

```

gap> g:= SymmetricGroup( 4 );
Sym( [ 1 .. 4 ] )
gap> tbl:= CharacterTable( g ); HasIrr( tbl );
false
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 . .
gap> tblmod2:= CharacterTable( tbl, 2 );
BrauerTable( Sym( [ 1 .. 4 ] ), 2 )
gap> tblmod2 = CharacterTable( tbl, 2 );
true
gap> tblmod2 = BrauerTable( tbl, 2 );
true
o5 : BrauerTable
16 : betti(t,Weights=>{0,true}
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 . .
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 ), ]
o6 : BettiTally
17 : 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
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
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}, 11) => 4
(3, {7, 4}, 11) => 4
(4, {5, 7}, 12) => 2
(4, {7, 5}, 12) => 2

```

Journal of Software for Algebra and Geometry

Divisor Package for Macaulay2

KARL SCHWEDE AND ZHAONING YANG

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, or 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 the `Divisor.m2` package for [Macaulay2], 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 formal linear combinations of height-1 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 code 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 in our approach.

We 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

Schwede was supported in part by the NSF FRG Grant DMS #1265261/1501115, NSF CAREER Grant DMS #1252860/1501102, NSF Grant #1801849 and a Sloan Fellowship. Yang was supported in part by the NSF CAREER Grant DMS #1252860/1501102.

MSC2010: primary 14C20; secondary 13B22.

Keywords: divisors, reflexive modules, Macaulay2.

Divisor.m2 version 0.3

execution speed for several functions as well as substantial improvements in code readability.

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 satisfies Serre’s second condition, see for example [Hartshorne 1977, Theorem 8.22A] or [Bruns and Herzog 1993, Proposition 2.2.21]. While one can talk about subvarieties of codimension 1 on more general schemes, the correspondence between divisors and reflexive sheaves is much more complicated, so we restrict ourselves to the normal case. For an introduction to the theory of rank-1-reflexive sheaves on “nice” schemes, see [Hartshorne 1994; 2007]; and for a more basic introduction see, for instance, [Hartshorne 1977, 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](#) describes how to determine if divisors satisfy various properties (for instance `isCartier` or `isSNC`). We conclude with a section on future plans.

2. CONSTRUCTION, CONVERSION AND GROUP OPERATIONS FOR DIVISORS.

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

```
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 = 3*Div(x, v) + 2*Div(x, u)
o3 : WeilDivisor on R
i4 : E = divisor(x)
o4 = Div(u, x) + Div(v, x)
o4 : WeilDivisor on R
i5 : F = divisor( (ideal(x,u))^2*(ideal(x,v))^3 )
o5 = 3*Div(v, x) + 2*Div(u, x)
o5 : WeilDivisor on R
```

The output is a formal sum of height-1 prime ideals. The first method requires a list of integers and a list of prime ideals. The third construction method finds a divisor defined by the given ideal in codimension 1.

We have different classes for \mathbb{Q} -divisors and \mathbb{R} -divisors (`QWeilDivisor` and `RWeilDivisor` respectively); these are constructed via the `divisor` function with the `CoeffType =>` option set or by multiplying a `WeilDivisor` by a rational or real number. See the documentation.

All types of divisors are ancestors of the `HashTable` class. Internally, they are hash tables where each key is a list of Gröbner basis generators for a prime height-1

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 that specifies the ambient ring and another key that points to a `CacheTable`.

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

```
i2 : R = QQ[x,y,u,v]/ideal(x*y-u*v);
i3 : D = divisor({1, -3}, {ideal(x,u), ideal(y,u)});
o3 : WeilDivisor on R
i4 : 1/1*D
o4 = -3*Div(y, u) + Div(x, u)
o4 : QWeilDivisor on R
i5 : toQWeilDivisor(D)
o5 = Div(x, u) + -3*Div(y, u)
o5 : QWeilDivisor on R
```

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. Alternatively, the functions `ceiling` and `floor` 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 an abelian group and one can add `WeilDivisor`/`QWeilDivisor`/`RWeilDivisor` 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);
o3 : WeilDivisor on R
o4 : WeilDivisor on R
i5 : 3*D+E
o5 = 4*Div(x, u) + -6*Div(x, v) + Div(u, y)
o5 : WeilDivisor on R
i6 : D - (1/2)*E
o6 = -2*Div(x, v) + 1/2*Div(x, u) + -1/2*Div(u, y)
o6 : QWeilDivisor on R
```

Since divisors are implemented as subclasses of hash tables, 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, we have the following:

```
i2 : R = QQ[x,y,z]/ideal(x*y-z^2); needsPackage "Divisor";
i3 : D = divisor(ideal(x, z));
o3 : WeilDivisor on R
i4 : OO(D)
o4 = image {-1} | x z |
      {-1} | z y |
o4 : R-module, submodule of R
i5 : divisor(o4)
o5 = -Div(z, x)
o5 : WeilDivisor on R
i6 : divisor(o4, IsGraded=>true)
o6 = Div(z, x)
o6 : WeilDivisor on R
```

The function `OO` produces a module M so that $\tilde{M} \cong \mathcal{O}_X(D)$ (and the gradings of M are set appropriately). The function `divisor(M)` only produces a divisor E such that $\mathcal{O}_X(E)$ is isomorphic to \tilde{M} . In particular, `divisor(OO(D))` will only produce a divisor linearly equivalent to D .

We use a straightforward strategy to compute `OO(D)`. If $D = \sum_{i=1}^m a_i P_i$ where the a_i are integers and the P_i are primes, then we can compute $\bigotimes P_i^{-a_i}$ (keeping in mind negative exponents mean applying $\text{Hom}_R(-, R)$) and compute the reflexification (see the method `reflexify`). We do several things to make this computation faster. Firstly, we break up the divisor into the positive and negative parts, and handle them separately (applying the `reflexify` method as little as possible). Then, instead of computing $P_i^{|a_i|}$, which can have many generators, we 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 substantial speed improvements using this technique.

The function `divisor(Module)` works as follows. First, it embeds the module as an ideal $I \subseteq R$ via the function `embedAsIdeal`. After we have an ideal I , we call `divisor(I)`. This finds a divisor D such that $\mathcal{O}_X(D)$ is isomorphic to the given ideal I (in a nongraded sense). The function `divisor(Ideal)` does this by looking at the minimal height-1 primes Q_i of the ideal I and finding the maximum power n_i such that $I \subseteq Q_i^{(n_i)}$ (the symbolic power). Note that because Q_i has height 1, we know $Q_i^{(n_i)} = (Q_i^{n_i})^{**}$ where $-^{**}$ denotes reflexification/S2-ification of the ideal. Finding this maximal power is done by a binary search. Again, for speed, we compute $(Q_i^{n_i})^{**}$ as $(Q_i^{[n_i]})^{**}$. If the `IsGraded` flag is set to `true`,

`divisor(Module)` 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 multidegree setting to solve the system of linear diophantine equations and find a monomial of the given multidegree.

Remark 3.1. *A variant of the function `embedAsIdeal` 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).*

Instead of calling `divisor(Module)`, one can call `divisor(Module, Section => f)`. This function finds the unique effective divisor D corresponding to a global section $f \in M$ of our module. The function `divisor(Ideal, Section => f)` behaves similarly. The strategy is the same as above, and additionally one tracks the section and adds a 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 `divisor(Module)`. If you wish to construct a canonical divisor on a projective variety, make sure to set the `IsGraded` option to `true`.

Pulling back divisors. Utilizing the module and divisor correspondence `pullBack` pulls back a divisor along a map $\text{Spec } S \rightarrow \text{Spec } R$ induced by a ring map $R \rightarrow S$. The user has a choice of two algorithms built into this function. The first works for nearly any map, provided that 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 `Strategy => Sheaves`, to use the second, use `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 `divisor(Ideal)` 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});
o4 : RingMap S <--- R
```

```

i5 : D = divisor(x*y*(x+y)*(x-y))
o5 = Div(x+y) + Div(-x+y) + Div(x) + Div(y)
o5 : WeilDivisor on R
i6 : pullback(f, D)
o6 = Div(a+1) + Div(a-1) + 4*Div(b) + Div(a)
o6 : WeilDivisor on S

```

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.

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 `basis(0, 00(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));
o3 : WeilDivisor on R
i4 : mapToProjectiveSpace(D)
o4 = map(R,QQ[YY , YY ],{v, x})
      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 $\mathcal{O}^{\oplus n} \rightarrow \mathcal{O}(D)$ where $H^0(X, \mathcal{O}(D))$ 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) );
o3 : WeilDivisor on R
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.m2` 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
i7 : isCartier(D, IsGraded => true)
o7 = true

```

The algorithm behind this function is as follows. We compute $\mathcal{O}_X(-D) \cdot \mathcal{O}_X(D)$ and check if 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`.

```

i8 : isQCartier(5, D)
o8 = 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 include `isPrincipal` 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, we can do this via Macaulay2 using the `prune` and `isFreeModule` commands. Unfortunately, we do not know an algorithm for deciding if a nongraded module is free (although we still try to prune the module and more). Therefore `isPrincipal` and `isLinearEquivalent` can give a false negative for non-graded divisors (the function warns you when this is the case). In the same way, the option `IsGraded` can be applied within `isLinearEquivalent`, 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, and finally it checks that every intersection 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. We have recently added the ability to check whether a divisor is very ample via the `isVeryAmple` function, which uses the `RationalMaps` package. However, there is much more to be done. Some basic intersection theory between divisors and smooth curves would be natural to include.

While the latest version of the package stores the outputs of some functions in the cache, this can still be improved. For example, there are likely ways to take advantage of knowing that a given divisor is Cartier or \mathbb{Q} -Cartier.

ACKNOWLEDGEMENTS. We thank Tommaso de Fernex, David Eisenbud, Daniel Grayson, Anurag Singh, Greg Smith, Mike Stillman, and the referees for useful conversations and comments on the development of this package. We also thank the referee for numerous useful comments on this paper.

SUPPLEMENT. The [online supplement](#) contains version 0.3 of `Divisor.m2`.

REFERENCES.

- [Bruns and Herzog 1993] W. Bruns and J. Herzog, *Cohen–Macaulay rings*, Cambridge Studies in Advanced Mathematics **39**, Cambridge University Press, 1993. [MR](#)
- [Hartshorne 1977] R. Hartshorne, *Algebraic geometry*, Graduate Texts in Mathematics **52**, Springer, 1977. [MR](#) [Zbl](#)
- [Hartshorne 1994] R. Hartshorne, “Generalized divisors on Gorenstein schemes”, *K-Theory* **8**:3 (1994), 287–339. [MR](#) [Zbl](#)
- [Hartshorne 2007] R. Hartshorne, “Generalized divisors and biliaison”, *Illinois J. Math.* **51**:1 (2007), 83–98. [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>.

RECEIVED: 31 Oct 2014

REVISED: 15 Jul 2018

ACCEPTED: 31 Aug 2018

KARL SCHWEDE:

schwede@math.utah.edu

Department of Mathematics, University of Utah, Salt Lake City, UT, United States

ZHAONING YANG:

zyy5054@gmail.com

<i>HeLP: a GAP package for torsion units in integral group rings</i> Andreas Bächle and Leo Margolis	1
<i>A software package to compute automorphisms of graded algebras</i> Simon Keicher	11
<i>A package for computations with classical resultants</i> Giovanni Staglianò	21
<i>The SpaceCurves package in Macaulay2</i> Mengyuan Zhang	31
<i>The ReesAlgebra package in Macaulay2</i> David Eisenbud	49
<i>A Macaulay2 package for computations with rational maps</i> Giovanni Staglianò	61
<i>ExteriorIdeals: a package for computing monomial ideals in an exterior algebra</i> Luca Amata and Marilena Crupi	71
<i>Software for computing conformal block divisors on $\overline{M}_{0,n}$</i> David Swinarski	81
<i>Divisor Package for Macaulay2</i> Karl Schwede and Zhaoning Yang	87