Die Fibonacci-Zahlen

1 Rekursion versus Iteration
Am Beispiel der Fibonacci-Zahlen

1.1 Vorbemerkung: Darstellungsformen für grosse ganze Zahlen (optional)

```
→ 100!;
(%o1)
933262154439441526816992388562[98 digits]916864000000000000000000000000
```

```
→ set_display(ascii);
(%o2)                         ascii
```

```
→ 100!;
(%o3) 9332621544394415268169923885626670049071596826438162146859296389521759999
93229915608941463976156518286253697920827223758251185210916864000000000000000000
00000000
```

1.2 Definition der rekursiven und der iterative Version der Fibonacci-Zahlen

Im folgenden Beispiel wird (zum Zwecke der Fallstudie) die Großschreibung Fib verwendet,
denn fib ist inzwischen eine Standardfunktion von Maxima.
In Maxima wird zwischen Gross- und Kleinschreibung unterschieden.

```
→ Fib(n) :=                          /* rekursiv */
   if n<=1 then n else Fib(n-1)+Fib(n-2) $ ;
```

```
→ Fib(20);
(%o2) 6765
```

```
→ Fib_it(n) :=                       /* iterativ */
   block([i : 0, f0 : 0, f1 : 1, f2 : 1],
     while i < n do
       (i : i+1, f0 : f1, f1 : f2, f2 : f1 + f0),
     f0 ) $;
```

```
→ Fib_it(20);
(%o4) 6765
```

```
→ Fib_it(100);
(%o5) 354224848179261915075
```

```
→ Fib_it(1000);
(%o6)
434665576869374564356885276750[149 Ziffern]516003704476137795166849228875
```

1.3 Kleiner Exkurs zur string-Verarbeitung (ergänzend)

```
→ Fib_it(50);
(%o7) 12586269025
```

Der Aufruf stringp(x) gibt als Funktionswert true oder false zurück, je nachdem ob x ein string ist oder nicht.

```
→ stringp(%)        /*  % = letzte Ausgabe  */ ;
(%o8) false
```

Das vorige Ergebnis war false, denn es war eine ganze Zahl und kein string.
Der Aufruf string(x) macht aus x einen string.

```
→ string(12586269025);
(%o9) 12586269025

→ stringp(%);
(%o10) true
```

Die Funktion slength ermittelt die Länge eines strings.

```
→ slength(string(12586269025))        /* string-length */;
(%o11) 11

→ slength(string(Fib_it(50)))        /* Funktions-Verschachtelung */ ;
(%o12) 11
```

2 *Laufzeitbetrachtungen*

2.1 System Tools (evtl. später)

```
→ functions;
(%o13) [Fib(n),Fib_it(n)]

→ timer(all);
(%o14) [Fib,Fib_it]

→ timer;
(%o15) [Fib_it,Fib]

→ Fib(30);
(%o16) 832040
```

```
➔ Fib_it(3000);
(%o17)
410615886307971260333568378719[567 Ziffern]658692285968043243656709796000
```

```
➔ timer_info();
```

(%o18)

function	time//call	calls	runtime	gctime
Fib_it	0.015 *sec*	1	0.015 *sec*	0
Fib	$9.913067118483424\ 10^{-5}$ *sec*	2692537	266.913 *sec*	0
total	$9.913620532003633\ 10^{-5}$ *sec*	2692538	266.928 *sec*	0

```
➔ timedate ();
(%o19) 2020−05−11 15:32:31+02:00
```

2.2 Berechnung der Fibonacci-Zahlen durch Anlegen einer Tabelle

```
➔ Fib_tab[n] := if n<=1 then n else Fib_tab[n-1]+Fib_tab[n-2] $ ;
```

```
➔ Fib_tab[20];
(%o22) 6765
```

```
➔ Fib_tab[100];
(%o23) 354224848179261915075
```

Die Version Fib_tab legt eine Tabelle an. (Man beachte die eckigen Klammern!)

Problem mit der Tabellen-Version: Gefahr des Speicherüberlaufs ("stack overflow").

```
➔ Fib_tab[1000];
(%o24)
434665576869374564356885276750[149 Ziffern]516003704476137795166849228875
```

```
➔ Fib_tab[10000];
Maxima encountered a Lisp error:
 Control stack exhausted (no more space for function call frames).
This is probably due to heavily nested or infinitely recursive function
calls, or a tail call that SBCL cannot or has not optimized away.
PROCEED WITH CAUTION.
Automatically continuing.
To enable the Lisp debugger set *debugger−hook* to nil.
```

2.3 Eine kleine Fallstudie zur Laufzeit

```
➔ showtime : true;
Evaluation took 0.0000 seconds (0.0000 elapsed) using 0 bytes.
(showtime) true
```

→ Fib(30) /* für (grobe) Zeitabschätzung */;

Evaluation took 12.8130 seconds (12.8090 elapsed) using 1499.595 MB.

(%o27) 832040

Neue Werte: 13 Sek statt 25 Sek ... einige Weltzeitalter weniger.

Eine konkrete (grobe) Messung ergibt: Fib(30) benötigt ca. 25 Sek
Extrapolation:
Vorbetrachtung: Fib(n) benötigt (weit) mehr Zeit als 2 * Fib(n-2).
===> Fib(32) benötigt mehr als 50 Sek = 2 * 25 Sek = 2^1 * 25 Sek
===> Fib(34) benötigt mehr als 100 Sek = 4 * 25 Sek = 2^2 * 25 Sek
===> Fib(36) benötigt mehr als 200 Sek = 8 * 25 Sek = 2^3 * 25 Sek
===> Fib(38) benötigt mehr als 400 Sek = 16 * 25 Sek = 2^4 * 25 Sek
===> Fib(40) benötigt mehr als 800 Sek = 32 * 25 Sek = 2^5 * 25 Sek
...
===> Fib(1000) benötigt mehr als 2^((1000-30)/2) * 25 Sek = mehr als 2^485 * 25 Sek

→ showtime: false;

(showtime) ***false***

→ (2^485)*25 /* Sekunden */;

(%o30)
2497398840252793785105277783 83[88 Ziffern]3163583805434306282450321408 00

In der "Gleitkomma"-Darstellung:

→ float((2^485)*25) /* Sekunden */;

(%o31) $2.497398840252794\ 10^{147}$

→ float((2^485)*25) / 60 /* Minuten */;

(%o32) $4.162331400421324\ 10^{145}$

→ float((2^485)*25) / (60*60) /* Stunden */;

(%o33) $6.937219000702204\ 10^{143}$

→ float((2^485)*25) / (60*60*24) /* Tage */;

(%o34) $2.890507916959252\ 10^{142}$

→ float((2^485)*25) / (60*60*24*365) /* Jahre */;

(%o35) $7.919199772491102\ 10^{139}$

→ float((2^485)*25) / (60*60*24*365*1000) /* Jahrtausende */;

(%o36) $7.919199772491101\ 10^{136}$

1 Weltzeitalter = Dauer der Existenz der bekannten Welt vom Urknall bis heute: ca. 20 Milliarden Jahre; d.h. 20*10^9 Jahre.
(vgl. Stephen W. Hawking: Eine kurze Geschichte der Zeit, Rowohlt Verlag 1991 (rororo Taschenbuch)

```
→ float((2^485)*25) / (60*60*24*365*(20*10^9))   /* Weltzeitalter */;
(%o37) 3.959599886245551 10^129
```

Algorithmen mit exponentieller Laufzeit sind nicht praktikabel (ausser für minimale Eingabewerte).

2.4 Standard-Darstellung: im folgenden

```
→ F(n) := Fib_it(n) $ ;

→ F(12);
(%o39) 144
```

2.5 Exakte Berechnung der Anzahl der rekursiven Aufrufe

```
→ AFib(n) := if n<2 then 1 else 1+AFib(n-1)+AFib(n-2) $ ;

→ AFib(0);
(%o41) 1

→ AFib(10);
(%o42) 177

→ makelist([n, F(n), AFib(n)], n, 0, 20);
(%o43) [[0,0,1],[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]]
```

3 Graphische Darstellungen

Beispiele: Siehe W. Haager (Grafiken mit Maxima) oder Ziegenbalg (Erste Hinweise ...zu Maxima)

3.1 Graphiken mit plot

```
→ Fib_Liste_Anfang(n) :=
    makelist( [i, F(i)], i, 0, n);
(%o44) Fib_Liste_Anfang(n):=makelist([i,F(i)],i,0,n)

→ LF : Fib_Liste_Anfang(20);
(LF) [[0,0],[1,1],[2,1],[3,2],[4,3],[5,5],[6,8],[7,13],[8,21],[9,34],[
10,55],[11,89],[12,144],[13,233],[14,377],[15,610],[16,987],[17,1597]
,[18,2584],[19,4181],[20,6765]]
```

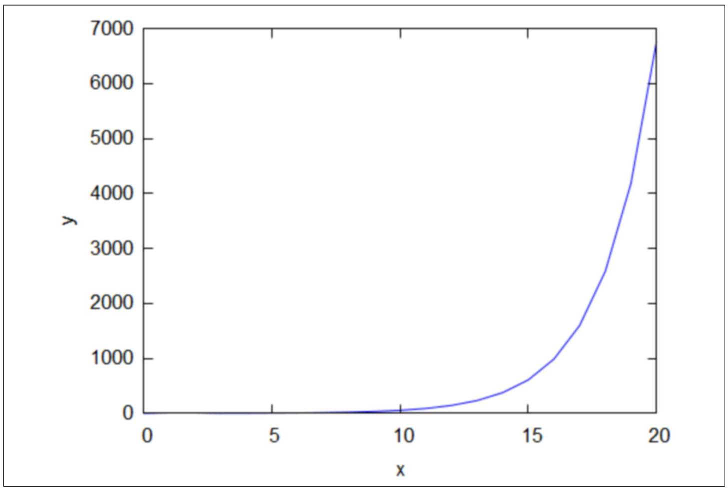
```
LF;
```

(%o46) [[0,0],[1,1],[2,1],[3,2],[4,3],[5,5],[6,8],[7,13],[8,21],[9,34],[10,55],[11,89],[12,144],[13,233],[14,377],[15,610],[16,987],[17,1597],[18,2584],[19,4181],[20,6765]]

Aus Gründen, die sich bisher allen Analyseversuchen entzogen haben, funktioniert der folgende Befehl manchmal nicht in der wxplot-Form
sondern nur in der plot-Form (siehe darauffolgende Zelle), bei der sich ein neues Graphik-Fenster öffnet, das zunächst geschlossen werden muss,
bevor man im Maxima-worksheet weiterarbeiten kann.

```
wxplot2d([discrete, LF]);
```

(%t47)



(%o47)

```
plot2d([discrete, LF]);        /* eigenes Fenster */
```

(%o78)

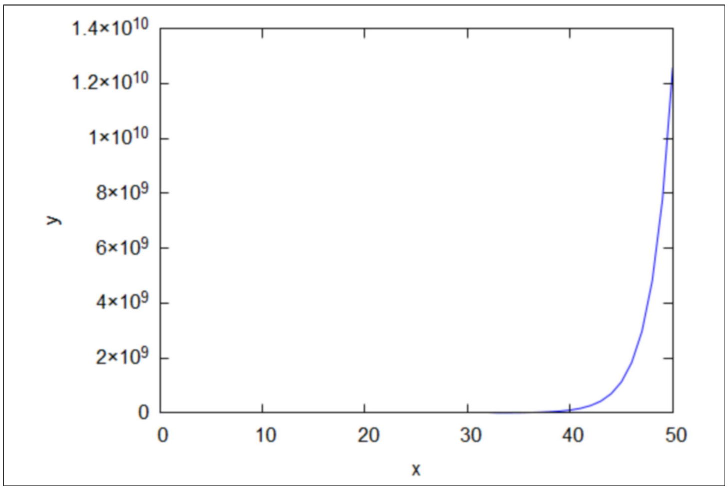
```
plot_options;
```

(%o44) [[t, − 3, 3], [grid, 30, 30], [transform_xy, false], [run_viewer, true], [axes, true], [plot_format, gnuplot], [color, blue, red, green, magenta, black, cyan], [point_type, bullet, circle, plus, times, asterisk, box, square, triangle, delta, wedge, nabla, diamond, lozenge], [palette, [hue, 0.25, 0.7, 0.8, 0.5], [hue, 0.65, 0.8, 0.9, 0.55], [hue, 0.55, 0.8, 0.9, 0.4], [hue, 0.95, 0.7, 0.8, 0.5]], [gnuplot_term, default], [gnuplot_out_file, false], [nticks, 29], [adapt_depth, 5], [gnuplot_preamble,], [gnuplot_default_term_command, set term pop], [gnuplot_dumb_term_command, set term dumb 79 22], [gnuplot_ps_term_command, se\ t size 1.5, 1.5;set term postscript eps enhanced color solid 24], [plot_realpart, false]]

Beispiel: Direkte Eingabe der Liste per Funktionsaufruf

→ wxplot2d([discrete, Fib_Liste_Anfang(50)]);

(%t48)



(%o48)

3.2 Graphiken mit draw

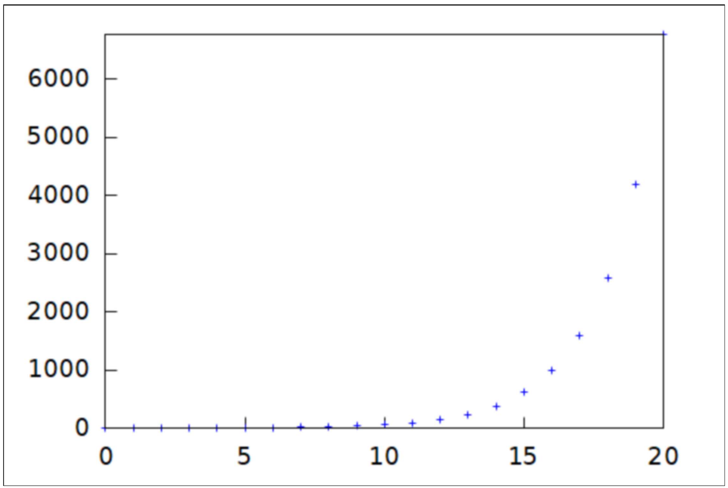
Das draw-Graphikpaket ist insgesamt flexibler als das plot-Paket.
Beispiele: Siehe W. Haager (Grafiken mit Maxima) oder Ziegenbalg (Erste Hinweise ...zu Maxima)

→ load(draw);

(%o49) *C:/maxima−5.43.2/share/maxima/5.43.2/share/draw/draw.lisp*

→ wxdraw2d(points(LF));

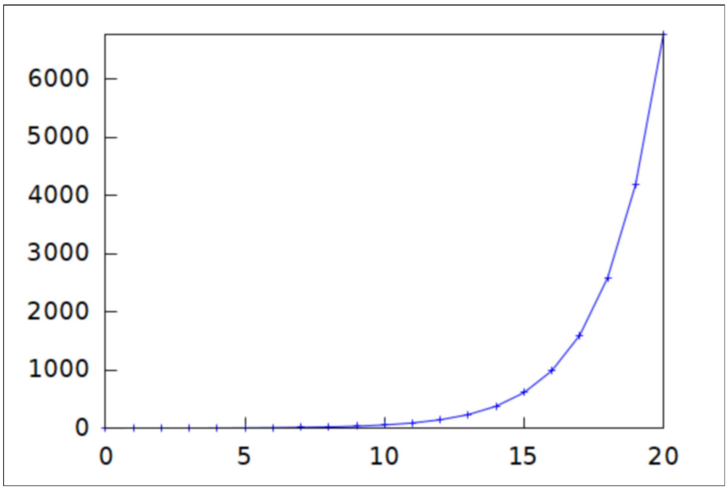
(%t51)



(%o51)

```
wxdraw2d([points_joined=true,points(LF)]);
```

(%t52)

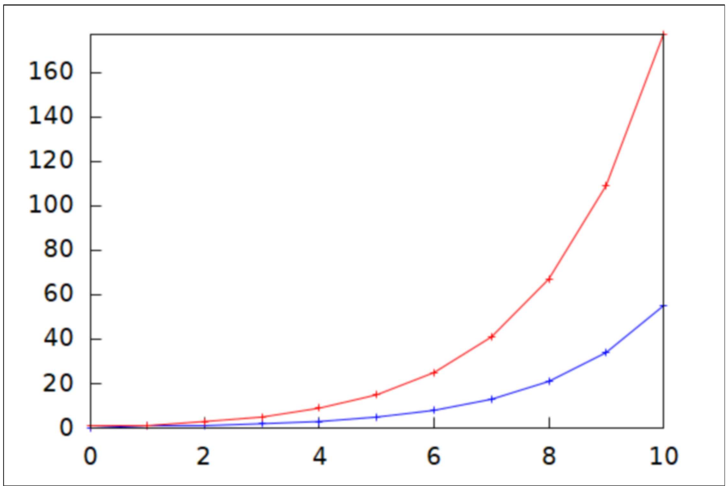


(%o52)

Die $ -Zeichen in der nächsten Zelle trennen (ähnlich wie der Stringpunkt ;) die verschiedenen Kommandos,
allerdings mit dem Unterschied: bei Verwendung von $ wird das Ergebnis nicht am Bildschirm dargestellt.

```
G1 : [points_joined=true,points(makelist([n,  F(n)], n, 0, 10))] $
G2 : [color=red, points_joined=true,points(makelist([n,  AFib(n)], n, 0, 10))] $
wxdraw2d(G1, G2);
```

(%t55)



(%o55)

Optionen für draw: (vgl. Haager, Seite 23)

set_draw_defaults(opts,. . .) Setzen von Defaultwerten für Optionen

terminal = term Ausgabeformat für die Grafik; mögliche Werte: screen (default), png, jpg, eps, eps_color, pdf.

file_name = "file" Ausgabeziel für die Grafik; default: maxima_out

user_preamble = "text" Gnuplot-Vorspann, enthält beliebige Gnuplot-Befehle, die vor dem Plot ausgeführt werden

dimensions = [width,height] Abmessungen der Grafik: in Bildpunkten bei Pixelgrafiken, in 1/10mm bei Vektorgrafiken

columns = n Anzahl der Spalten bei mehreren Szenen in einer Grafik

color = colorname Zeichenfarbe für Linien

background_color = name Hintergrundfarbe für das Diagramm

fill_color = name Füllfarbe für Rechtecke, Polygone und Kreise

x(yz)range = [min,max] Darstellungsbereich auf der x(yz)-Achse

logx(yz) = true/false Logarithmische Skalierung der x(yz)-Axchse

grid = true/false Zeichnen von Gitterlinien in der xy-Ebene

x(yz)tics = true/false Bestimmt, ob Skalenpukte auf der x(yz)-Achse automatisch gesetzt werden sollen

x(yz)tics_rotate = true/false Bestimmt, ob die Beschriftung der Skalenpukte um 90 Grad gedreht werden soll

title = "text" Diagrammtitel

key = "text" Angabe eines Funktionsnamens in der Legende (default: Leerstring)

x(yz)label = "text" Beschriftung der x(yz)-Achse

x(yz)axis = true/false Bestimmt, ob eine x(yz)-Achse gezeichnet werden soll

x(yz)axis_width = width Linienbreite für die entsprechende Achse

x(yz)axis_color = color Farbe für die entsprechende Achse

x(yz)axis_type = solid/dots Linientyp für die entsprechende Achse: durchgezogen (solid)

oder punktiert (dots), default: dots

line_width = width Linienbreite

line_type = solid/dots Linientyp (default: solid)

Hinweis: Die plot- und draw-Befehle gibt es in jeweils zwei Varianten:
1. Bei Verwendung von plot...(...) und draw...(...) wird ein neues Fenster aufgemacht, in das die Ausgabegraphik hineingeschrieben wird.
2. Bei Verwendung von wxplot...(...) und wxdraw...(...) wird die Ausgabegraphik direkt in das Arbeitsblatt hineingeschrieben.

ACHTUNG bei der Benutzung von plot / draw:
Man kann mit der Bearbeitung des Arbeitsblattes erst fortfahren, wenn man das Graphik-Fenster explizit geschlossen hat !
Es kann vorkommen,dass das Graphik-Fenster komplett hinter anderen Fenstern (insbesondere dem wxMaxima-Fenster)
verborgen ist.

4 Formeldarstellung (FB wegen Fibonacci mit Binet'scher Formel)

```
→ FB(n):= (1/sqrt(5))*(((1+sqrt(5))/2)^n - ((1-sqrt(5))/2)^n) $ ;
```

```
→ FB(10)            /* Maxima gibt die exakte Darstellung aus */;
```

(%o57) $$\frac{\frac{(\sqrt{5}+1)^{10}}{1024}-\frac{(1-\sqrt{5})^{10}}{1024}}{\sqrt{5}}$$

```
→ float(FB(10))          /* Gleitkomma-Darstellung; nicht exakt */;
```

(%o58) 55.00000000000001

```
→ makelist([k, FB(k)], k, 0, 6);
```

(%o59) $$[[0,0],[1,\frac{\frac{\sqrt{5}+1}{2}-\frac{1-\sqrt{5}}{2}}{\sqrt{5}}],[2,\frac{\frac{(\sqrt{5}+1)^{2}}{4}-\frac{(1-\sqrt{5})^{2}}{4}}{\sqrt{5}}],[3,\frac{\frac{(\sqrt{5}+1)^{3}}{8}-\frac{(1-\sqrt{5})^{3}}{8}}{\sqrt{5}}],[4,\frac{\frac{(\sqrt{5}+1)^{4}}{16}-\frac{(1-\sqrt{5})^{4}}{16}}{\sqrt{5}}],[5,\frac{\frac{(\sqrt{5}+1)^{5}}{32}-\frac{(1-\sqrt{5})^{5}}{32}}{\sqrt{5}}],[6,\frac{\frac{(\sqrt{5}+1)^{6}}{64}-\frac{(1-\sqrt{5})^{6}}{64}}{\sqrt{5}}]]$$

```
→ makelist([k, float(FB(k))], k, 0, 25);
```

(%o60) [[0,0.0],[1,1.0],[2,1.0],[3,2.0],[4,3.0],[5,5.000000000000001],[6,
8.000000000000002],[7,13.0],[8,21.0],[9,34.00000000000001],[10,
55.00000000000001],[11,89.00000000000003],[12,144.0000000000001],[13,
233.0000000000001],[14,377.0000000000002],[15,610.0000000000003],[16,
987.0000000000005],[17,1597.000000000001],[18,2584.000000000002],[19,
4181.000000000003],[20,6765.000000000005],[21,10946.00000000001],[22,
17711.00000000001],[23,28657.00000000002],[24,46368.00000000004],[25,
75025.00000000006]]

```
makelist([k, bfloat(FB(k))], k, 0, 25)
        /* bfloat (fuer bigfloat):  "lange" Gleitkommazahlen
          genauer als float, aber nicht exakt              */;
```

(%o61) [[0,0.0b0],[1,1.0b0],[2,1.0b0],[3,2.0b0],[4,3.0b0],[5,5.0b0],[6,8.0b0],[7,1.3b1],[8,2.1b1],[9,3.4b1],[10,5.5b1],[11,8.9b1],[12,1.44b2],[13,2.33b2],[14,3.77b2],[15,6.1b2],[16,9.87b2],[17,1.597b3],[18,2.584b3],[19,4.181b3],[20,6.765b3],[21,1.0946b4],[22,1.7711b4],[23,2.8657b4],[24,4.6368b4],[25,7.5025b4]]

```
ratsimp(FB(15))      /* ratsimp:  vereinfachte Darstellung, aber exakt */ ;
```

(%o62) 610

```
makelist([k, ratsimp(FB(k))], k, 0, 25);
```

(%o63) [[0,0],[1,1],[2,1],[3,2],[4,3],[5,5],[6,8],[7,13],[8,21],[9,34],[10,55],[11,89],[12,144],[13,233],[14,377],[15,610],[16,987],[17,1597],[18,2584],[19,4181],[20,6765],[21,10946],[22,17711],[23,28657],[24,46368],[25,75025]]

5 Brüche aus Fibonacci-Zahlen

```
f(n):=F(n+1)/F(n) $ ;
```

```
makelist(f(n),n, 1, 20);
```

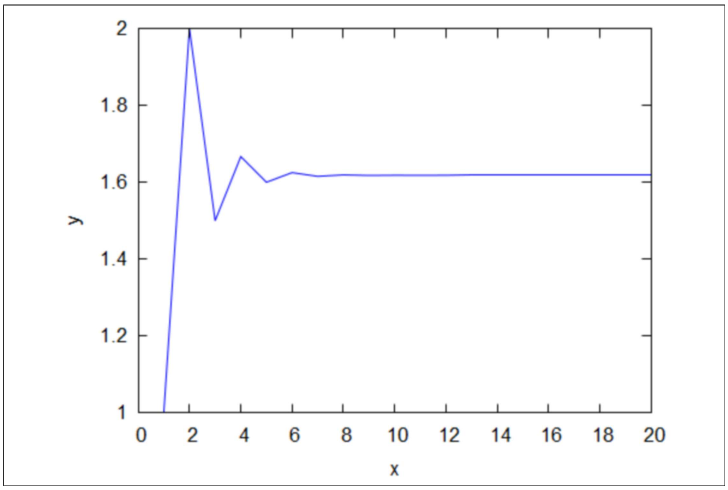
(%o65) $[1,2,\frac{3}{2},\frac{5}{3},\frac{8}{5},\frac{13}{8},\frac{21}{13},\frac{34}{21},\frac{55}{34},\frac{89}{55},\frac{144}{89},\frac{233}{144},\frac{377}{233},\frac{610}{377},\frac{987}{610},\frac{1597}{987},\frac{2584}{1597},\frac{4181}{2584},\frac{6765}{4181},\frac{10946}{6765}]$

```
Lf : makelist([n, bfloat(f(n))], n, 1, 20);
```

(Lf) [[1,1.0b0],[2,2.0b0],[3,1.5b0],[4,1.666666666666667b0],[5,1.6b0],[6,1.625b0],[7,1.615384615384615b0],[8,1.619047619047619b0],[9,1.617647058823529b0],[10,1.618181818181818b0],[11,1.617977528089888b0],[12,1.618055555555556b0],[13,1.618025751072961b0],[14,1.618037135278515b0],[15,1.618032786885246b0],[16,1.618034447821682b0],[17,1.618033813400125b0],[18,1.618034055727554b0],[19,1.618033963166707b0],[20,1.618033998521803b0]]

```
wxplot2d([discrete,Lf]);
```

(%t67)



(%o67)

Folgende Zelle auswerten, falls die "draw"-Umgebung noch nicht geladen ist.

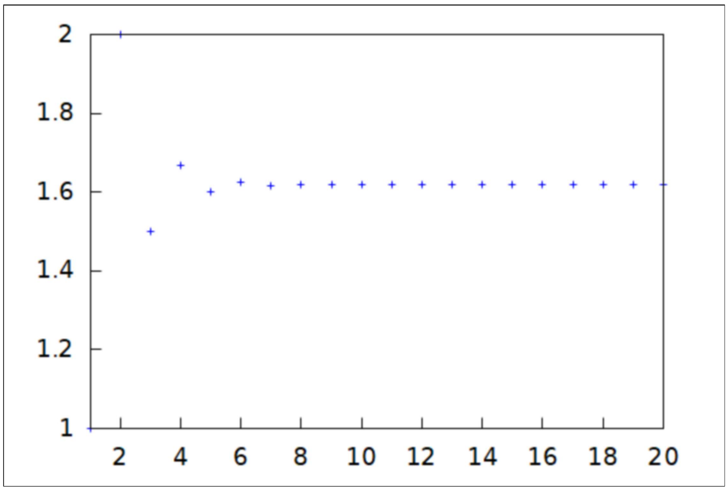
```
load(draw);
```

(%o64) C:/PROGRA~1/MAXIMA~1.2/share/maxima/5.31.2/share/draw/draw.lisp

Vor den nächsten Beispielen ggf. load(draw) !

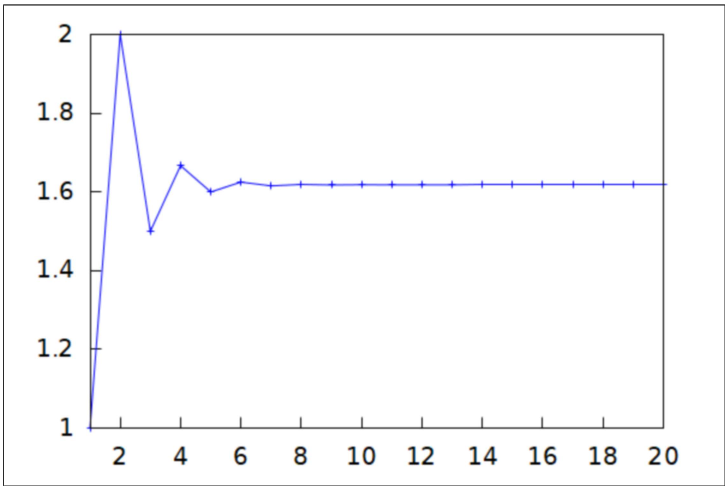
```
wxdraw2d(points(Lf))$
```

(%t68)



```
wxdraw2d([points_joined=true, points(Lf)])$
```

(%t69)



6 Experimente mit diversen Visualisierungen

(vgl. Figurierte Zahlen, Springer-Spektrum, Wiesbaden 2018)

6.1 1. Experiment (Quadrate und Rechtecke)

```
--> sum(F(i)^2, i, 0, 10);
(%o70) 4895
--> F(10)*F(11);
(%o71) 4895
--> sum(F(i)^2 , i, 0, 10) - F(10)*F(11);
(%o72) 0
--> makelist([sum(F(i)^2 , i, 0, k) , F(k)*F(k+1)], k, 0, 15);
(%o73) [[0,0],[1,1],[2,2],[6,6],[15,15],[40,40],[104,104],[273,273],[
714,714],[1870,1870],[4895,4895],[12816,12816],[33552,33552],[87841,
87841],[229970,229970],[602070,602070]]
```

6.2 2. Optische Täuschung mit Fibonacci

```
--> F(6)^2-F(5)*F(7);
(%o74) −1
--> F(7)^2-F(6)*F(8);
(%o75) 1
--> F(8)^2-F(7)*F(9);
(%o76) −1
--> makelist(F(n)^2-F(n-1)*F(n+1), n,2,50);
(%o77) [−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−
1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1,1,−1]
--> makelist([n, F(n)^2-F(n-1)*F(n+1)], n,2,50);
(%o78) [[2,−1],[3,1],[4,−1],[5,1],[6,−1],[7,1],[8,−1],[9,1],[10,−1],[
11,1],[12,−1],[13,1],[14,−1],[15,1],[16,−1],[17,1],[18,−1],[19,1],[
20,−1],[21,1],[22,−1],[23,1],[24,−1],[25,1],[26,−1],[27,1],[28,−1],[
29,1],[30,−1],[31,1],[32,−1],[33,1],[34,−1],[35,1],[36,−1],[37,1],[
38,−1],[39,1],[40,−1],[41,1],[42,−1],[43,1],[44,−1],[45,1],[46,−1],[
47,1],[48,−1],[49,1],[50,−1]]
--> functions    /* In diesem Arbeitsblatt definierte Funktionen */;
(%o79) [Fib(n),Fib_it(n),F(n),AFib(n),Fib_Liste_Anfang(n),FB(n),f(n)]
```

Fortsetzung folgt