CONSTITUENT TEST

Magilatin generating functions and constituents

(general form, with cubic data)

Notation:

L, S: magilatin, semimagic squares (all positive values).

ml: magilatin, except in g.f.'s.

l, s: normalized squares (symmetry types).

R: reduced squares (least element is 0).

r: reduced normalized squares (reduced symmetry types).

n: semimagic r.

gf: generating function in some form.

gfsum: generating function as a sum of simple terms.

c: Cubic (fixed strict upper bound; weak upper bound for reduced).

a: Affine (fixed magic sum).

enddegree: The number of terms of the L and l sequences to check in testing the constituents.

```
> enddegree:=1000;
```

p: Period of the quasipolynomial (known from geometry). (Period of the truncated quasipolynomial, in the affine count.)
d: Dimension of the geometry = degree of the quasipolynomials.
RtoLfactor: the rational function that multiplies Rgf to Lgf and rgf to lgf.

This is for cubic: set up main constants.

```
> d:=5; p:=60;
  RtoLfactor:=x^2/(1-x)^2;
```

We start by recomputing rs from the semimagic count. From the Latte results we get the closed Ehrhart g.f. of each flat, which depends on whether we're doing cubic or affine.

This is for cubic: set up simplex data.

```
> simplexname[1]:="OABC": ehrgf[1]:= 1/((1-x)^3*(1-x^2)) : dimen[1]:=3:
  simplexname[2]:="OEE2": ehrgf[2]:= 1/((1-x)*(1-x^2)*(1-x^3)) :
  dimen[2]:=2:
  simplexname[3]:="OAE2": ehrgf[3]:= 1/((1-x)*(1-x^2)^2) : dimen[3]:=2:
  simplexname[4]:="ADE2": ehrgf[4]:= 1/((1-x^2)^3) : dimen[4]:=2:
  simplexname[5]:="DE1E2": ehrgf[5]:= 1/((1-x^2)^2*(1-x^3)) : dimen[5]:=2:
  simplexname[6]:="OCE": ehrgf[6]:= 1/((1-x)^2*(1-x^3)) : dimen[6]:=2:
  simplexname[7]:="BDE1": ehrgf[7]:= 1/((1-x)*(1-x^2)*(1-x^3)) :
  dimen[7]:=2:
  simplexname[8]:="ABD": ehrgf[8]:= 1/((1-x)*(1-x^2)^2) : dimen[8]:=2:
  simplexname[9]:="FG1": ehrgf[9]:= 1/((1-x^3)*(1-x^5)) : dimen[9]:=1:
  simplexname[10]:="EF": ehrgf[10]:= 1/((1-x^3)^2) : dimen[10]:=1:
```

```
  simplexname[11]:="OG": ehrgf[11]:= 1/((1-x)*(1-x^4)) : dimen[11]:=1:
  simplexname[12]:="FG": ehrgf[12]:= 1/((1-x^3)*(1-x^4)) : dimen[12]:=1:
  simplexname[13]:="AF": ehrgf[13]:= 1/((1-x^2)*(1-x^3)) : dimen[13]:=1:
  simplexname[14]:="DG": ehrgf[14]:= 1/((1-x^2)*(1-x^4)) : dimen[14]:=1:
  simplexname[15]:="DG2": ehrgf[15]:= 1/((1-x^2)*(1-x^5)) : dimen[15]:=1:
  simplexname[16]:="DE": ehrgf[16]:= 1/((1-x^2)*(1-x^3)) : dimen[16]:=1:
  simplexname[17]:="H": ehrgf[17] := 1/(1-x^5) : dimen[17]:=0:
```

The closed E.g.f. is converted to the open E.g.f.

```
> for n from 1 to 17 do
    mu[n]:=(-1)^(dimen[1]-dimen[n]):
  od:
  mu[14]:=2*mu[14]:
  for n from 1 to 17 do
    openehrgf[n]:=simplify(-(-1)^dimen[n]*subs(x=1/x,ehrgf[n])):
  od:
```

Set up basic g.f.'s.

```
> for n from 1 to 17 do
    rsgfterm[n]:=openehrgf[n]:
  od:
  rsgfsum:=sum(mu[nn]*rsgfterm[nn],nn=1..17):
  rsgf:=rsgfsum:
  sgfsum:=RtoLfactor*rsgf:
  sgf:=sgfsum:
```

The additional faces and intersection polytopes involved in the magilatin computation. They depend on whether we're cubic or affine. These are for cubic.

```
> mlsimplexname[1]:="OAB": mlehrgf[1]:= 1 / ((1-x)^2*(1-x^2)) :
  mldimen[1]:=2  :
  mlsimplexname[2]:="OE": mlehrgf[2]:= 1 / ((1-x)*(1-x^3)) : mldimen[2]:=1
  :
  mlsimplexname[3]:="OAC": mlehrgf[3]:= 1 / ((1-x)^2*(1-x^2)) :
  mldimen[3]:=2  :
  mlsimplexname[4]:="AD": mlehrgf[4]:= 1/(1-x^2)^2 : mldimen[4]:=1 :
  mlsimplexname[5]:="DE1": mlehrgf[5]:= 1 / ((1-x^2)*(1-x^3)) :
  mldimen[5]:=1  :
  mlsimplexname[6]:="OBC": mlehrgf[6]:= 1/(1-x)^3 : mldimen[6]:=2 :
  mlsimplexname[7]:="OE2": mlehrgf[7]:= 1 / ((1-x)*(1-x^2)) :
  mldimen[7]:=1  :
  mlsimplexname[8]:="BD": mlehrgf[8]:= 1 / ((1-x)*(1-x^2)) : mldimen[8]:=1
  :
  mlsimplexname[9]:="DE2": mlehrgf[9]:= 1/(1-x^2)^2 : mldimen[9]:=1 :
  mlsimplexname[10]:="F": mlehrgf[10]:= 1/(1-x^3) : mldimen[10]:=0 :
  mlsimplexname[11]:="OB": mlehrgf[11]:= 1/(1-x)^2 : mldimen[11]:=1 :
  # for n from 1 to 11 do print(mlsimplexname[n], mldimen[n], mlehrgf[n]);
  od;
```

Now a general computation. First, open Ehrhart g.f.'s.

```
> for n from 1 to 11 do
    openmlehrgf[n]:=simplify(-(-1)^mldimen[n]*subs(x=1/x,mlehrgf[n]));
  od:
```

(-1)^3 n_OAB(1/x) equals mlehrgf[1]+mlehrgf[2], and hence n_OAB(x) is,   by another method that gives a nicer appearance, summing mu(,)E^o(x):

```
> mlnnew[1] := openmlehrgf[1]-openmlehrgf[2]:
```

(-1)^3 n_OAC(1/x) equals mlehrgf[3]+mlehrgf[4]+mlehrgf[5]. Hence n_OAC(x) equals

```
> mlnnew[2] := openmlehrgf[3]-openmlehrgf[4]-openmlehrgf[5]:
```

(-1)^3 n_OBC(1/x) equals mlehrgf[6]+mlehrgf[7]+mlehrgf[8]+mlehrgf[9]+mlehrgf[10]. So n_OBC(x) equals

```
> mlnnew[3] :=
  openmlehrgf[6]-openmlehrgf[7]-openmlehrgf[8]-openmlehrgf[9]+openmlehrgf[
  10]:
```

Finally, OB gives mlehrgf[11], so that n_OB(x) is

```
> mlnnew[4] := openmlehrgf[11]:
```

To compute r, we need rs=n from semimagic, which equals rgf:

```
> Rgfsum:=72*rsgf+36*(mlnnew[1]+mlnnew[2]+mlnnew[3])+12*mlnnew[4]:
  Rgf:=simplify(Rgfsum):
```

Hence L, the g.f. of the number of magilatin squares, equals

```
> Lgfsum:=RtoLfactor*Rgfsum:
  Lgf:=simplify(Lgfsum):
```

Now compute the number of reduced symmetry types:

```
> rgfsum:=rsgf+mlnnew[1]+mlnnew[2]+mlnnew[3]+mlnnew[4]:
  rgf:=simplify(rgfsum):
```

The g.f. of the total number of symmetry types, l_ml ("lgf"):

```
> lgfsum:=RtoLfactor*rgfsum:
  lgf:=simplify(lgfsum):
```

Generate the series expansions of the g.f.'s.

Expressing the rational function with standard denominator gives an orders-of-magnitude speedup in the series expansion.

Standard denominator (1-x^p)^{d+1}.

```
> pdenom:=(1-x^p):
  standenom:=pdenom^(d+1);
```

G.f. as rational function with standard denominator.

```
> Lgfstandnum:=simplify(numer(Lgf)*simplify(standenom/denom(Lgf)));
  Lgf:=Lgfstandnum/standenom;
```

G.f. as rational function with standard denominator.

```
> #Rgfstandnum:=simplify(numer(Rgf)*standenom/denom(Rgf));
  #Rgf:=Rgfstandnum/standenom;
```

G.f. as rational function with standard denominator.

```
> lgfstandnum:=simplify(numer(lgf)*simplify(standenom/denom(lgf)));
  lgf:=lgfstandnum/standenom;
```

G.f. as rational function with standard denominator.

```
> #rgfstandnum:=simplify(numer(rgf)*standenom/denom(rgf));
  #rgf:=rgfstandnum/standenom;
```

Expand the series to find the first few values of the number of squares.

```
> Lseries:=series(Lgf,x=0,enddegree+1):
```

Expand the series to find the first few values of the number of reduced squares.

```
> #Rseries:=series(Rgf,x=0,enddegree+1):
```

Expand the series to find the first few values of the number of symmetry types.

```
> lseries:=series(lgf,x=0,enddegree+1):
```

Expand the series to find the first few values of the number of reduced symmetry types.

```
> #rseries:=series(rgf,x=0,enddegree+1):
```

Find the constituents

Calculate the zeroth constituent of the magilatin symmetry-type counting function. Find its constant term.

```
> Lzeroth:=expand(
  sum(coeff(Lgfstandnum,x,p*jj)*binomial(d+t/p-jj,d),jj=0..d+1) ):
```

Extract the constituents of the total magilatin counting function.

```
> Lconstituent[0]:=Lzeroth:
  for r from 1 to p do
    Lconstituent[r]:=expand(sum(
  coeff(Lgfstandnum,x,p*jj+r)*binomial(d+(t-r)/p-jj,d),  jj=0..d)):
  od:
  print("Constituents of L are done.");
```

Calculate the zeroth constituent of the magilatin symmetry-type counting function. Find its constant term.

```
> lzeroth:=expand(
  sum(coeff(lgfstandnum,x,p*jj)*binomial(d+t/p-jj,d),jj=0..d+1) ):
```

Extract the constituents of the magilatin symmetry-type counting function.

```
> lconstituent[0]:=lzeroth:
  for r from 1 to p do
    lconstituent[r]:=expand(sum(
  coeff(lgfstandnum,x,p*jj+r)*binomial(d+(t-r)/p-jj,d),  jj=0..d)):
  od:
  print("Constituents of l are done.");
```

Now the test: do the constituents give the correct numbers?

Compare the sequence to the constituent values.

```
> for n from 1 to enddegree do
    co:=coeff(Lseries,x,n):
    r:=modp(n,p);
    q:=(n-r)/p;
    term:=eval(Lconstituent[r],t=n);
    if ( co <> term ) then print(n,term,co,r) fi;
```

```
  # if ( co = term ) then print(n,r) fi;
  od:
  print("L constituent test complete to ",enddegree);
```

```
> for n from 1 to enddegree do
   co:=coeff(lseries,x,n):
   r:=modp(n,p);
   q:=(n-r)/p;
   term:=eval(lconstituent[r],t=n);
   if ( co <> term ) then print(n,term,co,r) fi;
  # if ( co = term ) then print(n,r) fi;
  od:
  print("l constituent test complete to ",enddegree);
```