Changes to PosChar
Boise State University

1. Added genFrobeniusPower:

We added a function entitled genFrobeniusPower, which given an ideal I and an integer n computes the nth general Frobenius power of I i.e. $I^{[n]}$. By general Frobenius power we mean that if $I \subset R$ is an ideal such that $\operatorname{char} R = p > 0$ and $n \in \mathbb{N}$ hen

$$I^{[n]} = I^{n_0}(I_1^{n})^{[p]} \cdots (I^{n_e})^{[p^e]}$$

where $n = n_0 + n_1 p + \cdots + n_e p^e$ is the base p expansion of n. The code for is:

```
genFrobeniusPower = (I1,e1) ->(
     R1:=ring I1;
     p1:=char R1;
     E1:=basePExp(e1,p1);
     local answer;
     answer = product(#E1, q -> (frobeniusPower(I1^(E1#q),q)));
     answer
)
```

The following code provides a few test examples verifying this function:

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

I = ideal(x,y);
genFrobHand1 = I^2*frobeniusPower(I,1);
genFrobFunc1 = genFrobeniusPower(I,5);
genFrobFunc1 == genFrobHand1


J = ideal(x^3+y*x+y^2);
genFrobHand2 = J^2*frobeniusPower(J^2,1);
genFrobFunc2 = genFrobeniusPower(J,8);
genFrobFunc2 == genFrobHand2
```

This function seems to be fairly quick, with the following providing a possible example – all of which have run time of under one second, expect the 75th power which runs in roughly 10 seconds – of its effectiveness:

```
S = ZZ/11[x,y,z,w];
K = ideal(y^13+x*y^7+5*w^3, x*z, x^3+w^13-x^2*y^3*z^4);
time genFrobeniusPower(K, 200);
time genFrobeniusPower(K, 15);
time genFrobeniusPower(K, 20);
time genFrobeniusPower(K, 25);
time genFrobeniusPower(K, 35);
time genFrobeniusPower(K, 45);
time genFrobeniusPower(K, 75);
```

For comparison K^{25} takes over two second, K^{35} takes roughly 15 seconds, and K^{45} was given up on after a 10 minutes or so.

2. Modified fastExp:

The original code for `fastExp`:

```
fastExp = (f,N) ->
(
     p:=char ring f;
     E:=basePExp(N,p);
     product(#E, e -> (sum(terms f, g->g^(p^e)))^(E#e) )
```

sped up the process of computing f^n for an element $f \in R$, where $\text{char}(R) = p > 0$ by implementing the fact that $(g+h)^{p^e} = g^{p^e} + h^{p^e}$. first computing the base n expansion $n = n_0 + n_1 p + \cdots + n_e p^e$ and then doing the following computation:

$$f^n = f^{n_0}(f^p)^{n_1} \cdots (f^{p^e})^{n_e}.$$

Since raising elements to the n_i power is more difficult than raising elements to a pth power (because of the implementation of Frobenius) it seems inefficient to first raise things to a p^ith power (easy) and then raise the resulting larger polynomial to n_i (hard). This is nicely illustrated by trying to compute f^{p^e}. The original version of `fastExp` would preform the following computation:

$$f^{p^e} = (f^{p^e})^0 (f^{p^e})^0 \cdots (f^{p^e})^0 (f^{p^e})^1,$$

and so f^{p^e} is computed $e+1$ times. Of course if we switch the order we exponentiate we would have

$$f^{p^e} = (f^0)^{p^e} (f^0)^{p^e} \cdots (f^0)^{p^e} (f^1)^{p^e}.$$

and so we would only compute f^{p^e} once! With this in mind we modified `fastExp` so the order of exponentiation is reversed so that it now computes:

$$f^n = f^{n_0}(f^{n_1})^p \cdots (f^{n_e})^{p^e};$$

still making use of Frobenius to compute the pth powers. This change appears to have sped up this function as seen in these test cases were the new `fastExp` function is roughly 40% fast than the old function:

```
FastExpOld = (f,N) ->
(
     p:=char ring f;
     E:=basePExp(N,p);
     product(#E, e -> (sum(terms f, g->g^(p^e)))^(E#e) )
)

R = ZZ/7[x,y,z]

time FastExpOld(x+y, 7^10);
time fastExp(x+y,7^10);

time FastExpOld(x+y, 7^12);
time fastExp(x+y,7^12);
```

Thoughts: Currently `fastExp` computes f^{n_i} separately for each i this potentially seems inefficient if there is a good chance that many of the n_i may have repeated. For example, in the following computation

$$f = f^{100}(f^{100})^{101}(f^{100})^{101^2}(f^{100})^{101^3}$$

the function would compute $f^1 00$ four separate times. Considering that it seems that compute f^{n_i} is the difficult part of this computation this seems quite bad. So it might be worthwhile modifying this function so that given and f and n it first computes and stores $\{f, f^2, f^3, \ldots, f^t\}$ where $t = \max\{n_0, n_1, \ldots, n_e\}$ and then uses this list compute the f^{n_i}. Note: Since the n_i are between 0 and $p-1$ as e gets large it the number of repeats should increase, and so if we are interested in things as n gets large this might be worthwhile.

3. Modified frobeniusPower:

Given an ideal $I \subset R$ and an integer e, where $\text{char}(R) = p > 0$, the `frobeniusPower` function originally computed the eth Frobenius power of I by computing generators $\langle g_1, \ldots, g_t \rangle$ for I, and then raising each generator to the p^eth power. In particular, the original code for `frobeniusPower` was:

```
frobeniusPower(Ideal,ZZ) := (I1,e1) ->(
     R1:=ring I1;
     p1:=char R1;
     local answer;
     G1:=first entries gens I1;
     if (#G1==0) then answer=ideal(0_R1) else answer=ideal(apply(G1, j-> j^(p1^e1)));
     answer
);
```

This did not make use of the fact we are in characteristic p and so can utilize Frobenius to expedite the process of raising powers. With this in mind we modified `frobeniusPower` so it uses `fastExp` to compute $g_i^{p^e}$. This seems to have sped up this function.

4. Added nuHatList:

Given two ideals $I, J \subset R$, where $\text{char}(R) = p > 0$ and $I \subset \sqrt{J}$ define:

$$\hat{\nu}_I^J(p^e) = \max\{n \in N \ : \ I^{[n]} \not\subset J^{[p^e]}\}$$

we added a method `nuHatList` that computes $\hat{\nu}_I^J(p^e)$ for a specified range of e. Accepted inputs for this method are (`Ideal`,`Ideal`,`ZZ`), (`Ideal`,`ZZ`), (`RingElement`, `Ideal`, `ZZ`), and (`RingElement`, `ZZ`). When (I,J,e) is inputed this method will return a list of $\hat{\nu}_I^J(p^k)$ for $1 \le k \le e$ or an error message if $I \not\subset \sqrt{J}$. If (I,e) is inputed this method will return a list of $\hat{\nu}_I^{\mathfrak{m}}(p^k)$ for $1 \le k \le e$ where $\mathfrak{m}$ is the maximal ideal of R. Similarly when (f,J,e) is inputed this method will return a list of $\hat{\nu}_f^J(p^k)$ for $1 \le k \le e$ or an error message if $f \notin \sqrt{J}$. If (f,e) is inputed this method will return a list of $\hat{\nu}_f^{\mathfrak{m}}(p^k)$ for $1 \le k \le e$ where $\mathfrak{m}$ is the maximal ideal of R. (Note: when a ring element is given the function does not just evaluate `nuHatList(ideal(f),J,e)`, but instead utilizes `fastExp` to expedite the process.

In order to compute these we implement a binary search utilizing the `isSubset`, `genFrobeniusPower`, and `frobeniusPower`. (Note: this binary search is within a for loop which iterates through which $\hat{\nu}_I^J(p^e)$ we are computing.)The initial search range – the one used to compute $\hat{\nu}_I^J(p)$ – is 0 to pN where N is the smallest integer such that $I^N \subset J$. After this we make use of the fact that we have the following bounds

$$p\hat{\nu}_I^J(p^e) \le \hat{\nu}_I^J(p^{e+1}) \le p\hat{\nu}_I^J(p^e) + p,$$

and so conduct our binary search within these bounds. Here are following examples of this function:

```
----- Example #1 ------
----------------------


R = ZZ/3[x,y];
I = ideal(x);
J = ideal(x,y);

nuHatList(I,J,5)
nuHatList(I,5)
nuHatList(J,I,5)
```

This first example simply shows that `nuHatList` produces the expected output, and can be check in Macaulay2 with a bit of effort. Note `nuHatList(J,I,5)` returns an error, as it should, since J is not contained in the radical of I.

The following two examples were generated by Emily and Daniel, and were check against their theory. For all of these

```
----- Example #2 ------
-----------------------

R = ZZ/19[x,y];

I = ideal(x^5, y^4, x^4*y,x^3*y^2);
J = ideal(x^5+y^4+x^4*y+x^3*y^2);
K = ideal(x^5+y^4+5*x^4*y+10*x^3*y^2);
K1 = ideal(x^5+y^4+0*x^4*y+0*x^3*y^2);

nuLI = nuHatList(I,3)
nuLJ = nuHatList(J,3)
nuLK = nuHatList(K,3)
nuLK1 = nuHatList(K1,3)

----- Example #3 ------
-----------------------

R = ZZ/59[x,y];

I = ideal(x^5, y^4, x^4*y,x^3*y^2);
J = ideal(x^5+y^4+x^4*y+x^3*y^2);
K = ideal(x^5+y^4+5*x^4*y+10*x^3*y^2);
K1 = ideal(x^5+y^4+0*x^4*y+0*x^3*y^2);

nuI = nuHatList(I,3)
nuJ = nuHatList(J,3)
nuK = nuHatList(K,3)
nuK1 = nuHatList(K1,3)
```

The run-times and outputs for Example 2 and 3 are shown below. Notice they are relatively quick taking at most around 30 seconds.

```
----- Example #2 ------
-----------------------

i32 : time nuLI = nuHatList(I,3)
     -- used 32.8213 seconds

o32 = {8, 160, 3048}

o32 : List

i33 : time nuLJ = nuHatList(J,3)
     -- used 2.86942 seconds

o33 = {8, 160, 3048}

o33 : List
```

```
i34 : time nuLK = nuHatList(K,3)
     -- used 29.9829 seconds

o34 = {7, 151, 2887}

o34 : List

i35 : time nuLK1 = nuHatList(K1,3)
     -- used 0.031317 seconds

o35 = {7, 151, 2887}

o35 : List

----- Example #3 ------
-----------------------

i21 :       time nuI = nuHatList(I,2)
     -- used 147.207 seconds

o21 = {26, 1560}

o21 : List

i22 : time nuI = nuHatList(I,3)
Too many heap sections: Increase MAXHINCR or MAX_HEAP_SECTS
/bin/sh: line 1: 10356 Abort trap: 6          M2 --no-readline --print-width 176
```

Thoughts: See thoughts for nu.

5. Added nuHat:

This method is similar to nuHatList, however, it will only compute $\hat{\nu}_I^J(p^e)$ for a specific value of e. Accepted inputs for this method are (Ideal,Ideal,ZZ) and (Ideal,ZZ). When (I,J,e) is inputed this method will return $\hat{\nu}_I^J(p^e)$ or an error message if $I \not\subset \sqrt{J}$. If (I,e) is inputed this method will return a list of $\hat{\nu}_I^{\mathfrak{m}}(p^e)$ where $\mathfrak{m}$ is the maximal ideal of R.

In order to compute these we implement a binary search utilizing the isSubset, genFrobeniusPower, and frobeniusPower. The range for this binary search is 0 to p^eN where N is the smallest integer such that $I^N \subset J$. Here are examples of this function" NEDEDEDEDEDED

Thoughts: See thoughts for nu.

6. Added FTHatApproxList:

Given an ideals $I, J \subset R$ where $\operatorname{char}(R) = p > 0$ and $I \subset \sqrt{J}$ we define the F-Hat threshold to be

$$\hat{F}T(I,J) = \lim_{e\to\infty} \frac{\hat{\nu}_I^J(p^e)}{p^e}.$$

FTHatApproxList is a method to approximate the F-Hat threshold. The accepted input types for this method are (Ideal,Ideal,ZZ) and (RingElement,Ideal,ZZ). When (I,J,e) is inputed this will return a list with the following computations

$$\left\{\frac{\hat{\nu}_I^J(p)}{p}, \frac{\hat{\nu}_I^J(p^2)}{p^2}, \dots, \frac{\hat{\nu}_I^J(p^e)}{p^e}\right\}.$$

When (f, J, e) is inputed this will return a list with the following computations

$$\left\{\frac{\hat{\nu}_{(f)}^J(p)}{p}, \frac{\hat{\nu}_{(f)}^J(p^2)}{p^2}, \dots, \frac{\hat{\nu}_{(f)}^J(p^e)}{p^e}\right\}.$$

7. Re-Wrote nuList:

Thoughts: See thoughts for nu.

8. Re-Wrote nu:

Original nu was a function that given an element $f \in R$, an ideal $J \subset R$, where $\operatorname{char} R = p > 0$, and an integer $e \in \mathbb{Z}$ would compute

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

It would do this by implementing a binary search in the range of 0 and p^e as seen in the below code:

```
nu(Ideal, Ideal, ZZ) := (I1, J1, e1) -> ( --this does a fast nu computation
p1 := char ring I1;
local top;--for the binary search
local bottom1;--for the binary search
local middle;--for the binary search
local answer; --a boolean for determining if we go up, or down
N := 0;
myList := new MutableList;
curPower := 0;

bottom1 = 0;
top = p1^e1;
while (top - 1 > bottom1) do (--the bottom value is always not in m, the top is always in m
middle = floor((top + bottom1)/2);
answer = isJToAInIToPe(I1, middle, J1, e1);
if (answer == false) then bottom1 = middle else top = middle;
);
bottom1
)
```

That said this would not in general work! For example, letting $R = \mathbb{Z}/5[x]$, $f = x$, and $J = (x^6)$ we that $J^{[p]} = (x^6)^5 = (x^{30})$ and so $f^n = x^n$ is not in $J^{[p]}$ for all $n < 30$. So this function would not have worked in this situation.

With this in mind we re-wrote nu so that is is now correct, and is also more adaptable. The major changes we made to nu was making it a method, correcting the bounds on the search range so that it is correct, and replacing the `isJToAInIToPe` command with `isSubset` command, which appeared faster in practice.

As now implemented nu has four possible input types: NEDEDEDEDED

Thoughts: In our binary search loop we seem to compute `frobeniusPower(J1, e1)` during every iteration of the loop. This seems unnecessary, and we can probably speed up things if we move this computation outside the loop so it is only preformed once. Additionally, as it currently stands when we compute `I1^middle` from scratch for each iteration of our loop. This seems bad!! Note in the `genFrobeniusPower` section we saw that regular powers of ideals are hard to compute. I know we talked about doing this recursively, and were not sure how to make this efficient, but I think computing I^n might be a big bottle neck for this computation. This same thing applies to `nuList`, `nuHat`, and `nuHatList`.

9. Clean-Up:

In addition, to the above changes we also tried to clean up the code in various places. These changes include:

- The documentation at the bottom of the package were alphabetize by function name.
- We move `frobeniusPower` and `genFrobeniusPower` were moved up so that they appear before `nuList`, `nu`, `nuHatList`, and `nuHat` were these functions are used.
- Added additional comments inside sections to explain functions.
- Removed old versions of `nuList` and `nu`. (There were multiple versions of these functions with various names like `nuOld`, `nuFast`, etc. These were removed so there is only one of each of these functions. We did check that none of these were used later on.)