Testing the L picture and IFS fractal generation

Maple 13 commands

In order to make fractals with iterated function systems it is nice to have a test procedure to make sure you have picked your affine maps correctly (and to help you adjust them later if necessary.) The procedure TESTMAP below, takes a list of affine functions as its input, and the result is a mapping −L picture like the ones in the fractal blueprints in the directory http://www.math.utah.edu/~korevaar/fractals.

Before trying to generate fractal examples you should load the subroutines in this file, by placing the cursor in any command fields you will need, and pressing the <enter> (<return>) key. Alternately, you can choose Edit/Execute/Worksheet from the menu toolbar, and commands will be entered in document order.

```
> restart:with(plots):Digits:=4:
  #every time you start a new fractal save and close your
  #previous fractals work.  Then return to a fresh copy of this
  document
  #and restart;  otherwise you may overwhelm
  #Maple and crash!  Also, save frequently.
> TESTMAP:=proc(funclist)   #this procedure lets you test a list
  of
                     #functions in your iterated function system
    local num,#the number of functions
          i,  #dummy index
          F,  #current function in list
          S,  #corners of unit square
          L,  #corners of letter L
          Sq, #unit square
          Llet,  #letter L
          AS, #transf of square corners
          ASq,#transf of square
          AL, #transf of L corners
          ALlet,  #transf of letter L
          Pics;  #a list of pictures
  S:=[[0,0],[0,1],[1,1] ,[1,0]];
  L:=[[.1,.9],[.1,.65],[.2,.65],[.2,.675],[.125,.675],[.125,.9]]:
  Sq:=polygonplot(S,transparency=.9): #polygonplot connects the
  dots!
  Llet:=polygonplot(L,transparency=1):
  display({Sq,Llet});
     num:=nops(funclist):
     for i from 1 to num do
       F:=funclist[i]:  #select ith map
```

```
      AS[i]:=map(F,S):
      AL[i]:=map(F,L):
      ASq[i]:=polygonplot(AS[i],transparency=.9):
      ALlet[i]:=polygonplot(AL[i],transparency=.9):
        #a plot of the transformed square and letter:
      Pics[i]:=display({ASq[i],ALlet[i]}):
    od:
 #finally, display the unit square and all its images:
 display({Sq, seq(Pics[i],i=1..num)},scaling=constrained,
 title='fractal template');
 end:
```

Here is the standard affine map, which encodes

$$AFFINE1\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = x\begin{bmatrix} a \\ b \end{bmatrix} + y\begin{bmatrix} c \\ d \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

or, in the equivalent matrix representation:

$$AFFINE1\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} a & c \\ b & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

```
> AFFINE1:=proc(X,a,b,c,d,e,f)
  RETURN(evalf([a*X[1]+c*X[2]+e,
               b*X[1]+d*X[2]+f]));
  end:
```

Example 1: Encode the affine transformations which express (an isosceles) Sierpinski's triangle as a union of shrunken images:

```
> f1:=P->AFFINE1(P,.5,0,0,.5,0,0);
        #shrink by .5 and don't translate
  f2:=P->AFFINE1(P,.5,0,0,.5,.5,0);
        #same shrink, and translate 0.5 to the right
  f3:=P->AFFINE1(P,.5,0,0,.5,.25,.5);
        #shrink, then displace by [.25,.5]
> TESTMAP([f1,f2,f3]);  #Check your transformations!!
```

Since the template is correct, we may proceed with the iteration. Start with a single point. If you have "n" contraction mappings, then each iteration multiplies the number of points in your current set by the number "n". Your number "k" of iterations should keep the total number of points n^k under 300, 000, or Maple may stall out on you.

```
> S:={[0,0]}:#initial set consisting of one point
> 3^9; #good to keep point numbers well below 300,000,
       #so as not to strain Maple's memory
```

Based on the computation above I will do nine iterations below:

```
> for i from 1 to 9 do
  S1:=map(f1,S);
  S2:=map(f2,S);
  S3:=map(f3,S);
  S:=`union`(S1,S2,S3);
  od:
> pointplot(S,symbol=point,scaling=constrained,
      title=`Sierpinski triangle`);
>
```

Now if you copy and modify this file you can create as many contraction functions as you want, and see what fractal they generate. Alternately, you can use the L−picture template to reconstruct contractions for any interesting fractal you'd like to make!use these procedures to test a template and generate a fractal. Every time you restart you will need to return to this window to re−enter the procedures you need. Unfortunately the "point" resolution size in Maple 13 is pretty large, so pictures tend to overstate the actual fractal. (Maple 8 was actually better in this regard.)

Example 2: From the book "Fractals − Endlessly repeated geometric figures", by Hans Lauwerier, page 95. This system generates a bush−like fractal, with only two generating functions!

```
> f1:=P->AFFINE1(P,.6,.6,-.6,.6,0,0);
      #A rotation (by Pi/4), and rescaling
  f2:=P->AFFINE1(P,.53,0,0,.53,1,0);
      #pure scaling by .53, with translation

> TESTMAP([f1,f2]);
```

> 2^{16};

```
> S:={[0,0]}:
> for i from 1 to 16 do
  S1:=map(f1,S):
  S2:=map(f2,S):
  S:=`union`(S1,S2):
  od:

> pointplot(S,symbol=point,scaling=constrained,
  axes=none,   title=`Figure 5.16 page 95 Lauwerier`);
>
```

Printing:

The resulting fractal pictures can be printed or exported as .jpg (or other) files to other documents, using menu options at the top of the Maple 13 Window. In the Math Department, there seem to be issues in printing fractal pictures with a lot of points, such as those above. One way around that is to click on the fractal image, so that the "Plot" menu item at the top of the Maple window becomes active. You can then export the fractal picture as a .jpg file, using the "export" button at the bottom of the "Plot" options. You can open the .jpg file from your browser, and then print from there.

Testing for contractions: In order for the set map iterations to have a limit fractal, each of the transformations must be a contraction. Precisely, there must be some number strictly less than one so that distances between image points are always no more than this number times the corresponding distances between the domain points. Here is a procedure which will test a transformation to see if it is a contraction, in case you're not sure. It relies on the spectral theorem for symmetric matrices, and is related to singular value decomposition for transformations.

```
> with(linalg):  #old linear algebra library
> STRETCHTEST:=proc(fcn) #test affine map for contraction
  local Atemp,  #matrix of affine transformation
        temp; #hold eigenvector information of the transpose
             #of Atemp times Atemp - the square root of
             #the largest eigenvalue
             #is the largest stretch factor -
             #you want this to be less than one

  Atemp:=transpose
     (matrix([fcn([1.,0])-fcn([0,0]),fcn([0,1])-fcn([0,0])]));
  temp:=eigenvectors(transpose(Atemp)&*Atemp);
    if (temp[2]=2 and temp[1]>=1) #uniform stretch, factor >=1
       then return(print("expands uniformly, by " , temp[1] , "in
  all directions - not contraction!"));
    end if;
    if (temp[2]=2 and temp[1]<1) #uniform stretch, factor <1
       then return(print("contracts uniformly, by ", temp[1] , "in
  all directions - contraction!"));
    end if;
    if (temp[1][1]>temp[2][1] and temp[1][1]>=1)
        then return(print ("not a Euclidean contraction: maximum
  stretch factor" , sqrt(temp[1][1]) , "in direction" , temp[1][3]
  [1] ));
    end if;
    if (temp[1][1]<temp[2][1] and temp[2][1]>=1)
        then returnt(print ("not a Euclidean contraction: maximum
  stretch factor" , sqrt(temp[2][1]) , "in direction" , temp[2][3]
  [1] ));
    end if;
    if (temp[1][1]<1 and temp[2][1]<1)
                then print ("contraction!");
    end if;
  end:
```

Example:

```
> f1:=P->AFFINE1(P,.7,.8,.8,-.7,.6,3);
  f2:=P->AFFINE1(P,.5,-.6,.2,.3,.8,.5);
  f3:=P->AFFINE1(P,1.5,0,1,.5,.25,.5);
  f4:=P->AFFINE1(P,.7,.2,-.2,.7,.6,3);

> TESTMAP([f1,f2,f3,f4]);
> STRETCHTEST(f1);

> STRETCHTEST(f2);
> STRETCHTEST(f3);
> STRETCHTEST(f4);
>
```