DIVISOR PACKAGE FOR MACAULAY2

KARL SCHWEDE AND ZHAONING YANG

Abstract. This note describes a Macaulay2 package for handling divisors. Group operations for divisors are included. There are methods for converting divisors to reflexive or invertible sheaves. Additionally, there are methods for checking whether divisors are Cartier, $\mathbb{Q}$-Cartier, simple normal crossings, generate base point free linear systems or satisfy numerous other conditions.

1. Introduction

Divisors are fundamental objects of study within algebraic geometry and commutative algebra. In this package for Macaulay2 [GS] we provide a wrapper object for studying Weil and Cartier divisors. We include tools for studying divisors on both affine and projective varieties.

In this package, divisors are stored (roughly) as height one prime ideals with coefficients (from $\mathbb{Z}$, $\mathbb{Q}$ or $\mathbb{R}$). We include group and scaling operations for divisors, as well as various methods for constructing modules $\mathcal{O}_X(D)$ from divisors D (and vice versa). We also include a number of checks for determining whether divisors are linearly or $\mathbb{Q}$-linearly equivalent, and for checking whether divisors are Cartier or $\mathbb{Q}$-Cartier (or finding the non-Cartier locus). Finally, we also include a number of functions for handling reflexive modules, ideals and their powers.

We realize there is a Divisor class defined in a tutorial in the Macaulay2 help system. In that implementation, divisors are given as a pair of ideals – an ideal corresponding to the positive part and an ideal corresponding to the negative part. Our approach offers the advantage that it is easier for the user to see the structure of the divisor. Additionally, certain operations are much faster via our approach.

We should warn the user that when a divisor is created, Gröbner bases are constructed for each prime ideal defining a component of the divisor. Hence the construction phase may be slower than other potential implementations (and in fact slower than our initial implementation). However, we feel that this choice offers advantages of execution speed for several functions as well as substantial improvements in code readability.

Throughout this note, and within the package, it is tacitly assumed that the ambient ring on which we are working is normal. This includes the projective case so care should be taken to make sure the graded ring you are working on is S2. While one can talk about subvarieties of codimension 1 on more general schemes, the divisor–reflexive sheaf correspondence is much more complicated in that case, and so we restrict ourselves to the normal case. For an introduction to the theory of rank-1-reflexive sheaves on "nice" schemes, see [Har94, Har07] and of course for a more basic introduction see for instance [Har77, Chapter II, Sections 5–7].

This paper is structured as follows. We first give a brief introduction to the construction, conversion and group operation functions in Section 2. We then discuss the methods for converting divisors D to modules $\mathcal{O}_X(D)$ and converting modules back to divisors in Section 3. Section 4

Date: October 31, 2014.

2010 *Mathematics Subject Classification.* 14C20.

Key words and phrases. Divisors, Reflexive Modules, Macaulay2.

The first named author was supported in part by the NSF FRG Grant DMS #1265261, NSF CAREER Grant DMS #1252860 and a Sloan Fellowship.

The second named author was supported in part by the NSF CAREER Grant DMS #1252860.

describes the properties we can check divisors for (for instance `isCartier` or `isSNC`). We conclude with a section on future plans.

Acknowledgements. We would like to thank Tommaso de Fernex, David Eisenbud, Daniel Grayson, Anurag Singh and Mike Stillman for useful conversations and comments on the development of this package.

2. Construction, conversion and group operations for divisors

This package includes a number of ways to construct a divisor (an object of class `WDiv`), illustrated below.

```
i1 : needsPackage "Divisor";
i2 : R = QQ[x,y,u,v]/ideal(x*y-u*v);
i3 : D = divisor({2, 3}, {ideal(x,u), ideal(x, v)})
o3 = 2*Div(x, u) + 3*Div(x, v) of R
o3 : WDiv
i5 : F = divisor(x)
o5 = 1*Div(v, x) + 1*Div(u, x) of R
o5 : WDiv
i6 : F = divisor( (ideal(x,u))^2*(ideal(x,v))^3 )
o6 = 2*Div(u, x) + 3*Div(v, x) of R
o6 : WDiv
```

The output is a formal sum of height one prime ideals. In the first construction method, failure to provide integer coefficients or height one prime ideals will result in an error unless the `Unsafe =>` option is set to `true`. The third construction method finds a divisor that agrees with that ideal in codimension 1.

We have different classes for $\mathbb{Q}$-divisors and $\mathbb{R}$-divisors (`QDiv` and `RDiv` respectively), these are constructed via the `rationalDivisor, realDivisor` functions or via the `divisor` function with the `CoeffType=>` option set. See the documentation.

All types of divisors are descended from HashTable. Internally, they are HashTables where each key is a list of Gröbner basis generators for a prime height one ideal and each associated value is a list, the first entry of which is the coefficient of the prime divisor and the second entry is the prime ideal used to display the divisor (it tries to match how the user entered it for ease of reading). Besides the keys corresponding prime divisors, there is a key which specifies the ambient ring.

One can convert one type of divisor to another more general class, either by multiplication by appropriate coefficients or by calling appropriate functions

```
i4 : D = divisor({1, -3}, {ideal(x,u), ideal(y,u)});
i5 : 1/1*D
o5 = -3*Div(y, u) + 1*Div(x, u) of R
o5 : QDiv
i6 : toQDiv(D)
o6 = -3*Div(y, u) + 1*Div(x, u) of R
o6 : QDiv
```

One can convert $\mathbb{Q}$ or $\mathbb{R}$-divisors back to Weil divisors as follows.

```
i3 : D = divisor( {2/3, -1/2}, {ideal(x,u), ideal(y, v)}, CoeffType=>QQ)
o3 = 2/3*Div(x, u) + -1/2*Div(y, v) of R
o3 : QDiv
i4 : isWDiv(D)
o4 = false
```

```
i5 : isWDiv(6*D)
o5 = true
i6 : toWDiv(6*D)
o6 = 4*Div(x, u) + -3*Div(y, v) of R
o6 : WDiv
```

See the documentation for more examples. Alternately, the functions `ceilingDiv` and `floorDiv` will convert any $\mathbb{Q}$ or $\mathbb{R}$-divisor to a Weil divisor by taking the ceiling or floor of the coefficients respectively. More generally one can call the method `applyToCoefficients` to apply any function to the coefficients of a divisor (since divisors are a type of `HashTable`, this is just done via the `applyValues` function).

Divisors form a group and one can add `WDiv/QDiv/RDiv` to each other to obtain new divisors. Likewise one can scale by integers, rational numbers or real numbers.

```
i3 : D = divisor({1, -2}, {ideal(x,u), ideal(x, v)}); E = divisor(u);
i5 : 3*D+E
o5 = -6*Div(x, v) + 4*Div(x, u) + 1*Div(u, y) of R
o5 : WDiv
i6 : D - (1/2)*E
o6 = -2*Div(x, v) + 1/2*Div(x, u) + -1/2*Div(u, y) of R
o6 : QDiv
```

Note that since divisors are subclasses of HashTables, these operations are easily executed internally via the `merge` and `applyValues` commands.

3. Modules, ideals, divisors and applications

It is well known that divisors are so useful because of their connections with invertible and reflexive sheaves. This package includes many functions for conversion between these types of objects. For instance:

```
i1 : R = QQ[x,y,z]/ideal(x*y-z^2); needsPackage "Divisor";
i3 : D = divisor(ideal(x, z));
i4 : divisorToModule(D)
o4 = image {-1} | y z |
           {-1} | z x |
o4 : R-module, submodule of R
i5 : moduleToDivisor(o4)
o5 = -1*Div(z, y) of R
o5 : WDiv
i6 : moduleToDivisor(o4, IsGraded=>true)
o6 = 2*Div(z, x) + -1*Div(z, y) of R
o6 : WDiv
```

The function `divisorToModule` and `moduleToDivisor` do exactly what their names suggest. They convert divisors D to the corresponding reflexive modules $\mathcal{O}_X(D)$ and back. Note that while `divisorToModule` produces a module isomorphic to $\mathcal{O}_X(D)$, which is a unique output (and the gradings are set appropriately), `moduleToDivisor` is only guaranteed to produce a divisor E such that $\mathcal{O}_X(E)$ is isomorphic to the given module M. In particular, `moduleToDivisor(divisorToModule(`D`))` will only produce a divisor linearly equivalent to D. Note that setting the `IsGraded` option to `true` in `moduleToDivisor`, will find a divisor E so that $\mathcal{O}_X(E)$ is isomorphic to M as a graded module.

The execution of `divisorToModule(D)` is done via a straightforward strategy. If `D` corresponds to primes $P_1, \ldots, P_m$ with coefficients $a_1, \ldots, a_m$, then we can compute $\bigotimes P_i^{-a_i}$ (keeping in mind negative exponents mean applying $\mathrm{Hom}_R(\_\_, R)$) and then reflexifying (see the methods `reflexifyModule`

and `reflexifyIdeal`). We do several things make this computation faster. Firstly, we break up the divisor into the positive and negative parts, and handle them separately (reflexifying as little as possible). Then, instead of computing $P_i^{|a_i|}$, which can have huge numbers of generators, we compute $P_i^{[|a_i|]}$ which means form an ideal generated by the generators of P_i raised to the $|a_i|$th powers. Since this agrees with $P_i^{|a_i|}$ in codimension 1, it will give the correct answer up to reflexification. We have noticed very substantial speed improvements using this technique.

The function `moduleToDivisor` works as follows. First it embeds the module as an ideal $I \subseteq R$ via the function `moduleToIdeal`[1]. After we have an ideal, we call `idealToDivisor`. This finds a divisor D such that $\mathcal{O}_X(D)$ is isomorphic to the given ideal I (in a non-graded sense). `idealToDivisor` does this by looking at the minimal height one primes Q_i of the ideal I and finding the maximum power n_i such that $I \subseteq Q_i^{(n_i)} = (Q_i^{n_i})^{**}$. Here $\_^{**}$ denotes reflexification/S2ification of the ideal. Finding this maximal power is done by a binary search. Again for speed we compute $(Q_i^{n_i})^{**}$ via $(Q_i^{[n_i]})^{**}$. If the `IsGraded` flag is set to `true`, `moduleToDivisor` finally corrects the degree of the divisor by adding or subtracting the divisor of an element of appropriate degree (you can see this being done in the example above). Finding the element of appropriate degree is accomplished via the function `findElementOfDegree` which uses Smith normal form in the multi-degree setting to solve the system of linear diophantine equations and find a monomial of the given multi-degree.

Instead of calling `moduleToDivisor`, one can call `moduleWithSectionToDivisor` which finds the unique effective divisor D corresponding to a global section $\gamma \in M$ of our module. The function `idealWithSectionToDivisor` behaves similarly. The strategy is the same as above, additionally one tracks the section and adds an principal divisor corresponding to the section at the end.

It is worth mentioning that the function `canonicalDivisor` simply computes the canonical module via an appropriate Ext and then calls `moduleToDivisor`. If you wish to construct a canonical divisor on a projective variety, make sure to set the `IsGraded` option to `true`.

3.1. **Pulling back divisors.** Utilizing the module and divisor correspondence `divPullBack` pulls back a divisor along a map $\operatorname{Spec} S \longrightarrow \operatorname{Spec} R$ induced by a ring map $R \longrightarrow S$. The user has a choice of two algorithms built into this function. The first works for nearly any map provided the divisor is Cartier and it also works for arbitrary divisors in the flat or finite case. The second, which is the default strategy, only gives accurate answers if the map is flat, or if the map is finite (or if the prime components of the divisor are Cartier). It can be faster than the first algorithm, especially for divisors with large coefficients. To use the first algorithm, use is `Strategy=>Sheaves`, to use the second, use the `Strategy=>Primes`.

Let us briefly describe these two strategies. The first algorithm pulls back the sheaf $\mathcal{O}(D)$, keeping track of a section appropriately. The second algorithm extends each prime ideal defining a prime divisor of D to an ideal of S, then it calls `idealToDivisor` on each such ideal and sums them keeping track of coefficients appropriately.

Consider the following example where we look at pulling back a divisor after blowing up the origin (we only consider one chart of the blowup).

```
i2 : R = QQ[x,y];
i3 : S = QQ[a,b];
i4 : f = map(S, R, {a*b, b});
i5 : D = divisor(x*y*(x+y)*(x-y))
o5 = 1*Div(y) + 1*Div(x+y) + 1*Div(-x+y) + 1*Div(x) of R
```

[1]A variant of this function appeared in the Macaulay2 documentation in the Divisor tutorial, it also appeared in the work of Moty Katzman. Our version is slightly more robust than those as it tries to embed the module into the ring in several ways, including some random attempts (see the documentation for how to control the number of random attempts).

```
o5 : WDiv
i6 : divPullBack(f, D)
o6 = 1*Div(a-1) + 1*Div(a+1) + 1*Div(a) + 4*Div(b) of S
o6 : WDiv
```

Note one of the components was lost in this pull-back, as it should have been. The coefficient of the exceptional divisor is also 4, as it should be.

3.2. **Global sections.** There are only a few built in functions for dealing with global sections of modules corresponding to divisors in the current version (in the future we hope to add more tools to do this). Of course, the user may call things like `basis(0, divisorToModule(D))` to get the global sections of a module corresponding to a divisor. In this section, we describe briefly two functions for handling global properties of divisors.

The function `mapToProjectiveSpace` gets the global sections of $\mathcal{O}(D)$ and then computes the corresponding map to projective space. This of course assumes the divisor is graded. In the example below we project $\mathbb{P}^1 \times \mathbb{P}^1$ to one of its terms by calling `mapToProjectiveSpace` along a divisor of one of the rulings.

```
i2 : R = QQ[x,y,u,v]/ideal(x*y-u*v);
i3 : D = divisor(ideal(x,u));
i4 : mapToProjectiveSpace(D)
o4 = map(R,QQ[YY , YY ],{u, y})
                1    2
o4 : RingMap R <--- QQ[YY , YY ]
                         1    2
```

Still assuming the divisor is graded, the function `baseLocus` finds a defining ideal for the locus where $\mathcal{O}(D)$ is *not* generated by global sections. This is done by computing the cokernel of $\mathcal{O}^{\oplus n} \longrightarrow \mathcal{O}(D)$ where $H^0(\mathcal{O}(D))$ has a basis of n distinct global sections and the map is the obvious one. In the following example, we compute the base locus of a point on an elliptic curve, and also two times a point on an elliptic curve (which is degree 2 and hence base point free).

```
i2 : R = QQ[x,y,z]/ideal(y^2*z-x*(x+z)*(x-z));
i3 : D = divisor( ideal(x,y) );
i4 : baseLocus(D)
o4 = ideal (y, x)
o4 : Ideal of R
i5 : baseLocus(2*D)
o5 = ideal 1
o5 : Ideal of R
```

4. Checking properties of divisors

The package `Divisor` can check divisors for several properties. First we describe the method `isCartier`.

```
i2 : R = QQ[x,y,z]/ideal(x^2-y*z);
i3 : D = divisor(ideal(x,y));
i4 : isCartier(D)
o4 = false
i5 : nonCartierLocus(D)
o5 = ideal (z, y, x)
o5 : Ideal of R
i6 : isCartier(2*D)
```

```
o6 = true
```

The algorithm behind this function is as follows. We compute $\mathcal{O}_X(-D)\cdot\mathcal{O}_X(D)$ and check whether it is equal to $\mathcal{O}_X$. In general, $\mathcal{O}_X(-D)\cdot\mathcal{O}_X(D)$ always defines an ideal defining the non-Cartier locus of D, hence the command `nonCartierLocus`. If the option `IsGraded=>true` then the relevant functions saturate the ideals with respect to the irrelevant ideal.

We also briefly describe the method `isQCartier`.

```
i6 : isQCartier(5, D)
o6 = 2
```

This checks whether any multiples $n\cdot D$ of a Weil divisor or $\mathbb{Q}$-divisor D are Cartier for any integer n less than or equal to the first argument (in this case $n \leq 5$), it may actually search a little higher than the first argument in the $\mathbb{Q}$-Cartier case due to rounding issues. If it finds that nD is Cartier, it returns the integer n. If it doesn't find any Cartier divisors, it returns `0`.

Some other useful functions are `isDivPrincipal` and `isLinearEquivalent`. Checking whether a divisor is principal just comes down to checking whether $\mathcal{O}_X(D)$ is a free module and checking whether $D \sim E$ just boils down to checking whether $D - E$ is principal. In the graded case Macaulay2 does this via the `basis` command, unfortunately we don't know how to do this in general. Therefore `isDivPrincipal` and `isLinearEquivalent` can give a false negative for non-graded divisors (it warns you if it does this). Note that the option `IsGraded` can be applied within `isLinearEquivlavent` which checks that $\mathcal{O}_X(D-E)$ is principal of degree zero.

We can also check whether a divisor D has simple normal crossings by calling `isSNC`. This first checks that the ambient space of D is regular, then it checks that each prime divisor of D defines a regular scheme, finally it checks that every intersection of of prime divisors of D also defines a regular scheme of the appropriate dimension.

5. Future plans

There are a number of ways that this package should be expanded. One of the most important things to be done is to further develop the global methods related to divisors. For instance it would be very useful to be able to check whether a divisor is very ample (and thus try to see if a divisor is ample by checking multiples of it). Some basic intersection theory between divisors and smooth curves would also be natural to include.

In another direction, we could add a separate cache structure recording whether or not, for example, the divisor is Cartier or $\mathbb{Q}$-Cartier. This might be valuable because certain functions could be made faster for cases when divisors were known to satisfy various properties (at least if we knew the answer was already computed).

References

[GS] D. R. Grayson and M. E. Stillman: *Macaulay2, a software system for research in algebraic geometry.*

[Har77] R. Hartshorne: *Algebraic geometry*, Springer-Verlag, New York, 1977, Graduate Texts in Mathematics, No. 52. MR0463157 (57 #3116)

[Har94] R. Hartshorne: *Generalized divisors on Gorenstein schemes*, Proceedings of Conference on Algebraic Geometry and Ring Theory in honor of Michael Artin, Part III (Antwerp, 1992), vol. 8, 1994, pp. 287–339. MR1291023 (95k:14008)

[Har07] R. Hartshorne: *Generalized divisors and biliaison*, Illinois J. Math. **51** (2007), no. 1, 83–98 (electronic). MR2346188

Department of Mathematics, University of Utah, 155 S 1400 E Room 233, Salt Lake City, UT, 84112
E-mail address: schwede@math.utah.edu

Department of Mathematics, Pennsylvania State University, State College, PA, 16802
E-mail address: zyy5054@psu.edu