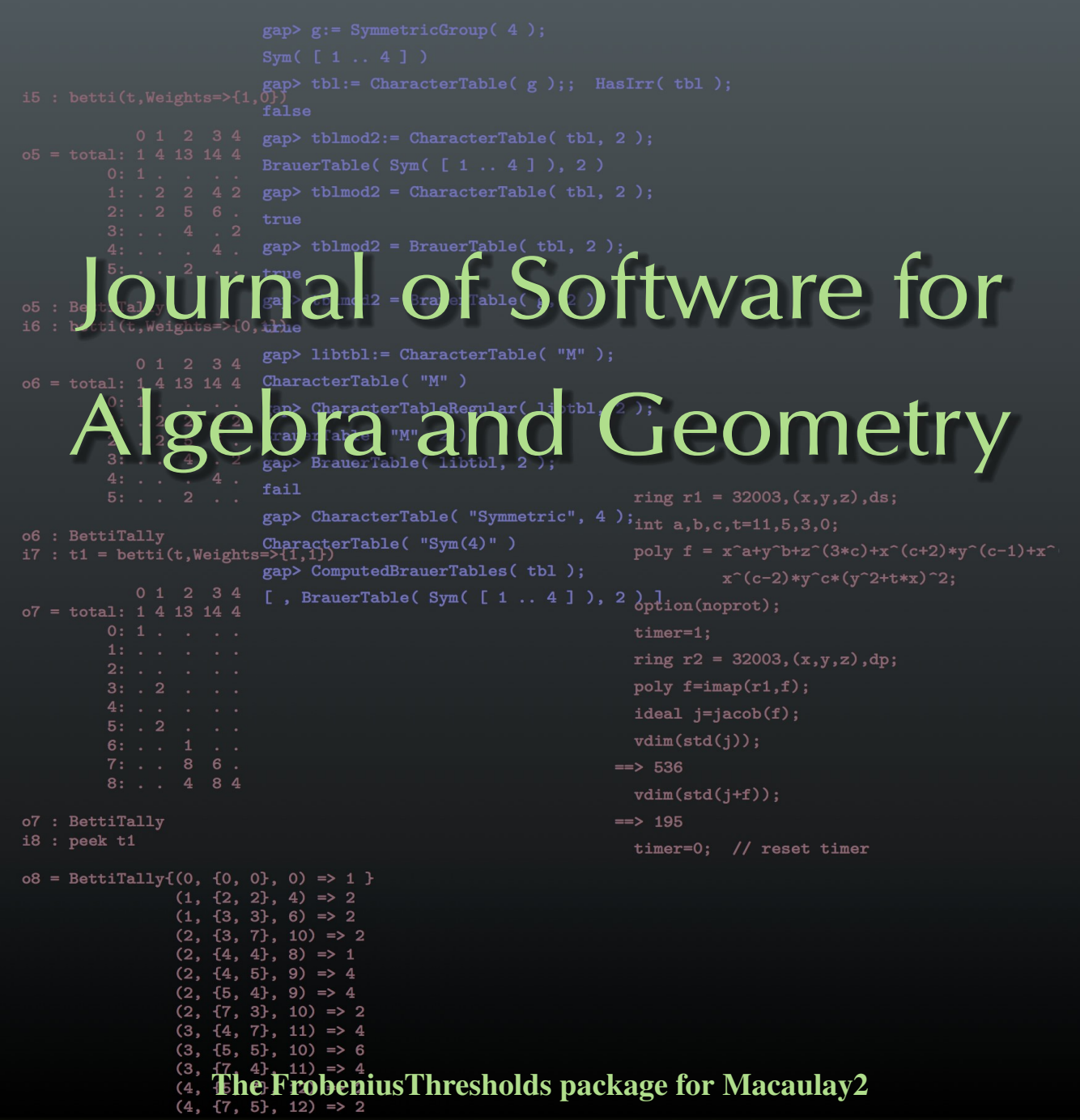
Journal of Software for Algebra and Geometry



The FrobeniusThresholds package for Macaulay2

Daniel J. Hernández, Karl Schwede, Pedro Teixeira and Emily E. Witt

vol 11 2021

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 *Macaulay2* 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 *Macaulay2* 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 *Macaulay2* [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

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

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 $\{f_1, \ldots, f_n\}$ (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 *Macaulay2*.

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 *Macaulay2* 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