which leads to bottlenecks in the computation. A truly distributed algorithm would be decentralized and asynchronous. Zelenberg [2018] discusses an asynchronous, decentralized distributed version of Buchberger's algorithm done generically with the potential of very good speedups. Zelenberg implemented a threaded version in [Python] to explore this further, and as a result some important discoveries were made. It should be noted that multithreaded algorithms are not necessarily distributed across distinct nodes; rather, threads are sharing computation and passing information back and forth.

Of the discoveries made, the most important is this: in order for a distributed process to be both generically usable and automated for the user, an effective algorithm will need to account for features of the polynomials (relative to the starting basis) when deciding what tasks to assign to what node. This is because transferring information between nodes is a very slow process and so needs to be minimized.

Given the need to analyze aspects of these polynomials, [Macaulay2] offers some clear advantages over *Python*. Moreover, the *Macaulay*2 engine is written in C/C++, a language well suited for writing distributed algorithms. Threads within *Macaulay*2 work differently than within *Python* and, as such,

MSC2020: 13P10.
Keywords: Gröbner basis, distributed computation, lineages, S-pairs.
ThreadedGB version 1.1

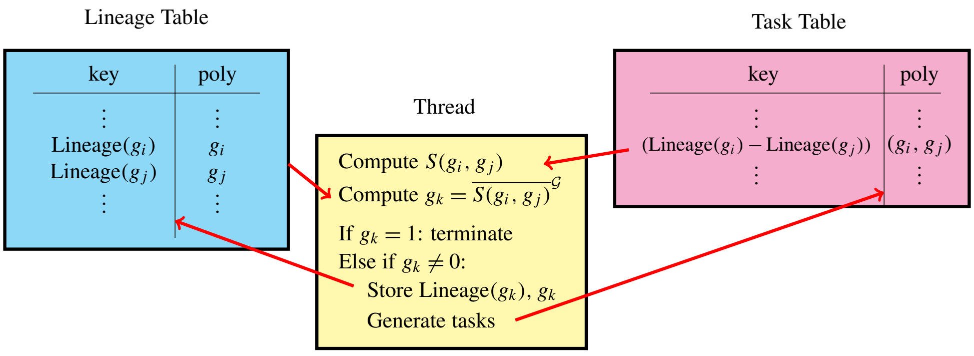

Figure 1. The lineage table is initialized with the starting basis, and the task table is initialized with a task for each pair of starting generators. Then, all threads perform the same task: they pull a pair of polynomials from the tasks hash table along with an associated lineage key. They compute the S-polynomial and reduce it with respect to the current basis. If the remainder, r, is nonzero, it is stored in the lineage table along with its lineage key, and, for each g in the current basis, a new task indexed by the pair (g, r) is added to the task table. If the remainder 1 is found, the process of creating tasks stops. The process is repeated using n parallel threads, with n specified by the user, until the task table is empty.

some design changes were necessary from the implementation discussed in [Zelenberg 2018]. Queues are replaced altogether with hash tables — an improvement since threads access the most up-to-date version of the generating set at the time of reduction. Even so, this cannot eliminate redundancies as threads may compute the same result (virtually) simultaneously.

One of the goals of our package *ThreadedGB* is to allow a user to analyze what we refer to as *lineages* of polynomials in a Gröbner basis.

Definition 1.1. Let G be a Gröbner basis of $I = (f_0, \dots, f_k)$. A *lineage* of a polynomial in G is a natural number, or an ordered pair of lineages, tracing its history in the given Gröbner basis computation. It is defined recursively as follows:

- For the starting generating set, $\text{Lineage}(f_i) = i$,
- For any subsequently created S-polynomial $S(f, g)$, the lineage of its remainder r on division is the pair $\text{Lineage}(r) = (\text{Lineage}(f), \text{Lineage}(g))$.

To illustrate, suppose $I = (x^2 - y, x^3 - z) \subset \mathbb{Q}[x, y, z]$ with graded reverse lexicographic order. Then $\text{Lineage}(x^2 - y) = 0$ and $\text{Lineage}(x^3 - z) = 1$. Two additional elements are added to create a (nonminimal) Gröbner basis: $xy + z$ and $y^2 - xz$, with lineages $(0, 1)$ and $((0, 1), 0)$, respectively. According to $\text{Lineage}(y^2 - xz)$, this element is constructed from $S(xy + z, x^2 - y)$. Lineages are expressions of the starting generating set and thus dependent on the choice and order of its elements. More importantly, a lineage is not necessarily unique, as the same polynomial can be constructed multiple ways. The lineage tables produced by *ThreadedGB* (see Figure 1) do not provide all possible lineages — only a particular choice based on the order in which the generators are provided by the user.

2. Effect of ordering of polynomials on lineages: a simple example. Consider the polynomial ring $\mathbb{Q}[x_1, x_3, x_0, x_4, x_2]$ with lexicographic order and the ideal of the rational normal curve in $\mathbb{P}^4$. The six generators have lineages $0, \dots, 5$, and Buchberger's algorithm adds three new elements to the Gröbner basis before final reduction. This can be seen by turning on the `gbTrace` option in *Macaulay*2, which tells us three new polynomials are added to the basis. The function `tgb` lets us know exactly which ones and their lineages. Specifically, a run of `tgb` reveals these are $x_0x_4 - x_2^2$, $-x_3x_0x_4 + x_3x_2^2$, $-x_0x_4x_2 + x_2^3$ with lineages $(2, 3)$, $(1, 4)$, $(1, 2)$, respectively.

```
i1 : needsPackage "ThreadedGB";

i2 : QQ[x_1,x_3,x_0,x_4,x_2,MonomialOrder=>Lex];

i3 : rnc = minors(2, matrix{{x_0..x_3},{x_1..x_4}});
o3 : Ideal of QQ[x , x , x , x , x ]
                  1   3   0   4   2

i4 : allowableThreads = 4;

i5:  g = tgb(rnc)
                                            3
o5 = LineageTable{(1, 2) => - x x x  + x    }
                               0 4 2    2
                                           2
                  (1, 4) => - x x x  + x x
                               3 0 4    3 2
                                         2
                  (2, 3) => x x  - x
                             0 4    2
                              2
                  0 => - x  + x x
                          1    0 2
                  1 => - x x  + x x
                          1 2    3 0
                                2
                  2 => x x  - x
                        1 3    2
                  3 => - x x  + x x
                          1 3    0 4
                  4 => x x  - x x
                        1 4    3 2
                          2
                  5 => - x  + x x
                          3    4 2
o5 : LineageTable
```

Running the command `reduce g` will produce a reduced Gröbner basis; in particular, the lineage table entries with keys $(1, 2)$, $(1, 4)$ and 2 will be replaced by `null`. This allows the user to see which nonzero polynomials produced during the computation turn out not to be needed. Of course, to continue computing with the given basis, one wishes to have it in standard *Macaulay*2 format, which is a matrix.

```
i6 : matrix reduce g
o6 = | x_1^2-x_0x_2 x_1x_2-x_3x_0 x_1x_3-x_2^2
       x_1x_4-x_3x_2 x_3^2-x_4x_2 x_0x_4-x_2^2 |
                                           1                               6
o6 : Matrix (QQ[x , x , x , x , x ])  <--- (QQ[x , x , x , x , x ])
                 1   3   0   4   2               1   3   0   4   2
```

One can use the package to study, for example, how reordering the input basis affects the algorithm. In Gröbner computations, *Macaulay*2 creates and processes S-polynomials in lexicographic order of pairs (first and second, then first and third, and so on). Let $S = \mathbb{Q}[a, b, c, d]$ and $I = (abc - 1, abc, a + bd - c)$; clearly $I = S$. But the order of generators listed affects the complexity of the particular run; namely, listing the quadratic first makes the algorithm perform more steps. The method `tgb` can be verbose and can tell us what is going on behind the scenes for each lineage.

```
i7 : QQ[a, b, c, d];

i8 : I =  ideal (a*b*c, a*b*c - 1, a+b*d-c);

i9 :  tgb(I,Verbose=>true)
Scheduling a task for lineage (0,1)
Scheduling a task for lineage (0,2)
Scheduling a task for lineage (1,2)
Adding the following remainder to GB: 1 from lineage (0,1)
Found a unit in the Groebner basis; reducing now.
o9 = LineageTable{(0, 1) => 1}
                  0 => null
                  1 => null
                  2 => null
o9 : LineageTable
```

Compare this to the following run of the threaded Buchberger's algorithm under a different input order.

```
i10 : I = ideal (a+b*d-c, a*b*c-1, a*b*c);
o10 : Ideal of QQ[a, b, c, d]

i11 : tgb(I)
o11 = LineageTable{(0, 1) => null}
                   (0, 2) => null
                   (1, 2) => 1
                   0 => null
                   1 => null
                   2 => null
o11 : LineageTable
```

Three new elements are added to the basis, namely (0,1), (0,2), (1,2), if the quadratic generator is listed first, but if it is listed last, then only the polynomial with lineage (0,1) is added — because it already equals 1 — and the algorithm stops.

3. Nuts and bolts.

Given a list of polynomials L or an ideal I and an integer n, the main method `tgb` uses `Tasks` in *Macaulay*2 to compute a Gröbner basis of I or (L) using n threads. It returns an object of type `LineageTable`, which is an instance of `HashTable`, whose values are a Gröbner basis of I or (L). The keys are polynomial lineages.

The starting basis L (meaning, the input list `L` or `L=I`$_*$) populates the entries of a lineage table G with keys from 0 to one less than the number of elements of L. The method creates all possible S-polynomials of L and schedules their reduction with respect to G as tasks. Throughout the computation, every nonzero remainder added to the basis is added to G, with its lineage, as defined above, being the key. Each such remainder also triggers the creation of S-polynomials using it and every element in G and scheduling the reduction thereof as additional tasks. The process is done when there are no remaining tasks.

There is a way to track the tasks being created by turning on the option `Verbose`, or provide the reduced or a minimal Gröbner basis using the functions `reduce` or `minimalize`, respectively. The users who expect just a Gröbner basis in usual *Macaulay*2 format, without the lineages, can call `matrix` on the `LineageTable`.

4. Improvements and speed-ups.

As with any *Macaulay*2 package, improvements are easy to make via GitHub. Our package's GitHub repository will be made public shortly, so other users can implement any extensions or add improvements to this threaded implementation of Buchberger's algorithm.

These may include known speed-ups as optional ways to run the algorithm; for example, if one wishes to study lineages produced by the F4 algorithm [Faugére 1999], then one can build that option into this threaded computation.

The current goal is to explore algorithm performance and complexity and how input basis features affect these; the lineages are designed specifically to aid in this goal. Of course, speed-ups should come "naturally" from a threaded implementation. However, with *Macaulay*2's current implementation of threads, speed-ups aren't observed with interpreted code, hence to achieve effective speed-ups in practice, we plan to implement `tgb` in the engine, using C/C++.

SUPPLEMENT. The online supplement contains version 1.1 of *ThreadedGB*.

REFERENCES.

[Faugére 1999] J.-C. Faugére, "A new efficient algorithm for computing Gröbner bases (F_4)", pp. 61–88 in *Effective methods in algebraic geometry* ((Saint-Malo, 1998)), vol. 139, 1999. MR Zbl

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

[Mityunin and Pankrat'ev 2005] V. A. Mityunin and E. V. Pankrat'ev, "Parallel algorithms for the construction of Gröbner bases", *Sovrem. Mat. Prilozh.* 30 (2005), 46–64. Translated in *J. of Math. Statistics*, **142**(4): 2248–2266, 2007. MR

[Python] G. Van Rossum and F. L. Drake, *Python* 3 *Reference Manual*, CreateSpace, Scotts Valley, CA.

[Zelenberg 2018] S. J. Zelenberg, *Distributed computational systems*, Ph.D. thesis, 2018, available at etda.libraries.psu.edu/catalog/15329sxj937.

RECEIVED: 19 Jul 2020 REVISED: 6 Aug 2021 ACCEPTED: 8 Oct 2021

SONJA PETROVIĆ:
sonja.petrovic@iit.edu
Department of Applied Mathematics, Illinois Institute of Technology, Chicago, IL, United States

SHAHRZAD ZELENBERG:
szelenberg@mx.lakeforest.edu
Lake Forest College, Department of Mathematics and Computer Science, Lake Forest, IL, United States

JSAG **11** (2021), 129–142
https://doi.org/10.2140/jsag.2021.11.129

The Journal of Software for
Algebra and Geometry

Standard pairs of monomial ideals over nonnormal affine semigroups in SageMath

BYEONGSU YU

ABSTRACT: We present `StdPairs`, a `SageMath` library to compute standard pairs of a monomial ideal over a pointed (nonnormal) affine semigroup ring. Moreover, `StdPairs` provides the associated prime ideals, the corresponding multiplicities, and an irredundant irreducible primary decomposition of a monomial ideal. The library expands on the `standardPairs` function on `Macaulay2` over polynomial rings, and is based on algorithms from Matusevich and Yu (2020). We also provide methods that allow the outputs from this library to be compatible with the `Normaliz` package of `Macaulay2` and `SageMath`.

1. INTRODUCTION.

Affine semigroup rings are the object of many studies in combinatorial commutative algebra. The goal of this article is to present the `SageMath` library `StdPairs`, which systematizes computations for monomial ideals in affine semigroup rings. The algorithms implemented here are based on the notion of *standard pairs*, introduced for monomial ideals in polynomial rings by [Sturmfels et al. 1995], and generalized to the semigroup ring case in [Matusevich and Yu 2020]. Standard pairs are combinatorial structures that contain information on primary and irreducible decompositions of monomial ideals, as well as multiplicities. One of the main contributions of [Matusevich and Yu 2020] is that standard pairs and the associated algebraic concepts can be effectively computed over affine semigroup rings.

The `SageMath` library `StdPairs` implements the algorithms of [Matusevich and Yu 2020] to calculate standard pairs for monomial ideals in any pointed (nonnormal) affine semigroup ring. This library can be regarded as a generalization of the `standardPairs` function in `Macaulay2` implemented by [Hoşten and Smith 2002]. This library is provided as an online supplement to this paper.

Outline. Section 2 provides background on affine semigroup rings, their monomial ideals, and related combinatorial notions. It also explains their implementation as `SageMath` classes. Section 3 presents the implementation of algorithms to find standard pairs, proposed in [Matusevich and Yu 2020, Section 4]. Section 4 shows compatibility with the `Normaliz` package by introducing methods to translate objects in `SageMath` into objects in `Macaulay2` using `Normaliz`.

MSC2020: primary 13F65, 68W30, 90C90; secondary 13P25, 20M25.
Keywords: affine semigroups, standard pairs, monomial ideals, semigroup rings.
`StdPairs` version 1.0

1.1. ***Notation.*** We denote a semigroup of nonnegative integers including 0 by $\mathbb{N} = \{0, 1, 2, \ldots\}$. A ring of integers is $\mathbb{Z}$. All nonnegative real numbers are represented by $\mathbb{R}_{\geq 0}$. Boldface uppercase letters and boldface lowercase letters denote matrices and vectors, respectively. This will be used when we call a cone $\mathbb{R}_{\geq 0}\boldsymbol{A}$ for some $d \times n$ matrix $\boldsymbol{A}$ over $\mathbb{Z}$. Let $\mathbb{K}$ be an arbitrary field.

2. Affine semigroup, ideal, and proper pair as classes of SageMath.

2.1. ***Mathematical background.*** An *affine semigroup* is a semigroup of $\mathbb{Z}^d$ generated by finitely many vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ for some $n \in \mathbb{N}$. We let $\boldsymbol{A}$ be a $d \times n$ matrix whose column vectors are $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n \in \mathbb{Z}^d$. The set of all nonnegative integer linear combinations of $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$, denoted by $\mathbb{N}\boldsymbol{A}$, is an affine semigroup. These columns $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ are the *generators* of the affine semigroup $\mathbb{N}\boldsymbol{A}$; the matrix $\boldsymbol{A}$ is called the *generating matrix*. Since $\mathbb{N}\boldsymbol{A}$ contains 0, an affine semigroup is a commutative monoid. Given a field $\mathbb{K}$, we are concerned with the *affine semigroup ring* $\mathbb{K}[\mathbb{N}\boldsymbol{A}]$. A natural first example is the polynomial ring in d-variables; in this case, $\boldsymbol{A}$ is the $d \times d$ identity matrix. We refer to [Miller and Sturmfels 2005, Section 7] for more background on this topic. Throughout this article, we assume that the affine semigroup $\mathbb{N}A$ under consideration is *pointed*, which means that the cone $\mathbb{R}_{\geq 0}\boldsymbol{A}$ does not contain lines.

An *ideal* of an affine semigroup is a set $I \subset \mathbb{N}\boldsymbol{A}$ such that $I + \mathbb{N}\boldsymbol{A} \subseteq I$. There is a one-to-one correspondence between monomial ideals of $\mathbb{K}[\mathbb{N}\boldsymbol{A}]$ and ideals of $\mathbb{N}\boldsymbol{A}$. Therefore, the definition of prime, irreducible, and primary ideals of $\mathbb{K}[\mathbb{N}\boldsymbol{A}]$ can be naturally extended to the ideals of an affine semigroup. The *standard monomials* of an ideal $I \subset \mathbb{N}\boldsymbol{A}$ are all elements of $\mathbb{N}\boldsymbol{A} \smallsetminus I$. Let $\operatorname{std}(I)$ be a set of all standard monomials with respect to I.

A *face* of an affine semigroup $\mathbb{N}\boldsymbol{A}$ is a subsemigroup $\mathbb{N}\boldsymbol{F} \subseteq \mathbb{N}\boldsymbol{A}$ such that the complement $\mathbb{N}\boldsymbol{A} \smallsetminus \mathbb{N}\boldsymbol{F}$ is an ideal of $\mathbb{N}\boldsymbol{A}$ [Miller and Sturmfels 2005, Definition 7.8]. Equivalently, it is a subsemigroup $\mathbb{N}\boldsymbol{F}$ with the property that $\boldsymbol{a} + \boldsymbol{b} \in \mathbb{N}\boldsymbol{F}$ if and only if $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{N}\boldsymbol{F}$. The faces of an affine semigroup form a lattice which is isomorphic to the face lattice of a (real) cone over the affine semigroup [Bruns and Herzog 1993; Miller and Sturmfels 2005]. Thus, we may represent a face $\mathbb{N}\boldsymbol{F}$ as a submatrix $\boldsymbol{F}$ of $\boldsymbol{A}$.

A *pair* is a tuple $(\boldsymbol{a}, \boldsymbol{F})$ of an element $\boldsymbol{a}$ in $\mathbb{N}\boldsymbol{A}$ and a face $\boldsymbol{F}$ of $\mathbb{N}\boldsymbol{A}$ [Matusevich and Yu 2020]. A *proper pair* of an ideal I is a pair $(\boldsymbol{a}, \boldsymbol{F})$ such that $\boldsymbol{a} + \mathbb{N}\boldsymbol{F} \subseteq \operatorname{std}(I)$. A pair $(\boldsymbol{a}, \boldsymbol{F})$ *divides* $(\boldsymbol{b}, \boldsymbol{G})$ if there exists $\boldsymbol{c} \in \mathbb{N}\boldsymbol{A}$ such that $\boldsymbol{a} + \boldsymbol{c} + \mathbb{N}\boldsymbol{F} \subseteq \boldsymbol{b} + \mathbb{N}\boldsymbol{G}$ [Matusevich and Yu 2020]. The set of all proper pairs of an ideal I is partially ordered $\prec$ by inclusion. In other words, $(\boldsymbol{a}, \boldsymbol{F}) \prec (\boldsymbol{b}, \boldsymbol{G})$ if $\boldsymbol{a} + \mathbb{N}\boldsymbol{F} \subset \boldsymbol{b} + \mathbb{N}\boldsymbol{G}$. The *standard pairs* of an ideal I are the maximal elements of the set of all proper pairs of I with this partial order. We denote by $\operatorname{Std}(I)$ the set of all standard pairs of an ideal I.

We remark that our notation here differs from existing notation for standard pairs over polynomial rings. Over the polynomial ring $\mathbb{K}[x_1, x_2, \ldots, x_n]$, a pair is a tuple $(x^{\boldsymbol{a}}, V)$ where $x^{\boldsymbol{a}}$ is a monomial $x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$ for some integer vector $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ and V is a set of variables [Hoşten and Smith 2002; Sturmfels et al. 1995]. From the viewpoint of affine semigroup rings, the polynomial ring is a special case when the underlying affine semigroup is generated by an $n \times n$ identity matrix $\boldsymbol{I}$. Since the cone $\mathbb{R}_{\geq 0}\boldsymbol{I}$ is a simplicial cone, i.e., every subset of rays form a face, we may interpret V as a face. The following example shows the different notations for the standard pairs of a monomial ideal

$$I = \left\langle x^{\left[\begin{smallmatrix}1\\3\\1\end{smallmatrix}\right]}, x^{\left[\begin{smallmatrix}1\\2\\2\end{smallmatrix}\right]}, x^{\left[\begin{smallmatrix}0\\3\\2\end{smallmatrix}\right]}, x^{\left[\begin{smallmatrix}0\\2\\3\end{smallmatrix}\right]} \right\rangle$$

in the polynomial ring $\mathbb{K}[x_1, x_2, x_3]$:

In `Macaulay2`,

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

i2 : I = monomialIdeal(x*y^3*z, x*y^2*z^2, y^3*z^2, y^2*z^3)
                          3      2 2    3 2    2 3
o2 = monomialIdeal (x*y z, x*y z , y z , y z )

o2 : MonomialIdeal of R

i3 : standardPairs I
o3 = {{1, {x, z}}, {y, {x, z}}, {1, {x, y}}, {z, {y}},
   2          2 2
{y z, {x}}, {y z , {}}}

o3 : List
```

whereas in the given library `StdPairs` in `SageMath`,

```
sage: from stdpairs import *

sage: A = matrix(ZZ,[[1,0,0],[0,1,0],[0,0,1]])

sage: Q = AffineMonoid(A)

sage: M = matrix(ZZ,[[1,1,0,0],[3,2,3,2],[1,2,2,3]])

sage: I = MonomialIdeal(M,Q)

sage: I.standard_cover()
{(1,): [([[0], [0], [1]]^T,[[0], [1], [0]])],
 (0, 2): [([[0], [1], [0]]^T,[[1, 0], [0, 0], [0, 1]]),
  ([[0], [0], [0]]^T,[[1, 0], [0, 0], [0, 1]])],
 (0, 1): [([[0], [0], [0]]^T,[[1, 0], [0, 1], [0, 0]])],
 (): [([[0], [2], [2]]^T,[[], [], []])],
 (0,): [([[0], [2], [1]]^T,[[1], [0], [0]])]}
```

`()`, `(0,)`, `(0, 2)`, `(0, 1)`, and `(1,)` in `StdPairs` of `SageMath` are indices of columns of `A`. These denote `{}`, `{x}`, `{x, y}`, `{x, z}`, and `{y}`, respectively, in `Macaulay2`. Therefore, for example, the pair `([[0], [0], [1]]^T,[[0], [1], [0]])` will represent `{z, {y}}`, while the pair `([[0], [0], [0]]^T,[[1, 0], [0, 0], [0, 1]])` represents `{1, {x, z }}`, and so on. Therefore, this example shows that `StdPairs` is consistent with `Macaulay2`.

2.2. ***Classes in StdPairs.*** We implement three classes related to affine semigroups, semigroup ideals, and proper pairs respectively. This implementation is based on `SageMath` 9.1 with `Python` 3.7.3. and the `4ti2` package. Detailed usage and examples of each method or object can be found using the command `<method_name>?` in `SageMath` or https://byeongsuyu.github.io/StdPairs/, the documentation of `StdPairs` made by the `Sphinx` package.

Class AffineMonoid. This class is constructed by using an integer matrix $\boldsymbol{A}$. The name follows the convention of `SageMath` which distinguishes monoid from semigroup. In `SageMath`, $\boldsymbol{A}$ can be expressed as a 2-dimensional `NumPy.ndarray` type or an integer matrix of `SageMath`. For example,

```
sage: from stdpairs import *
sage: A = matrix(ZZ,[[1,2],[0,2]])
sage: Q = AffineMonoid(A)
```

generates Q as a type of `AffineMonoid`. This class has several methods as explained below:

- `Q.gens()` returns a matrix generating an affine monoid Q as `NumPy.ndarray` type. This may not be a minimal generating set of Q.
- `Q.mingens()` returns a minimal generating matrix of an affine monoid of Q.
- `Q.poly()` returns a real cone $\mathbb{R}_{\geq 0}Q$ represented as a type of `Polyhedron` in `SageMath`. If one generates Q with `True` parameter, i.e.,

  ```
  sage: from stdpairs import *
  sage: A = matrix(ZZ,[[1,2],[0,2]])
  sage: Q = AffineMonoid(A,is_normaliz=True)
  ```

  then `Q.poly()` is of a class of `Normaliz` integral polyhedron. This requires the `PyNormaliz` package. See [Köppe and Labbé 2019] for more details.
- `Q.face_lattice()` returns a finite lattice containing all faces of the affine semigroup. A face in the lattice is saved as a tuple storing column numbers of generators A. This lattice is of type `FiniteLatticePoset` in `SageMath`. For example,

  ```
  sage: Q.face_lattice()
  Finite lattice containing 5 elements

  sage: Q.face_lattice().list()
  [(-1,), (), (0,), (1,), (0, 1)]
  ```
- `Q.index_to_face()` returns a `dictionary` type object whose keys are tuples denoting indices of column vectors consisting of faces, and whose items are corresponding faces of `Q.poly()`. For example,

  ```
  sage: Q.index_to_face()
  {(-1,): A -1-dimensional face of a Polyhedron in ZZ^2,
   (): A 0-dimensional face of a Polyhedron in ZZ^2
   defined as the convex hull of 1 vertex,
   (0,): A 1-dimensional face of a Polyhedron in ZZ^2
   defined as the convex hull of 1 vertex and 1 ray,
   (1,): A 1-dimensional face of a Polyhedron in ZZ^2
   defined as the convex hull of 1 vertex and 1 ray,
   (0,1): A 2-dimensional face of a Polyhedron in ZZ^2
    defined as the convex hull of 1 vertex and 2 rays}
  ```
- `Q.index_of_face(matrix face)` returns a face as a tuple of indices of column vectors of a generator A corresponding to a given submatrix `face` of A. For example,

  ```
  sage: M = matrix(ZZ,[[2],[2]])

  sage: Q.index_of_face(M)
  (1,)
  ```

  Here, `face` should be a submatrix of `Q.gens()` which form a face.

- `Q.face(tuple index)` returns a face as a submatrix of a generator $\boldsymbol{A}$ corresponding to a given tuple `index`. For example,

```
sage: Q.face((1,))
array([[2],
       [2]])
```

- `Q.integral_support_vectors()` return a `dictionary` type object whose keys are tuples denoting faces and whose items are integral support functions of facets containing $\boldsymbol{F}$ as a vector form. An *integral support function* $\phi_{\boldsymbol{H}}$ of a facet $\boldsymbol{H}$ is a linear function $\phi_{\boldsymbol{H}} : \mathbb{R}^d \to \mathbb{R}$ such that $\phi_{\boldsymbol{H}}(\mathbb{Z}^d) = \mathbb{Z}$, $\phi_{\boldsymbol{H}}(\boldsymbol{a}) \geq 0$ for all column vectors $\boldsymbol{a}$ of generators A, and $\phi_{\boldsymbol{H}}(\boldsymbol{a}) = 0$ if and only if $\boldsymbol{a} \in \boldsymbol{H}$. By linearity, $\phi_{\boldsymbol{H}}(\boldsymbol{a}) = \boldsymbol{b} \cdot \boldsymbol{a}$ for some rational vector $\boldsymbol{b}$. We call $\boldsymbol{b}$ an *integral support vector*. Each item of `Q.integral_support_vectors()` is a matrix as `NumPy.ndarray` type whose rows are integral support vectors of facets containing the given face. For example,

```
sage: Q.integral_support_vectors()
{(): array([[ 0,  1],
           [ 1, -1]]),
 (0,): array([[0, 1]]),
 (1,): array([[ 1, -1]]),
 (0, 1): array([], dtype=int64)}
```

  In this code, `()` denoting 0 has two integral support vectors, since it is an intersection of two facets `(0,)` and `(1,)`, while `(0,1)` has no such integral support vectors since it is not a proper face but the affine semigroup itself. See [Matusevich and Yu 2020, Definition 2.1] for the precise definition of a (primitive) integral support function.

- `Q.is_empty()` returns a boolean value indicating whether Q is a trivial affine semigroup or not. A *trivial affine semigroup* is an empty set as an affine semigroup.

- `Q.is_pointed()` returns a boolean value indicating whether Q is a pointed affine semigroup or not.

- `Q.is_element(vector` $\boldsymbol{b}$`)` returns nonnegative integral inhomogeneous solutions (minimal integer solutions) of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ using `zsolve` in [4ti2]. If $\boldsymbol{b}$ is not an element of an affine semigroup Q, then it returns an empty matrix. The vector $\boldsymbol{b}$ should be a `NumPy.ndarray` type 2-dimensional object with one column, or a matrix of `SageMath` with only one column.

- `Q.save_txt()` returns a string containing information about `Q`. This can be loaded again using `txt_to_affinemonoid(string info)`, which will be explained in Section 2.3.

- `Q.save(string path)` saves the given object `Q` as binary file on the given path. This can be loaded again using `load(path)`, a pre-existing global function of `SageMath`.

Moreover, one can directly compare affine semigroups using the equality operator `==` in `SageMath`.

Class MonomialIdeal. This class is constructed by an affine semigroup Q and generators of an ideal as a matrix form, say $\boldsymbol{M}$, which is a 2-dimensional `NumPy.ndarray` object or an integer matrix of `SageMath`.

For example,

```
sage: M = matrix(ZZ,[[4,6],[4,6]])
sage: I = MonomialIdeal(M,Q)
sage: I
An ideal whose generating set is
[[4]
 [4]]
```

As shown in the example above, this class stores only minimal generators of the ideal. The attributes and methods are explained below:

- `I.gens()` returns the minimal generators of I as a `NumPy.ndarray` type object.
- `I.ambient_monoid()` returns the ambient affine semigroup of I.
- `I.standard_cover(verbose = False)` returns the *standard cover* of I, a `dictionary` object whose keys are faces and whose items are lists consisting of `ProperPair` type objects whose face is equal to the corresponding key. `ProperPair` objects will be explained in Section 2.2. The definition of the standard cover will be given in Section 3.1. Users can check the progress of the computation if `verbose=True`.
- `I.overlap_classes()` returns a `dictionary` object whose keys are tuples denoting faces and whose items are lists of lists representing overlap classes of I. An *overlap class* of an ideal I is a set of standard pairs such that their representing submonoids intersect nontrivially.
- `I.maximal_overlap_classes()` returns all maximal overlap classes of I with divisibility. An overlap class is *maximal with divisibility* if every pair in the overlap class can divide only pairs in itself. See [Matusevich and Yu 2020, Section 3] for the detail.
- `I.irreducible_decomposition()` returns a list of components of the irredundant irreducible primary decomposition of I.
- `I.associated_primes()` returns all associated prime ideals of I as a `dictionary` type. In other words, the function returns a dictionary whose keys are faces of the affine semigroup as a `tuple` and whose values are associated prime ideals corresponding to the face in its key.
- `I.multiplicity(ideal P or face` $\boldsymbol{F}$`)` returns a multiplicity of I over the given associated prime P. Since there is a one-to-one correspondence between monomial prime ideals and faces of an affine semigroup, this method takes the face $\boldsymbol{F}$ (as a tuple) corresponding to a prime ideal P as a valid input instead.
- `I.is_element(vector` $\boldsymbol{b}$`)` returns nonnegative integral inhomogeneous solutions (minimal integer solutions) of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} - \boldsymbol{a}$ for each generator $\boldsymbol{a}$ of I using `zsolve` in [4ti2]. If $\boldsymbol{b}$ is an element of the ideal, then it returns a list $[\boldsymbol{x}, \boldsymbol{a}]$ for some generator $\boldsymbol{a}$ such that $\boldsymbol{a} + \boldsymbol{A}\boldsymbol{x}^T = \boldsymbol{b}$. Otherwise, it returns an empty matrix. The vector $\boldsymbol{b}$ should be a `NumPy.ndarray` type 2-dimensional object with one column, or a matrix of `SageMath` with only one column.

- `I.is_standard_monomial(vector` $\boldsymbol{b}$`)` returns a boolean value indicating whether the given vector $\boldsymbol{b}$ is a standard monomial or not.
- `I.is_principal()` returns a boolean value indicating whether I is principal or not. Likewise, the similar methods `I.is_empty()`, `I.is_irreducible()`, `I.is_primary()`, `I.is_prime()`, and `I.is_radical()` return a boolean value indicating whether I has the properties implied by their name or not.
- `I.radical()` returns the radical of I as a `MonomialIdeal` object.
- `I.intersect(J)` returns an intersection of two ideals I and J as a `MonomialIdeal` object. Likewise, addition +, multiplication $*$, and comparison == are defined between two objects. The following example shows an addition of two monomial ideals in `SageMath`:

```
sage: I = MonomialIdeal(matrix(ZZ,[[4,6],[4,6]]),Q)

sage: J = MonomialIdeal(matrix(ZZ,[[5],[0]]),Q)

sage: I.intersect(J)
An ideal whose generating set is
[[9]
 [4]]

sage: I+J
An ideal whose generating set is
[[5 4]
 [0 4]]

 sage: I*J
An ideal whose generating set is
[[9]
 [4]]
```

- `I.save_txt()` returns a string which can be used to recover the object I and its precalculated properties without calculation. That is, it contains not only the generators of I, but also its standard cover, overlap classes, associated primes, and irreducible primary decompositions if they were calculated. This can be loaded again using `txt_to_monomialideal(string info)` (see Section 2.3).
- `I.save(string path)` saves the given object `I` as a binary file on the given path. This can be loaded again using `load(path)`, a pre-existing global function of `SageMath`.

Class ProperPair. A proper pair $(\boldsymbol{a}, \boldsymbol{F})$ of an ideal I can be declared in `SageMath` by specifying an ideal I, a standard monomial $\boldsymbol{a}$ as a matrix form (or `NumPy` 2D array), and a face $\boldsymbol{F}$ as a tuple. If $(\boldsymbol{a}, \boldsymbol{F})$ is not proper, then `SageMath` calls a `ValueError`. The following example shows two ways of defining a proper pair:

```
sage: import numpy as np

sage: I = MonomialIdeal(matrix(ZZ,[[4,6],[4,6]]),Q)

sage: PP = ProperPair(np.array([2,0])[np.newaxis].T,(0,),I)

sage: PP
([[2], [0]]^T,[[1], [0]])

sage: QQ = ProperPair(np.array([2,0])[np.newaxis].T,(0,),I,
....: properness =True)

sage: QQ
([[2], [0]]^T,[[1], [0]])
```

The second line tests whether the pair is a proper pair of the given ideal I or not before generating PP. However, the fourth line with properness=True generates QQ without checking whether QQ is a proper pair of I or not. Use the third parameter with True only if the generating pair is proper a priori. In any case, each PP and QQ denotes a proper pair whose initial monomial is $\left[\begin{smallmatrix}2\\0\end{smallmatrix}\right]$ and whose face is $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$.

The attributes and methods are explained below. We assume that PP denotes a proper pair $(\boldsymbol{a}, \boldsymbol{F})$.

- PP.monomial(), PP.face(), and PP.ambient_ideal() return the initial monomial $\boldsymbol{a}$ (as a NumPy 2D array), the face $\boldsymbol{F}$ (as a tuple), and its ambient ideal (as an object of AffineMonoid) respectively.
- PP.is_maximal() returns a boolean value indicating whether the given pair is maximal with respect to the divisibility of proper pairs of the ambient ideal. If *PP* is generated without testing whether its monomial is in the given ideal I or not, this method raises a warning instead of returning a boolean value.
- PP.is_element(vector $\boldsymbol{b}$) returns nonnegative integral inhomogeneous solutions (minimal integer solutions) of $\boldsymbol{a} + \boldsymbol{F}\boldsymbol{x} = \boldsymbol{b}$ using zsolve in [4ti2]. If $\boldsymbol{b}$ is not an element of the submonoid $\boldsymbol{a} + \mathbb{N}\boldsymbol{F}$, then it returns an empty matrix.
- Like AffineMonoid or MonomialIdeal, one can directly compare proper pairs using the equality operator == in SageMath.

2.3. *Global functions.*

- prime_ideal(tuple face, AffineMonoid Q) returns a prime ideal of the given AffineMonoid object Q corresponding to the tuple object face as an object of MonomialIdeal.

```
sage: prime_ideal((1,),Q)
An ideal whose generating set is
[[1]
 [0]]
```

- div_pairs(pair PP, pair QQ) will return a matrix whose column $\boldsymbol{u}$ is a minimal solution of $\boldsymbol{a} + \boldsymbol{u} + \mathbb{N}\boldsymbol{F} \subseteq \boldsymbol{b} + \mathbb{N}\boldsymbol{G}$ if $PP = (\boldsymbol{a}, \boldsymbol{F})$ and $QQ = (\boldsymbol{b}, \boldsymbol{G})$. The returned value is a nonempty matrix if and only if a pair *PP* divides a pair *QQ*. For example, suppose two pairs *PP* and *QQ* are $\left[\begin{smallmatrix}2\\0\end{smallmatrix}\right] + \mathbb{N}\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ and $\left[\begin{smallmatrix}3\\0\end{smallmatrix}\right] + \mathbb{N}\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ respectively. Then,

```
sage: I = MonomialIdeal(matrix(ZZ,[[4,6],[4,6]]),Q)

sage: PP = ProperPair(matrix(ZZ,[[2],[0]]),(0,),I)

sage: QQ = ProperPair(matrix(ZZ,[[3],[0]]),(0,),I)

sage: div_pairs(PP,QQ)
[1]
[0]

sage: div_pairs(QQ,PP)
[0]
[0]
```

since $\left(\left[\begin{smallmatrix}2\\0\end{smallmatrix}\right] + \mathbb{N}\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]\right) \supseteq \left(\left[\begin{smallmatrix}3\\0\end{smallmatrix}\right] + \mathbb{N}\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]\right)$.

- `txt_to_affinemonoid(string info)` (or `txt_to_monomialideal(string info)`) loads an `AffineMonoid` object (or a `MonomialIdeal` object) in the string `info`, which is generated by the `AffineMonoid.save_txt()` (or `MonomialIdeal.save_txt()`) methods. These are useful for users who want to avoid repeating calculations which were previously done. For example, the ideal J in

```
sage: I = MonomialIdeal(matrix(ZZ,[[4,2],[4,0]]),Q)

sage: I.standard_cover()
{(): [([[1], [0]]^T,[[], []]),
  ([[0], [0]]^T,[[], []]),
  ([[3], [2]]^T,[[], []]),
  ([[2], [2]]^T,[[], []])]}

sage: J=txt_to_monomialideal(I.save_txt())

sage: J.standard_cover()
{(): [([[1], [0]]^T,[[], []]),
  ([[0], [0]]^T,[[], []]),
  ([[3], [2]]^T,[[], []]),
  ([[2], [2]]^T,[[], []])]}
```

  is a new `MonomialIdeal` object; however, it does not need time to calculate its standard cover, since precalculated information of the standard cover was stored in `I.save_txt()` and transferred to J.

- `pair_difference(ProperPair PP, ProperPair QQ)` is a global function which decomposes $PP \setminus QQ$ as a finite union of pairs. See Theorem 1 and subsequent arguments for details.

- `from_macaulay2(string var_name)` and `to_macaulay2(MonomialIdeal I)` are global functions used for communicating with `Macaulay2` objects. See Section 4 for details.

3. Implementation of an algorithm finding standard pairs.

3.1. *Case 1: Principal ideal.*

A *cover* of standard monomials of an ideal I is a set of proper pairs of I such that the union of all subsemigroups $\boldsymbol{a}+\mathbb{N}\boldsymbol{F}$ corresponding to an element $(\boldsymbol{a}, \boldsymbol{F})$ of the cover is equal to the set of all standard monomials. The *standard cover* of an ideal I is a cover of I whose elements are standard pairs. The standard cover of a monomial ideal I is unique by the maximality of standard pairs among all proper pairs of I. A key idea in [Matusevich and Yu 2020, Section 4] is to construct covers containing all standard pairs. Once a cover is obtained, we can then produce the standard cover.

The following result helps to compute the standard cover in the special case of a principal ideal.

Theorem 1 [Matusevich and Yu 2020, Theorem 4.1]. *Let $\boldsymbol{b}, \boldsymbol{b}' \in \mathbb{N}\boldsymbol{A}$ and let $\boldsymbol{G}, \boldsymbol{G}'$ be faces of A such that $\boldsymbol{G} \cap \boldsymbol{G}' = \boldsymbol{G}$. There exists an algorithm to compute a finite collection C of pairs over faces of $\boldsymbol{G}$ such that*

$$(\boldsymbol{b}+\boldsymbol{G}) \smallsetminus (\boldsymbol{b}'+\boldsymbol{G}') = \bigcup_{(\boldsymbol{a},\boldsymbol{F})\in C} (\boldsymbol{a}+\boldsymbol{F}).$$

The *pair difference* of the pairs $(\boldsymbol{b}, \boldsymbol{G})$ and $(\boldsymbol{b}', \boldsymbol{G}')$ is a finite collection of pairs over faces of $\boldsymbol{G}$ given by Theorem 1.

Corollary 2. *Given a principal ideal $I = \langle \boldsymbol{b} \rangle$, the pair difference of pairs $(0, \boldsymbol{A})$ and $(\boldsymbol{b}, \boldsymbol{A})$ is the standard cover of I.*

Proof. Theorem 1 implies that the pair difference is a cover of I. To see it is the standard cover, suppose that the ambient affine semigroup is generated by $\boldsymbol{A} = [\boldsymbol{a}_1 \cdots \boldsymbol{a}_n]$. Let $(\boldsymbol{c}, F)$ be a proper pair in the pair difference. Without loss of generality, we assume that $F = [\boldsymbol{a}_1 \cdots \boldsymbol{a}_m]$ for some $m < n$ by renumbering indices. By the proof of Theorem 1 in [Matusevich and Yu 2020], $\boldsymbol{c} = A \cdot \boldsymbol{u}$ where $x^{\boldsymbol{u}} \in \mathbb{K}[\mathbb{N}^n]$ is a standard monomial such that $(x^{\boldsymbol{u}}, \{x_1, \dots, x_m\})$ is a standard pair of some monomial ideal J in $\mathbb{K}[\mathbb{N}^n]$.

Suppose that there exists $(\boldsymbol{d}, \boldsymbol{G})$ such that $F \subseteq \boldsymbol{G}$ and $\boldsymbol{d} + \boldsymbol{g} = \boldsymbol{c}$ for some $\boldsymbol{g} \in G$. Since $\boldsymbol{d} \in \mathbb{N}\boldsymbol{A}$, $\boldsymbol{d} = A\boldsymbol{w}$ for some $\boldsymbol{w} \in \mathbb{N}^n$. Since A is pointed, $\boldsymbol{w}$ is, coordinatewise, less than $\boldsymbol{u}$. Thus, $(x^{\boldsymbol{w}}, \{x_1, \dots, x_m\})$ contains $(x^{\boldsymbol{u}}, \{x_1, \dots, x_m\})$. Lastly, $(x^{\boldsymbol{w}}, \{x_1, \dots, x_m\})$ is a proper pair of J, otherwise, there exists $x^{\boldsymbol{v}} \in \mathbb{K}[x_1, \dots, x_m] \subseteq \mathbb{K}[\mathbb{N}^n]$ such that $x^{\boldsymbol{w}+\boldsymbol{v}} \in J$. Then, $x^{\boldsymbol{g}} x^{\boldsymbol{w}+\boldsymbol{v}} \in J \Longrightarrow x^{\boldsymbol{u}+\boldsymbol{v}} \in J \cap (x^{\boldsymbol{u}}, \{x_1, \dots, x_m\}) = \varnothing$ leads to a contradiction.

Thus, by maximality of the standard pair, $\boldsymbol{w} = \boldsymbol{u}$. This implies $\boldsymbol{d} = \boldsymbol{c}$. Moreover, $G = F$, otherwise there exists $j \in \{1, 2, \dots, n\} \setminus \{1, \dots, m\}$ such that $x^{\boldsymbol{u}} x_j^l \notin J$ for any l, which implies that $(x^{\boldsymbol{u}}, \{x_1, \dots, x_m, x_j\})$ is a proper pair of J strictly containing a standard pair $(x^{\boldsymbol{u}}, \{x_1, \dots, x_m\})$ of J, a contradiction. □

Theorem 1 is implemented as a method `pair_difference(`$(\boldsymbol{b}, \boldsymbol{F})$`, `$(\boldsymbol{b}', \boldsymbol{F}')$`)` within the library `StdPairs`. The two input arguments should be of type `ProperPair`. It returns the pair difference of pairs $(\boldsymbol{b}, \boldsymbol{F})$ and $(\boldsymbol{b}', \boldsymbol{F}')$ with `dictionary` type, called `Cover`. `Cover` classifies pairs by their faces. For example, the code below shows the pair difference of pairs $(0, \boldsymbol{A})$ and $((0, 2), \boldsymbol{A})$, which are

$$\left(0, \begin{bmatrix}2\\0\end{bmatrix}\right), \left(\begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}2\\0\end{bmatrix}\right), \left(\begin{bmatrix}1\\2\end{bmatrix}, \begin{bmatrix}2\\0\end{bmatrix}\right), \text{ and } \left(\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}2\\0\end{bmatrix}\right).$$

```
sage: from stdpairs import *
sage: Q = AffineMonoid(matrix(ZZ, [[2,0,1],[0,1,1]]))
sage: I = MonomialIdeal(matrix(ZZ,0),Q)
sage: C = ProperPair(np.array([[0,0]]).T, (0,1,2), I )
sage: D = ProperPair(np.array([[0,2]]).T, (0,1,2), I )
sage: print(pair_difference(C,D))
{(0,): [([[1], [2]]^T,[[2], [0]]), ([[1], [1]]^T,[[2], [0]]),
([[0], [1]]^T,[[2], [0]]), ([[0], [0]]^T,[[2], [0]])]}
```

By Corollary 2, this is the standard cover of the ideal $I = \langle (0, 2) \rangle$ in an affine semigroup $\mathbb{N}\left[\begin{smallmatrix}2&0&1\\0&1&1\end{smallmatrix}\right]$.

The method `pair_difference(`$(\boldsymbol{b}, \boldsymbol{F})$`, `$(\boldsymbol{b}', \boldsymbol{F}')$`)` uses `standardPairs` of `Macaulay2` internally to find standard pairs of a polynomial ring, which is implemented by [Hoşten and Smith 2002]. Briefly, the method `pair_difference(`$(\boldsymbol{b}, \boldsymbol{F})$`, `$(\boldsymbol{b}', \boldsymbol{F}')$`)` calculates minimal solutions of the integer linear system

$$\begin{bmatrix} F & -F' \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} = \boldsymbol{b}' - \boldsymbol{b}$$

using `zsolve` in `4ti2`. The solutions form an ideal J of a polynomial ring in the proof of Theorem 1 on `Macaulay2`. Next, `standardPairs` derives standard pairs of J. Lastly, the method `pair_difference` constructs proper pairs based on the standard pairs of J, and classifies the proper pairs based on their faces and returns the pair difference.

3.2. *Case 2: General ideal.* [Matusevich and Yu 2020, Proposition 4.4] gives an algorithm to find the standard cover of nonprincipal monomial ideals.

Proposition 3 [Matusevich and Yu 2020, Proposition 4.4]. *Let I be a monomial ideal in $\mathbb{K}[\mathbb{N}\boldsymbol{A}]$. There is an algorithm whose input is a cover of the standard monomials of I, and whose output is the standard cover of I.*

According to the proof of [Matusevich and Yu 2020, Proposition 4.4], this is achieved by repeating the procedures below.

(1) Input: C_0, an initial cover of I.

(2) For each $(\boldsymbol{a}, F) \in C_0$, find minimal solutions of $(\boldsymbol{a} + \mathbb{R}F) \cap \mathbb{N}\boldsymbol{A}$ using the primitive integral support functions. (See [Matusevich and Yu 2020, Lemma 4.2] for the detail.)

If $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_m$ are the minimal solutions of $(\boldsymbol{a} + \mathbb{R}F) \cap \mathbb{N}\boldsymbol{A}$, then we construct pairs such as $(\boldsymbol{b}_1, F), (\boldsymbol{b}_2, F), \ldots, (\boldsymbol{b}_m, F)$ and store them in the attribute C_1.

(3) For each pair $(\boldsymbol{b}, F) \in C_1$, construct $(\boldsymbol{b}, G)$ for any face G which is not strictly contained in F. If $(\boldsymbol{b}, G)$ is a proper pair of I, save $(\boldsymbol{b}, G)$ on the attribute C_2.

(4) If C_0 is equal to C_2, we are done. Otherwise, set $C_0 := C_2$ and repeat the above process.

The method `_czero_to_cone(`C_0, I`)` in the hidden module `_stdpairs` of `StdPairs` implements (2) to return C_1. It calls the method `_minimal_holes(vector` $\boldsymbol{a}$`, face F, affine semigroup A)` internally, which is the implementation of Lemma 4.2 in [Matusevich and Yu 2020]. The method `_cone_to_ctwo(`C_1, I`)` implements (3). Since the constructor function of the class `ProperPair` checks whether the pair is proper or not, the method `_cone_to_ctwo(`C_1, I`)` tries to construct proper pairs as an attribute in `SageMath` and records them if it is successful.

Now we are ready to find the standard cover of a general ideal I whose minimal generators are $\langle \boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \rangle$. One can find standard pairs as in [Matusevich and Yu 2020, Theorem 4.5] as described below:

(1) Find the standard cover C of $\langle \boldsymbol{b}_1 \rangle$ using pair difference.

(2) For $i = 2$ to n:
   (a) For each pair $(\boldsymbol{b}, F)$ in C, replace it with elements of the pair difference of pairs $(\boldsymbol{b}, F)$ and $(\boldsymbol{b}_i, A)$. After this process, C is a cover of an ideal $\langle \boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_i \rangle$.
   (b) Using the algorithm of Proposition 3, find the standard cover C' of $\langle \boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_i \rangle$.
   (c) Replace C with C'.

(3) Return C.

The returned value C is now the standard cover of I.

The library `StdPairs` implements [Matusevich and Yu 2020, Theorem 4.5] as a hidden method `_standard_pairs(`I`)`. This method has an input I whose type is `MonomialIdeal`. It returns a cover whose type is `dictionary`, classifying standard pairs by its face. For example, the code below shows that the standard cover of an ideal generated by

$$\begin{bmatrix} 2 & 2 & 2 \\ 0 & 1 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

in an affine semigroup

$$\mathbb{N}\boldsymbol{A} = \mathbb{N}\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

is

$$\left\{ \left(0, \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}\right), \left(\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}\right), \quad \text{and} \quad \left(\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}\right) \right\}.$$

```
sage: from stdpairs import *
sage: Q=AffineMonoid(matrix(ZZ,[[0,1,1,0],[0,0,1,1],[1,1,1,1]]))
sage: I=MonomialIdeal(matrix(ZZ,[[2,2,2],[0,1,2],[2,2,2]]),Q)
sage: I.standard_cover()
{(0, 3): [([[1], [0], [1]]^T,[[0, 0], [0, 1], [1, 1]]),
  ([[1], [1], [1]]^T,[[0, 0], [0, 1], [1, 1]]),
  ([[0], [0], [0]]^T,[[0, 0], [0, 1], [1, 1]])]}
```

4. Compatibility with Normaliz package in SageMath and Macaualay2.

`Normaliz` is a package in `SageMath` and `Macaulay2` for finding Hilbert bases of rational cones and their normal affine monoid [Bruns and Ichim 2010]. `StdPairs` has methods translating classes in Section 2 into objects in the `Normaliz` package. If an affine semigroup $\mathbb{N}\boldsymbol{A}$ is *normal*, i.e., $\mathbb{N}\boldsymbol{A} = \mathbb{Z}^d \cap \mathbb{R}_{\geq 0}A$, then this translation works well. However, if it is not normal, then this translates $\mathbb{N}\boldsymbol{A}$ into its saturation described in Section 2.

For `SageMath`, one can have a polyhedron over $\mathbb{Z}$ with the `Normaliz` package in `SageMath` by adding an argument `True` on the constructor of `AffineMonoid`. For example, the code below gives an `AffineMonoid` class attribute Q whose attribute `Q.poly()` is a polyhedron over $\mathbb{Z}$ with `Normaliz`. Therefore, you can use all methods on `Normaliz` object. For example,

```
sage: from stdpairs import *
sage: Q=AffineMonoid(matrix(ZZ, [[0,1,1,0],[0,0,1,1],[1,1,1,1]]),
is_normaliz=True)
sage: Q.poly().hilbert_series([0,0,1])
(t + 1)/(-t^3 + 3*t^2 - 3*t + 1)
```

The method `to_macaulay2( MonomialIdeal I)` returns a dictionary storing attributes of `Macaulay2` computations. This dictionary contains an affine semigroup ring, a list of generators of an ideal, and a list of standard pairs in `Macaulay2`. For example,

```
sage: from stdpairs import *
sage: Q=AffineMonoid(matrix(ZZ,[[0,1,1,0],[0,0,1,1],[1,1,1,1]]))
sage: I=MonomialIdeal(matrix(ZZ,[[2,2,2],[0,1,2],[2,2,2]]),Q)
```

```
sage: S=to_macaulay2(I)

sage: S
{'AffineSemigroupRing': ZZ[c, a*c, a*b*c, b*c]

 monomial subalgebra of PolyRing,
 'MonomialIdeal':   2 2   2   2   2 2 2
 {a c , a b*c , a b c }

 List,
 'StandardCover': {{1, {c, b*c}}, {a*c, {c, b*c}}, {a*b*c, {c, b*c}}}

 List}
```

Moreover, `Macaulay2` objects `AffineSemigroupRing`, `MonomialSubalgebra`, and `list` of a standard cover can be accessed via `macaulay.eval(string)` method with strings `R`, `I`, and `SC`. For instance, the example below shows how to access such `Macaulay2` objects:

```
sage: macaulay2.eval('R')
ZZ[c, a*c, a*b*c, b*c]

monomial subalgebra of PolyRing

sage: macaulay2.eval('I')
  2 2   2   2   2 2 2
{a c , a b*c , a b c }

List

sage: macaulay2.eval('SC')
{{1, {c, b*c}}, {a*c, {c, b*c}}, {a*b*c, {c, b*c}}}

List
```

In `Macaulay2`, a type `MonomialSubalgebra` in the `Normaliz` package may correspond to an affine semigroup ring. Since `Normaliz` has no attributes for a monomial ideal of the type `MonomialSubalgebra`, the ideal is stored as a list of its generators. The standard cover of I is also sent to `Macaulay2` as a nested list, similar to the output of the method `standardPairs` in `Macaulay2`.

In the other direction, `from_macaulay(Macaulay2` S`)` translates `monomialSubalgebra` object S of `Macaulay2` into an `AffineMonoid` object in `StdPairs`. For example,

```
sage: R = macaulay2.eval('ZZ[x,y,z]')

sage: macaulay2.needsPackage('"Normaliz"')
Normaliz

sage: macaulay2.eval('S=createMonomialSubalgebra {x^2*y, x*z, z^3}')
    2         3
ZZ[x y, x*z, z ]

monomial subalgebra of ZZ[x..z]

sage: Q=from_macaulay2('S')

sage: Q
An affine semigroup whose generating set is
[[2 1 0]
 [1 0 0]
 [0 1 3]]
```

Acknowledgements. We would like to express our deepest appreciation to Laura Matusevich for conversations and helpful comments on draft versions of this paper. Also, we are grateful to Matthias Köppe for advice on using `zsolve` in `4ti2`. Lastly, we are indebted to an anonymous reviewer for insightful comments on the documentation and distribution of `StdPairs`.

SUPPLEMENT. The online supplement contains version 1.0 of `StdPairs`.

REFERENCES.

[4ti2] 4ti2 team, “4ti2—A software package for algebraic, geometric and combinatorial problems on linear spaces”, available at https://4ti2.github.io.

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

[Bruns and Ichim 2010] W. Bruns and B. Ichim, “Normaliz: algorithms for affine monoids and rational cones”, *J. Algebra* **324**:5 (2010), 1098–1113. MR Zbl

[Hoşten and Smith 2002] S. Hoşten and G. G. Smith, “Monomial ideals”, pp. 73–100 in *Computations in algebraic geometry with Macaulay* 2, edited by D. Eisenbud et al., Algorithms Comput. Math. **8**, Springer, 2002. MR Zbl

[Köppe and Labbé 2019] M. Köppe and J.-P. Labbé, “The Normaliz backend for polyhedral computations”, Sage module, 2019, available at https://doc.sagemath.org/html/en/reference/discrete_geometry/sage/geometry/polyhedron/backend_normaliz.html.

[Matusevich and Yu 2020] L. F. Matusevich and B. Yu, “Standard pairs for monomial ideals in semigroup rings”, 2020. arXiv

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

[Sturmfels et al. 1995] B. Sturmfels, N. V. Trung, and W. Vogel, “Bounds on degrees of projective schemes”, *Math. Ann.* **302**:3 (1995), 417–432. MR Zbl

RECEIVED: 22 Oct 2020 REVISED: 12 Jul 2021 ACCEPTED: 10 Aug 2021

BYEONGSU YU:
byeongsu.yu@tamu.edu
Department of Mathematics, Texas A&M University, College Station, TX, United States

JSAG **11** (2021), 143–153
https://doi.org/10.2140/jsag.2021.11.143

The Journal of Software for
Algebra and Geometry

Computations with rational maps between multi-projective varieties

GIOVANNI STAGLIANÒ

ABSTRACT: We briefly describe the algorithms behind some of the functions provided by the *Macaulay*2 package *MultiprojectiveVarieties*, a package for multi-projective varieties and rational maps between them.

INTRODUCTION. This paper is a natural sequel of [Staglianò 2018], where we presented some of the algorithms implemented in the *Macaulay*2 package [Cremona], related to computations with rational and birational maps between closed subvarieties of projective spaces.

Here we describe methods for working with rational and birational maps between multiprojective varieties, that is, closed subvarieties of products of projective spaces. For instance, we explain how to compute the degrees of such maps, their graphs, and the inverses when they exist. All these methods are implemented in the Macaulay2 package *MultiprojectiveVarieties*.

From a theoretical point of view, we know that every multiprojective variety is isomorphic, via the Segre embedding, to a projective variety embedded into a single projective space. Therefore, every rational map between multiprojective varieties can be regarded as a rational map between ordinary subvarieties of projective spaces. This, however, introduces a lot of new variables, making computation more difficult.

Moreover, basic constructions on rational maps naturally lead one to consider rational maps between multiprojective varieties. For instance, the graph of a rational map is a closed subvariety of the product of the source and of the target of the map. Using the package *Cremona*, it is generally easy to verify that the first projection from the graph is birational, but to calculate, for instance, its inverse we need the tools provided by the package presented here.

In Section 1, we give a concise overview of the theory of rational maps between multiprojective varieties, emphasizing the computational aspects and making clear how they can be represented in a computer. For more details on the theory see, e.g., [Harris 1992; Hartshorne 1977]. In Section 2, with the help of an example, we show how one can work with such maps using *Macaulay*2.

1. AN OVERVIEW OF RATIONAL MAPS BETWEEN MULTIPROJECTIVE VARIETIES.

1A. ***Notation and terminology.*** Throughout this paper, we keep the following notation. Let K denote an arbitrary field. Consider the polynomial ring

$$R = K[x_0^{(1)}, \dots, x_{n_1}^{(1)}; \dots; x_0^{(r)}, \dots, x_{n_r}^{(r)}]$$

in r groups of variables, equipped with the $\mathbb{Z}^r$-grading, where the degree of each variable is a standard

MSC2020: 14E05, 14Q15.
Keywords: multiprojective variety, rational map, birational map, multidegree.
MultiprojectiveVarieties version 2.3

basis vector. More precisely, we set $\deg(x_i^{(j)}) = (0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{Z}^r$, where 1 occurs at position j; we call this the *standard $\mathbb{Z}^r$-grading* on R. The polynomial ring R is the homogeneous coordinate ring of the product of r projective spaces

$$\boldsymbol{P}^{n_1,\dots,n_r} = \mathbb{P}^{n_1} \times \cdots \times \mathbb{P}^{n_r}.$$

The closed subsets (of the Zariski topology) of $\boldsymbol{P}^{n_1,\dots,n_r}$ are of the form

$$V(\mathfrak{a}) = \{p \in \boldsymbol{P}^{n_1,\dots,n_r} : F(p) = 0 \text{ for all homogeneous } F \in \mathfrak{a}\},$$

where $\mathfrak{a}$ is a homogeneous ideal in R. For any homogeneous ideal $\mathfrak{a} \subseteq R$, the *multisaturation* of $\mathfrak{a}$ is the homogeneous ideal

$$\operatorname{sat}(\mathfrak{a}) = \Bigl(\cdots \Bigl(\bigl(\mathfrak{a} : \bigl(x_0^{(1)}, \dots, x_{n_1}^{(1)}\bigr)^{\infty}\bigr) : \bigl(x_0^{(2)}, \dots, x_{n_2}^{(2)}\bigr)^{\infty}\Bigr) : \cdots \Bigr) : \bigl(x_0^{(r)}, \dots, x_{n_r}^{(r)}\bigr)^{\infty}.$$

One says that $\mathfrak{a}$ is *multisaturated* if $\mathfrak{a} = \operatorname{sat}(\mathfrak{a})$. Two homogeneous ideals $\mathfrak{a}, \mathfrak{a}' \subseteq R$ define the same subscheme of $\boldsymbol{P}^{n_1,\dots,n_r}$ if and only if $\operatorname{sat}(\mathfrak{a}) = \operatorname{sat}(\mathfrak{a}')$, and they define the same subset if and only if $\sqrt{\operatorname{sat}(\mathfrak{a})} = \sqrt{\operatorname{sat}(\mathfrak{a}')}$.

We fix a homogeneous absolutely prime ideal $I \subset R$, and we may also assume that I is multisaturated. The graded domain R/I is the homogeneous coordinate ring of an absolutely irreducible multiprojective variety

$$X = V(I) \subseteq \boldsymbol{P}^{n_1,\dots,n_r} = \mathbb{P}^{n_1} \times \cdots \times \mathbb{P}^{n_r}.$$

There is a similar correspondence between homogeneous ideals in R/I and closed subsets of X. The two most important invariants of X are: the dimension (as a topological space), which is the (Krull) dimension of the homogeneous coordinate ring R/I minus r, and the multidegree, an integral homogeneous polynomial of degree $\operatorname{codim} X = n_1 + \cdots + n_r - \dim X$ in r variables (see [Harris 1992, Lecture 19] and [Miller and Sturmfels 2005, p. 165]).

Similarly, let us take another polynomial ring in s groups of variables,

$$S = K[y_0^{(1)}, \dots, y_{m_1}^{(1)}; \dots; y_0^{(s)}, \dots, y_{m_s}^{(s)}],$$

equipped with the standard $\mathbb{Z}^s$-grading. Let $J \subset S$ be a multisaturated homogeneous absolutely prime ideal, and let

$$Y = V(J) \subseteq \boldsymbol{P}^{m_1,\dots,m_s} = \mathbb{P}^{m_1} \times \cdots \times \mathbb{P}^{m_s}$$

be the absolutely irreducible multiprojective variety defined by J.

1B. ***Rational maps to an embedded projective variety.*** In this subsection we consider the particular case when $s = 1$, and we set $\mathbb{P}^m = \boldsymbol{P}^{m_1,\dots,m_s}$. Then $Y \subseteq \mathbb{P}^m$ is an embedded projective variety.

Definition of rational map. We call *multiform* (or simply *form*) a homogeneous element of R/I. To a vector $\boldsymbol{F} = (F_0, \dots, F_m)$ of $m+1$ forms in R/I of the same multidegree, which are not all zero, we associate a continuous map

$$\phi_{\boldsymbol{F}} : X \setminus V(\boldsymbol{F}) \longrightarrow \mathbb{P}^m, \qquad \text{defined by } p \in X \setminus V(\boldsymbol{F}) \overset{\phi_{\boldsymbol{F}}}{\longmapsto} (F_0(p), \dots, F_m(p)) \in \mathbb{P}^m.$$

If $\boldsymbol{G} = (G_0, \dots, G_m)$ is another such vector of forms in R/I of the same multidegree, then we say that $\boldsymbol{F} \sim \boldsymbol{G}$ if $\phi_{\boldsymbol{F}}(p) = \phi_{\boldsymbol{G}}(p)$ for each $p \in X \setminus (V(\boldsymbol{F}) \cup V(\boldsymbol{G}))$. We have $\boldsymbol{F} \sim \boldsymbol{G}$ if and only if $\phi_{\boldsymbol{F}} = \phi_{\boldsymbol{G}}$ on some nonempty open subset U of $X \setminus (V(\boldsymbol{F}) \cup V(\boldsymbol{G}))$; in particular $\sim$ is an equivalence relation. A *rational map* $\Phi : X \dashrightarrow Y$ is defined as an equivalence class of nonzero vectors of $m+1$ forms $\boldsymbol{F} = (F_0, \dots, F_m)$ in R/I of the same multidegree, with respect to the relation $\sim$, such that for some (and hence every) representative $\boldsymbol{F}$ we have that the image of $\phi_{\boldsymbol{F}}$ is contained in Y. If $p \in X \setminus V(\boldsymbol{F})$ for some representative $\boldsymbol{F}$, we set $\Phi(p) = \phi_{\boldsymbol{F}}(p)$ and we say that Φ is *defined* at p. The *domain* of Φ, denoted by $\mathrm{Dom}(\Phi)$, is the set of points where Φ is defined, that is, it is the largest open subset of X such that the map $\phi_{\boldsymbol{F}}$ is defined for some representative $\boldsymbol{F}$. The complementary set in X of the domain of Φ is called *base locus*. A rational map $\Phi : X \dashrightarrow Y$ is called a *morphism* if it everywhere defined, that is, if its base locus is empty.

Establishing the equality of rational maps. Notice that if a vector $\boldsymbol{F} = (F_0, \dots, F_m)$ of forms in R/I represents a rational map $\Phi : X \dashrightarrow Y$, then also the vector $H \cdot \boldsymbol{F} = (HF_0, \dots, HF_m)$ represents Φ, for each nonzero form H in R/I. More generally, two vectors $\boldsymbol{F} = (F_0, \dots, F_m)$ and $\boldsymbol{G} = (G_0, \dots, G_m)$, as the ones considered above, represent the same rational map $\Phi : X \dashrightarrow Y$ if and only if

$$\mathrm{rk}\begin{pmatrix} F_0 & \cdots & F_m \\ G_0 & \cdots & G_m \end{pmatrix} < 2,$$

that is, if and only if $F_iG_j - F_jG_i$ vanishes identically on X, for every $i, j = 0, \dots, m$.

Determining the domain of a rational map. Let $\Phi : X \dashrightarrow Y$ be a rational map and let $\boldsymbol{F} = (F_0, \dots, F_m)$ be one of its representatives. A *syzygy* of $\boldsymbol{F}$ is a vector $\boldsymbol{H} = (H_0, \dots, H_m)$ of forms in R/I such that $\sum_{i=0}^m H_iF_i = 0$. Let $M_{\boldsymbol{F}}$ be a matrix whose columns form a set of generators for the module of syzygies of $\boldsymbol{F}$. The following result is proved in [Simis 2004, Proposition 1.1], although stated there only for $r = 1$.

Proposition 1.1. The representatives of the rational map Φ correspond bijectively to the homogeneous vectors in the rank one graded (R/I)-module

$$\ker(M_{\boldsymbol{F}}^t) \subset (R/I)^{m+1}.$$

Let $\boldsymbol{F}_1, \dots, \boldsymbol{F}_p$ be a set of minimal homogeneous generators of $\ker(M_{\boldsymbol{F}}^t)$. The base locus of Φ is the closed subset of X where all the entries of $\boldsymbol{F}_i$, for $i = 1, \dots, p$, vanish. The sequence of multidegrees $(\deg \boldsymbol{F}_1, \dots, \deg \boldsymbol{F}_p)$, defined up to ordering, is called the *degree sequence* of Φ.

Example 1.2. In the case when R/I is a unique factorization domain (e.g., $X = \mathbb{P}^{n_1} \times \cdots \times \mathbb{P}^{n_r}$), then a rational map $\Phi : X \dashrightarrow Y$ is uniquely represented up to proportionality, that is, the degree sequence of Φ consists of a unique element.

Direct and inverse images via rational maps. Let $\Phi : X \dashrightarrow Y$ be a rational map, and let $\mathcal{M}$ be a set of generators for the (R/I)-module of representatives of Φ. For $\boldsymbol{F} = (F_0, \dots, F_m) \in \mathcal{M}$, we consider the graded K-algebra homomorphism $\varphi_{\boldsymbol{F}} : S/J \to R/I$ defined by $\varphi_{\boldsymbol{F}}(y_i) = F_i \in R/I$.

For each homogeneous ideal $\mathfrak{a} \subseteq R/I$ (resp. $\mathfrak{b} \subseteq S/J$), we have a closed subset $V(\mathfrak{a}) \subseteq X$ (resp. $V(\mathfrak{b}) \subseteq Y$). The *direct image* of $V(\mathfrak{a})$ via Φ, denoted by $\overline{\Phi(V(\mathfrak{a}))}$, and the *inverse image* of $V(\mathfrak{b})$ via Φ, denoted by $\overline{\Phi^{-1}(V(\mathfrak{b}))}$, as sets, are given by the closure

$$\overline{\Phi(V(\mathfrak{a}))} = \overline{\{\Phi(p) : p \in \mathrm{Dom}(\Phi) \cap V(\mathfrak{a})\}}, \quad \overline{\Phi^{-1}(V(\mathfrak{b}))} = \overline{\{p \in \mathrm{Dom}(\Phi) : \Phi(p) \in V(\mathfrak{b})\}}.$$

The following result follows from elementary commutative algebra, and it tells us how to calculate direct and inverse images.

Proposition 1.3. The following formulas hold:

$$\overline{\Phi(V(\mathfrak{a}))} = \bigcup_{\boldsymbol{F} \in \mathcal{M}} V(\varphi_{\boldsymbol{F}}^{-1}(\mathfrak{a})) = V\Big(\bigcap_{\boldsymbol{F} \in \mathcal{M}} \varphi_{\boldsymbol{F}}^{-1}(\mathfrak{a})\Big);$$

$$\overline{\Phi^{-1}(V(\mathfrak{b}))} = \bigcup_{\boldsymbol{F} \in \mathcal{M}} V(\varphi_{\boldsymbol{F}}(\mathfrak{b}) : (\boldsymbol{F})^{\infty}) = V\Big(\bigcap_{\boldsymbol{F} \in \mathcal{M}} \varphi_{\boldsymbol{F}}(\mathfrak{b}) : (\boldsymbol{F})^{\infty}\Big).$$

As a consequence, we obtain that if $\boldsymbol{F}$ is any of the representatives of Φ, then

$$\overline{\Phi(X)} = V(\ker \varphi_{\boldsymbol{F}}).$$

The direct image $\overline{\Phi(X)}$ is called the (closure of the) *image* of Φ. We say that Φ is *dominant* if $\overline{\Phi(X)} = Y$.

1C. ***Rational maps to a multiprojective variety.*** We now consider the general case when $s \geq 1$, and hence $Y \subseteq \boldsymbol{P}^{m_1,\dots,m_s} = \mathbb{P}^{m_1} \times \cdots \times \mathbb{P}^{m_s}$ is a multiprojective variety. Let us denote by $\pi_i : \boldsymbol{P}^{m_1,\dots,m_s} \to \mathbb{P}^{m_i}$ the i-th projection, and let $Y_i = \pi_i(Y)$.

Definition of multirational map. We define a *multirational map* (or simply rational map)

$$\Phi : X \dashrightarrow Y$$

as an s-tuple of rational maps $\Phi_i : X \dashrightarrow \mathbb{P}^{m_i}$ such that the image of Φ_i is contained in Y_i, for $i = 1, \dots, s$. The domain of a multirational map Φ is the intersection

$$\mathrm{Dom}(\Phi) = \bigcap_{i=1}^{s} \mathrm{Dom}(\Phi_i).$$

In other words, Φ is defined at a point $p \in X$ if and only if Φ_i is defined at p for all $i = 1, \dots, s$, and in that case we set $\Phi(p) = (\Phi_1(p), \dots, \Phi_s(p)) \in \boldsymbol{P}^{m_1,\dots,m_s}$. Analogously with the case $s = 1$, we call the *base locus* the complementary set in X of the domain of Φ, and we say that Φ is a *morphism* if

$X = \mathrm{Dom}(\Phi)$. We say that Φ is *dominant* if for some (and hence every) open subset U of the domain of Φ, the set $\{\Phi(p) : p \in U\}$ is dense in Y.

Composition of multirational maps. If $\Psi = (\Psi_1, \ldots, \Psi_t) : Y \dashrightarrow Z$ is another multirational map, then Φ and Ψ can be composed if $\Phi(\mathrm{Dom}(\Phi)) \cap \mathrm{Dom}(\Psi) \neq \varnothing$; in particular, this happens when either Φ is dominant or Ψ is a morphism. If $\boldsymbol{F}^{(1)}, \ldots, \boldsymbol{F}^{(s)}$ are, respectively, representatives of $\Phi_1, \ldots, \Phi_s$, and if $\boldsymbol{G}^{(j)}$ is a representative of Ψ_j, then the vector $\boldsymbol{G}^{(j)}(\boldsymbol{F}^{(1)}, \ldots, \boldsymbol{F}^{(s)})$ is a representative of $(\Psi \circ \Phi)_j = \Psi_j \circ \Phi$.

So we can consider the category of (multi)-projective varieties and dominant (multi)-rational maps. An "*isomorphism*" in this category is called a birational map, that is, $\Phi : X \dashrightarrow Y$ is a birational map if it admits an inverse, namely a multirational map $\Phi^{-1} : Y \dashrightarrow X$ such that $\Phi^{-1} \circ \Phi = \mathrm{id}_X$ and $\Phi \circ \Phi^{-1} = \mathrm{id}_Y$ as (multi)-rational maps. A birational morphism $\Phi : X \dashrightarrow Y$ is called an *isomorphism* if Φ^{-1} is a morphism. Also (multi)-projective varieties and morphisms form a category.

Example: *the Segre embedding.* Let $N = (n_1 + 1) \cdots (n_r + 1) - 1$, and let us consider $\mathbb{P}^N$ with the homogeneous coordinate ring $K[z_{(\iota_1,\ldots,\iota_r)} : \iota_j = 0, \ldots, n_j,\ j = 1, \ldots, r]$, where the variables are the entries of the generic r-dimensional matrix of shape $(n_1 + 1) \times \cdots \times (n_r + 1)$. The *Segre embedding* of $\mathbb{P}^{n_1} \times \cdots \times \mathbb{P}^{n_r}$ into $\mathbb{P}^N$ is the rational map

$$\mathfrak{S}_{n_1,\ldots,n_r} : \mathbb{P}^{n_1} \times \cdots \times \mathbb{P}^{n_r} \dashrightarrow \mathbb{P}^N,$$

represented by the ring map

$$K[z_{(\iota_1,\ldots,\iota_r)} : \iota_j = 0, \ldots, n_j,\ j = 1, \ldots, r] \to K[x_0^{(1)}, \ldots, x_{n_1}^{(1)}, \ldots, x_0^{(r)}, \ldots, x_{n_r}^{(r)}],$$
$$z_{(\iota_1,\ldots,\iota_r)} \mapsto x_{\iota_1}^{(1)} \cdots x_{\iota_r}^{(r)}.$$

This ring map (or better the forms defining it) represents uniquely up to proportionality the rational map $\mathfrak{S}_{n_1,\ldots,n_r}$, and it is also clear that it is an injective morphism. The image of $\mathfrak{S}_{n_1,\ldots,n_r}$ is the projective variety of all r-dimensional matrices of rank 1. If we consider $\mathfrak{S}_{n_1,\ldots,n_r}$ as a rational map onto its image, then we have that $\mathfrak{S}_{n_1,\ldots,n_r}$ is an isomorphism. Indeed, for $j = 1, \ldots, r$, the module of representatives of the j-th component $\mathfrak{T}_j$ of the inverse $\mathfrak{T} = \mathfrak{S}_{n_1,\ldots,n_r}^{-1}$ is generated by the $(n_1+1) \cdots (n_{j-1}+1)(n_{j+1}+1) \cdots (n_r+1)$ vectors $(z_{(\iota_1,\ldots,\iota_r)} : \iota_j = 0, \ldots, n_j)$, as $\iota_1, \ldots, \iota_{j-1}, \iota_{j+1}, \ldots, \iota_r$ vary. Note, in particular, that $\mathfrak{T}_j$ is not uniquely represented up to proportionality, provided that $n_1, \ldots, n_{j-1}, n_{j+1}, \ldots, n_r$ are not all zero.

Multirational maps as ordinary rational maps. Let $\Phi = (\Phi_1, \ldots, \Phi_s) : X \dashrightarrow Y$ be a multirational map. Then, by composing Φ with the restriction to Y of the Segre embedding $\mathfrak{S}_{m_1,\ldots,m_s} : \mathbb{P}^{m_1} \times \cdots \times \mathbb{P}^{m_s} \to \mathbb{P}^M$, where $M = (m_1 + 1) \cdots (m_s + 1) - 1$, we get an ordinary rational map $\widetilde{\Phi} : X \dashrightarrow \mathfrak{S}_{m_1,\ldots,m_s}(Y) \subseteq \mathbb{P}^M$. The rational map $\widetilde{\Phi}$ is the unique rational map that makes the following diagram commutative:

$$\begin{tikzcd}
X \arrow[rr, dashed, "\widetilde{\Phi}"] \arrow[dr, dashed, "\Phi_1"'] \arrow[drr, dotted] \arrow[drrr, dashed, "\Phi_s"] & & \mathfrak{S}_{m_1,\ldots,m_s}(\mathbb{P}^{m_1} \times \cdots \times \mathbb{P}^{m_s}) \arrow[dl] \arrow[d, dotted] \arrow[dr] & \\
& \mathbb{P}^{m_1} & \cdots & \mathbb{P}^{m_s}
\end{tikzcd}$$

Since $\mathfrak{S}_{m_1,\dots,m_s}$ is an isomorphism onto its image, we have that Φ is a morphism (resp. birational; resp. an isomorphism) if and only if $\widetilde{\Phi}$ is a morphism (resp. birational; resp. an isomorphism). Thus, from a theoretical point of view, it would be enough to consider only "*ordinary*" rational maps. In practice, however, this complicates things considerably since the ambient space of the target of $\mathfrak{S}_{m_1,\dots,m_s}$ is much larger with respect to the source, and moreover the homogeneous coordinate ring of the image of $\mathfrak{S}_{m_1,\dots,m_s}$ is no longer a unique factorization domain (ruling out trivial cases).

Graph of a (multi)-rational map. Let $\boldsymbol{F}^{(1)},\dots,\boldsymbol{F}^{(s)}$ be, respectively, representatives of the components $\Phi_1,\dots,\Phi_s$ of a multirational map $\Phi : X \dashrightarrow Y$. Consider the $\mathbb{Z}^r \times \mathbb{Z}^s$-graded coordinate ring of

$$\mathbb{P}^{n_1} \times \cdots \times \mathbb{P}^{n_r} \times \mathbb{P}^{m_1} \times \cdots \times \mathbb{P}^{m_s}, \tag{1-1}$$

given by

$$T = K[\boldsymbol{x}_1;\dots;\boldsymbol{x}_r;\boldsymbol{y}_1;\dots;\boldsymbol{y}_s],$$

where $\boldsymbol{x}_j = (x_0^{(j)},\dots,x_{n_j}^{(j)})$ and $\boldsymbol{y}_i = (y_0^{(i)},\dots,y_{m_i}^{(i)})$, for $j = 1,\dots,r$ and $i = 1,\dots,s$. Moreover, let $t_1,\dots,t_s$ be new variables, and consider the extended polynomial ring

$$\overline{T} = K[t_1,\dots,t_s;\boldsymbol{x}_1;\dots;\boldsymbol{x}_r;\boldsymbol{y}_1;\dots;\boldsymbol{y}_s].$$

We define an ideal in $\overline{T}$ as the following sum of ideals (by abuse of notation we also denote by $\boldsymbol{F}^{(i)}$ chosen lifts of $\boldsymbol{F}^{(i)}$ to R):

$$\mathcal{I}_{(\boldsymbol{F}^{(1)},\dots,\boldsymbol{F}^{(s)})} := I + \sum_{i=1}^{s} (\boldsymbol{y}_i - t_i\,\boldsymbol{F}^{(i)}). \tag{1-2}$$

The *graph* $\Gamma(\Phi)$ of the multirational map Φ is the subvariety of (1-1) defined by the contraction ideal

$$\mathcal{I}_{(\boldsymbol{F}^{(1)},\dots,\boldsymbol{F}^{(s)})} \cap T, \tag{1-3}$$

which no longer depends on the choice of the representatives $\boldsymbol{F}^{(i)}$. Equivalently, we can consider the homogeneous ideal in T given by

$$\mathcal{J}_{(\boldsymbol{F}^{(1)},\dots,\boldsymbol{F}^{(s)})} := I + \left(2\times 2 \text{ minors of } \begin{pmatrix} y_0^{(i)} & \cdots & y_{m_i}^{(i)} \\ F_0^{(i)} & \cdots & F_{m_i}^{(i)} \end{pmatrix},\ i = 1,\dots,s\right), \tag{1-4}$$

and then we can calculate the ideal of $\Gamma(\Phi)$ by the saturation:

$$(\cdots(\mathcal{J}_{(\boldsymbol{F}^{(1)},\dots,\boldsymbol{F}^{(s)})} : (\boldsymbol{F}^{(1)})^{\infty}) : \cdots) : (\boldsymbol{F}^{(s)})^{\infty}. \tag{1-5}$$

We point out that the homogeneous coordinate ring of $\Gamma(\Phi)$ is also known as "*Rees algebra*"; see [Eisenbud 2018]. We have two projections (which are morphisms) that fit in a commutative diagram

$$\begin{tikzcd} & \Gamma(\Phi) \arrow[dl, "\pi_1"'] \arrow[dr, "\pi_2"] & \\ X \arrow[rr, dashed, "\Phi"] & & Y \end{tikzcd}$$

The first projection $\pi_1 : \Gamma(\Phi) \to X$ is also known as the *blowing up of X along B*, where $B = X \setminus \mathrm{Dom}(\Phi)$ is the base locus of Φ. It is a birational morphism, and it is an isomorphism if and only if Φ is a morphism. See, e.g., [Hartshorne 1977, Chapter II, Section 7] for more details. The second projection $\pi_2 : \Gamma(\Phi) \to Y$ is birational if and only if Φ is birational, and in that case the graph of Φ^{-1} is the same as that of Φ, by exchanging the two projections. Moreover, π_2 and Φ have always the same image in Y; in particular, we can calculate the homogeneous ideal of the image of Φ as the contraction of the ideal of $\Gamma(\Phi)$ to $S = K[\boldsymbol{y}_1; \dots ; \boldsymbol{y}_s]$.

Computing the inverse map of a birational map. Keep the notation as above, and assume moreover that $\Phi : X \dashrightarrow Y$ is birational. We want to find the components $\Psi_j : Y \dashrightarrow \mathbb{P}^{n_j}$, for $j = 1, \dots, r$, of the inverse multirational map $\Psi : Y \dashrightarrow X$ of Φ.

Fix a minimal set of multiforms generating the homogeneous ideal of the graph $\Gamma(\Phi)$ in the $\mathbb{Z}^r \times \mathbb{Z}^s$-graded coordinate ring of (1-1). For each $j = 1, \dots, r$, we select in this set those of multidegree $(0, \dots, 0, 1, 0, \dots, 0; d_1, \dots, d_s)$, where 1 occurs at position j, and $d_1, \dots, d_s$ are not subject to conditions. Let us denote these multiforms by $H_1(\boldsymbol{x}_j, \boldsymbol{y}_1, \dots, \boldsymbol{y}_s), \dots, H_q(\boldsymbol{x}_j, \boldsymbol{y}_1, \dots, \boldsymbol{y}_s)$. Thus, for $k = 1, \dots, q$, we can write

$$H_k(\boldsymbol{x}_j, \boldsymbol{y}_1, \dots, \boldsymbol{y}_s) = x_0^{(j)} G_0^{(j,k)}(\boldsymbol{y}_1, \dots, \boldsymbol{y}_s) + \cdots + x_{n_j}^{(j)} G_{n_j}^{(j,k)}(\boldsymbol{y}_1, \dots, \boldsymbol{y}_s),$$

for suitable uniquely determined forms $G_{\iota_j}^{(j,k)} \in S = K[\boldsymbol{y}_1, \dots, \boldsymbol{y}_s]$. We regard the $q \times (n_j{+}1)$-matrix

$$\mathfrak{J}^{(j)} = \left(G_{\iota_j}^{(j,k)}\right)_{k=1,\dots,q}^{\iota_j=0,\dots,n_j}$$

as a matrix over the homogeneous coordinate ring S/J of Y.

Proposition 1.4. The (S/J)-module of representatives of Ψ_j is given by $\ker(\mathfrak{J}^{(j)})$. More explicitly we have that the rank of $\mathfrak{J}^{(j)}$ is n_j, and Ψ_j is represented by the vector of signed $n_j \times n_j$-minors of any full rank $n_j \times (n_j{+}1)$-submatrix of $\mathfrak{J}^{(j)}$.

A proof of the previous result can be found in [Simis 2004, Theorem 2.4], in the particular case when $r = s = 1$ (see also [Doria et al. 2012] and [Busé et al. 2020, Theorem 4.4] for the case when $s = 1$ and the source is a product of projective varieties). The proof in the general case is not so different; its main ingredients are: the description of the equations of the graph $\Gamma(\Phi)$ given by (1-4) and (1-5), and the fact that $\Gamma(\Phi)$ can be identified with $\Gamma(\Psi)$. We leave the details to the reader.

Direct and inverse images via multirational maps. If $Z \subseteq X$ is an irreducible subvariety such that $Z \cap \mathrm{Dom}(\Phi) \neq \varnothing$, we can consider the restriction of Φ to Z, $\Phi|_Z : Z \dashrightarrow Y$, defined as usual by the composition of the inclusion $Z \hookrightarrow X$ with Φ. Note that the graph (and hence the image) of $\Phi|_Z$, can be calculated as above, just by replacing in (1-2) the ideal I with the multisaturated homogeneous ideal of Z, and by choosing the representatives $\boldsymbol{F}^{(i)}$ such that $Z \nsubseteq V(\boldsymbol{F}^{(i)})$. This gives us a way to calculate the direct image $\overline{\Phi(Z)} = \overline{\Phi|_Z(Z)}$.

If $W \subseteq Y$ is a subvariety, using Proposition 1.3, we can calculate the inverse image $\overline{\Phi^{-1}(W)} \subseteq X$ as $\overline{\Phi^{-1}(W)} = \overline{\widetilde{\Phi}^{-1}(\mathfrak{S}_{m_1,\dots,m_s}(W))}$. Alternatively (and more efficiently), let $I_W \subseteq S$ be the defining ideal of W, and let $\varphi_{(\boldsymbol{F}^{(1)},\dots,\boldsymbol{F}^{(s)})} : S \to R/I$ be the map defined by $y^{(i)}_{\iota_i} \mapsto F^{(i)}_{\iota_i} \in R/I$, for $i = 1, \dots, s$ and $\iota_i = 0, \dots, m_i$. Then the saturation of the extended ideal $(\varphi_{(\boldsymbol{F}^{(1)},\dots,\boldsymbol{F}^{(s)})}(I_W)) \subseteq R/I$ with respect to all the ideals $(\boldsymbol{F}^{(i)})$, for $i = 1, \dots, s$, gives us the ideal of the closure of $\overline{\Phi^{-1}(W)} \setminus V(\boldsymbol{F}^{(1)}, \dots, \boldsymbol{F}^{(s)})$.

Multidegree of a multirational map. Let $\Phi : X \dashrightarrow Y$ be a rational map. The *projective degrees*

$$d_0(\Phi), d_1(\Phi), \dots, d_{\dim X}(\Phi)$$

of Φ are defined as the components of the multidegree of the graph, embedded as a subvariety of

$$\mathfrak{S}_{n_1,\dots,n_r}(\mathbb{P}^{n_1} \times \cdots \times \mathbb{P}^{n_r}) \times \mathfrak{S}_{m_1,\dots,m_s}(\mathbb{P}^{m_1} \times \cdots \times \mathbb{P}^{m_s}) \subset \mathbb{P}^N \times \mathbb{P}^M,$$

where $N = \Pi^r_{j=1}(n_j + 1) - 1$ and $M = \Pi^s_{i=1}(m_i + 1) - 1$. It follows that the composition $\widetilde{\Phi} : X \dashrightarrow \mathbb{P}^M$ of Φ with the restriction to Y of the Segre embedding $\mathfrak{S}_{m_1,\dots,m_s}$ has the same projective degrees as Φ. If L denotes the intersection of Y with $\dim X - i$ general hypersurfaces of multidegree $(1, \dots, 1)$, then we have

$$d_i(\Phi) = \deg(\mathfrak{S}_{n_1,\dots,n_r}(\overline{\Phi^{-1}(L)})),$$

if $\dim(\overline{\Phi^{-1}(L)}) = i$ and $d_i(\Phi) = 0$ otherwise. See also [Harris 1992, Example 19.4, p. 240]. This gives us a probabilistic algorithm to compute the projective degrees, as already remarked in [Staglianò 2018]. A nonprobabilistic algorithm can be obtained by calculating the multidegree of the graph of Φ as a subvariety of $\boldsymbol{P}^{n_1,\dots,n_r} \times \boldsymbol{P}^{m_1,\dots,m_s}$ and then applying the following remark.

Remark 1.5. Let

$$P(a_1, \dots, a_r, b_1, \dots, b_s) \in \mathbb{Z}[a_1, \dots, a_r, b_1, \dots, b_s]$$

be the multidegree of a k-dimensional subvariety of $\boldsymbol{P}^{n_1,\dots,n_r} \times \boldsymbol{P}^{m_1,\dots,m_s}$. Then the multidegree of the same variety embedded as a subvariety of $\mathfrak{S}_{n_1,\dots,n_r}(\boldsymbol{P}^{n_1,\dots,n_r}) \times \mathfrak{S}_{m_1,\dots,m_s}(\boldsymbol{P}^{m_1,\dots,m_s}) \subset \mathbb{P}^N \times \mathbb{P}^M$, is given by

$$\sum_{i=\max(0,k-M)}^{\min(k,N)} d_i a^{N-i} b^{M-k+i} \in \mathbb{Z}[a, b],$$

where d_i denotes the coefficient of the monomial $a_1^{n_1} \cdots a_r^{n_r} b_1^{m_1} \cdots b_s^{m_s}$ in the polynomial

$$(a_1 + \cdots + a_r)^i (b_1 + \cdots + b_s)^{k-i} P(a_1, \dots, a_r, b_1, \dots, b_s).$$

In particular, when $m_1 = \cdots = m_s = 0$, we get the degree of the variety embedded in $\mathbb{P}^N$ from its multidegree as a subvariety of $\boldsymbol{P}^{n_1,\dots,n_r}$.

The last projective degree $d_{\dim X}(\Phi)$ is the degree of $\mathfrak{S}_{n_1,\dots,n_r}(X) \subseteq \mathbb{P}^N$. The first projective degree $d_0(\Phi)$ is the product of the degree of $\mathfrak{S}_{m_1,\dots,m_s}(\overline{\Phi(X)}) \subseteq \mathbb{P}^M$ with the *degree* of Φ. We have that Φ is birational onto its image if and only if its degree is 1, that is, if and only if $d_0(\Phi) = \deg(\mathfrak{S}_{m_1,\dots,m_s}(\overline{\Phi(X)}))$. Thus we can determine whether Φ is birational without computing its inverse.

2. Implementation in Macaulay2. The *Macaulay*2 package *MultiprojectiveVarieties* provides support for multiprojective varieties and multirational maps. It implements, among other things, the methods described in the previous section. As we previously said, a multirational map can be represented by a list of rational maps having as target a projective space. Partial support for this particular kind of rational maps is provided by the package Cremona, on which the first one depends.

Here we give just one simple example to illustrate how one can work with these packages. We refer to the online documentation of *Macaulay*2 for more examples and technical details.

It is classically well known that a smooth cubic hypersurface $X \subset \mathbb{P}^5$ containing two disjoint planes is birational to $\mathbb{P}^2 \times \mathbb{P}^2$, and that the inverse map $\mathbb{P}^2 \times \mathbb{P}^2 \dashrightarrow X$ is not defined along a K3 surface of degree 14. We now analyze this example using *Macaulay*2.

In the following lines of code, we first define the two projections $f : \mathbb{P}^5 \dashrightarrow \mathbb{P}^2$ and $g : \mathbb{P}^5 \dashrightarrow \mathbb{P}^2$ from two disjoint planes in $\mathbb{P}^5$, then we define the multirational map $(f, g) : \mathbb{P}^5 \dashrightarrow \mathbb{P}^2 \times \mathbb{P}^2$ and restrict it to a smooth cubic hypersurface X containing the two planes. So we get a multirational map $\Phi : X \dashrightarrow \mathbb{P}^2 \times \mathbb{P}^2$.

```
M2 --no-preload
Macaulay2, version 1.18
i1 : needsPackage "MultiprojectiveVarieties"; -- version 2.2
i2 : K = QQ, K[t,u,v,x,y,z];
i3 : f = rationalMap {t,u,v};
o3 : RationalMap (linear rational map from PP^5 to PP^2)
i4 : g = rationalMap {x,y,z};
o4 : RationalMap (linear rational map from PP^5 to PP^2)
i5 : Phi = rationalMap {f,g};
o5 : MultirationalMap (rational map from PP^5 to PP^2 x PP^2)
i6 : X = projectiveVariety ideal(t*u*x-u^2*x+u*v*x-v^2*x+t*x^2-u*x^2+t^2*y-t*u*y-t*v*y-t*x*y
                                 -v*x*y-t*y^2+t*u*z+v^2*z-t*x*z-u*y*z-v*y*z-t*z^2+u*z^2);
o6 : ProjectiveVariety, hypersurface in PP^5
i7 : Phi = Phi|X;
o7 : MultirationalMap (rational map from X to PP^2 x PP^2)
```

Next, we verify that Φ is dominant and birational, compute the inverse map Φ^{-1}, and “describe” the base locus of Φ^{-1}.

```
i8 : image Phi == target Phi
o8 = true
i9 : degree Phi
o9 = 1
i10 : inverse Phi;
o10 : MultirationalMap (birational map from PP^2 x PP^2 to X)
i11 : describe baseLocus inverse Phi;
o11 = ambient:.............. PP^2 x PP^2
      dim:.................. 2
      codim:................ 2
      degree:............... 14
      multidegree:.......... 2 T_0^2 + 5 T_0 T_1 + 2 T_1^2
      generators:........... (2,1)^1 (1,2)^1
      purity:............... true
      dim sing. l.:......... -1
```

Now we take the graph of Φ with the two projections $p_1 : \Gamma(\Phi) \to X$ and $p_2 : \Gamma(\Phi) \to \mathbb{P}^2 \times \mathbb{P}^2$. We calculate the projective degrees of p_1 and p_2, the inverse of p_2, and verify that $p_1 \circ p_2^{-1} = \Phi^{-1}$ and that p_2 is a morphism but not an isomorphism.

```
i12 : (p1,p2) = graph Phi;

i13 : (multidegree p1, multidegree p2)
o13 = ({141, 63, 25, 9, 3}, {141, 78, 40, 18, 6})

i14 : inverse p2;
o14 : MultirationalMap (birational map from PP^2 x PP^2 to 4-dimensional
                        subvariety of PP^5 x PP^2 x PP^2)

i15 : (inverse p2) * p1 == inverse Phi, isMorphism p2, isIsomorphism p2
o15 = (true, true, false)
```

We now calculate the *exceptional locus* of the first projection p_1; this is the inverse image of the base locus of p_1^{-1}.

```
i16 : baseLocus Phi == baseLocus inverse p1
o16 = true

i17 : E = p1^* (baseLocus Phi);
o17 : ProjectiveVariety, threefold in PP^5 x PP^2 x PP^2

i18 : dim E, degree E
o18 = (3, 48)
```

Finally, we take the first projection $h : \Gamma(p_2) \to \Gamma(\Phi)$ from the graph of p_2. This multirational map, regarded as a rational map between embedded projective varieties, has as source a fourfold of degree 771 in $\mathbb{P}^{485}$ and as target a fourfold of degree 141 in $\mathbb{P}^{53}$.

```
i19 : h = first graph p2;
o19 : MultirationalMap (birational map from 4-dimensional subvariety of
                        PP^5 x PP^2 x PP^2 x PP^2 x PP^2 to 4-dimensional
                        subvariety of PP^5 x PP^2 x PP^2)

i20 : degree source h, degree target h
o20 = (771, 141)
```

By construction, we know (and *Macaulay*2 knows) that the map h is birational. We can also verify this experimentally, by reducing to prime characteristic and calculating the fiber of h at a random point p on its source.

```
i21 : h = h ** (ZZ/1000003),;

i22 : p = point source h;
o22 = ProjectiveVariety, a point in PP^5 x PP^2 x PP^2 x PP^2 x PP^2

i23 : p == h^* h p
o23 = true
```

On a standard laptop, the time to execute the 23 lines of code above is less than 5 seconds.

SUPPLEMENT. The online supplement contains version 2.3 of *MultiprojectiveVarieties*.

REFERENCES.

[Busé et al. 2020] L. Busé, Y. Cid-Ruiz, and C. D'Andrea, "Degree and birationality of multi-graded rational maps", *Proc. Lond. Math. Soc.* (3) **121**:4 (2020), 743–787. MR Zbl

[Cremona] G. Staglianò, "Cremona: a Macaulay2 package for working with rational maps between projective varieties", available at https://faculty.math.illinois.edu/Macaulay2/doc/Macaulay2/share/doc/Macaulay2/Cremona/html/index.html.

[Doria et al. 2012] A. V. Doria, S. H. Hassanzadeh, and A. Simis, "A characteristic-free criterion of birationality", *Adv. Math.* **230**:1 (2012), 390–413. MR Zbl

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

[Harris 1992] J. Harris, *Algebraic geometry: a first course*, Graduate Texts in Mathematics **133**, Springer, 1992. MR Zbl

[Hartshorne 1977] R. Hartshorne, *Algebraic geometry*, Graduate Texts in Mathematics **52**, Springer, 1977. MR Zbl

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

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

[Simis 2004] A. Simis, "Cremona transformations and some related algebras", *J. Algebra* **280**:1 (2004), 162–179. MR Zbl

[Staglianò 2018] G. Staglianò, "A Macaulay2 package for computations with rational maps", *J. Softw. Algebra Geom.* **8** (2018), 61–70. MR Zbl

RECEIVED: 12 Jan 2021 REVISED: 27 Jul 2021 ACCEPTED: 31 Aug 2021

GIOVANNI STAGLIANÒ:
giovannistagliano@gmail.com
Dipartimento di Matematica e Informatica, Università degli Studi di Catania, Catania, Italy

JOURNAL OF SOFTWARE FOR ALGEBRA AND GEOMETRY vol 11, no 1, 2021