

Projekt 5: Eine ganz besondere Spirale

Marc Kieseheuer & Oliver Kamp
5. Februar 2003

Beginne mit einem Quadrat der Kantenlänge 1. Lege ein weiteres daneben. Lege nun an die lange Kante ein Quadrat mit ebendieser Kantenlänge. Und wieder...

– Initialisierung

```
> restart;  
#'mod' := modp:  
with(plottools):  
with(plots):  
Warning, the names arrow and changecoords have been redefined
```

– Einleitung: Grafische Ausgabe der Quadrate

```
> fib1 := proc(n)  
  option remember:  
  if n <= 2 then  
    1;  
  else  
    procname(n-1)+procname(n-2);  
  end if;  
end proc:  
> quadrate:=proc(n)  
  local  
  a,b,c,d,cst,cct,a2,b2,c2,d2,a3,b3,c3,d3,a4,b4,c4,d4,a5,b5,c5,  
  d5,quadrat1, quadrat2, quadrat3, quadrat4,Bild,Bild2,R,i,z;  
  a:=<0,0>:  
  b:=<-1,0>:  
  c:=<-1,1>:  
  d:=<0,1>:  
  cst:=<1,1>:  
  cct:=<-1,1>:  
  z:=0.0:  
  quadrat1:=polygonplot(  
    map(convert,[a,b,c,d],list),  
    color=COLOR(HUE,z), scaling=constrained):  
  
  z:=z+0.3;  
  a2:=c+fib1(2)*a: b2:=c+fib1(2)*d: c2:=c+fib1(2)*cst:  
  d2:=c+fib1(2)*(-b): quadrat2:=polygonplot(  
    map(convert,[a2,b2,c2,d2],list),  
    color=COLOR(HUE,z), scaling=constrained,axes=None):  
  
  z:=z+0.3;  
  a3:=c2+fib1(3)*a: b3:=c2+fib1(3)*(-b): c3:=c2+fib1(3)*(-cct):  
  d3:=c2+fib1(3)*(-d):
```

```

quadrat3:=polygonplot(
  map(convert,[a3,b3,c3,d3],list),
  color=COLOR(HUE,z), scaling=constrained,axes=None):

z:=z+0.3;
a4:=c3+fib1(4)*a: b4:=c3+fib1(4)*(-d): c4:=c3+fib1(4)*(-cst):
d4:=c3+fib1(4)*(+b):
quadrat4:=polygonplot(
  map(convert,[a4,b4,c4,d4],list),
  color=COLOR(HUE,z), scaling=constrained,axes=None):

if n=1 then Bild:=(quadrat1): end if;
if n=2 then Bild:=(quadrat1,quadrat2): end if;
if n=3 then Bild:=(quadrat1,quadrat2,quadrat3): end if;
if n>=4 then Bild:=(quadrat1,quadrat2,quadrat3,quadrat4):
end if;

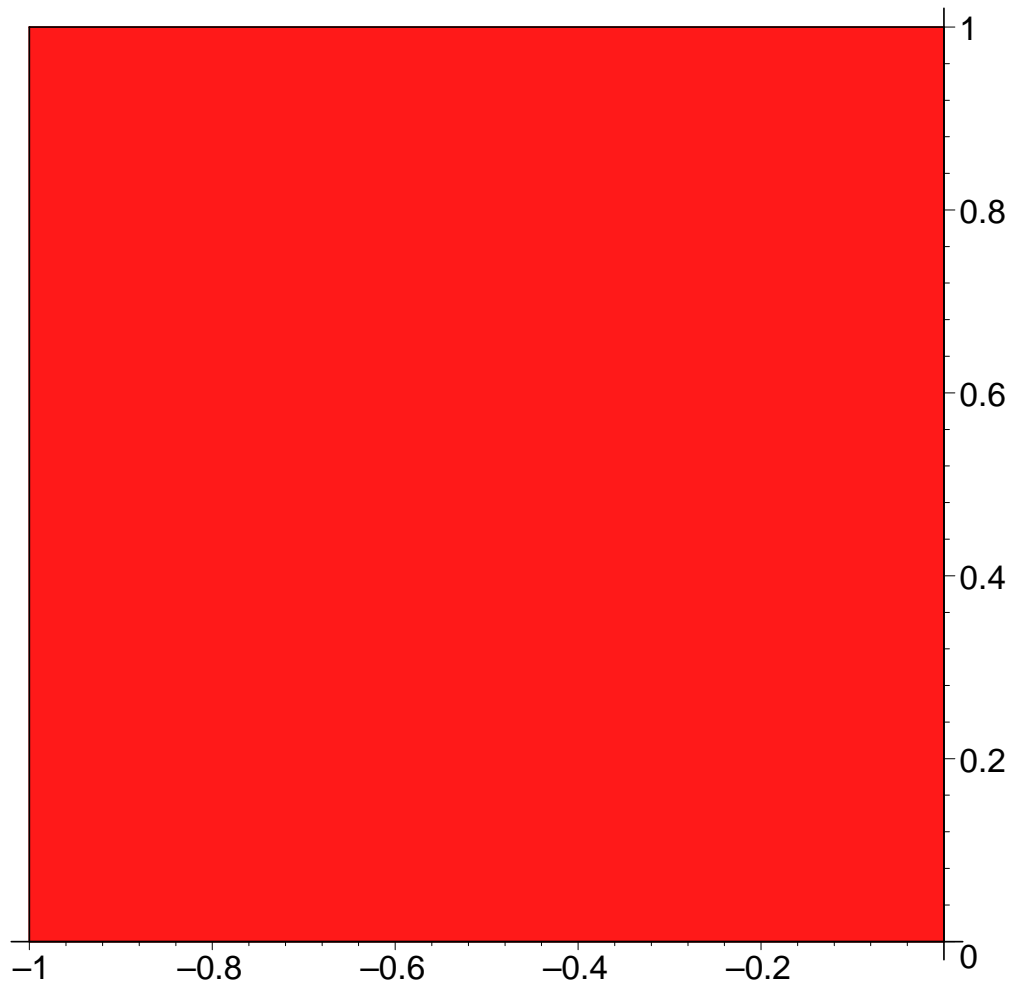
for i from 5 to n do
  if i mod 4 = 1 then
    z:=z+0.3;
    if z>=1 then z:=z-1; end if;
    a5:=c4+fib1(i)*a: b5:=c4+fib1(i)*(+b):
c5:=c4+fib1(i)*cct: d5:=c4+fib1(i)*d:
    R[i]:=polygonplot(
      map(convert,[a5,b5,c5,d5],list),
      color=COLOR(HUE,z), scaling=constrained,axes=None):
    end if;
  if i mod 4 = 2 then
    z:=z+0.3;
    if z>=1 then z:=z-1; end if;
    a2:=c5+fib1(i)*a: b2:=c5+fib1(i)*d: c2:=c5+fib1(i)*cst:
d2:=c5+fib1(i)*(-b):
    R[i]:=polygonplot(
      map(convert,[a2,b2,c2,d2],list),
      color=COLOR(HUE,z), scaling=constrained,axes=None):
    end if;
  if i mod 4 = 3 then
    z:=z+0.3;
    if z>=1 then z:=z-1; end if;
    a3:=c2+fib1(i)*a: b3:=c2+fib1(i)*(-b):
c3:=c2+fib1(i)*(-cct): d3:=c2+fib1(i)*(-d):
    R[i]:=polygonplot(
      map(convert,[a3,b3,c3,d3],list),
      color=COLOR(HUE,z), scaling=constrained,axes=None):
    end if;
  if i mod 4 = 0 then
    z:=z+0.3;
    if z>=1 then z:=z-1; end if;
    a4:=c3+fib1(i)*a: b4:=c3+fib1(i)*(-d):
c4:=c3+fib1(i)*(-cst): d4:=c3+fib1(i)*(+b):
    R[i]:=polygonplot(
      map(convert,[a4,b4,c4,d4],list),
      color=COLOR(HUE,z), scaling=constrained,axes=None):

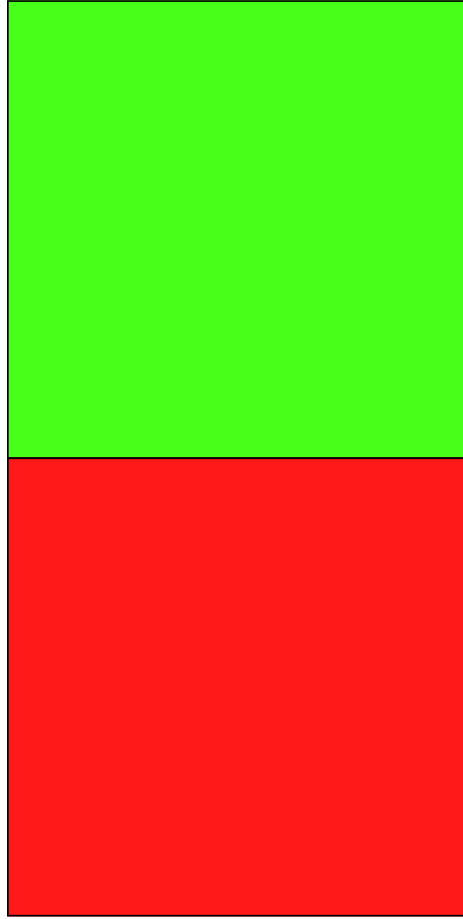
```

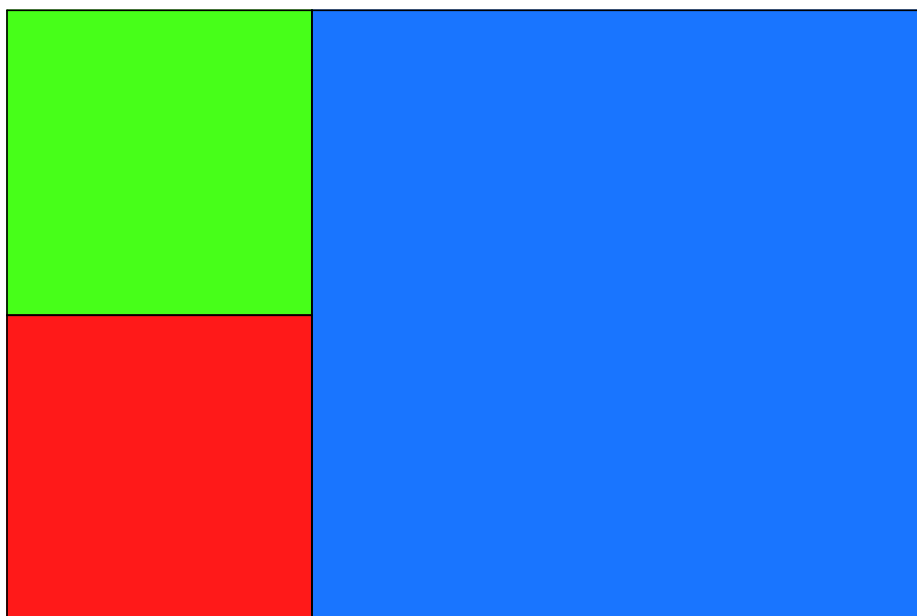
```
    end if;  
  end do;  
  Bild2:=seq(R[d], d=5..i-1):  
  display(Bild,Bild2);  
end proc:
```

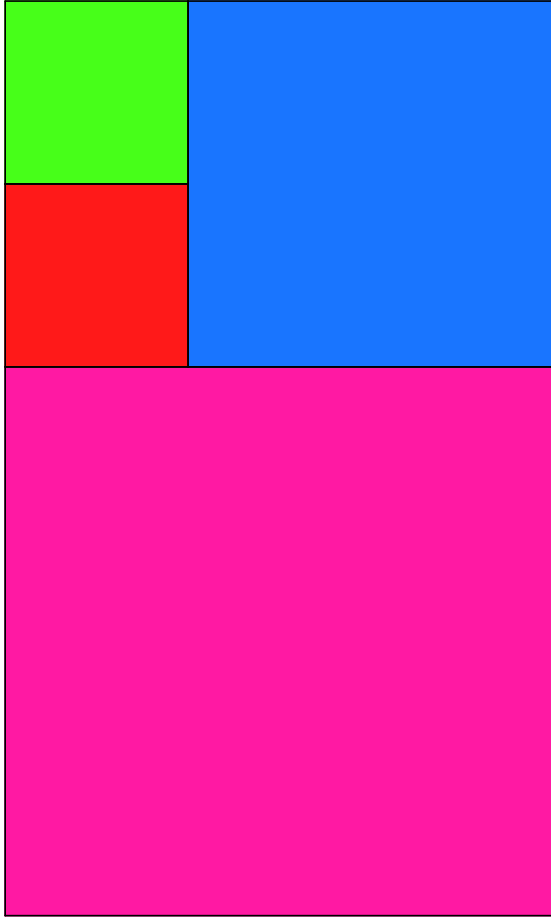
Ausgabe der Quadrate

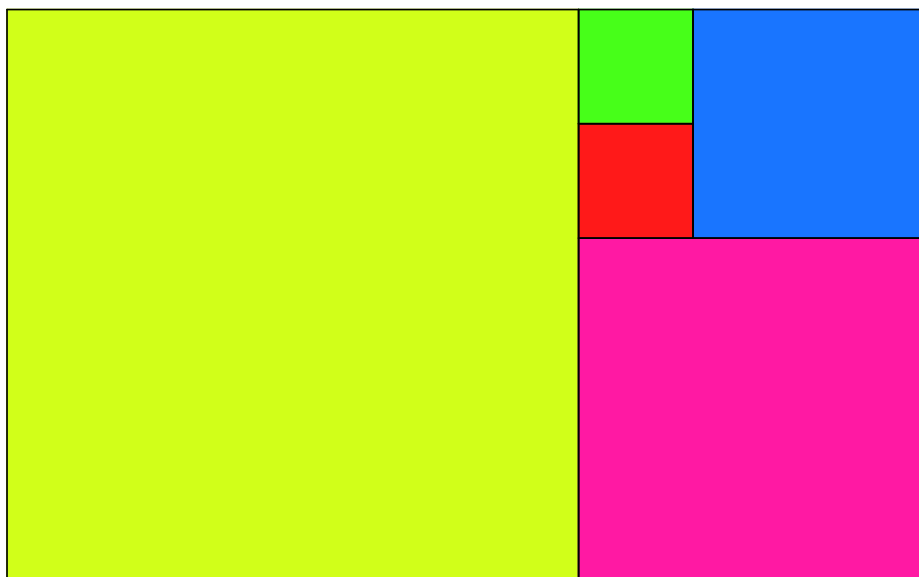
```
> for i from 1 to 11 do quadrate(i); end do;
```

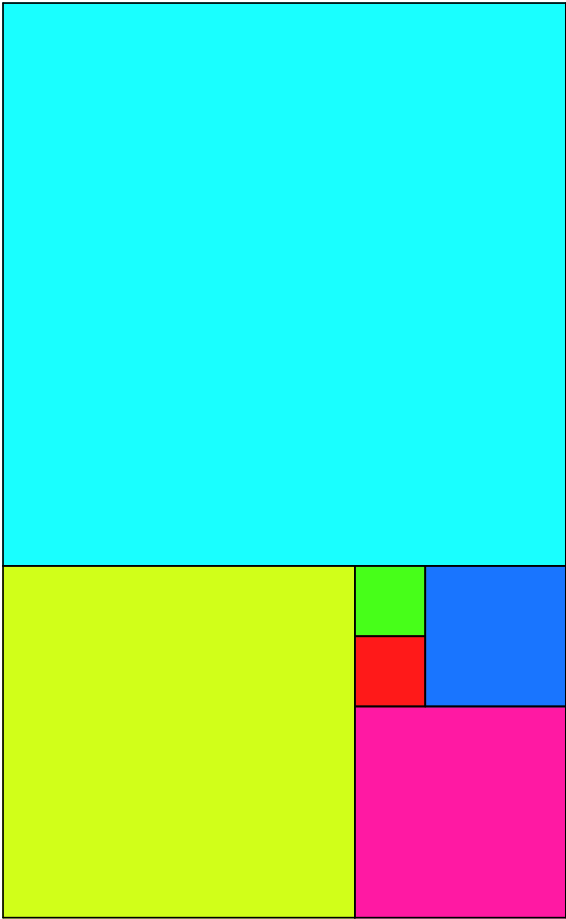


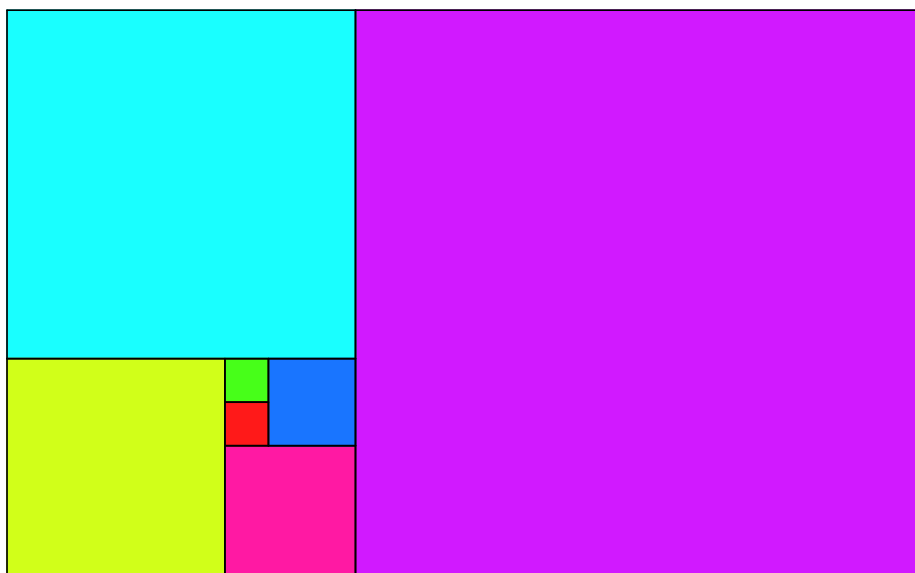


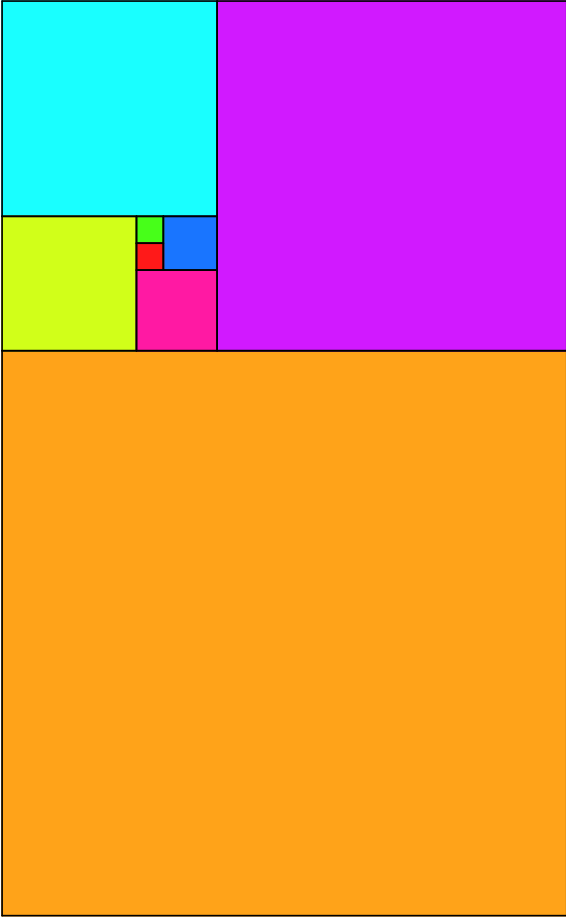


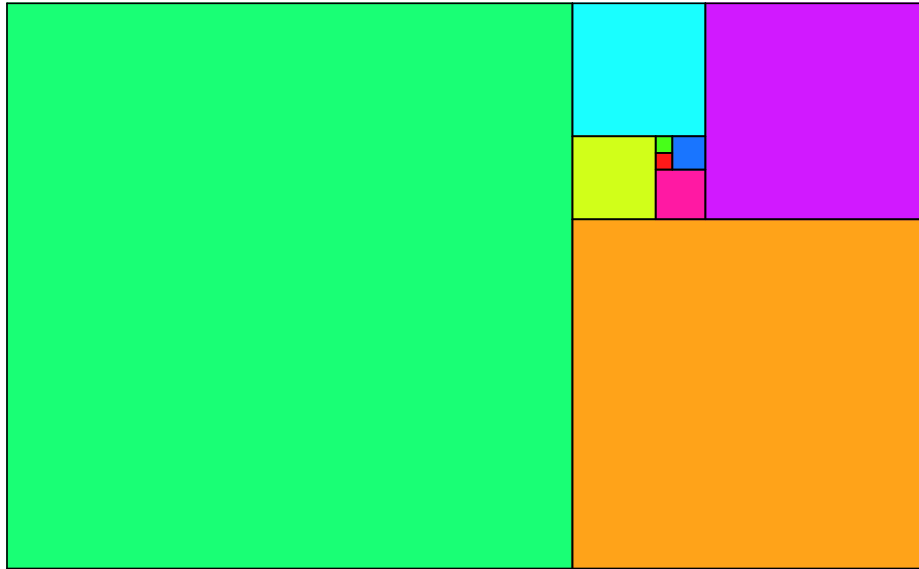


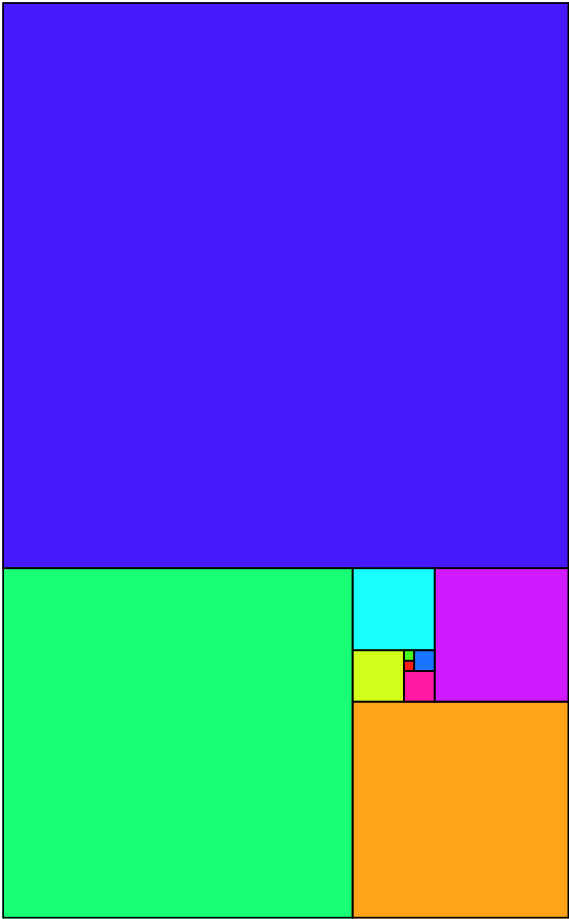


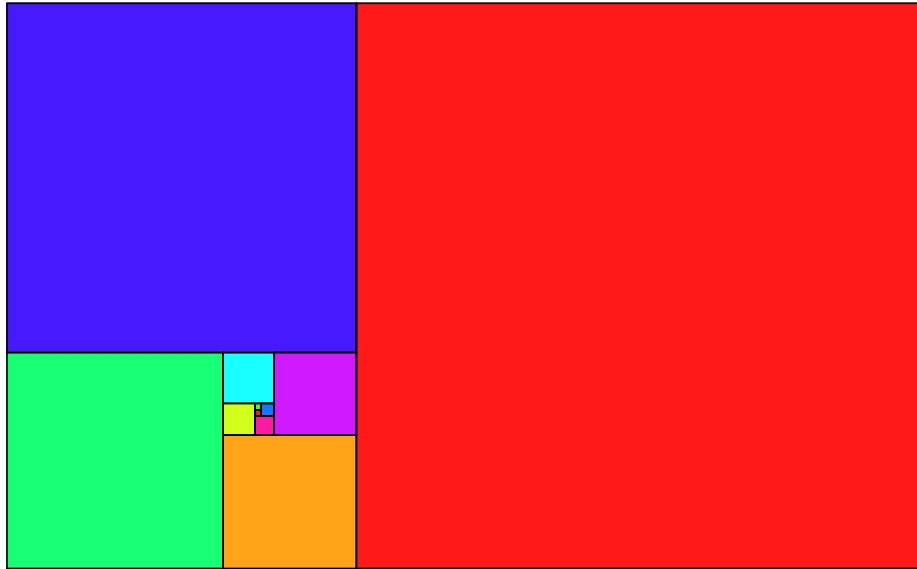






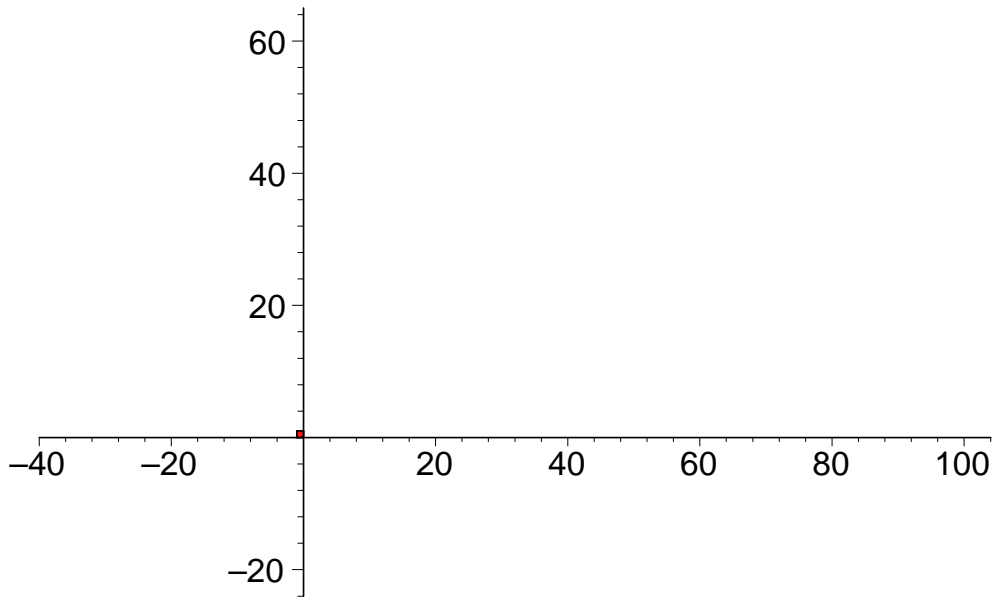






Film der Quadrate

```
> p:=NULL:  
  for i from 1 to 11 do p:=p,display(quadrate(i)); end do:  
  display([p],insequence=true,scaling=constrained);
```



– Aufgabe P5.1

Es sei $a(n)$ die Kantenlänge des Quadrates, also $a_1=1$, $a_2=1$, $a_3=2$, $a_4=3$,...

1. Bestimme die Rekursionsformel für $a(n)$, $n \geq 4$:

$$\mathbf{a(n) = a(n-1) + a(n-2)}$$

2. Schreibe eine Prozedur, welche $a(n)$ berechnet.

– Das Modul Fib.

```
> Fib:=module()
```

```

    local
LastCalculated,F,FPrev,Calls,RecursiveRememberExec
;
    export
Recursive,Iterative,IterativeOptimized,ByFormula,

RecursiveRemember,RecursiveCount,GetCalls,ResetCalls;
    description "compute Fibonacci numbers";
    option package;

Recursive:=proc(N::nonnegative);
    if N>=2 then
        return procname(N-2)+procname(N-1)
    else
        return N
    end if
end proc;

RecursiveRememberExec:=proc(N::nonnegative)
    option remember;
    if N>=2 then
        return procname(N-2)+procname(N-1)
    else
        return N
    end if
end proc;

RecursiveRemember:=proc(N::nonnegative)
    forget(RecursiveRememberExec);
    RecursiveRememberExec(N)
end proc;

RecursiveCount:=proc(N::nonnegative);
    Calls:=Calls+1;
    if N>=2 then
        return
RecursiveCount(N-2)+RecursiveCount(N-1)
    else
        return N
    end if
end proc;

```

```

ResetCalls:=proc();
  Calls:=0
end proc;

GetCalls:=proc():integer;
  return Calls
end proc;

Iterative:=proc(N::nonnegative)
  local F::integer,FPrev::integer,i::integer;
  if N<2 then
    return N
  else
    F:=1;
    FPrev:=1;
    for i from 3 to N do
      F,FPrev:=F+FPrev,F
    end do;
    return F
  end if
end proc;

IterativeOptimized:=proc(N::nonnegative)
  local Start::integer,i::integer;
  if N<2 then
    F:=N;
    FPrev:=0;
  else
    if N<LastCalculated then
      F:=1;
      FPrev:=1;
      Start:=3;
    else
      Start:=LastCalculated+1;
    end if;
    for i from Start to N do
      F,FPrev:=F+FPrev,F
    end do;
  end if;
  LastCalculated:=N;
  return F
end proc;

```

```
ByFormula:=N->simplify(1/sqrt(5)*((1/2+sqrt(5)/2)^N-(1/2-sqrt(5)/2)^N));
```

```
LastCalculated:=1:  
F:=1:  
FPrev:=0:  
Calls:=0:  
end module:
```

Normale Syntax für Aufrufe von Prozeduren eines Moduls:

```
> Fib:-Recursive(25);  
75025  
> Fib:-RecursiveRemember(25);  
75025  
> Fib:-Iterative(25);  
75025
```

Einbinden der von Fib exportierten Prozeduren (und ggf. Variablen):

```
> with(Fib):
```

Dann sind die vom Modul exportierten Prozeduren und Variablen ohne explizite Referenz auf das Modul aufrufbar.

```
> IterativeOptimized(25);  
75025  
> ByFormula(30);  
832040
```

Vergleich der Ergebnisse der verschiedenen Algorithmen (zur Kontrolle):

```
> for i from 0 to 15 do  
i,Recursive(i),Iterative(i),IterativeOptimized(i),  
ByFormula(i); end do;  
0,0,0,0,0  
1,1,1,1,1  
2,1,1,1,1  
3,2,2,2,2  
4,3,3,3,3  
5,5,5,5,5
```

6, 8, 8, 8, 8
7, 13, 13, 13, 13
8, 21, 21, 21, 21
9, 34, 34, 34, 34
10, 55, 55, 55, 55
11, 89, 89, 89, 89
12, 144, 144, 144, 144
13, 233, 233, 233, 233
14, 377, 377, 377, 377
15, 610, 610, 610, 610

3. Untersuche das Verhältnis $a(n+1)/a(n)$ mit wachsendem n .

– Berechnung des Verhältnisses. Der goldene Schnitt.

```
> fib2:= proc(n)
  local i,j, L, Liste;
  if n<2 then
    return 1;
  end if;
  for i from 0 to 1 do
    L[i]:= 1;
  end do;
  for i from 2 to n do
    L[i]:=0;
  end do;
  for i from 2 to n do
    for j from 1 to 2 do
      L[i]:= L[i]+L[i-j]
    end do;
  end do;
  L[i-1];
end proc:
> fiblist:= proc(n)
  local i, L:
  for i from 0 to n do
    L[i]:= fib2(i);
  end do;
  [seq(L[l], l=0..i-1)]:
end proc:
> fiblist(50);
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765,
```

```
10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040,  
1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817, 39088169,  
63245986, 102334155, 165580141, 267914296, 433494437, 701408733, 1134903170,  
1836311903, 2971215073, 4807526976, 7778742049, 12586269025, 20365011074]
```

```
> fiblist2:= proc(n)  
  local i, L,K;  
    for i from 0 to n do  
      L[i]:= fib2(i);  
    end do;  
    for i from 1 to n do  
      K[i]:= L[i]/L[i-1];  
    end do;  
    [seq(K[l], l=1..i-1)]:  
end proc:
```

```
> fiblist2(50);
```

```
[1, 2,  $\frac{3}{2}$ ,  $\frac{5}{3}$ ,  $\frac{8}{5}$ ,  $\frac{13}{8}$ ,  $\frac{21}{13}$ ,  $\frac{34}{21}$ ,  $\frac{55}{34}$ ,  $\frac{89}{55}$ ,  $\frac{144}{89}$ ,  $\frac{233}{144}$ ,  $\frac{377}{233}$ ,  $\frac{610}{377}$ ,  $\frac{987}{610}$ ,  $\frac{1597}{987}$ ,  $\frac{2584}{1597}$ ,  $\frac{4181}{2584}$ ,  $\frac{6765}{4181}$   
10946 17711 28657 46368 75025 121393 196418 317811 514229 832040  
6765' 10946' 17711' 28657' 46368' 75025' 121393' 196418' 317811' 514229'  
1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169  
832040' 1346269' 2178309' 3524578' 5702887' 9227465' 14930352' 24157817'  
63245986 102334155 165580141 267914296 433494437 701408733 1134903170  
39088169' 63245986' 102334155' 165580141' 267914296' 433494437' 701408733'  
1836311903 2971215073 4807526976 7778742049 12586269025 20365011074  
1134903170' 1836311903' 2971215073' 4807526976' 7778742049' 12586269025]
```

```
> evalf(%);
```

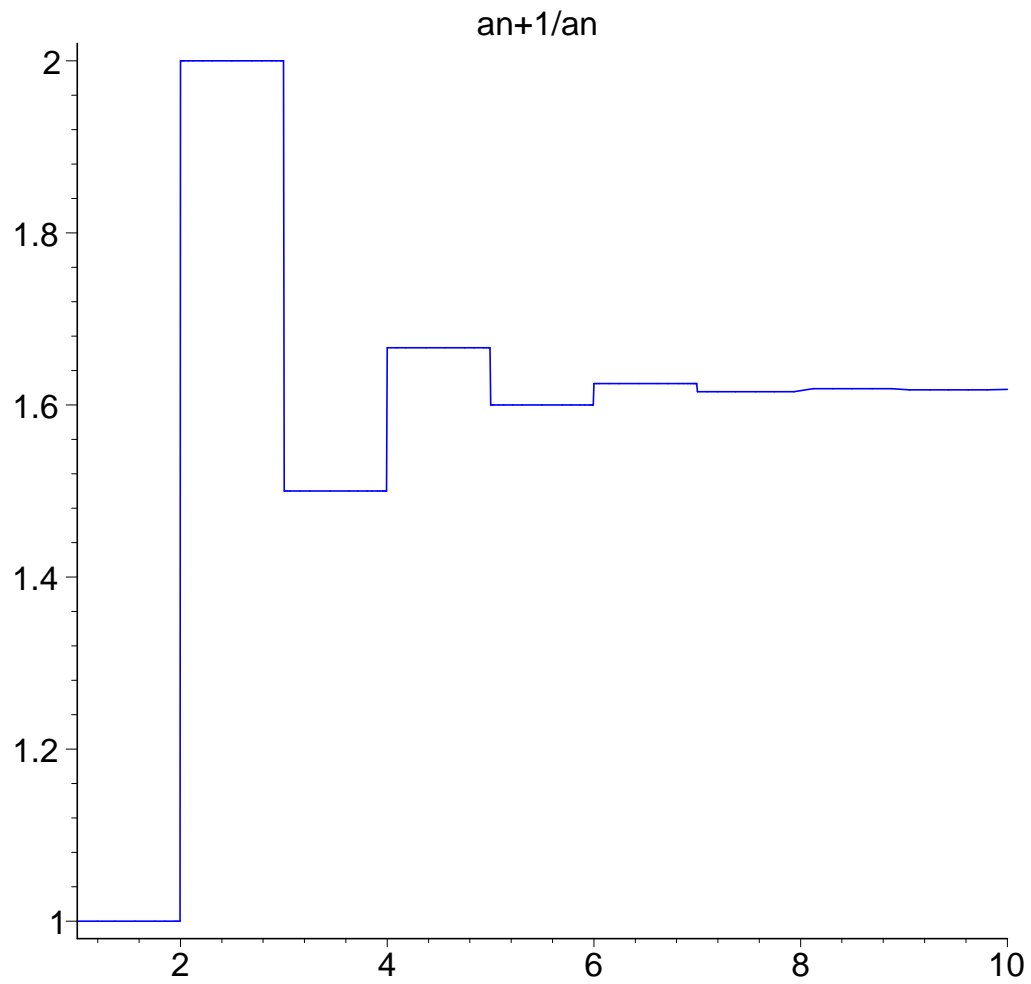
```
[1., 2., 1.500000000, 1.666666667, 1.600000000, 1.625000000, 1.615384615,  
1.619047619, 1.617647059, 1.618181818, 1.617977528, 1.618055556, 1.618025751,  
1.618037135, 1.618032787, 1.618034448, 1.618033813, 1.618034056, 1.618033963,  
1.618033999, 1.618033985, 1.618033990, 1.618033988, 1.618033989, 1.618033989,  
1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989,  
1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989,  
1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989,  
1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989,  
1.618033989]
```

scheint zu konvergieren...

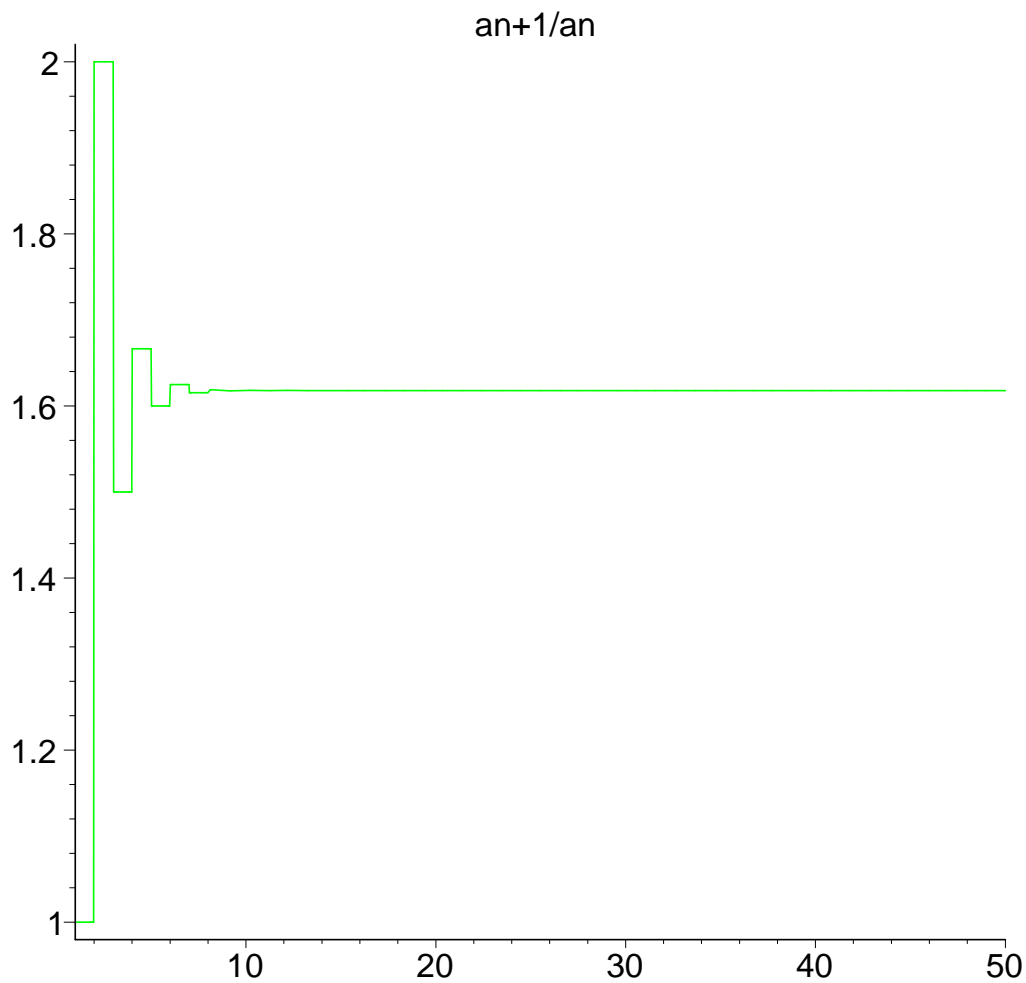
```
> f:=x ->evalf(fib2(x)/fib2(x-1));
```

$$f := x \rightarrow \text{evalf}\left(\frac{\text{fib2}(x)}{\text{fib2}(x-1)}\right)$$

```
> plot(f,1..10,color=blue, title="an+1/an");
```



```
> plot(f,1..50,color=green, title="an+1/an");
```



```

[ > with(combinat, fibonacci):
[ > g:=x->evalf(fibonacci(x)/fibonacci(x-1));
                                g := x → evalf(
                                 $\frac{\text{fibonacci}(x)}{\text{fibonacci}(x-1)}$ )
[ > g(50);
                                1.618033989
[ > limit(g(x), x=infinity);
                                 $\lim_{x \rightarrow \infty} \frac{\text{fibonacci}(x)}{\text{fibonacci}(x-1)}$ 
[ > evalf(%);
                                1.618033989

```

konvergiert tatsächlich! Zwei benachbarte Fibonaccizahlen stehen also im Verhältnis 1.618:1 (wenn sie groß genug sind)

```

> tau:=%;
                                     τ := 1.618033989
> 1/tau;
                                     0.6180339887
> tau-1/tau;
                                     1.000000000

```

Es ergibt sich folgende Gleichung:

```

> GL:='tau'=1+1/'tau';
                                     GL := τ = 1 +  $\frac{1}{\tau}$ 
> solve(x=1+1/x);
                                      $\frac{\sqrt{5}}{2} + \frac{1}{2}, \frac{1}{2} - \frac{\sqrt{5}}{2}$ 
> L:=[evalf(%)];
                                     L := [1.618033988, -0.6180339880]

```

Dieser Wert für Tau ist gerade das aus der Geometrie bekannte Zahlenverhältnis des goldenen Schnittes:

```

> GL2:='tau'=(1+sqrt(5))/2;
                                     GL2 := τ =  $\frac{\sqrt{5}}{2} + \frac{1}{2}$ 

```

Das Verhältnis zwischen zwei benachbarten Fibonacci-Zahlen (genügender Größe) ist also identisch mit dem des goldenen Schnittes. (War wahrscheinlich schon bei Platon bekannt, Blüte des goldenen Schnittes bei Leonardo da Vinci und Albrecht Dürer in der Malerei in der Renaissance, aber auch hauptsächlich in der Architektur. Ist häufig in der Natur anzutreffen A. Zeisig schreibt: "Der Goldene Schnitt ist das Grundprinzip aller, nach Schönheit drängenden Gestaltung im Reich der Natur (...), der die vollkommene Realisation erst im Menschen erfahren hat.)

Das Bildungsgesetz der Fibonacci-Folge erlaubt es, Quadrate mit diesen Zahlen als Seitenlänge spiralförmig aneinander zu setzen.

4. Finde eine Spirale.

— Grafische Ausgabe der Spirale

```

> spirale:=proc(n)
  local
  a,b,c,d,cst,cct,a2,b2,c2,d2,a3,b3,c3,d3,a4,b4,c4,d4,a5,b5,
  c5,d5,quadrat1,quadrat2,quadrat3,

```

```

quadrat4,Bild,Bild2,R,i,dd,dd2,dd3,dd4,dd5,k1,k2,k3,k4;
a:=<0,0>;
b:=<-1,0>;
c:=<-1,1>;
d:=<0,1>;
cst:=<1,1>;
cct:=<-1,1>;

dd:=convert(d,'list');
k1:=arc(dd,fib1(1),Pi..3*Pi/2);

a2:=c+fib1(2)*a: b2:=c+fib1(2)*d: c2:=c+fib1(2)*cst:
d2:=c+fib1(2)*(-b):

dd2:=convert(d2,'list');
k2:=arc(dd2,fib1(2),Pi/2..Pi);

a3:=c2+fib1(3)*a: b3:=c2+fib1(3)*(-b):
c3:=c2+fib1(3)*(-cct): d3:=c2+fib1(3)*(-d):

dd3:=convert(d3,'list');
k3:=arc(dd3,fib1(3),0..Pi/2);

a4:=c3+fib1(4)*a: b4:=c3+fib1(4)*(-d):
c4:=c3+fib1(4)*(-cst): d4:=c3+fib1(4)*(+b):

dd4:=convert(d4,'list');
k4:=arc(dd4,fib1(4),3*Pi/2..2*Pi);

if n=1 then Bild:=(k1): end if;
if n=2 then Bild:=(k1,k2): end if;
if n=3 then Bild:=(k1,k2,k3): end if;
if n>=4 then Bild:=(k1,k2,k3,k4): end if;

for i from 5 to n do
  if i mod 4 = 1 then
    a5:=c4+fib1(i)*a: b5:=c4+fib1(i)*(+b):
c5:=c4+fib1(i)*cct: d5:=c4+fib1(i)*d:
    dd5:=convert(d5,'list'):
    R[i]:=arc(dd5,fib1(i),Pi..3*Pi/2):
  end if;
  if i mod 4 = 2 then
    a2:=c5+fib1(i)*a: b2:=c5+fib1(i)*d:
c2:=c5+fib1(i)*cst: d2:=c5+fib1(i)*(-b):
    dd2:=convert(d2,'list'):
    R[i]:=arc(dd2,fib1(i),Pi/2..Pi):
  end if;
  if i mod 4 = 3 then
    a3:=c2+fib1(i)*a: b3:=c2+fib1(i)*(-b):
c3:=c2+fib1(i)*(-cct): d3:=c2+fib1(i)*(-d):
    dd3:=convert(d3,'list'):
    R[i]:=arc(dd3,fib1(i),0..Pi/2):
  end if;
end for;

```

```

    if i mod 4 = 0 then
      a4:=c3+fib1(i)*a: b4:=c3+fib1(i)*(-d):
c4:=c3+fib1(i)*(-cst): d4:=c3+fib1(i)*(+b):
      dd4:=convert(d4,'list'):
      R[i]:=arc(dd4,fib1(i),3*Pi/2..2*Pi):
    end if;
end do:
Bild2:=seq(R[d], d=5..i-1):
display(Bild,Bild2);
end proc:

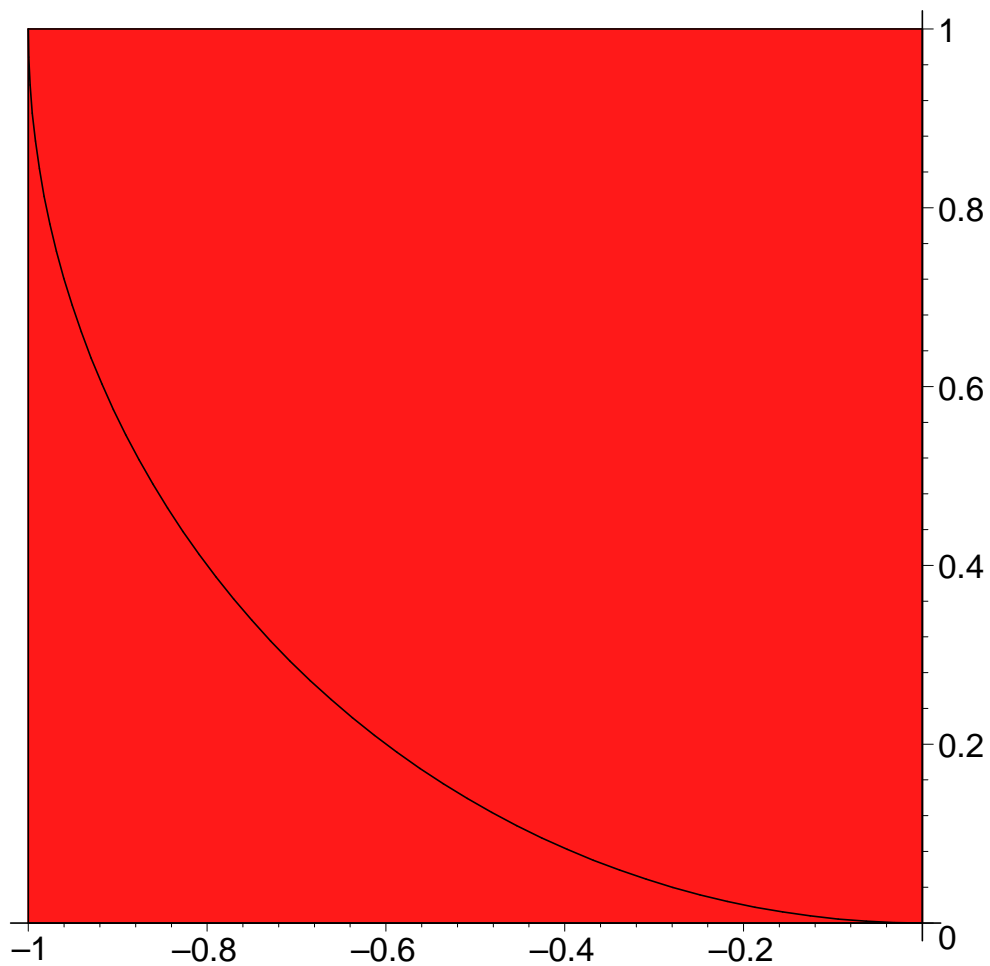
```

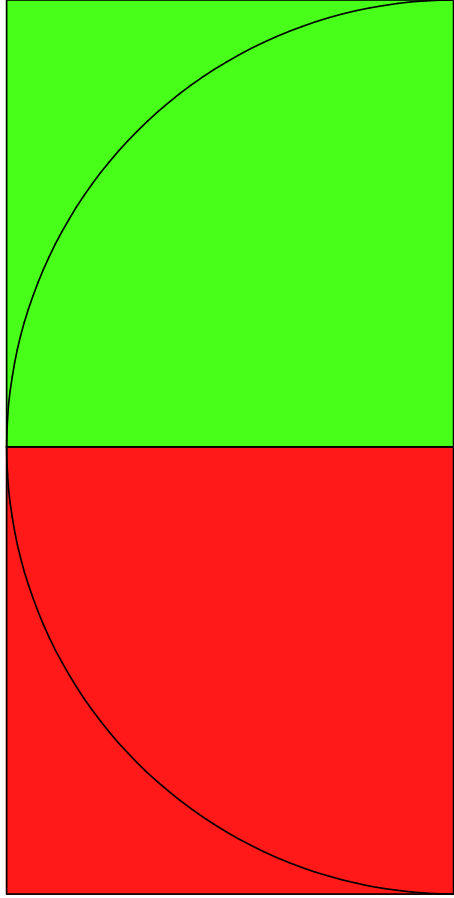
Ausgabe der Quadrate mit Spirale

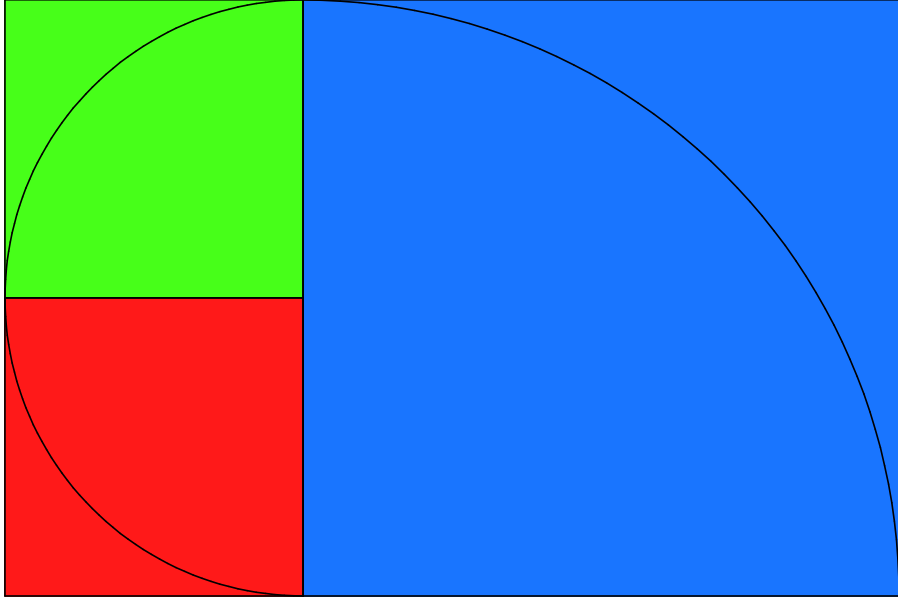
```

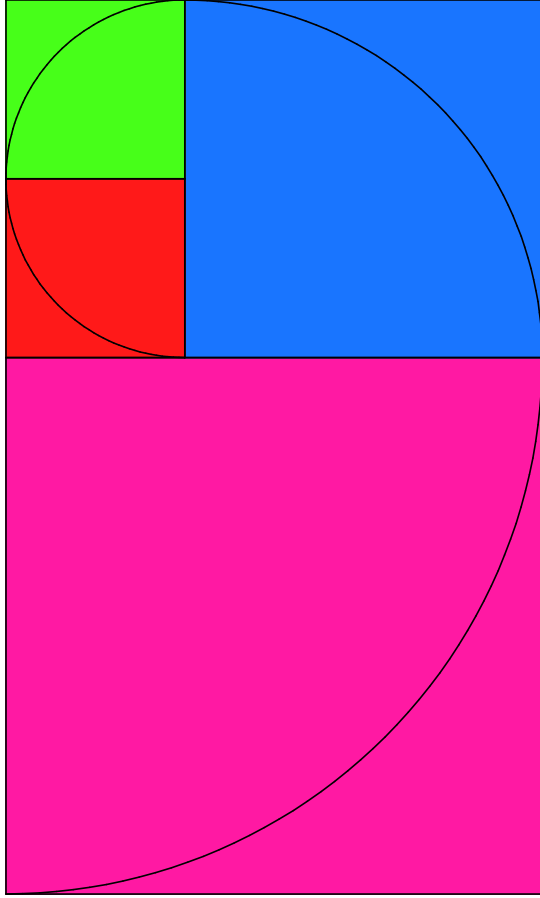
> beides:=proc(n)
  display(quadrate(n),spirale(n));
end proc:
> for i from 1 to 11 do beides(i); end do;

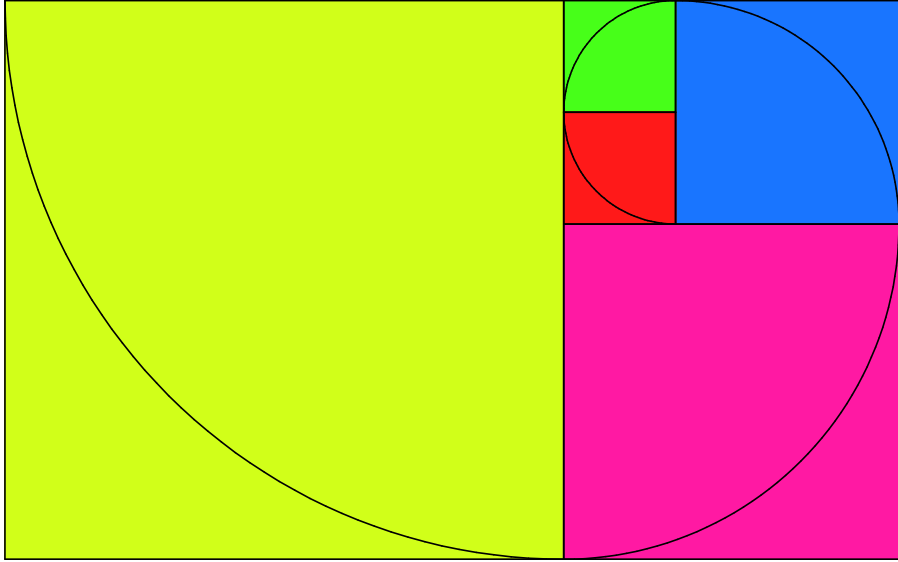
```

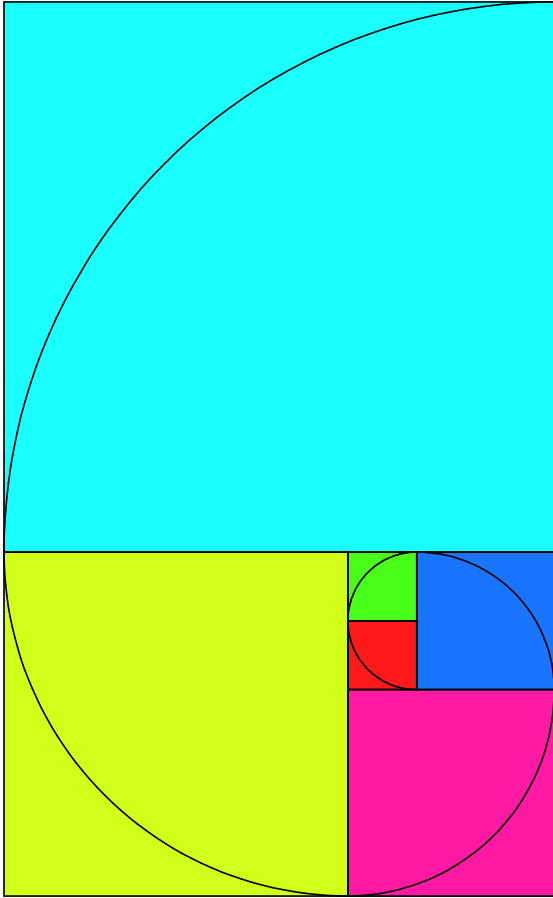


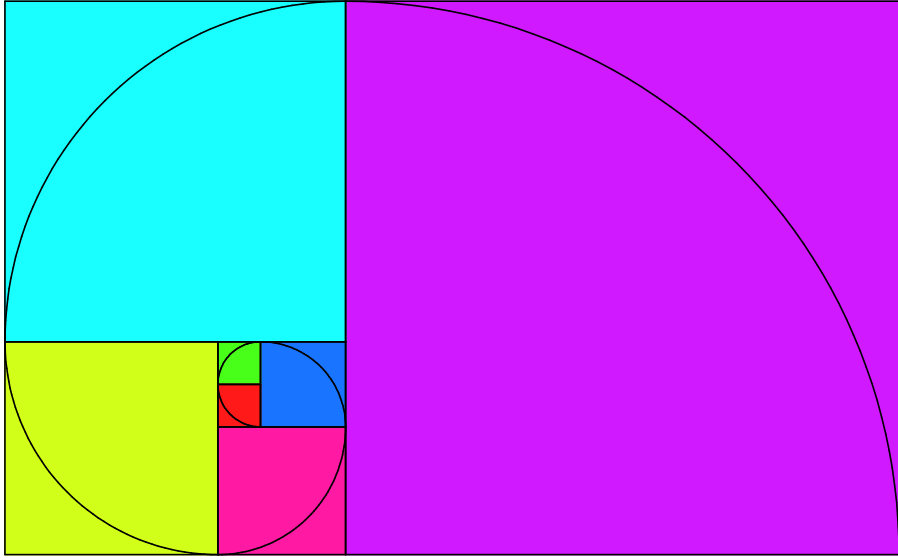


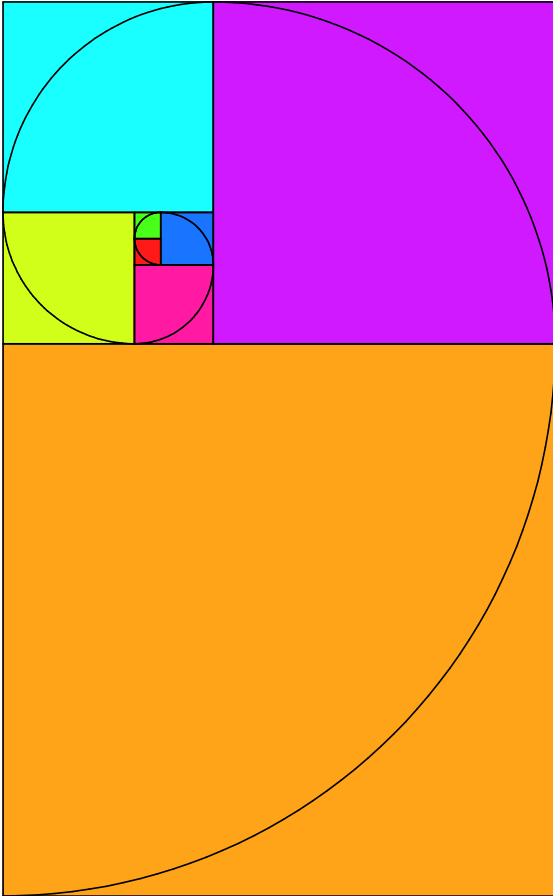


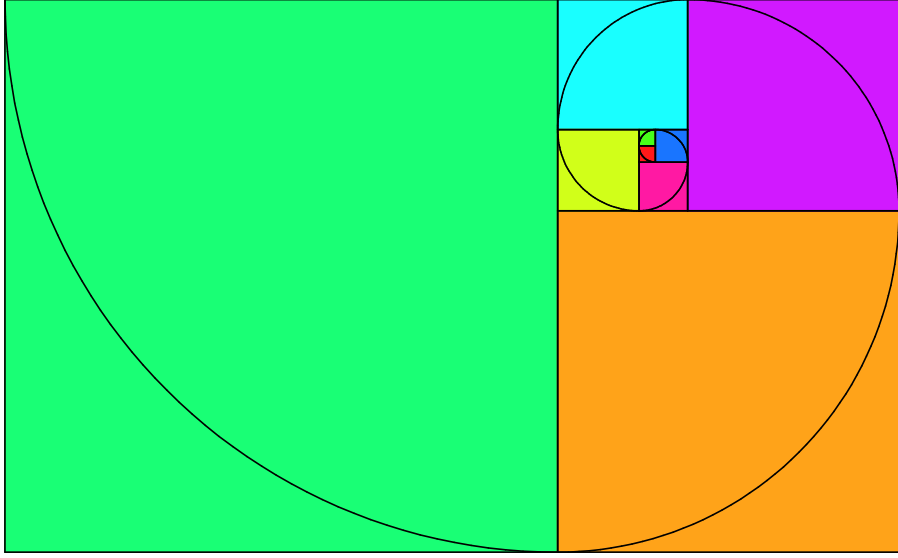


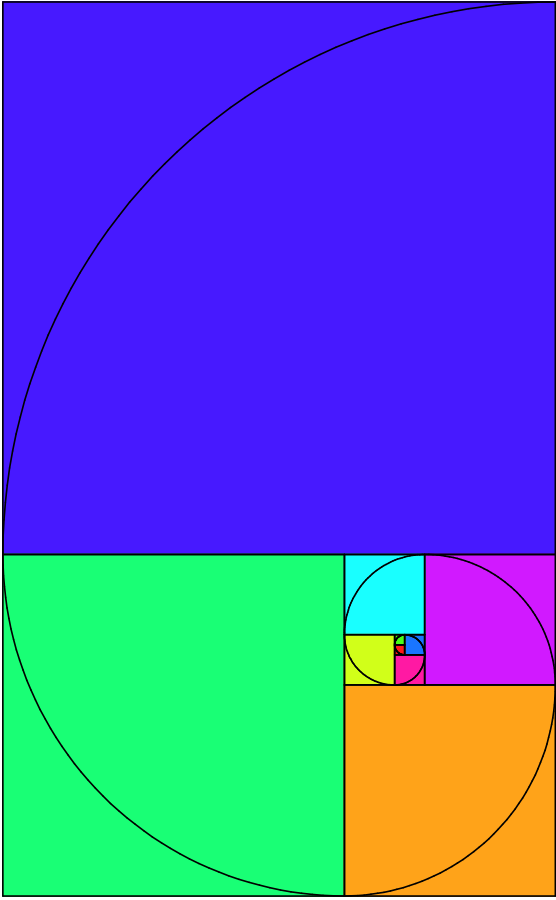


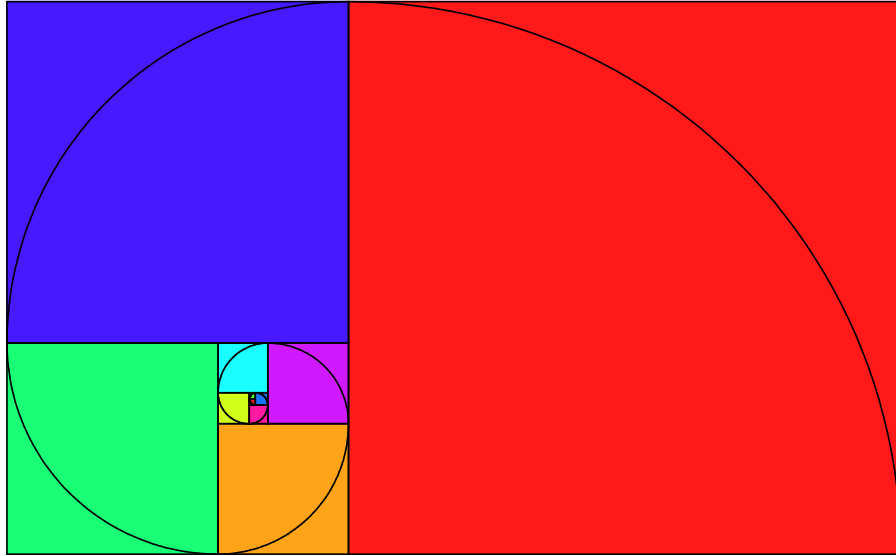






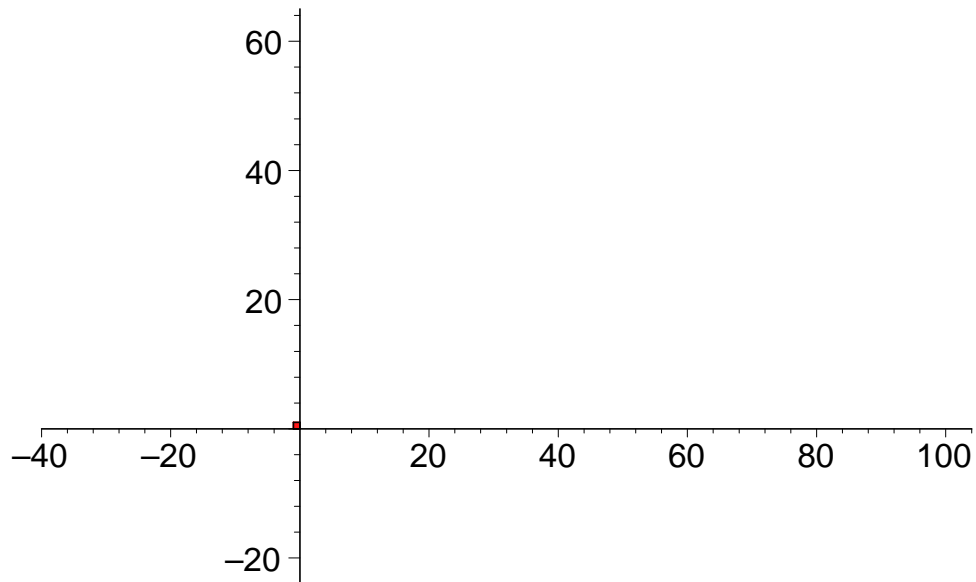






Film der Quadrate mit Spirale

```
> spirale2:=proc(n)
  display(quadrate(11),spirale(n));
end proc:
> p:=NULL:
  o:=NULL:
  for i from 1 to 11 do p:=p,display(quadrate(i)); end do:
  for i from 1 to 11 do o:=o,display(spirale2(i)); end do:
  display([p],[o],insequence=true,scaling=constrained);
```



5. Eine neue Rekursionsformel.

– **Rekursionsformel mit Matrizen**

```

> b := proc(n)
  local A;
  option remember;
  A:=<<0,1>|<1,1>>;
  if n <= 1 then
    <0,1>;
  else
    A . procname(n-1);
  end if;
end proc;
> b(8);

```

$\begin{bmatrix} 13 \\ 21 \end{bmatrix}$

Aufgabe P5.2

Eigenwerte

```
> with( LinearAlgebra );  
> A:=<<0,1>|<1,1>>;  

$$A := \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$
  
> CharacteristicPolynomial( A, T );  

$$T^2 - T - 1$$
  
> solve( % );  

$$\frac{\sqrt{5}}{2} + \frac{1}{2}, \frac{1}{2} - \frac{\sqrt{5}}{2}$$
  
> evalf(%);  
1.618033988, -0.6180339880
```

Die Eigenwerte der Matrix A aus 5.1.5 sind gerade die Lösungen aus der Gleichung GL aus der Untersuchung des Grenzwertes:

```
> L;  
[1.618033988, -0.6180339880]
```

Bestimme $J, Q := \text{JordanForm}(A, \text{output}=['J', 'Q'])$.

Jordanform

```
> J, Q := JordanForm( A, output=['J', 'Q'] );  

$$J, Q := \begin{bmatrix} \frac{1}{2} - \frac{\sqrt{5}}{2} & 0 \\ 0 & \frac{\sqrt{5}}{2} + \frac{1}{2} \end{bmatrix}, \begin{bmatrix} \frac{(\sqrt{5}+1)\sqrt{5}}{10} & \frac{(-1+\sqrt{5})\sqrt{5}}{10} \\ -\frac{\sqrt{5}}{5} & \frac{\sqrt{5}}{5} \end{bmatrix}$$

```

Der Befehl `JordanForm` gibt also die Jordanform der Matrix A und die entsprechende Matrix Q aus!

Prüfe $Q \cdot J \cdot Q^{-1}$.

QJQ⁽⁻¹⁾

```
> Q . J . Q^(-1);  

$$\left[ \frac{(\sqrt{5}+1)\sqrt{5} \left( \frac{1}{2} - \frac{\sqrt{5}}{2} \right)}{10} + \frac{(-1+\sqrt{5})\sqrt{5} \left( \frac{\sqrt{5}}{2} + \frac{1}{2} \right)}{10}, \right.$$
  

$$\left. \frac{(\sqrt{5}+1)\sqrt{5} \left( \frac{1}{2} - \frac{\sqrt{5}}{2} \right)^2}{10} + \frac{(-1+\sqrt{5})\sqrt{5} \left( \frac{\sqrt{5}}{2} + \frac{1}{2} \right)^2}{10} \right]$$

```

```

[
  [
    [
      
$$\left[ -\frac{\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)\sqrt{5}}{5} + \frac{\left(\frac{\sqrt{5}}{2}+\frac{1}{2}\right)\sqrt{5}}{5}, -\frac{\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)^2\sqrt{5}}{5} + \frac{\left(\frac{\sqrt{5}}{2}+\frac{1}{2}\right)^2\sqrt{5}}{5} \right]$$

    ]
  ]
]
> simplify( % );
      [ 0  1 ]
      [ 1  1 ]

```

Finde eine geschlossene Formel für an und schreibe eine Prozedur.

— Geschlossene Formel

```

[
  > A;
      [ 0  1 ]
      [ 1  1 ]
  > for i from 1 to 6 do b(i); end do;
      [ 0 ]
      [ 1 ]
      [ 1 ]
      [ 1 ]
      [ 2 ]
      [ 2 ]
      [ 3 ]
      [ 3 ]
      [ 5 ]
      [ 5 ]
      [ 8 ]
  > w:=b(1);
      w := [ 0 ]
           [ 1 ]
  > A . w;
      [ 1 ]
      [ 1 ]
  > A^2 . w;
      [ 1 ]
      [ 2 ]
  > A^5 . w;

```

```
[ 5 ]  
[ 8 ]
```

```
> fib3:= proc(n)  
  local A,w, Erg;  
  A:=<<0,1>|<1,1>>;  
  w:= <0,1>;  
  Erg:=A^(n-1) . w;  
  Erg[2];  
end proc;  
> fib3(6);  
  
> for i from 1 to 10 do fib3(i); end do;  
  
1  
1  
2  
3  
5  
8  
13  
21  
34  
55
```

Laufzeitbetrachtungen

– Die Rekursionsformel: Explorierende Anzahl der Aufrufe

```
> with(plots):  
  with(Fib):  
> for i to 20 do ResetCalls(): print(i,RecursiveCount(i),  
  GetCalls()) end do;  
  
1, 1, 1  
2, 1, 3  
3, 2, 5  
4, 3, 9  
5, 5, 15  
6, 8, 25  
7, 13, 41  
8, 21, 67  
9, 34, 109  
10, 55, 177  
11, 89, 287  
12, 144, 465  
13, 233, 753  
14, 377, 1219
```

15, 610, 1973
16, 987, 3193
17, 1597, 5167
18, 2584, 8361
19, 4181, 13529
20, 6765, 21891

– GetTimeAdaptive: Eine Prozedur zur komfortablen Zeitmessung

GetTimeAdaptive bekommt eine Prozedur und einen Zeitrahmen (in s) für die Messung übergeben.

Es erfolgt eine Schätzung der benötigten Zeit. Liegt diese unterhalb der im Zeitrahmen erlaubten Zeit,

so wird die Prozedur mehrfach ausgeführt und die Laufzeit gemittelt. Dadurch erhält man auch bei

sehr kurzen Laufzeiten noch vernünftige Messwerte. Ein größerer Zeitrahmen erhöht die Stabilität

gegenüber Störeinflüssen (Hintergrundaktivitäten, etc.)..

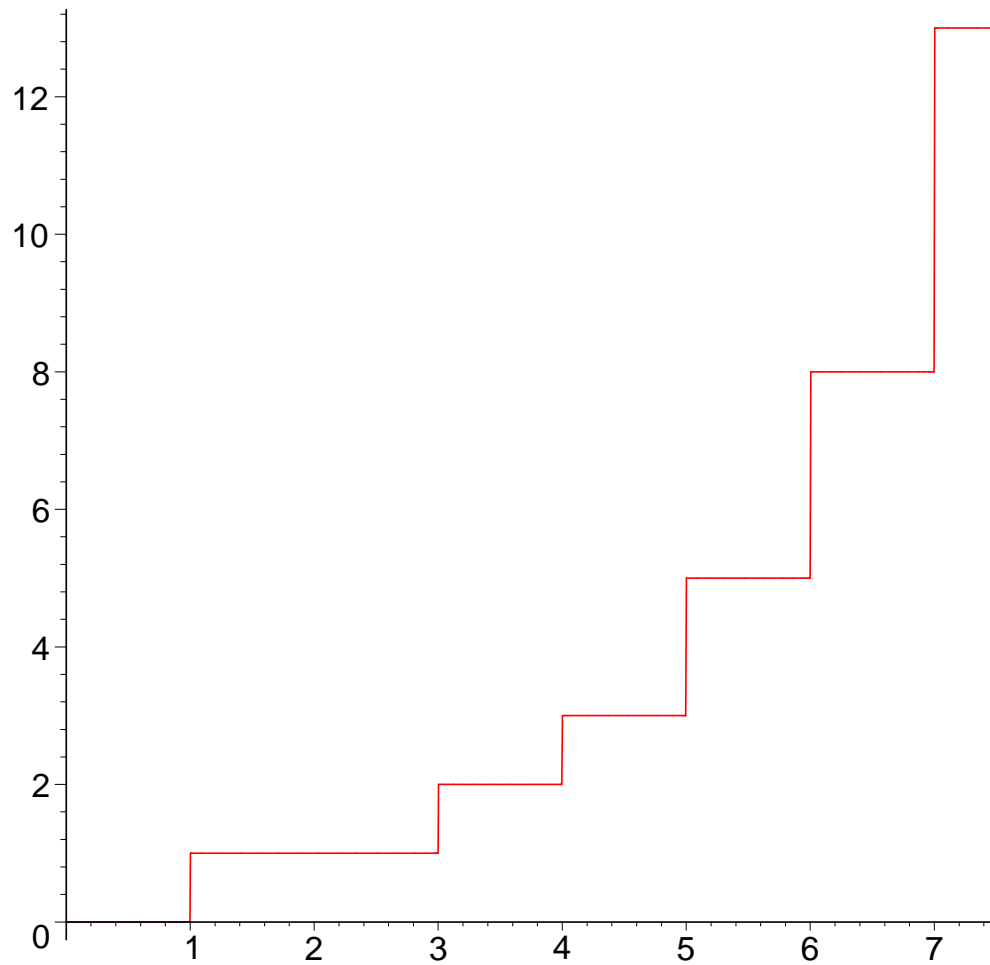
Die Rückgabe ist die Zeit für einen Prozedurlauf in Sekunden.

```
> GetTimeAdaptive:=proc(P::function,Time::float)
  local ReqTime,Scale,Count,st,i::integer;
  ReqTime:=0;
  Scale:=1;
  while ReqTime<0.1 do
    st:=time();
    for i from 1 to Scale do
      eval(P)
    end do;
    ReqTime:=time()-st;
    if ReqTime<0.1 then
      Scale:=Scale*5
    end if;
  end do;
  if ReqTime<Time then
    Count:=floor(Time/ReqTime*Scale);
    st:=time();
    for i from 1 to Count do
      eval(P)
    end do;
    ReqTime:=(time()-st)/Count
  else
    ReqTime:=ReqTime/Scale
  end if;
  return ReqTime;
end proc;
```

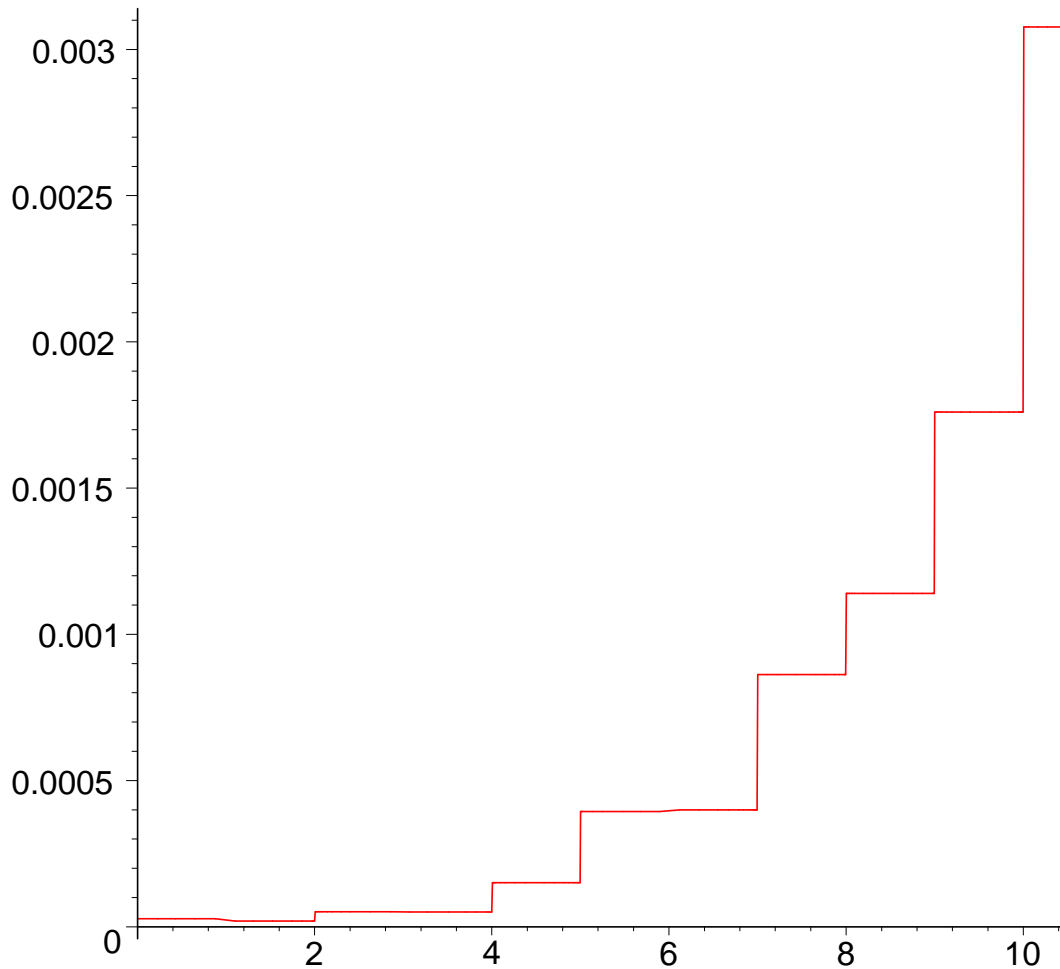
– ListPlot: Eine Prozedur zum Plotten von diskreten Funktionen ohne unnötige Funktionsaufrufe

```
> ListPlot:=proc(P::function,X1::integer,X2::integer)
  local L,f;
```

```
L:=seq(eval(P),i=X1..X2);  
f:=n->L[floor(n-X1+1)];  
plot(f,X1..X2+0.5)  
end proc:  
> ListPlot('IterativeOptimized(i)',0,7);
```



```
> ListPlot('GetTimeAdaptive('Fib:-Recursive(i)',0.2)',0,10);
```



Die gemessenen Zeiten für die verschiedenen Algorithmen

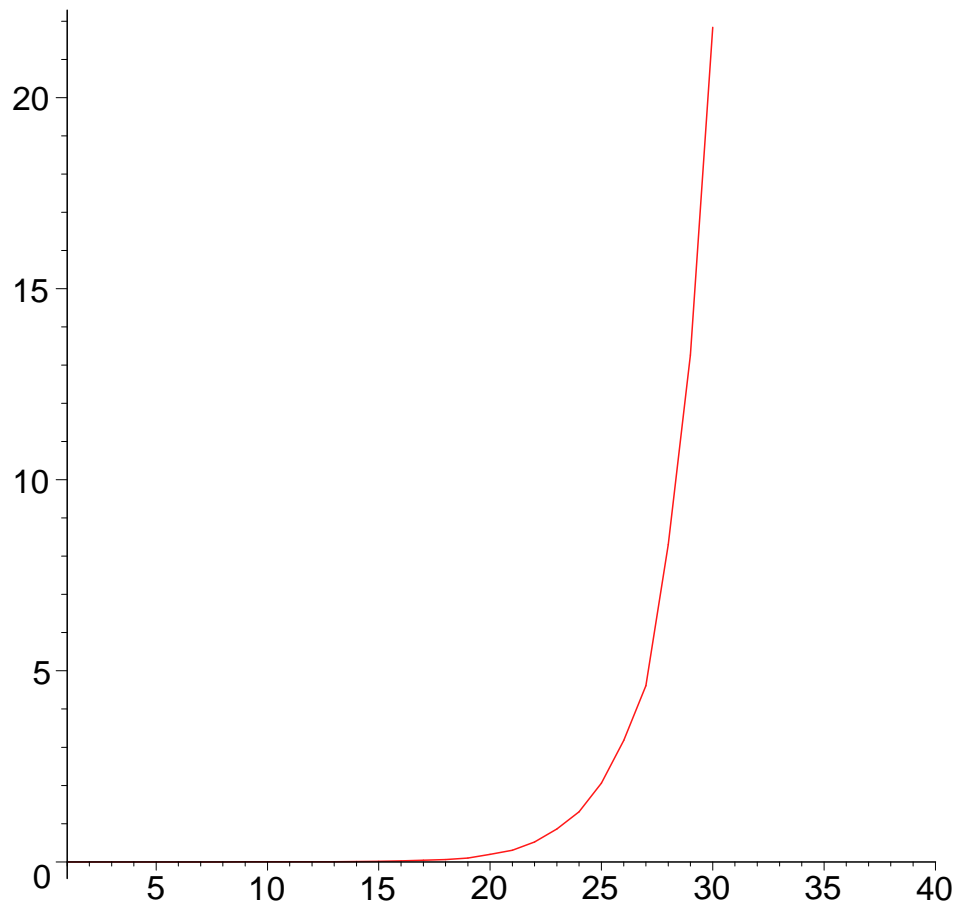
Die nachfolgenden Berechnungen können (in Abhängigkeit vom gewählten Zeitrahmen für GetTimeAdaptive) einige Zeit in Anspruch nehmen, daher sind die fertigen Grafiken mitgeliefert.
 Vorsicht: Eine Neuberechnung mit den vorgegebenen Parametern dauert mehrere Minuten.

```
> MTime:=1.0:
```

Der Rekursive Algorithmus (vergrößert)

Die Laufzeit explodiert schon für kleine n.

```
> f:=n->GetTimeAdaptive('Recursive(floor(n))',MTime):
  TRecursive1:=plot(f,1..40,sample=[seq(i,i=1..30)],color=
  COLOR(HUE,0),adaptive=false):
> display(TRecursive1);
```

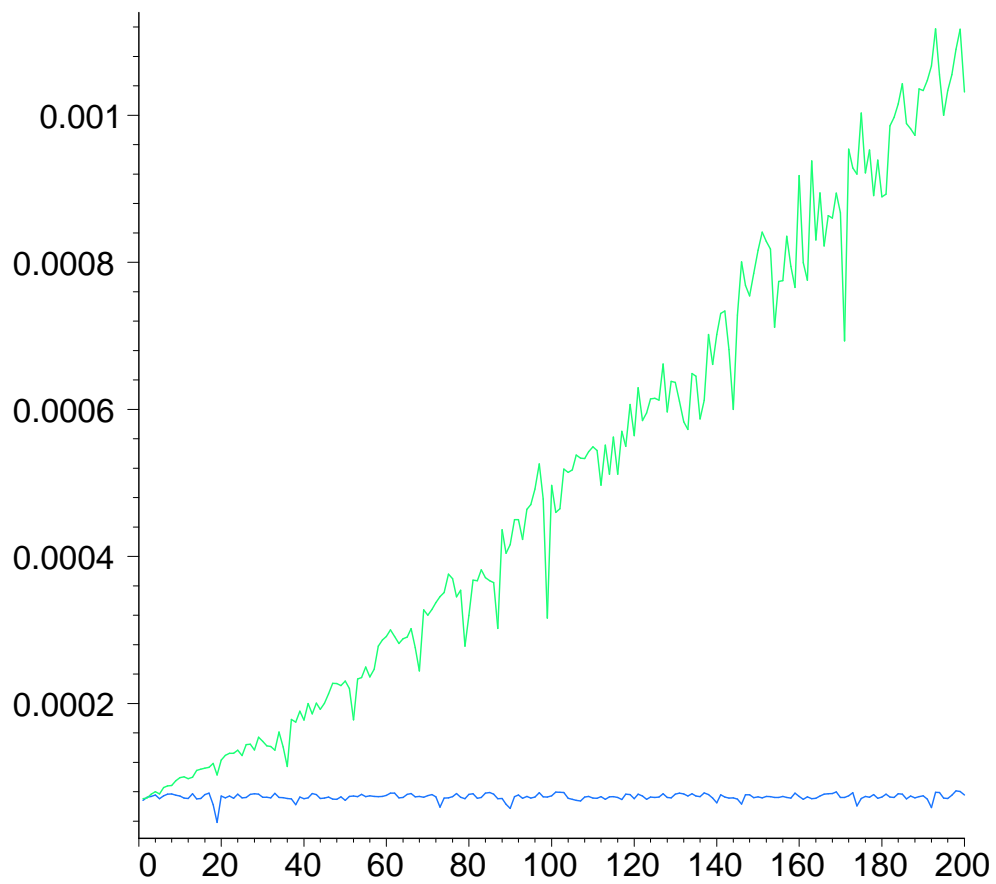


– **Vergleich der Algorithmen**

```
[ > x1,x2:=0,200:
```

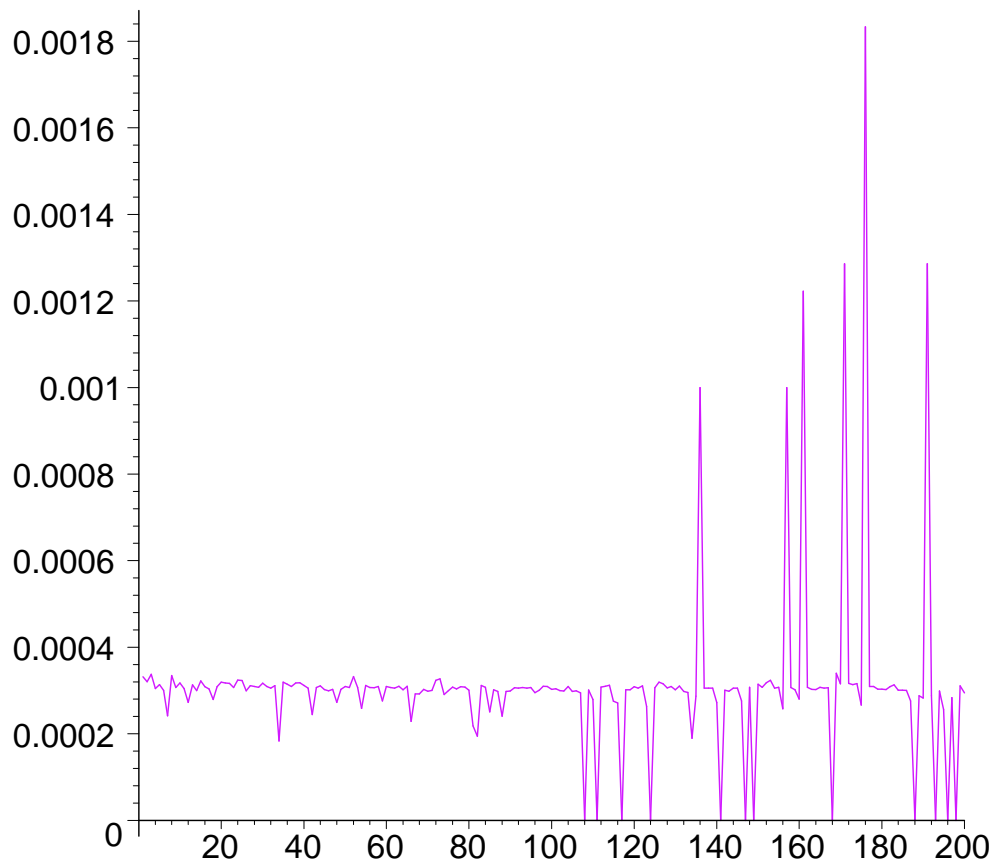
– **Die iterativen Algorithmen**

```
[ > f:=n->GetTimeAdaptive('Iterative(floor(n))',MTime):  
  TIterative:=plot(f,x1..x2,sample=[seq(i,i=1..200)],co  
  lor=COLOR(HUE,0.4),adaptive=false):  
[ > f:=n->GetTimeAdaptive('IterativeOptimized(floor(n))',  
  MTime):  
  TIterativeOptimized:=plot(f,x1..x2,sample=[seq(i,i=1.  
  .200)],color=COLOR(HUE,0.6),adaptive=false):  
[ > display(TIterative,TIterativeOptimized);
```



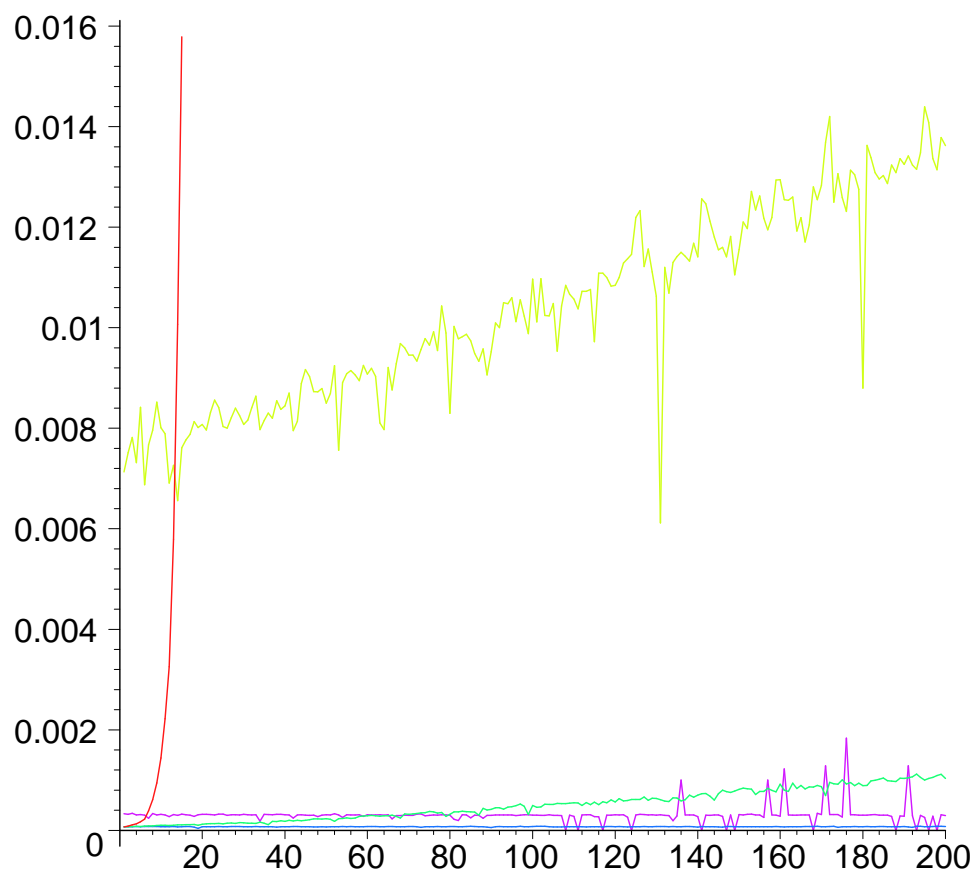
— **Die geschlossene Formel**

```
[ > f:=n->GetTimeAdaptive('ByFormula(floor(n))',MTime):  
  TByFormula:=plot(f,X1..X2,sample=[seq(i,i=1..200)],co  
  lor=COLOR(HUE,0.8),adaptive=false):  
[ > display(TByFormula);
```



— **Alle Algorithmen im Vergleich**

```
[ > f:=n->GetTimeAdaptive('Recursive(floor(n))',MTime):
  TRecursive:=plot(f,X1..X2,sample=[seq(i,i=1..15)],col
  or=COLOR(HUE,0),adaptive=false):
[ > f:=n->GetTimeAdaptive('RecursiveRemember(floor(n))',M
  Time):
  TRecursiveRemember:=plot(f,X1..X2,sample=[seq(i,i=1..
  200)],color=COLOR(HUE,0.2),adaptive=false):
[ > display(TRecursive,TRecursiveRemember,Titerative,Tite
  rativeOptimized,TByFormula);
```



[>