Magilatin generating functions and sequences

(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).

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.

enddegree: The number of terms desired in the sequences, from degree 1 (but initial zeros will be omitted).

The number of terms desired of each sequence is "enddegree".

```
> enddegree:=100;
```

This is for cubic.

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

We start by recomputing r_s=rsgf 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.

Set up the simplex data for the faces and intersection polytopes in the semimagic part of the magilatin series.

```
> 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. The first step is to compute the Mobius function of the intersection poset.

```
> 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:
```

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

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

```
> 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 :
```

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*rsgfsum+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

```
> Lgf:=simplify(RtoLfactor*Rgf);
```

Now compute the number of reduced symmetry types:

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

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

```
> lgf:=simplify(RtoLfactor*rgf);
```

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):
  print("Series computed.");
```

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

```
> Rseries:=series(Rgf,x=0,enddegree+1):
  print("Series computed.");
```

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

```
> lseries:=series(lgf,x=0,enddegree+1):
  print("Series computed.");
```

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

```
> rseries:=series(rgf,x=0,enddegree+1):
  print("Series computed.");
```

Find the counting sequences

Generate the labelled sequence of magilatin square numbers of all four kinds. The first step is to compute the degree of the first non-zero term.

```
> Lgfdegree:=ldegree(numer(Lgf),x);
  Rgfdegree:=ldegree(numer(Rgf),x);
  lgfdegree:=ldegree(numer(lgf),x);
  rgfdegree:=ldegree(numer(rgf),x);
```

List the coefficients of each series, i.e., the terms of the counting sequences.
The comment symbol # is for controlling the output. With large "enddegree" the output is huge so it's more convenient to run each sequence's output separately and copy it from the worksheet.

```
> for n from Lgfdegree to enddegree do
   co:=coeff(Lseries,x,n):
  # printf("%d  %d \n",n,co);
  od:
  print("Sequence complete.",n,co);
```

```
> for n from Rgfdegree to enddegree do
   co:=coeff(Rseries,x,n):
  # printf("%d  %d \n",n,co);
  od:
  print("Sequence complete.",n,co);
```

```
> for n from lgfdegree to enddegree do
   co:=coeff(lseries,x,n):
  # printf("%d  %d \n",n,co);
  od:
  print("Sequence complete.",n,co);
```

```
> for n from rgfdegree to enddegree do
   co:=coeff(rseries,x,n):
  # printf("%d  %d \n",n,co);
  od:
  print("Sequence complete.",n,co);
```

```
>
```