ENDRE ZIBOLEN*

CAS procedures and fractals

CAS eljárások és fraktálok
Fraktálok (véges térbe sűrített végtelen, önhasonló geometriai alakzatok) sok helyen fellépnek a természetben, de találkozhatunk velük még a káosz elméletben és ismeretes gyakorlati alkalmazásuk is. E cikk igyekszik közérthető meghatározását adni a legismertebb fraktáloknak, köztük a Mandelbrot halmaznak is. Néhány, fraktált előállító Maple eljárás (számítógépes program) közlésével és elemzésével megmutatni kívánja, hogy számítógép-algebrai rendszerek (CAS) segítségével milyen tömör és viszonylag gyors programokat lehet írni.

Introduction

One of the Maple V **C**omputer **A**lgebra **S**ystem's eminent capabilities – although not planned for it – is calculating and displaying fractals. Displaying the images may take several seconds, depending on the computer's power and amount of memory, but the results are dazzling.

If somebody have a look at the next procedures, he will see that writing fractals in the mentioned CAS environment is very easy, in most cases it is only necessary to write a few lines and after that use the commands **plot** or **plot3d** for display.

Introduction into fractals

Now we briefly explain what fractals are and how they could be produced. The properties of fractals are presented with the Mandelbrot set.

A fractal is an generally self-similar infinite geometrical pattern condensed into a finite space.

The "Mandelbrot set" was invented by BENOID MANDELBROT in 1979 and it consists of every c point of the complex plane for which the points $z_0 = c$, $z_{i+1} = z_i^2 + c$, $i = 1, 2, ..., n$ are closer to the origin as r, where the numbers n and r are fixed.

* BGF Kereskedelmi, Vendéglátóipari és Idegenforgalmi Főiskolai Kar, Informatikai Intézet, főiskolai docens.

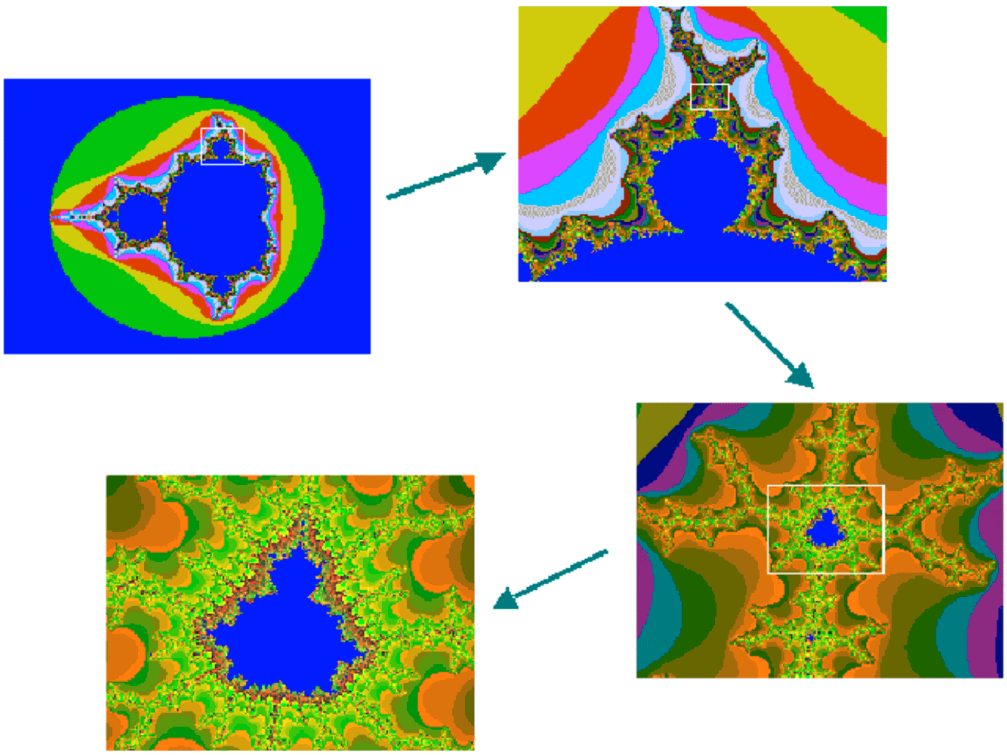

Figure 1
The magnification of the Mandelbrot set (self-similarity)

Let us determine whether the point $0.5 + 0.3*i$ does it belong to the Mandelbrot set or not for $n = 30$ and $r = 2$? To get the answer it is enough to use the next two basic properties of the complex numbers: $i*i = -1$ and the distance of point $z = x + i*y$ from the origin is $|z| = |x + i * y| = \sqrt{x^2 + y^2}$. Thus

$z_0 = c = 0.5 + 0.3*i$,

$|z_0| = |0.5 + 0.3 * i| = \sqrt{0.5^2 + 0.3^2} = 0.58 < 2$;

$z_1 = z_0^2 + c = (0.5 + 0.3*i)*(0.5 + 0.3*i)+(0.5 + 0.3*i) =$
$= 0.5^2 + 0.5*0.3*i + 0.3*i*0.5 + 0.3*i*0.3*i + 0.5 + 0.3*i =$
$= 0.25 + 0.15*i + 0.15*i + 0.3^2*i*i + 0.5 + 0.3*i =$
$= 0.25 + 0.3*i + 0.09*(-1) + 0.5 + 0.3*i = 0.25 - 0.09 + 0.5 + 0.3*i + 0.3*i = 0.66 + 0.6*i$
and

$|z_1| = |0.66 + 0.6 * i| = \sqrt{0.66^2 + 0.6^2} = 0.89 < 2$; etc.

$z_5 = 3.038901609 + 3.259282682*I$;
$|z_5| = 4.456214379 > 2$, thus $c = 0.5 + 0.3*i$ does not belong to the Mandelbrot set.

For another point $c = 0.2 + 0.5*i$

$z_0 = 0.2 + 0.5*i$, $\quad |z_0| = 0.5385164807 < 2$;

$z_1 = -0.01 + 0.7*i$, $\quad |z_1| = 0.7000714129 < 2$;

$z_2 = -0.2899 + 0.4860*i$, $\quad |z_2| = 0.5659 < 2$;

...,

$z_{30} = -0.0256317732 + 0.4123085163*i$, $\quad |z_{30}| = 0.4131044667 < 2$.

It is easy verify that $|z_i| < 2$ for every $i = 0, 1, ..., 30$, therefore the original point $c = 0.2 + 0.5*i$ is an element of the Mandelbrot set.

Drawing fractals on a PC

The algorithm of 13 rows that produces a fractal is really simple. The example of a Mandelbrot set:

```
> restart: with(plots):
> mandelbrot := proc(x, y)
>    local c, z, m;
>    c := evalf(x+y*I);
>    z := c;
>    for m from 0 to 30 while abs(z) < 2 do
>       z := z^2+c
>    od;
>    m
> end:
> plot3d(0, -2 .. 0.7, -1.2 .. 1.2, orientation=[-90,0], grid=[250, 250],
> style=patchnogrid, scaling=constrained, color=mandelbrot);
```

The plot you can se on the *Figure 2*.

The procedure Mandelbrot gives back the value m of the completed iterations. It is a clever thing to avoid long computation times by the aid of **plot3d** and its **color** option. The trouble in displaying fractals with Maple V is that it is impossible to directly put points on the screen but it is needed to store the points in a list for later plotting. This procedure avoids this trouble.

This procedure is based on that Maple can color objects with respect to a function. Besides, displaying the 0 function keeps the figure in a plane.

Increasing bigger values of **grid** in command **plot3d** leads to a better resolution.

We can look at special parts of the Mandelbrot set by modifying the 2nd and 3rd parameters of the command **plot3d**.

There are different fractals in the nature, e.g. mountains, trees, clouds and ferns, etc.

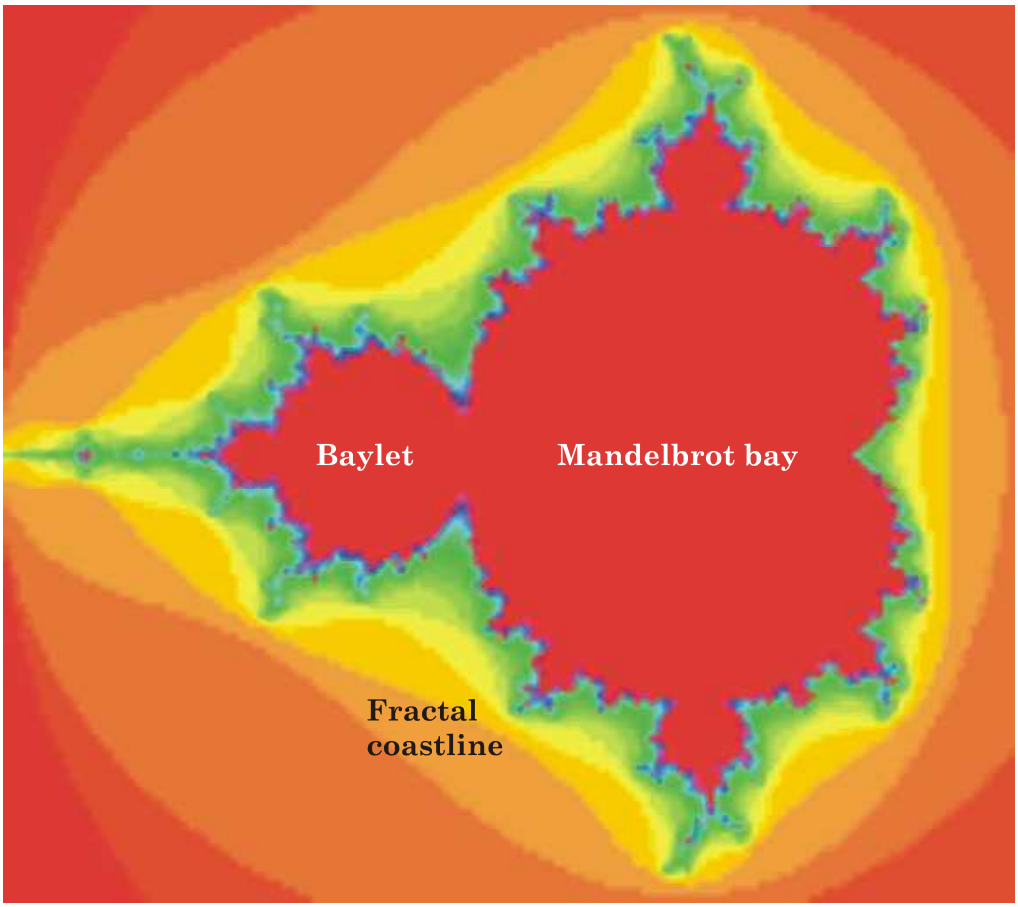

Figure 2
Example of a Mandelbrot set

Julia sets

Julia sets are similar related to the Mandelbrot set, because the formula producing the sets is the same. In the Mandelbrot set *c* is the point [x, y] being iterated, but for Julia sets *c* remains constant for every point being iterated from the plane. The procedure generating a Julia set we can receive by changing the 4th row of Mandelbrot procedure by a constant assigning:

```
>    c := evalf(x+y*I); # the old line
>    c := 1.25+0*I; # the new one
```

In the *Table 1* we give some interesting Mandelbrot's and Julia's representations.

Table 1
Parameters of some interesting Mandelbrot's and Julia's representations

Type of set	c	Iteration's formula	Escape condition	n	Domain for x_0	Domain for y_0
Mandelbrot		$z := z^2 + c$	$\|z\| < 2$	31	$-2 \le x_0 \le 0.7$	$-1.2 \le y_0 \le 1.2$
Julia	–1.25	$z := z^2 + c$	$\|z\| < 3$	31	$-2 \le x_0 \le 2$	$-1.3 \le y_0 \le 1.3$
Mandelbrot (biomorph)		$z := z^2 * c^{0.1} + c$	$\|z\| < 2$	30	$-2 \le x_0 \le 0.7$	$-1.2 \le y_0 \le 1.2$
Julia	–1.25	$z := \sin(z) + z^2 + c$	$\|z\| < 3$	31	$-2 \le x_0 \le 2$	$-1.3 \le y_0 \le 1.3$
Julia Logistic eq.	$0.85 + I*0.6$	$z := c * z * (1 - z)$	$\|z\| < 3$	31	$-1.5 \le x_0 \le 2.5$	$-1.5 \le y_0 \le 1.5$
Julia	$1+I*0.4$	$z := \sin(z)*c$	$\|z\| < 4$	101	$-4 \le x_0 \le 4$	$-3 \le y_0 \le 3$
Julia	$1+I*0.4$	$z := \sin(z)*c$	$\|z\| < 4$	101	$-1.5 \le x_0 \le 1.5$	$-1.1 \le y_0 \le 1.1$

Fast Mandelbrot

This fast procedure speeds up the first Mandelbrot algorithm by the **evalhf** command, which executes floating-point operations by using the mathematical hardware coprocessor.

To speed up the algorithm we can do the followings: let us substitute the iteration's formula by its real and imaginary parts.

Solution 1

Let $z = x + I*y$ and $c = a + I*b$, then from $z = z\text{^}2 + c$ it follows
$x + I*y = (x + I*y)^2 + a + I*b = x^2 + 2*x*y*I + I^2*y^2 + a + I*b =$
$= x^2 + 2*x*y*I - y^2 + a + I*b = x^2 - y^2 + a + (2*x*y + b)*I$, thus
$z = z^2 + c \Leftrightarrow x = x^2 - y^2 + a, \qquad y = 2xy + b.$

Because of

$$|z| = |x + I * y| = \sqrt{x^2 + y^2} \ ,$$

the iteration's condition:

$$|z| < 2 \Leftrightarrow \sqrt{x^2 + y^2} < 2 \ .$$

Solution 2

The previous decomposition can be given automatically by the following commands:

```
> z:=x+I*y: c:=a+I*b: real_part:=evalc(Re(z^2+c));
  imaginary_part:=evalc(Im(z^2+c));
  real_part := x^2-y^2+a
  imaginary_part := 2*x*y+b
```

In according to that the Mandelbrot set can be found also by the next fast mandelbrot procedure:

```
> fastmandelbrot:=proc(x, y)
>    local xn, xnold, yn, m;
>    xn:=x; yn:=y;
>    for m from 0 to 100 while sqrt(evalhf(xn^2+yn^2)) < 2 do
>       xnold:=xn;
>       xn:=evalhf(xn^2-yn^2+x); yn:=evalhf(2*xnold*yn+y);
>    od;
>    m
> end:
```

Now we verify that $z := \sin(z) + z^2 + c$ involved in our table too can be written in the following form:

$z_{n+1} = \sin(z_n)*c \equiv \sin(x_n + y_n*I)*(1 + 0.4*I) \rightarrow$

$x_n = \sin(x_n)*\cosh(y_n) - \cos(x_n)*\sinh(y_n)*0.4$

$y_n = \cos(x_n)*\sinh(y_n) + \sin(x_n)*\cosh(y_n)*0.4$

```
> z:=sin(x+I*y)*(1+I*0.4);
  z:=(1.+.4I)sin(x+Iy)
> evalc(z);
  1.sin(x)cosh(y)-.4cos(x)sinh(y)+
  I(.4sin(x)cosh(y)+1.cos(x)sinh(y))
> real_part:=evalc(Re(z));
  real_part:= 1.sin(x)cosh(y)-.4cos(x)sinh(y)
> imaginary_part:=evalc(Im(z));
  imaginary_part:= 4sin(x)cosh(y)+1.cos(x)sinh(y)
```

We note that the proof of this mathematical relationship „by hand" would be considerable longer.

Cantor set

GEORG CANTOR published the Cantor set in 1883. It is created as follows:

1. consider the interval [0, 1];
2. remove the middle one third part of the interval, it is: (1/3, 2/3);
3. repeat step 2n times to the left parts.

The Cantor Set is constituted by those points not removed by the algorithm, e.g. 0, 1/3, 2/3, 1/9, and so on.

The following procedure plots the first phases of the Cantor set's construction.

```
> with(plots):
> line := proc(a: list, b: list)
> cree_segment := (a,b,h) -> line([a,h],[b,h],color=red):
> f1:=x->x/3: f2:=x->(x+2)/3:
> f := s -> s union map(f1, s) union map(f2, s):
```

```
> sequence_of_segments := proc(l,h)
> local accu, i;
> accu := NULL;
> for i to nops(l) by 2 do
> accu := accu,cree_segment(l[i], l[i+1], h) od;
> accu
> end:
> Cantor := proc(n) local s, i;
> option remember;
> s := sequence_of_segments([0,1], 0);
> for i from 1 to n do
> s := s, sequence_of_segments(sort([op((f@@i)({0,1}))]), i/n)
> od;
> display({s}union{seq(textplot([[0,i/n, `0`], [1, i/n, `1`]],
> align=ABOVE), i=0 .. n)}, color=blue,axes=NONE,thickness=0)
> end:
> Cantor(5);
```

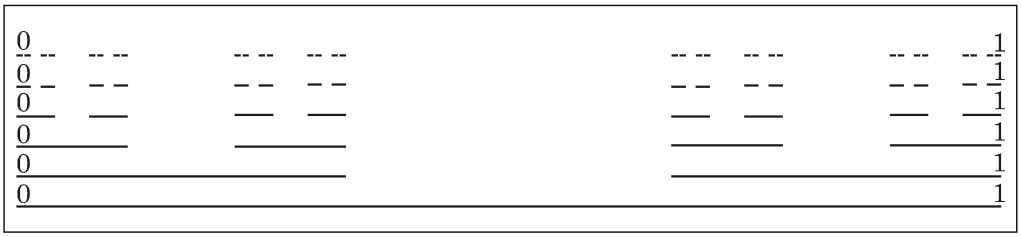

Figure 3
The „Cantor set" plotted by the command Cantor(5)

The remember option indicates to Maple that it should store results of the calls to the procedure Cantor in a remember table to avoid the unnecessary calculations and to make the procedure faster.

Analysis of the execution

In consequence of Cantor(5) and Cantor(n) n=5.

Explanation of `s := sequence_of_segments([0,1], 0);`

Owning to `sequence_of_segments(l,h) l=[0,1], h=0` and `nops(l)= nops([0,1])=2` →
`accu = cree_segment(l[1], l[2], h) = cree_segment(0, 1, 0) = cree_segment(a,b,h)` → `a=0, b=1, h=0`
`cree_segment := (a,b,h)` → `line([a,h],[b,h],color=red)` →
`cree_segment(0, 1, 0)= cree_segment(a, b, h)=line([0,0],[1,0],color=red).`
From `line(a:: list, b:: list)` → `a=[0,0], b=[1,0]`
Thus
`line([0,0],[1,0])= line(a,b)=plot([a, b],style=line)= plot([[0,0],[1,0]],style=line)`
Therefore `s := sequence_of_segments([0,1],0)` prepares the s plot of line segment with endpoints [0,0],[1,0].

We examine next loop for i=1:

```
> for i from 1 to n do
> s := s, sequence_of_segments(sort([op((f@@i)({0,1}))]), i/n)
> od;
```

i=1 and n=6, in consequence the result:

```
s := s, sequence_of_segments(sort([op((f@@1)({0,1}))]), 1/6),
```

which can be written into the form s=s0,s1, where

```
s0= sequence_of_segments([0,1], 0) and
s1= sequence_of_segments(sort([op((f@@1)({0,1}))]), 1/6)
```

Explanation of `s1= sequence_of_segments(sort([op((f@@1)({0,1}))]),1/6)`

```
f@@1)({0,1})=f({0,1})=f(s)= s union map(f1, s) union map(f2, s)=
=s union f1(s) union f2(s)={0,1} union f1({0,1}) union f2({0,1})=
={0,1} union {f1(0),f1(1)} union {f2(0),f2(1)}=
={0,1} union {0/3,1/3)} union {(0+2)/3,(1+2)/3}=
={0,1} union {0/3,1/3)} union {2/3,1}={0,1,1/3,2/3}, thus
op((f@@1)({0,1}))= op({0,1,1/3,2/3})=0,1,1/3,2/3
```

In this way

```
op((f@@1)({0,1}))= op({0,1,1/3,2/3})=0,1,1/3,2/3
[op((f@@1)({0,1}))]=[0,1,1/3,2/3], and
sort([{0,1,1/3,2/3}])=[0,1/3,2/3,1] →
s1= sequence_of_segments(sort([op((f@@2)({0,1}))]), 1/6)=
  = sequence_of_segments([0/9,1/3,2/3,1], 1/6)
```

On account of

```
sequence_of_segments(l,h) l=[0,1/3,2/3,1], h=1/6
nops(l)=nops([0,1/3,2/3,1])=4 →
... accu=
= cree_segment(l[1],l[2],h), cree_segment(l[3],l[4],h)=
= cree_segment(0,1/3,1/6), cree_segment(2/3,1,1/6)=...
= line([0,1/6],[1/3,1/6]), line([0,1/6],[1/3,1/6]) =
= plotting linesegments of the endpoints [0,1/6], [1/3,1/6]
  plotting linesegments of the endpoints [2/3,1/6], [1/3,1/6].
```

So `s1 = sequence_of_segments(sort([op((f@@1)({0,1}))]), 1/6)` removes the middle one third part of `[0,1]`, plot's name is `s1`.

We examine next loop `for i=2`:

```
> for i from 1 to n do
> s := s, sequence_of_segments(sort([op((f@@i)({0,1}))]), i/n)
> od;
```

i=2 and n=6, so the result:

```
s := s, sequence_of_segments(sort([op((f@@2)({0,1}))]),2/6),
```

which can be written into the form `s=s0,s1,s2,` where

```
s2= sequence_of_segments(sort([op((f@@2)({0,1}))]), 2/6)
```

Interpretation of `s2= sequence_of_segments(sort([op((f@@2)({0,1}))]),1/6)`

```
(f@@2)({0,1})=f(f({0,1}))=…=f({0,1,1/3,2/3})=f(s)=
=s union map(f1, s) union map(f2, s)=
=s union f1(s) union f2(s)={ 0,1,1/3,2/3} union f1({0,1,1/3,2/3})
 union f2({0,1,1/3,2/3})=
```

```
={0,1,1/3,2/3} union {f1(0),f1(1), f1(1/3),f1(2/3)} union
{f2(0),f2(1), f2(1/3),f2(2/3)}=
={0,1,1/3,2/3} union {0/3,1/3,1/9,2/9} union
{0/3+2/3,1/3+2/3,1/9+2/3,2/9+2/3}=
={0/9,9/9,3/9,6/9} union {0/9,3/9,1/9,2/9} union
{0/9+6/9,3/9+6/9,1/9+6/9,2/9+6/9}=
={0/9,9/9,3/9,6/9} union {0/9,3/9,1/9,2/9} union {6/9,9/9,7/9,8/9}=
={0/9,9/9,3/9,6/9, 1/9,2/9,7/9,8/9}, →
op((f@@2)({0,1}))= op({0/9,9/9,3/9,6/9,
1/9,2/9,7/9,8/9})=0/9,9/9,3/9,6/9,1/9,2/9,7/9,8/9
```

and

```
[op((f@@1)({0,1}))]=[0/9,9/9,3/9,6/9, 1/9,2/9,7/9,8/9],
```

in other word

```
sort([0/9,9/9,3/9,6/9,
1/9,2/9,7/9,8/9])=[0/9,1/9,2/9,3/9,6/9,7/9,8/9,9/9]
```

follows, in this manner

```
s2=sequence_of_segments(sort([op((f@@2)({0,1}))]), 2/6)=
  =sequence_of_segments([0/9,1/9,2/9,3/9,6/9,7/9,8/9,9/9],2/6)
```

Because of `sequence_of_segments(l,h)` →

```
l=[0/9,1/9,2/9,3/9,6/9,7/9,8/9,9/9], h=2/6
nops(l)=nops([0/9,1/9,2/9,3/9,6/9,7/9,8/9,9/9])=8 ... accu=
= cree_segment(l[1], l[2], h), cree_segment(l[3], l[4], h),
  cree_segment(l[5], l[6], h), cree_segment(l[7], l[8], h)=
= cree_segment(0/9,1/9, 2/6), cree_segment(2/9,3/9, 2/6),
  cree_segment(6/9,7/9, 2/6), cree_segment(8/9,9/9, 2/6)=
= line([0/9,2/6],[1/9,2/6]), line(2/9,2/6],[3/9,2/6]),
 line( [6/9,2/6],[7/9,2/6]), line(8/9,2/6],[9/9,2/6])=
= plotting linesegment of endpoints [0/9,2/6], [1/9,2/6] and
  plotting linesegment of endpoints [2/9,2/6], [3/9,2/6] and
  plotting linesegment of endpoints [6/9,2/6], [7/9,2/6] and
  plotting linesegment of endpoints [8/9,2/6], [9/9,2/6].
```

Thus `s2= sequence_of_segments(sort([op((f@@2)({0,1}))]),1/6)` removes the middle one third part of intervals of s1, the plot of the remainder parts is called `s2`.

From the loop

```
> for i from 1 to n do
> s := s, sequence_of_segments(sort([op((f@@i)({0,1}))]), i/n)
> od;
```

it follows that `s = s0, s1, s2, …, s6.`

The procedures of present topic are based mainly on internet resources of JOHN OPREA and ALAIN SCAUBER.

Irodalom

[1] MOLNÁRKA, GERGÓ, WETTL, HORVÁTH, KALLÓS: A Maple V és alkalmazásai, Springer, 1996.

[2] M. B. MONAGAN, K. O. GEDDES, K. M. HEAL, G. LABAHN, S. M. VORKOETTER: Maple V Programming Guide, Springer, 1998.

[3] ANDRÉ HECK: Introduction to Maple, Springer-Verlag New York, 1996.
[4] http://www.mapleapps.com/categories/graphics/gallery/html/mandelbrot.html
http://www.math.utsa.edu/mirrors/maple/mfrmand.htm
http://seawolf.uofs.edu/~monks/software/chaos/chaos.html
http://www.ostium.ch/old/english/fractal.html
http://spanky.triumf.ca/www/fractint/fractint.html