

# Projekt P5 zu Mathe am Computer, WS

## 2002/03

David Flore & Rudi Penner, Paderborn  
5. Februar 2003

### Eine ganz besondere Spirale

#### Aufgabe P5.1 (Spiralige Quadrate)

(i) Bestimme eine Rekursionsformel für  $a_n$ ,  $4 \leq n$  !

```
> restart;  
'mod' := modp;
```

Die Kantenlängen  $a_n$  der ersten vier Quadrate sind bereits vorgegeben. Die weiteren Kantenlängen kann man dann wie in der Aufgabenstellung beschrieben berechnen. Wir geben einmal die Kantenlängen der ersten 7  $a_n$ 's an:

```
> a[1]=1;  
a[2]=1;  
a[3]=2;  
a[4]=3;  
a[5]=5;  
a[6]=8;  
a[7]=13;
```

$$a_1 = 1$$

$$a_2 = 1$$

$$a_3 = 2$$

$$a_4 = 3$$

$$a_5 = 5$$

$$a_6 = 8$$

$$a_7 = 13$$

Man erkennt sehr schnell, dass die Rekursionsformel für die **Fibonaccizahlen** wie folgt lautet:  $a_n := a_{n-1} + a_{n-2}$ . Diese Formel gilt offensichtlich auch schon für  $n \geq 3$ !

Als nächstes prüfen wir die Formel für die ersten 15 Quadrate.

```
> a[1]:=1;  
a[2]:=1;  
for i from 3 to 15 do  
  a[i]:=a[i-1]+a[i-2];  
end do;
```

$$a_1 := 1$$

$$a_2 := 1$$

$a_3 := 2$   
 $a_4 := 3$   
 $a_5 := 5$   
 $a_6 := 8$   
 $a_7 := 13$   
 $a_8 := 21$   
 $a_9 := 34$   
 $a_{10} := 55$   
 $a_{11} := 89$   
 $a_{12} := 144$   
 $a_{13} := 233$   
 $a_{14} := 377$   
 $a_{15} := 610$

**Antwort:** Wir halten die Rekursionsformel

$a_n = a_{n-1} + a_{n-2}$  fest, die bereits für  $n \geq 3$  gilt!

- (ii) *Schreibe eine Prozedur  $a := \text{proc}(n) \dots$ , welche  $a_n$  berechnet. Dabei darf die Prozedur sich selbst aufrufen; in MAPLE geht das, indem man beispielsweise  $\text{procname}(n-1)$  einbaut. (procname ist ein MAPLE-Kommando)!*

Auch hier sind die Kantenlängen der ersten 2 Quadrate wieder bekannt. Die Prozedur ruft sich über  $\text{procname}$  selbst wieder auf. Eigentlich besteht die Prozedur nur aus der Rekursionsformel. Allerdings ist sie wegen einer Kleinigkeit, auf die wir später noch eingehen, noch nicht so gut geeignet, deswegen nennen wir diese Prozedur *avorl* und entwickeln sie später weiter zu unserer gewünschten Prozedur

```

> avorl := proc(n)
    if n=1 then return 1; end if;
    if n=2 then return 1; end if;
    if n>=3 then
        procname(n-1)+procname(n-2);
    end if;
end proc;
avorl := proc(n)
    if n = 1 then return 1 end if;
    if n = 2 then return 1 end if;
    if 3 ≤ n then procname(n - 1) + procname(n - 2) end if
end proc

```

Ein kleiner Test, der zeigt, dass die Prozedur korrekt arbeitet:

```
> avorl(15);
```

610

Nun messen wir die Zeit, die die Prozedur benötigt:

```
> st:=time():
  avorl(30);
  time()-st;
```

832040

18.116

```
> st:=time():
  timelimit( 600, avorl(40) );
  time()-st;
Error, (in avorl) time expired
```

600.013

Die Prozedur ist noch nicht schnell genug, für  $n=40$  braucht sie bereits so lange (auf jeden Fall über 10 Minuten), dass wir sie abgebrochen haben. Aus diesem Grund bauen wir das MAPLE-Kommando "**option remember**" zur Beschleunigung in unsere Prozedur ein.

```
> a:=proc(n)
  option remember;
  if n=1 then return 1; end if;
  if n=2 then return 1; end if;
  if n>=3 then
    procname(n-1)+procname(n-2);
  end if;
end proc;
```

**a := proc(n)**

**option remember;**

**if n = 1 then return 1 end if;**

**if n = 2 then return 1 end if;**

**if 3 ≤ n then procname(n - 1) + procname(n - 2) end if**

**end proc**

Noch einmal ein Test, ob die Prozedur funktioniert:

```
> a(15);
```

610

Jetzt überprüfen wir noch die benötigte Zeit der Prozedur:

```
> st:=time():
  a(30);
  time()-st;
```

832040

0.

```
> st:=time():
  a(999);
  time()-st;
```

2686381002448535938614672720214292396761660931898695234012317\  
5997617981700247881689338369654483356564191827856161443356312\  
9766736422103503246348504103776803673341511728991697231970827\  
63985615764450078474174626

0.020

Wir sehen: Nun dauert selbst die Berechnung der Kantenlänge des 999-ten Quadrates nur noch den Bruchteil einer Sekunde. Diese Prozedur scheint uns geeignet, und wir halten sie gleich für  $n \geq 3$  fest, da sie auch schon für  $n=3$  gilt:

```
a:=proc(n)  
option remember;  
if n=1 then return 1; end if;  
if n=2 then return 1; end if;  
if n>=3 then  
  procname(n-1)+procname(n-2);  
end if;  
end proc;
```

– (iii) *Untersuche das Verhältnis  $\frac{a_{n+1}}{a_n}$  mit wachsendem  $n$ .*

[ Zur Lösung dieser Aufgabe verwenden wir unsere oben programmierte Prozedur a.

Nun lassen wir uns die Quotienten  $\frac{a_{n+1}}{a_n}$  für  $n=1..50$  als Fließkommazahl (Dezimalbruch

mit dem MAPLE-Befehl "evalf") ausgeben. Damit wir erkennen, welchem  $n$  sie zugeordnet sind, geben wir diesem Quotienten die Bezeichnung "quot(n)". Als Dezimalbruch lassen wir sie uns ausgeben, um sie besser vergleichen zu können.

```
> for n from 1 to 50 do  
  quot[n]:=evalf(a(n+1)/a(n));  
end do;
```

$quot_1 := 1.$

$quot_2 := 2.$

$quot_3 := 1.500000000$

$quot_4 := 1.666666667$

$quot_5 := 1.600000000$

$quot_6 := 1.625000000$

$quot_7 := 1.615384615$   
 $quot_8 := 1.619047619$   
 $quot_9 := 1.617647059$   
 $quot_{10} := 1.618181818$   
 $quot_{11} := 1.617977528$   
 $quot_{12} := 1.618055556$   
 $quot_{13} := 1.618025751$   
 $quot_{14} := 1.618037135$   
 $quot_{15} := 1.618032787$   
 $quot_{16} := 1.618034448$   
 $quot_{17} := 1.618033813$   
 $quot_{18} := 1.618034056$   
 $quot_{19} := 1.618033963$   
 $quot_{20} := 1.618033999$   
 $quot_{21} := 1.618033985$   
 $quot_{22} := 1.618033990$   
 $quot_{23} := 1.618033988$   
 $quot_{24} := 1.618033989$   
 $quot_{25} := 1.618033989$   
 $quot_{26} := 1.618033989$   
 $quot_{27} := 1.618033989$   
 $quot_{28} := 1.618033989$   
 $quot_{29} := 1.618033989$   
 $quot_{30} := 1.618033989$   
 $quot_{31} := 1.618033989$   
 $quot_{32} := 1.618033989$   
 $quot_{33} := 1.618033989$   
 $quot_{34} := 1.618033989$   
 $quot_{35} := 1.618033989$   
 $quot_{36} := 1.618033989$   
 $quot_{37} := 1.618033989$   
 $quot_{38} := 1.618033989$   
 $quot_{39} := 1.618033989$   
 $quot_{40} := 1.618033989$

$quot_{41} := 1.618033989$   
 $quot_{42} := 1.618033989$   
 $quot_{43} := 1.618033989$   
 $quot_{44} := 1.618033989$   
 $quot_{45} := 1.618033989$   
 $quot_{46} := 1.618033989$   
 $quot_{47} := 1.618033989$   
 $quot_{48} := 1.618033989$   
 $quot_{49} := 1.618033989$   
 $quot_{50} := 1.618033989$

Wir können erkennen, dass sich das Verhältnis der Kantenlängen zweier aufeinanderfolgenden Quadrate ab dem 24-ten bis zum 50-ten Quadrat bis mindestens zur 9-ten Stelle nach dem Komma nicht mehr ändert. Um diese These zu erhärten, lass wir uns auch noch die Quotienten ab dem 985-ten bis zum 1000-ten ausgeben.

```
> for n from 985 to 1000 do  
    quot[n]:=evalf(a(n+1)/a(n));  
end do;
```

$quot_{985} := 1.618033989$   
 $quot_{986} := 1.618033989$   
 $quot_{987} := 1.618033989$   
 $quot_{988} := 1.618033989$   
 $quot_{989} := 1.618033989$   
 $quot_{990} := 1.618033989$   
 $quot_{991} := 1.618033989$   
 $quot_{992} := 1.618033989$   
 $quot_{993} := 1.618033989$   
 $quot_{994} := 1.618033989$   
 $quot_{995} := 1.618033989$   
 $quot_{996} := 1.618033989$   
 $quot_{997} := 1.618033989$   
 $quot_{998} := 1.618033989$   
 $quot_{999} := 1.618033989$   
 $quot_{1000} := 1.618033989$

Auch hier lässt sich keine Veränderung feststellen. Als letztes untersuchen wir noch die Quadrate 1000 bis 10000 in tausender Schritten.

```
> for n from 1000 to 10000 by 1000 do
  quot[n]:=evalf(a(n+1)/a(n));
end do;
```

$quot_{1000} := 1.618033989$

$quot_{2000} := 1.618033989$

$quot_{3000} := 1.618033989$

$quot_{4000} := 1.618033989$

$quot_{5000} := 1.618033989$

$quot_{6000} := 1.618033989$

$quot_{7000} := 1.618033989$

$quot_{8000} := 1.618033989$

$quot_{9000} := 1.618033989$

$quot_{10000} := 1.618033989$

Da sich auch hier nichts mehr an dem Quotienten ändert, können wir feststellen,

dass der Quotient  $\frac{a_{n+1}}{a_n}$  gegen 1,618033989 konvergiert!

Nun lassen wir uns die Zahl des Goldenen Schnittes, nämlich  $\frac{1+\sqrt{5}}{2}$ , als

Fließkommazahl ausgeben:

```
> evalf((1+sqrt(5))/2);
```

1.618033988

Der Grenzwert des Quotienten  $\frac{a_{n+1}}{a_n}$  stimmt also bis zur achten Nachkommastelle mit

der Zahl  $\frac{1+\sqrt{5}}{2}$  vom Goldenen Schnitt überein! (Diese Zahl wird uns später noch einmal begegnen!) Damit können wir folgenden Antwortsatz formulieren:

**Antwort: Der Quotient  $\frac{a_{n+1}}{a_n}$  konvergiert mit wachsendem n (also  $n \rightarrow \infty$ ) gegen 1,618033989 - das ist ziemlich genau  $\frac{1+\sqrt{5}}{2}$  !**

– (iv) *Finde eine parametrisierte Kurve, die jeweils durch diejenige der beiden äußeren Ecken eines Quadrates unserer Konstruktion verläuft, wo nächste Quadrat angesetzt wird.*

```
> with(plots):
```

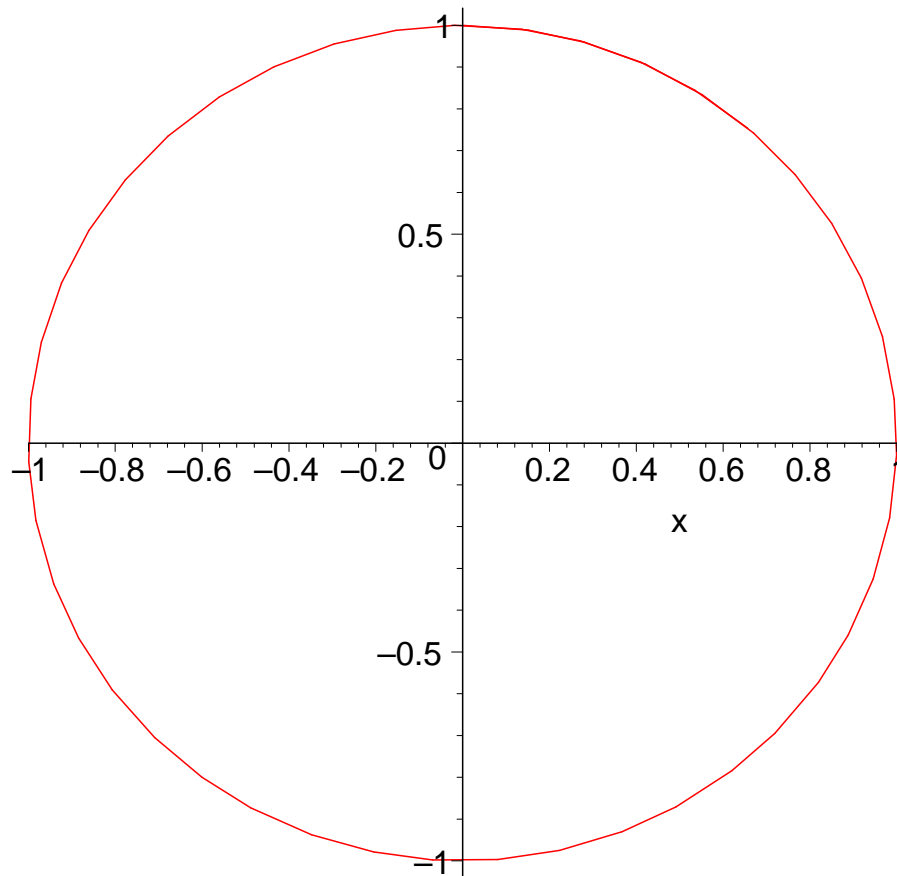
```
with(plottools):  
with(LinearAlgebra):  
Warning, the name changecoords has been redefined  
Warning, the name arrow has been redefined
```

Als erstes schauen wir nach, wie man in Maple eine parametrisierte Kurve zeichnen kann:

```
> # ?parametric
```

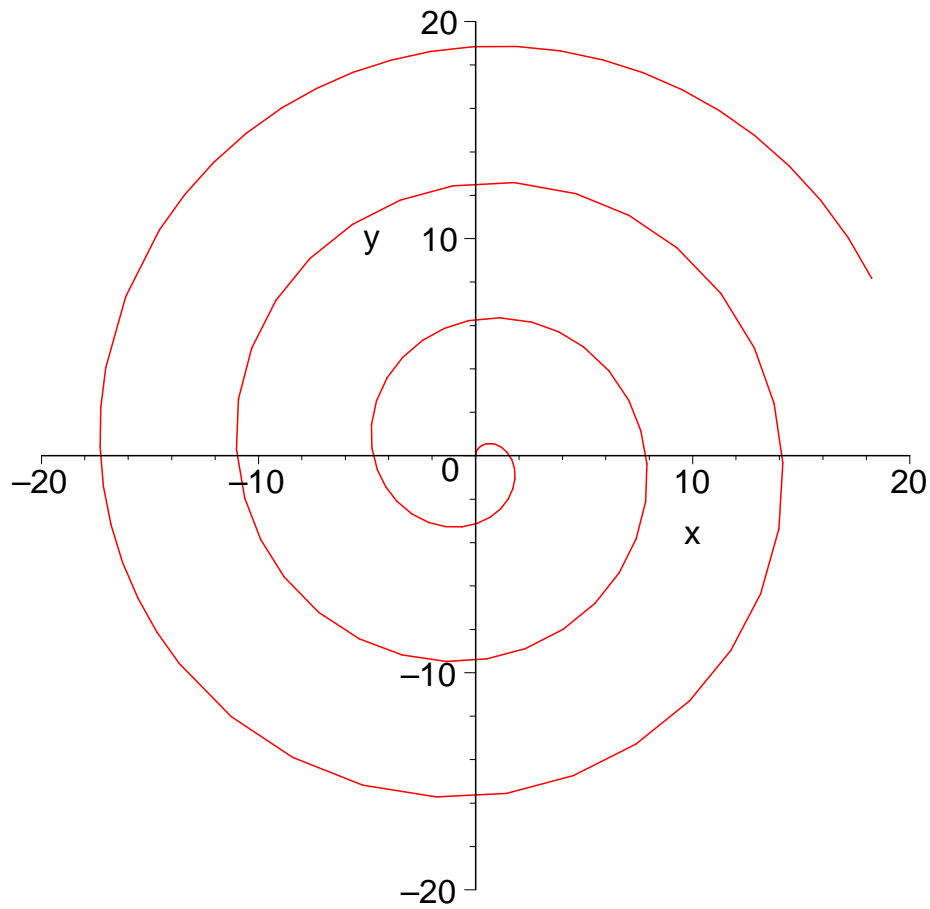
Wir tasten uns langsam an die Kurve heran und zeichnen zunächst einen Kreis:

```
> plot([sin(t),cos(t),t=0..7],x=-1..1);
```



Jetzt kommen wir der Spirale schon ein ganzes Stück näher!

```
> plot([t*sin(t),t*cos(t),t=0..20],x=-20..20,y=-20..20);
```



Um die Spirale richtig zeichnen zu können, benötigen wir Viertelkreise, aus denen sich dann die Spirale zusammensetzt. Dabei sieht der Viertelkreis, je nachdem in welchem Quadranten wir uns befinden, jeweils anders aus: Die Prozedur "Viertelkreis" ist abhängig von dem Mittelpunkt, dem Radius für das jeweilige Quadrat und dem Quadranten.

```
> Viertelkreis:=proc(m,r,q)
    plot([m[1]+r*cos(t),m[2]+r*sin(t),
    t=Pi-q*Pi/2..Pi/2-q*Pi/2], color=black,thickness=2);
end proc;
```

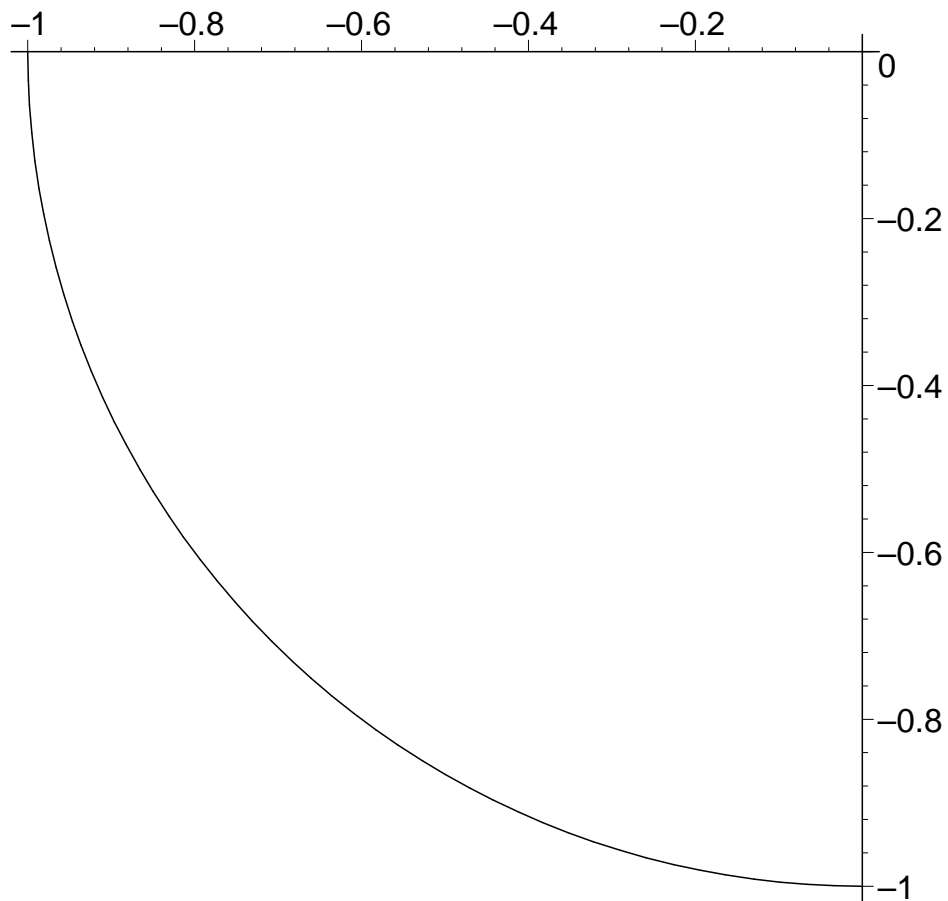
```
Viertelkreis := proc(m, r, q)
```

```
    plot([m[1]+r*cos(t), m[2]+r*sin(t), t = pi - 1/2*q*pi .. pi/2 - 1/2*q*pi],
    color = black, thickness = 2)
```

```
end proc
```

Wir testen unserer Prozedur "Viertelkreis" im 3. Quadranten:

```
> Viertelkreis([0,0],1,3);
```



Jetzt können wir eine Prozedur "zeichne" schreiben, die die gewünschte Spirale zeichnet. Wie schon gesagt, besteht die Spirale aus vielen einzelnen Viertelkreisen. Dabei muss der Mittelpunkt für jeden neuen Viertelkreis entsprechend verschoben werden:

```
> zeichne:=proc(n)
  local M,p,r,i,q;

  p := table():
  r := table():
  M := [0,0]:

  for i from 0 to n do
    q := ((i-1) mod 4) + 1;
    p[i] := Viertelkreis(M,a(i+2),q);

    if q = 4 then
      r[i] :=
rectangle(M,M+[-a(i+2),a(i+2)],color=COLOR(HUE,
i/(n+1)));
      M[2] := M[2] - a(i+1);

    elif q = 1 then
      r[i] :=
```

```
rectangle(M,M+[a(i+2),a(i+2)],color=COLOR(HUE,
i/(n+1)));
    M[1] := M[1] - a(i+1);

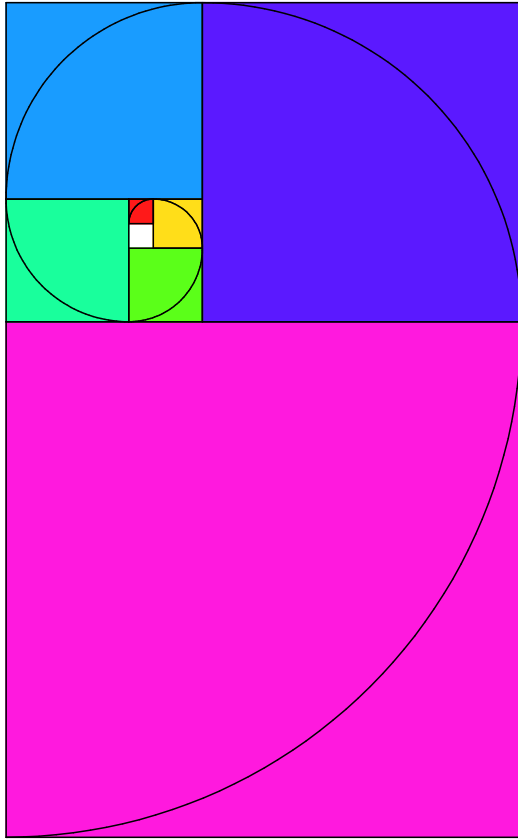
    elif q = 2 then
        r[i] :=
rectangle(M,M+[a(i+2),-a(i+2)],color=COLOR(HUE,
i/(n+1)));
        M[2] := M[2] + a(i+1);

    else
        r[i] :=
rectangle(M,M+[-a(i+2),-a(i+2)],color=COLOR(HUE,
i/(n+1)));
        M[1] := M[1] + a(i+1);

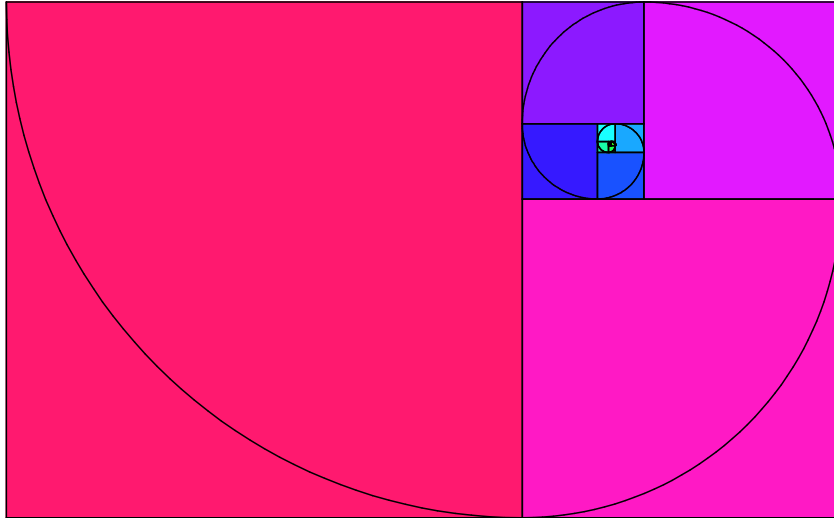
    end if;
end do;
display([op(convert(r,'list')),op(convert(p,'list'))],sc
aling=constrained,axes=none);
end proc;
```

Wir gucken, ob die Spirale tatsächlich passt:

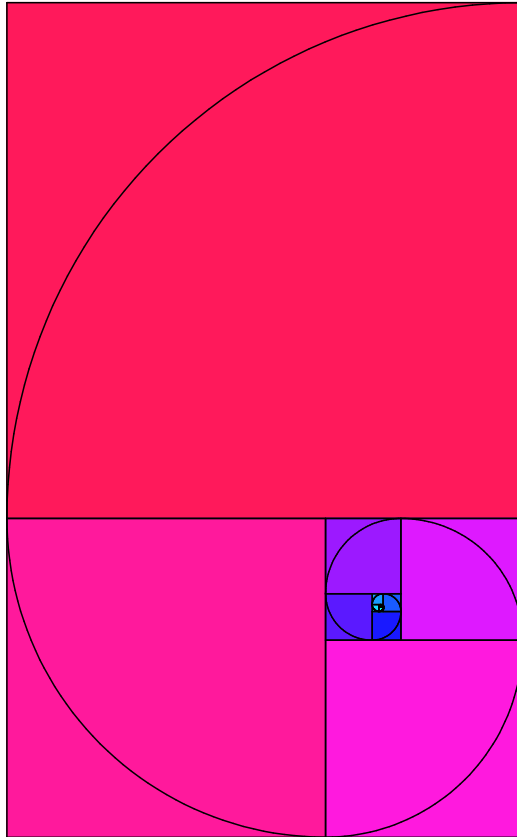
```
> zeichne(6);
```



```
> zeichne(15);
```



```
> zeichne(20);
```



[ Die Spirale hat also genau die gewünschte Form!

– (v) *Schreibe die Rekursionsformel um  $b_n := \langle a_{n-1} / a_n \rangle$ .*

```
[ > restart;
  with(LinearAlgebra):
  > a:=proc(n)
    option remember;
    if n=1 then return 1; end if;
    if n=2 then return 1; end if;
    if n>=3 then
      procname(n-1)+procname(n-2);
    end if;
  end proc;
a := proc(n)
option remember;
  if n = 1 then return 1 end if;
  if n = 2 then return 1 end if;
  if 3 ≤ n then procname(n - 1) + procname(n - 2) end if
end proc
```

Wir suchen eine Matrix A, so dass  $b_n = A b_{n-1}$ , weil es sich um eine lineare Abbildung handelt:

Exkurs an der Tafel -> Matrix A

```
> A:=Matrix([[0,1],[1,1]]);
```

$$A := \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Wir überprüfen allgemein, ob die Matrix A richtig gewählt ist und berechnen  $b_n$ :

```
> n:='n';
```

```
n := n
```

```
> A.<a[n-1],a[n]>;
```

$$\begin{bmatrix} a_n \\ a_{n-1} + a_n \end{bmatrix}$$

Das passt, weil der Ergebnisvektor oben nach der Rekursionsformel für  $a_n$  gerade der Vektor  $\langle a_n | a_{n+1} \rangle = b_n$  ist! Also gilt  $A b_{n-1} = b_n$ !!

Zur nochmaligen "experimentellen" Überprüfung,  $b_n = A b_{n-1}$  gilt, berechnen wir  $b_n$  für  $n=3..11$  nach der Definition und vergleichen das Ergebnis mit dem Ergebnis, das unsere Formel liefert. Dabei können wir erst mit  $b_2$  starten, da  $a_0$  und somit  $b_1$  nicht definiert ist!

```
> for i from 3 to 11 do
  b[i-1]:=<a(i-2),a(i-1)>;
  b[i]:=<a(i-1),a(i)>, A.b[i-1];
end do;
```

$$b_2 := \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$b_3 := \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$b_3 := \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$b_4 := \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$b_4 := \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$b_5 := \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

$$b_5 := \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

$$b_6 := \begin{bmatrix} 5 \\ 8 \end{bmatrix}, \begin{bmatrix} 5 \\ 8 \end{bmatrix}$$

$$b_6 := \begin{bmatrix} 5 \\ 8 \end{bmatrix}$$

$$b_7 := \begin{bmatrix} 8 \\ 13 \end{bmatrix}, \begin{bmatrix} 8 \\ 13 \end{bmatrix}$$

$$b_7 := \begin{bmatrix} 8 \\ 13 \end{bmatrix}$$

$$b_8 := \begin{bmatrix} 13 \\ 21 \end{bmatrix}, \begin{bmatrix} 13 \\ 21 \end{bmatrix}$$

$$b_8 := \begin{bmatrix} 13 \\ 21 \end{bmatrix}$$

$$b_9 := \begin{bmatrix} 21 \\ 34 \end{bmatrix}, \begin{bmatrix} 21 \\ 34 \end{bmatrix}$$

$$b_9 := \begin{bmatrix} 21 \\ 34 \end{bmatrix}$$

$$b_{10} := \begin{bmatrix} 34 \\ 55 \end{bmatrix}, \begin{bmatrix} 34 \\ 55 \end{bmatrix}$$

$$b_{10} := \begin{bmatrix} 34 \\ 55 \end{bmatrix}$$

$$b_{11} := \begin{bmatrix} 55 \\ 89 \end{bmatrix}, \begin{bmatrix} 55 \\ 89 \end{bmatrix}$$

Wir erkennen, dass die  $a_n$  mit dieser Formel richtig berechnet werden und halten sie deshalb fest:

**Die gesuchte Rekursionsformel ist**  $b_n = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} * b_{n-1}$ .

Dazu schreiben wir noch eine kleine Prozedur, um die  $b_n$  sofort berechnen zu können:

```
> b:=proc(n)
  option remember;
  if n=1 then print("b[1] ist nicht definiert!"); end
  if;
  if n=2 then return <1,1>; end if;
  if n>=3 then return A.b(n-1); end if;
```

```

end proc;
b := proc(n)
option remember;
  if n = 1 then print("b[1] ist nicht definiert!") end if;
  if n = 2 then return <1, 1> end if;
  if 3 ≤ n then return A . (b(n - 1)) end if
end proc

```

Wieder ein kleiner Test der Prozedur, der zeigt, dass sie funktioniert:

```

> for i from 1 to 7 do
  b(i);
end do;

```

"b[1] ist nicht definiert!"

```

[ 1 ]
[ 1 ]
[ 1 ]
[ 2 ]
[ 2 ]
[ 3 ]
[ 3 ]
[ 5 ]
[ 5 ]
[ 8 ]
[ 8 ]
[13 ]

```

Und wir prüfen, wie schnell die Prozedur ist:

```

> st:=time():
  b(400);
  time() -st;

```

```

[108788617463475645289761992289049744844995705477812699099751\
202749393926359816304226]
[176023680645013966468226945392411250770384383304492191886725\
992896575345044216019675]

```

2.884

```

> st:=time():
  b(800);
  time() -st;

```

```

[428192994374323026176320535226795887595441254172350710190250\
6901195098209974588835496480994146847025305900315860745966290\
7055750510443512830937550167717484757372564701]
[692830818642247171362900776813285182733991243852048207189660\

```

**Alternativ kann man für  $b_n$  eine weitere Rekursionsformel angeben, die wie die Rekursionsformel für  $a_n$  die Rekursionstiefe 2 besitzt. Diese Formel ist zwar nicht so schön, funktioniert aber auch, so dass wir sie der Vollständigkeit halber präsentieren:**

Zunächst berechnen wir die ersten 10  $b_n$ , wobei wir erst bei  $b_2$  beginnen können, da  $a_0$  und somit  $b_1$  nicht definiert sind:

```
> for i from 2 to 11 do  
    b[i] := <a(i-1), a(i)>;  
end do;
```

$$b_2 := \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$b_3 := \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$b_4 := \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$b_5 := \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

$$b_6 := \begin{bmatrix} 5 \\ 8 \end{bmatrix}$$

$$b_7 := \begin{bmatrix} 8 \\ 13 \end{bmatrix}$$

$$b_8 := \begin{bmatrix} 13 \\ 21 \end{bmatrix}$$

$$b_9 := \begin{bmatrix} 21 \\ 34 \end{bmatrix}$$

$$b_{10} := \begin{bmatrix} 34 \\ 55 \end{bmatrix}$$

$$b_{11} := \begin{bmatrix} 55 \\ 89 \end{bmatrix}$$

Durch genaues Hinsehen erkennen wir, dass  $b_n = b_{n-1} + b_{n-2}$  gilt, was sich auch durch eine einfache Schlußfolgerung aus unserer Rekursionsformel für  $b_n$  ergibt, denn:  $b_n = \langle a_{n-1} | a_n \rangle = \langle a_{n-2} + a_{n-3} | a_{n-1} + a_{n-2} \rangle = \langle a_{n-2} | a_{n-1} \rangle + \langle a_{n-3} | a_{n-2} \rangle = b_{n-1} + b_{n-2}$ . Damit können wir auch hier für  $b_n$  eine Prozedur schreiben, die wir *alternativb* nennen.

```
> alternativb:=proc(n)
  option remember;
  if n<=1 then print("b[1] ist nicht definiert!"); end
  if;
  if n=2 then return <1,1>; end if;
  if n=3 then return <1,2>; end if;
  if n>=4 then return procname(n-1)+procname(n-2); end
  if;
end proc;
```

*alternativb* := **proc**(n)

**option** remember;

**if** n ≤ 1 **then** print("b[1] ist nicht definiert!") **end if**;

**if** n = 2 **then** return <1, 1> **end if**;

**if** n = 3 **then** return <1, 2> **end if**;

**if** 4 ≤ n **then** return procname(n - 1) + procname(n - 2) **end if**

**end proc**

Ein Test, ob die Prozedur richtig läuft:

```
> alternativb(1);
alternativb(2);
alternativb(8);
alternativb(100);
```

"b[1] ist nicht definiert!"

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

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

$$\begin{bmatrix} 218922995834555169026 \\ 354224848179261915075 \end{bmatrix}$$

Und auch hier messen wir die Laufzeit:

```
> st:=time();
alternativb(400);
time() -st;
```

[108788617463475645289761992289049744844995705477812699099751\

202749393926359816304226]

[176023680645013966468226945392411250770384383304492191886725\

992896575345044216019675]

```

> st:=time():
alternativb(800);
time() -st;
[428192994374323026176320535226795887595441254172350710190250\
6901195098209974588835496480994146847025305900315860745966290\
7055750510443512830937550167717484757372564701]
[692830818642247171362900776813285182733991243852048207189660\
4059769143558727838311227716196753253067537417085740474301762\
3467220361778016172106855838975759985190398725]
3.135

```

Diese Prozedur ist auch langsamer als die erste, d.h. die zugehörige Rekursionsformel ist nur die 2. Wahl!

**Antwort: Wir erhalten die alternative Rekursionsformel:**

$$b_n = b_{n-1} + b_{n-2}$$

## Aufgabe P5.2 (Verfeinerungen)

(i) Bestimme  $J, Q := \text{JordanForm}(\langle\langle 0, 1 \rangle | \langle 1, 1 \rangle \rangle, \text{output}=['J', 'Q'])$ .

```

> restart:
with(LinearAlgebra):
> J, Q := JordanForm ( <<0,1> | <1,1>>, output= ['J', 'Q'] );

```

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

(ii) Was tut **JordanForm**? (Frag Maple!)

JordanForm liefert die Jordanform  $J$  der angegebenen Matrix, in unserem Fall der Matrix  $A = \langle\langle 0 | 1 \rangle | \langle 1, 1 \rangle \rangle$ . Die Matrix  $J$  ist äquivalent zu der Matrix  $A$ , d.h. sie beschreibt die gleiche lineare Abbildung wie  $A$ . Die Matrix  $J$  besteht aus den sogenannten Jordanblöcken und ist bis auf die Reihenfolge dieser Jordanblöcke eindeutig bestimmt. Auf der Hauptdiagonalen von  $J$  stehen die Eigenwerte von  $A$ , in unserem

Fall  $\frac{1}{2} + \frac{\sqrt{5}}{2}$  und  $\frac{1}{2} - \frac{\sqrt{5}}{2}$ . Die zweite Matrix  $Q$ , die JordanForm wegen der

Anweisung in der *output*-Option mitliefert, ist die sogenannte Transformationsmatrix, d.h. die Matrix, mit der man die Matrix  $A$  auf die Jordanform  $J$  bringt, d.h.

$$J = Q^{(-1)} A Q. \text{ Also gilt: } Q J Q^{(-1)} = A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

(iii) Prüfe  $Q J Q^{(-1)} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$

```

> (Q^(-1));

```

$$\begin{bmatrix} 1 & \frac{1}{2} + \frac{\sqrt{5}}{2} \\ 1 & \frac{1}{2} - \frac{\sqrt{5}}{2} \end{bmatrix}$$

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

$$\begin{bmatrix} \frac{(-1+\sqrt{5})\sqrt{5}\left(\frac{1}{2}+\frac{\sqrt{5}}{2}\right)}{10} + \frac{(1+\sqrt{5})\sqrt{5}\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)}{10}, \\ \frac{(-1+\sqrt{5})\sqrt{5}\left(\frac{1}{2}+\frac{\sqrt{5}}{2}\right)^2}{10} + \frac{(1+\sqrt{5})\sqrt{5}\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)^2}{10} \\ \left[ \frac{\left(\frac{1}{2}+\frac{\sqrt{5}}{2}\right)\sqrt{5}}{5} - \frac{\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)\sqrt{5}}{5}, \frac{\left(\frac{1}{2}+\frac{\sqrt{5}}{2}\right)\sqrt{5}}{5} - \frac{\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)\sqrt{5}}{5} \right] \end{bmatrix}$$

> simplify(%);

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

> A:=(%);

$$A := \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

**Wir sehen also, dass Maple uns tatsächlich die richtigen Ergebnisse geliefert hat!**

**(iv) Finde eine geschlossene Formel für  $n$ .**

Zunächst erinnern wir uns an die Rekursionsformel für  $n$  und die zugehörige Prozedur:

```
> a:=proc(n)
  option remember;
  if n=1 then return 1; end if;
  if n=2 then return 1; end if;
  if n>=3 then
    procname(n-1)+procname(n-2);
  end if;
end proc;

a := proc(n)
  option remember;
  if n = 1 then return 1 end if;
  if n = 2 then return 1 end if;
  if 3 ≤ n then procname(n - 1) + procname(n - 2) end if
end proc

> for i from 1 to 8 do
  i, A^i;
```

```
end do;
```

```
1, [ 0  1 ]  
   [ 1  1 ]  
2, [ 1  1 ]  
   [ 1  2 ]  
3, [ 1  2 ]  
   [ 2  3 ]  
4, [ 2  3 ]  
   [ 3  5 ]  
5, [ 3  5 ]  
   [ 5  8 ]  
6, [ 5  8 ]  
   [ 8 13 ]  
7, [ 8 13 ]  
   [13 21 ]  
8, [13 21 ]  
   [21 34 ]
```

Wenn man die Matrix A immer wieder potenziert, fällt auf, dass in zweiten Spalte der potenzierten Matrix zwei aufeinanderfolgende Kantenlängen stehen, und zwar wie folgt: In der Matrix  $A^n$  steht an der Stelle 1,2 das Element  $a_n$  und an der Stelle 2,2 das Element  $a_{n+1}$ . (An 1,1 steht  $a_{n-1}$  und an 1,2 steht noch einmal  $a_n$ ).

Um diese Vermutung zu bestätigen, berechnen wir für  $i=10..17$  einmal mit Hilfe unserer Prozedur und einmal wie gerade beschrieben:

```
> for n from 10 to 17 do  
  (A^n)[1,2], a(n);  
end do;
```

```
55, 55  
89, 89  
144, 144  
233, 233  
377, 377  
610, 610  
987, 987  
1597, 1597
```

**Super!! Die Ergebnisse stimmen überein!**

**Wir sehen also, dass man  $a_n$  erhält, indem man  $A^n$  berechnet und dann**

den Eintrag an der Stelle 1,2 nimmt! Also kommen wir an eine geschlossene Formel für  $a_n$ , indem wir  $A^n$  berechnen und dann den Eintrag an der Stelle 1,2 betrachten!

Da  $A = Q J Q^{(-1)}$  gilt, folgt:  $A^n = (Q J Q^{(-1)})^n = Q J^n Q^{(-1)}$ . Und da  $J$  eine Diagonalmatrix ist, erhält man  $J^n$ , indem man die Diagonaleinträge, also  $J_{1,1}$  und  $J_{2,2}$  jeweils mit  $n$  potenziert und an den Stellen 1,2 und 2,1 jeweils die Null stehen läßt:

```
> n := 'n' ;
```

```
> (Q.<<J[1,1]^n,0>|<0,J[2,2]^n>>.Q^(-1));
```

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

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

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

```
> simplify(%);
a[n]:= %[1,2];
```

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

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

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

$$a_n := \frac{\sqrt{5} \left( \frac{1+\sqrt{5}}{2} \right)^n}{5} - \frac{\sqrt{5} \left( \frac{1-\sqrt{5}}{2} \right)^n}{5}$$

Der Eintrag von  $A^n$  an der Stelle 1,2 liefert uns  $a_n$  und damit die geschlossene Formel für  $a_n$ . Sie lautet also:

$$a_n = \frac{1 \left( \left( \frac{1+\sqrt{5}}{2} \right)^n - \left( \frac{1-\sqrt{5}}{2} \right)^n \right)}{\sqrt{5}}$$

(v) **Schreibe eine zweite Prozedur `a2:=proc(n)...`, welche  $a_n$  mit Hilfe der obigen geschlossenen Formel berechnet.**

Mit Hilfe der gerade gefundenen geschlossenen Formel für  $a_n$  können wir jetzt sofort die zugehörige Prozedur schreiben. Sie besteht eigentlich nur aus der geschlossenen Formel. Der *simplify*-Befehl liefert gleich schöne Ergebnisse, und *option remember*-Befehl beschleunigt die Prozedur `a2` ebenso wie die Prozedur `a`.

```
> a2:=proc(n)
  option remember;
  simplify( (1/sqrt(5))* ( ((1+sqrt(5))/2)^n -
    ((1-sqrt(5))/2)^n ) );
end proc;
```

```
a2 := proc(n)
```

```
option remember;
```

```
simplify(((1/2 + 1/2*sqrt(5))^n - (1/2 - 1/2*sqrt(5))^n) / sqrt(5))
```

```
end proc
```

Wir prüfen diese Prozedur einmal an den ersten 15 Gliedern:

```
> for i from 0 to 15 do
  a2(i);
end do;
```

0

1

1

2

3

5

8  
13  
21  
34  
55  
89  
144  
233  
377  
610

*Wir sehen, dass diese Prozedur tatsächlich die  $a_n$  berechnet!*

(vi) **Vergleiche die Laufzeiten von a und a2.**

```
[ > restart;
> a:=proc(n)
  option remember;
  if n=1 then return 1; end if;
  if n=2 then return 1; end if;
  if n>=3 then
    procname(n-1)+procname(n-2);
  end if;
end proc;
a := proc(n)
  option remember;
  if n = 1 then return 1 end if;
  if n = 2 then return 1 end if;
  if 3 ≤ n then procname(n - 1) + procname(n - 2) end if
end proc
> a2:=proc(n)
  # option remember;
  simplify( (1/sqrt(5))* ( ((1+sqrt(5))/2)^n -
    ((1-sqrt(5))/2)^n ) );
end proc;
a2 := proc(n)
  simplify(((1 / 2 + 1 / 2*sqrt(5))^n - (1 / 2 - 1 / 2*sqrt(5))^n) / sqrt(5))
end proc
> st:=time():
a(20);
time()-st;

st1:=time():
a2(20);
time()-st1;

6765
0.
6765
```

```

0.
> st:=time():
a(500);
time()-st;

st1:=time():
a2(500);
time()-st1;
1394232245616978801397243828704072839500702565876973072641089\
62948325571622863290691557658876222521294125
0.010
1394232245616978801397243828704072839500702565876973072641089\
62948325571622863290691557658876222521294125
0.671
> st:=time():
a(2000);
time()-st;

st1:=time():
a2(2000);
time()-st1;
422469633392304878706725602341482782579852840250681098010280\
1373143085843701307072241235996391415110884460875389096036076\
4019471164359602927198331259873732625355580260699158591522949\
2453904998722256795316982874482472992263901833716778060607011\
6154978867198798583114688708762645973690867228840236544222952\
4334796448013951534956297208765265606952980649984197744872015\
5612802665404554171717881930324025204312082516817125
0.130
422469633392304878706725602341482782579852840250681098010280\
1373143085843701307072241235996391415110884460875389096036076\
4019471164359602927198331259873732625355580260699158591522949\
2453904998722256795316982874482472992263901833716778060607011\
6154978867198798583114688708762645973690867228840236544222952\
4334796448013951534956297208765265606952980649984197744872015\
5612802665404554171717881930324025204312082516817125
15.061

```

**Antwort: Die Laufzeit der Prozedur a2 ist deutlich größer als die der Prozedur a. Also ist es schneller,  $a_n$  rekursiv zu berechnen als über die geschlossene Formel!**