# A BRIEF TUTORIAL ON THE F-SINGULARITIES PACKAGE, VERSION 0.1

PACKAGE BY: MORDECHAI KATZMAN, SARA MALEC, KARL SCHWEDE, EMILY WITT

This is a package for use in Macaulay2 for doing computations of $F$-pure thresholds, test ideals, non-sharply $F$-pure ideals, and related notions. This document gives a brief tutorial on how to use the package.

## 1. Loading the package

This package has been developed using Macaulay2 version 1.6. Use other versions at your own risk.

Place the package `PosChar.m2` in a folder that Macaulay2 can see (for example, the folder from which you are starting Macaulay2 or emacs), and run the command

```
loadPackage "PosChar"
```

If there are no errors, then it worked.

If you would like to automatically load the package on startup, please edit your `.Macaulay2/init.m2` and add a line such as

```
if fileExists "/home/myUserName/F-sing/PosChar.m2" then path=append(path,
"/home/myUserName/F-sing/")
if fileExists "/home/myUserName/F-sing/PosChar.m2" then loadPackage("PosChar")
```

You should edit your path appropriately.

## 2. $F$-pure thresholds

The default function here is `estFPT`. To use it, create a polynomial ring over a finite field, say `R = ZZ/5[x,y]` and then create an element, say $\mathtt{f} = \mathtt{x}^2 - \mathtt{y}^3$. Then running `estFPT(f, 1)` will try to compute the $F$-pure threshold (if it fails, it should give a range in which the FPT can be found, or it crash due to lack of RAM). You can replace 1 by any integer `e`, larger integers `e` may result in longer running time, but perhaps also more success (additional details as to where `e` is used can be found below).

It does it in the following way. (To watch the progress, run `estFPT` with the `Verbose=>true` switch).

(1) It first checks whether or not the polynomial `f` is diagonal (using the function `isDiagonal`). If so, it computes the FPT using the methods from D. Hernández's thesis, by calling the function `diagonalFPT`. You can turn this check off by including the parameter inside the function call, `DiagonalCheck=>false`. `diagonalFPT` is maintained by Emily Witt.

(2) Next it checks whether or not the polynomial `f` is binomial (using the function `isBinomial`). If so, it computes the FPT using the methods from D. Hernández's thesis, by calling the function `binomialFPT`. You can turn this check off by including the parameter inside the function call, `BinomialCheck=>false`. `binomialFPT` is maintained by Emily Witt.

(3) If these operations fail, using the `e` you provided, it will compute the largest $\nu_e$ such that $\mathtt{f}^{\nu_\mathtt{e}} \notin \langle \mathtt{x}_1, \dots, \mathtt{x}_\mathtt{n} \rangle$. To compute this $\nu_\mathtt{e}$ yourself, use the functions `nu` and `nuList` (maintained by Emily Witt). Regardless, we now know that $\frac{\nu_\mathtt{e}}{\mathtt{p}^\mathtt{e}-1} \leq \mathrm{FPT}(\mathtt{f}) \leq \frac{\nu_\mathtt{e}+1}{\mathtt{p}^\mathtt{e}}$.

(4) We check whether $(\mathtt{R}, \mathtt{f}^{\frac{\nu_e}{p^e-1}})$ is strongly $F$-regular by calling the function `isFRegularPoly`, if it is not, then $\frac{\nu_e}{p^e-1} = \mathrm{FPT}(\mathtt{f})$. Indeed, if the denominator of the FPT is not divisible by $p > 0$, then this method will always return the FPT as long as you provide a sufficiently divisible `e`. You can turn this check off by including the parameter inside the function call, `NuPEMinus1Check=>false`. This part of the function and `isFRegularPoly` is maintained by Karl Schwede.

(5) If the previous check failed, then the $F$-signature of $(\mathtt{R}, \mathtt{f}^{\frac{\nu_e}{p^e}})$ and $(\mathtt{R}, \mathtt{f}^{\frac{\nu_e-1}{p^e}})$ are computed using the `fSig` function. This can be very slow, but currently it cannot be turned off. You can try to run this in a multithreaded way with `MultiThread=>true`, but it provides no performance gains at the moment. Substantial performance gains can be found by using a different monomial order, or making sure your polynomial is quasi-homogeneous. Regardless, the secant line between

$$\left(\frac{\nu_e}{p^e}, (\mathtt{R}, \mathtt{f}^{\frac{\nu_e}{p^e}})\right) \text{ and } \left(\frac{\nu_e-1}{p^e}, (\mathtt{R}, \mathtt{f}^{\frac{\nu_e-1}{p^e}})\right)$$

intersects the $x$-axis at a point $\mathtt{a} \leq \mathrm{FPT}(\mathtt{f})$. This part is maintained by Karl Schwede.

(6) Finally, the program checks whether $(\mathtt{R}, \mathtt{f}^{\mathtt{a}})$ is strongly $F$-regular again using the function `isFRegularPoly`. If not, then $a = \mathrm{FPT}(\mathtt{f})$. Otherwise, the range $[\mathtt{a}, \frac{\nu_e+1}{p^e}]$ is returned. This method may never find the FPT, even for large and divisible $e$.

In the future, we hope to be able to compute FPTs for wider classes of quasi-homogeneous polynomials, for non-principal ideals, and for more general ambient rings. If you need some of this functionality *now*, please contact us and we may be able to help.

You can check whether a particular value is the FPT by running `isFPTPoly` (for instance $\mathtt{isFPTPoly}(\mathtt{x}^2 - \mathtt{y}^3, 5/6)$). You can also use `guessFPT` to try to find potential values of the FPT. You can also see `isFJumpingNumberPoly`

## 3. Test ideals and non-sharply-$F$-pure ideals

There are several functions for computing test ideals. We list them below. These are maintained by Karl Schwede, but they rely heavily on the `ethRoot` function (which computes $\bullet^{[1/p^e]}$ also denoted by $I_e(\bullet)$) written and maintained by Mordechai Katzman.

(1) `tauPoly(f,t)` Suppose `R` is a polynomial ring containing an element `f` and $\mathtt{t} \geq 0$ is a rational number. This computes the test ideal $\tau(\mathtt{R}, \mathtt{f}^{\mathtt{t}})$. This is done by writing $\mathtt{t} = \frac{\mathtt{a}}{(p^b-1)p^c}$, first computing $\tau(\mathtt{R}, \mathtt{f}^{\frac{a}{p^b-1}})$ via $\mathtt{tauAOverPEMinus1Poly}(\mathtt{f}, \mathtt{a}, \mathtt{b})$. And then using the formula

$$\tau(\mathtt{R}, \mathtt{f}^{\frac{a}{p^b-1}})^{[1/p^c]} = \tau(\mathtt{R}, \mathtt{f}^{\frac{a}{(p^b-1)p^c}}).$$

The $[1/p^c]$ implementation was originally written by Katzman.

(2) `tauAOverPEMinus1Poly(f,a,e)` Suppose `R` is a polynomial ring containing an element `f` and $\mathtt{a}, \mathtt{e} \geq 1$ are integers. This computes the test ideal $\tau(\mathtt{R}, \mathtt{f}^{\frac{a}{p^e-1}})$. This uses the same strategy as the work of Katzman for computing parameter test ideals.

(3) `tauQGor(R, e, f, t)` Suppose that `R` is a $\mathbb{Q}$-Gorenstein normal ring containing an element `f` and that the index of $K_{\mathtt{R}}$ divides $p^e - 1$. Further $t \geq 0$ is a rational number. Then this computes the test ideal $\tau(\mathtt{R}, \mathtt{f}^{\mathtt{t}})$. The strategy is similar to `tauPoly(f,t)` above.

(4) `tauQGorAmb(R, e)` Suppose that `R` is a $\mathbb{Q}$-Gorenstein normal ring and that the index of $K_{\mathtt{R}}$ divides $p^e - 1$. Then this computes the test ideal $\tau(\mathtt{R})$.

Using these functions, we also have implemented $F$-regularity checks: for polynomial rings `isFRegularPoly` and for $\mathbb{Q}$-Gorenstein rings `isFRegularQGor`.

There is also support for non-sharply-$F$-pure ideals.

(1) `sigmaAOverPEMinus1Poly(f, a, e)` Suppose $\mathtt{R}$ is a polynomial ring containing an element $\mathtt{f}$ and $\mathtt{a}, \mathtt{e} \geq \mathtt{1}$ are integers. This computes the non-sharply-$F$-pure ideal $\sigma(\mathtt{R}, \mathtt{f}^{\frac{\mathtt{a}}{\mathtt{p}^{\mathtt{e}}-1}})$.
(2) `sigmaQGorAmb(R, g)` Suppose that $\mathtt{R}$ is a $\mathbb{Q}$-Gorenstein normal ring and that the index of $K_{\mathtt{R}}$ divides $\mathtt{p}^{\mathtt{g}} - \mathtt{1}$. Then this computes the non-sharply-$F$-pure ideal $\sigma(\mathtt{R})$.
(3) `sigmaQGorAOverPEMinus1(f, a, e, g)` Suppose that $\mathtt{R}$ is a $\mathbb{Q}$-Gorenstein normal ring containing an element $\mathtt{f}$ and that the index of $K_{\mathtt{R}}$ divides $\mathtt{p}^{\mathtt{g}} - \mathtt{1}$. Further $\mathtt{a}, \mathtt{e} \geq \mathtt{1}$ are integers. Then this computes the non-sharply-$F$-pure ideal $\sigma(\mathtt{R}, \mathtt{f}^{\frac{\mathtt{a}}{\mathtt{p}^{\mathtt{e}}-1}})$.

In each of these cases, $\sigma$ is the stable image of certain $p^{-e}$-linear maps, which can be reduced to twists of `ethRoot`. Using these functions, we also have implemented a sharp $F$-purity check for polynomial rings `isSharplyFPurePoly`.

In the future we also hope to include parameter test module and ideal computations and $F$-rationality / $F$-injectivity checks. These have previously been implemented by Katzman. We also hope to include within this package tools for computing compatibly split ideals, and more generally $\phi$-fixed ideals. See the recent work of Boix-Katzman (and also Katzman-Zhang).