Robert Salvador salvador@htlinn.ac.at

Fraktale Kurven

- Mathematische / Fachliche Inhalte in Stichworten:

  Fraktale, Komplexe Zahlen, Iteration

- Kurzzusammenfassung

  Als geometrische Anwendung der komplexen Zahlen wird, ausgehend von ganz einfachen ebenen Kurven durch Iteration ein immer komplexer werdendes Gebilde erzeugt - im Grenzfall eine fraktale Kurve.

- Didaktische Überlegungen / Zeitaufwand:

  Nur teilweise (komplexe Zahlen, Rotation durch Multiplikation, Prinzip der Iteration) für den HTL-Unterricht zu gebrauchen. Der geometrisch-fraktale Aspekt geht weit über die HTL-Bedürfnisse hinaus und könnte wohl nur im Rahmen eines Freifaches Mathematik an den (besonders interessierten) Schüler gebracht werden.

- Lehrplanbezug (bzw. Gegenstand / Abteilung / Jahrgang):

  Rechenoperationen mit komplexen Zahlen und ihre geometrische Interpretation, Iterationsalgorithmen.

- Mathcad-Version:

  Mathcad 2000

- Literaturangaben:

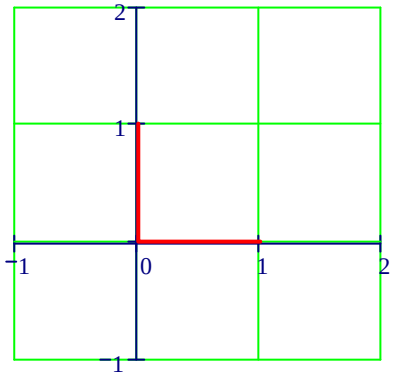
  Peitgen H.-O., Jürgens H., Saupe D.: Chaos - Bausteine der Ordnung, Klett-Kotta/Springer-Verlag
  Zeitler H., Pagon D.: Fraktale Geometrie - Eine Einführung, Vieweg

- Anmerkungen bzw. Sonstiges:

In den beiden oben angeführten Büchern ist von einer Mehrfach-Verkleinerungs-Kopier-Maschine (MVKM) bzw. Barnsley-Maschine die Rede. Darunter verstehen die Autoren eine fiktive "Maschine", eine Vorschrift, wie man - ausgehend von einer sehr einfachen geometrischen Figur - durch Erzeugung einer verkleinerten Kopie (oder mehrerer verkleinerter Kopien) und Zusammensetzen dieser Kopien zu einer neuen Figur, die dann demselben Mechanismus unterworfen wird, selbstähnliche, also fraktale Kurven erzeugen kann.
Wie dies mit Mathcad geschehen könnte, soll hier an einigen Beispielen gezeigt werden.

1. Die Drachenkurve:
Wie schon angedeutet beginnt die Konstruktion mit einer einfachen Kurve; in unserem Fall werden drei Punkte in der (komplexen) Ebene definiert, die durch einen Polygonzug verbunden werden. Dass dazu die komplexe Ebene verwendet wird anstatt eines reellen kartesischen Koordinatensystems hat keine wesentlichen mathematischen Vorteile Lediglich die Rotation der Figur lässt sich etwas kompakter formulieren als mit Vektoren und Drehmatrizen.

Unsere Startfigur ist definiert durch $Drachen\_start := \begin{pmatrix} j \\ 0 \\ 1 \end{pmatrix}$; die Punkte werden in der Reihenfolge ihrer Vektor-Indizes verbunden:



Im nächsten Schritt erstellen wir eine Kopie dieser Figur, und unterwerfen sie einem für die Endkurve charakteristischen Algorithmus. Das "Kochrezept" dafür lautet:
Kehre die Reihenfolge der Punkte in der Kopie um, drehe die so geänderte Kopie im Uhrzeigersinn um 90 Grad und verschiebe sie so, dass ihr Anfangspunkt mit dem Endpunkt des Originals zusammenfällt. Das Ganze um den Faktor $\frac{1}{\sqrt{2}}$ verkleinert und um 45 Grad im Uhreigersinn gedreht vollendet die erste Iteration.
Diese Schritte werden nun einzeln berechnet:

Erzeugung der Kopie:

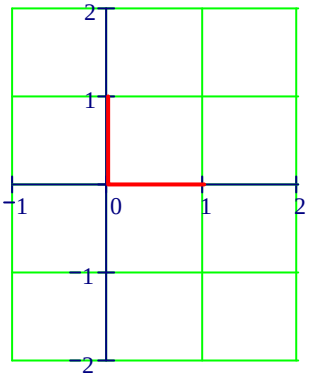
$Kopie := Drachen\_start$ $\quad Kopie = \begin{pmatrix} j \\ 0 \\ 1 \end{pmatrix}$

Umkehrung der Punktreihenfolge in der Kopie:

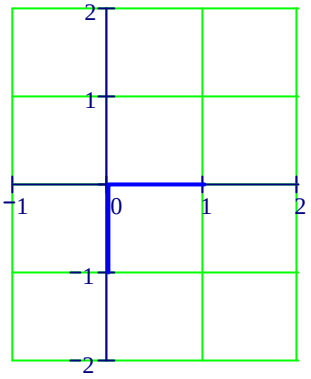
$Kopie\_u := umkehren(Kopie)$ $\quad Kopie\_u = \begin{pmatrix} 1 \\ 0 \\ j \end{pmatrix}$

Drehung um 90 Grad im Uhrzeigersinn:

$Kopie\_u\_90 := Kopie\_u \cdot (-j)$ $\quad Kopie\_u\_90 = \begin{pmatrix} -j \\ 0 \\ 1 \end{pmatrix}$



Startfigur



umgekehrte Reihenfolge und gedreht

Berechnung der komplexen Differenz zwischen dem Endpunkt des Originals und dem Anfangspunkt der umgekehrten, gedrehten Kopie (wird gebraucht um die beiden zusammenzuhängen!)

$diff := Kopie_2 - Kopie\_u\_90_0$

Vor dem Zusammenhängen der beiden Teile wird der letzte Punkt im Original entfernt damit dieser Punkt nicht doppelt vorkommt:

$Kopie\_kurz := submatrix(Kopie, 0, 1, 0, 0)$

$$Kopie\_kurz = \begin{pmatrix} j \\ 0 \end{pmatrix}$$

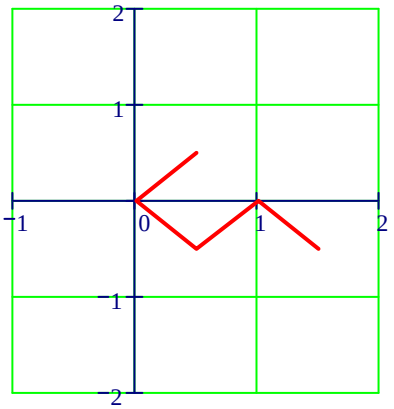
Nun entsteht aus beiden Teilen ein längeres Gebilde aus 5 Punkten:

$Kopie\_u\_90\_neu := stapeln(Kopie\_kurz, Kopie\_u\_90 + diff)$

$$Kopie\_u\_90\_neu = \begin{pmatrix} j \\ 0 \\ 1 \\ 1 + j \\ 2 + j \end{pmatrix}$$

Das Ganze wird noch verkleinert um den Faktor $\frac{1}{\sqrt{2}}$ und um 45 Grad im Uhrzeigersinn gedreht:

$$Kopie\_u\_90\_neu\_45\_klein := \frac{Kopie\_u\_90\_neu}{\sqrt{2}} \cdot e^{-j\cdot\frac{\pi}{4}}$$



Ergebnis der 1. Iteration

Die nächste Iteration verwendet dieses Ergebnis als Startfigur, ...

Mit Hilfe der Programmiermöglichkeiten von Mathcad wird der Vorgang iteriert:

Festlegung der Startfigur: $Drachen\_start := \begin{pmatrix} j \\ 0 \\ 1 \end{pmatrix}$

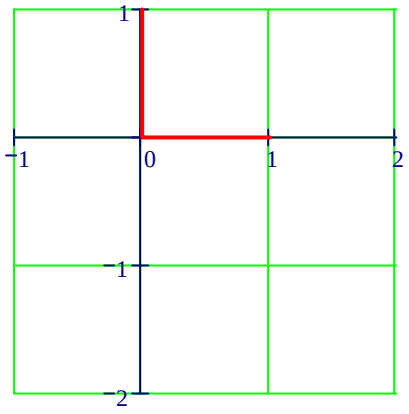
```
Drachen_iteration(N) :=
    N ← max(0, floor(N))
    vektor ← Drachen_start
    vektor  if  N = 0
    otherwise
        n ← 0
        while  n ≤ N − 1
            L ← letzte(vektor)
            short ← submatrix(vektor, 0, L − 1, 0, 0)
            vektor2 ← umkehren[vektor · (−j)]
            differenz ← vektor_L − vektor2_0
            vektor ← stapeln(short, vektor2 + differenz) / √2
            vektor ← vektor · e^(−j·π/4)
            n ← n + 1
    vektor
```

Erklärungen zur Funktion *Drachen_iteration(N)*:
Der Funktionsparameter *N* gibt die Zahl der Iterationen an.
Die erste Programmzeile sorgt dafür, dass unsinnige Eingaben für *N* in sinnvolle umgewandelt werden: negative Werte werden zu 0. Dezimalzahlen auf die nächst kleinere nichtnegative ganze Zahl zurechtgestutzt.
Die Variable *vektor* wird jeweils der Iteration unterzogen. *vektor* wird anfangs gleich *Drachen_start* gesetzt und nach jeder Iteration durch das soeben entstandene letzte Iterationsergebnis ersetzt.

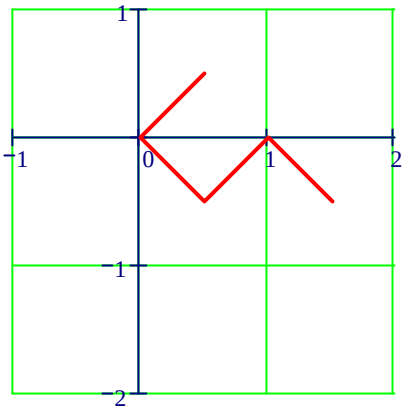
Falls *N* = 0 ist, wird *vektor* unverändert (also *Drachen_start*) zurückgeliefert.
Andernfalls wird der Schleifenzähler *n* mit 0 initialisiert, *vektor* der 1. Iteration unterzogen (wie oben im Detail demonstriert), das Ergebnis wieder in *vektor* gespeichert und der Schleifenzähler inkrementiert bis die Abbruchbedingung erfüllt ist.

Wie die ersten paar Iterationsergebnisse aussehen zeigen folgende Bilder:

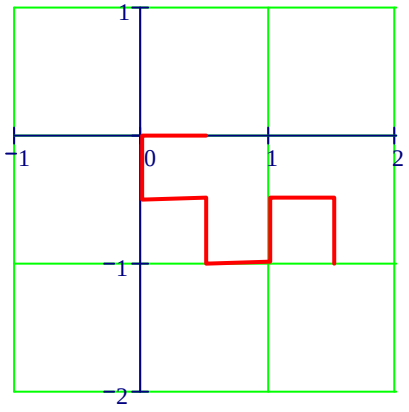
Startkurve



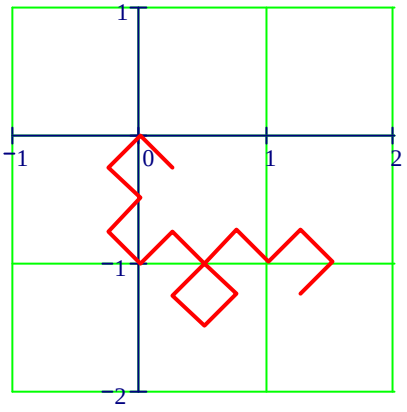
1. Iteration



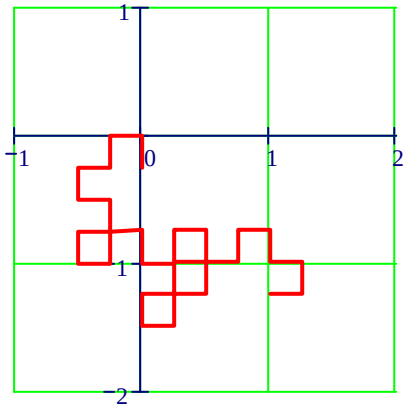
2. Iteration



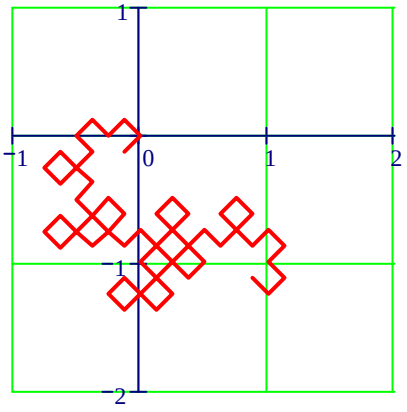
3. Iteration



4. Iteration

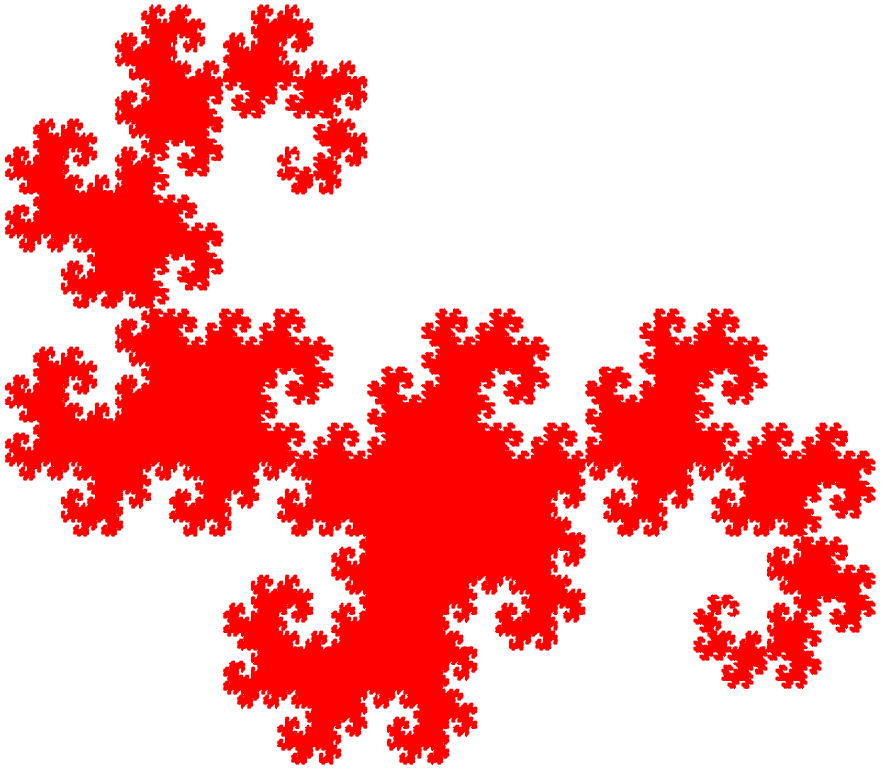


5. Iteration



Drache := *Drachen_iteration*(16)

16. Iteration



Dass man mit Mathcad (kreativ) spielen kann, beweisen (zumindest dem Autor des Artikels) die folgenden Bilder. Sie entstehen dadurch, dass man die durch eine entsprechende Anzahl von Iterationen entstandenen komplexen Zahlen als Argumente einer (fast beliebigen) Funktion verwendet und erst dann grafisch darstellt. Geben Sie der Ästhetik eine Chance und lassen Sie Ihrer Fantasie freien Lauf!

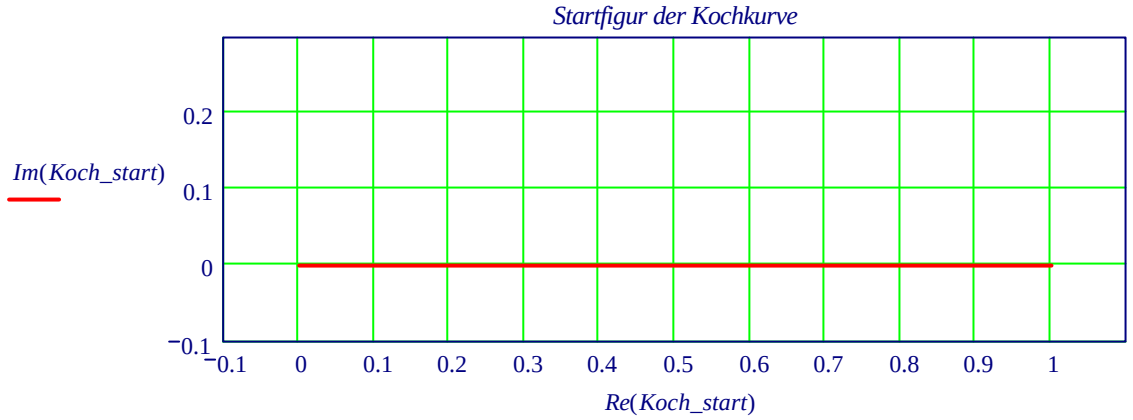
$f_1(z) := z^{2.5}$ $\qquad$ $Drache_1 := \overrightarrow{f_1(Drache)}$

Anmerkung: Aus Speicherplatzgründen wurden die im Mathcad-Artikel folgenden 3 Beispiele zur "kreativen Gestaltung" der Drachenkurve für die PDF-Datei herausgeschnitten!
Gegebenenfalls also bitte diese Beispiele den Mathcad-Files entnehmen !!

2. Die Koch´sche Kurve:

In diesem Fall gehen wir von einer noch einfacheren Grundkurve aus, nämlich einer Strecke zwischen den Punkten 0 und 1 der komplexen Ebene. Unsere MVKM hat allerdings kompliziertere Anweisungen auszuführen:
Zunächst einmal sind drei Kopien des Originals zu erzeugen. Die erste Kopie wird - um 60 Grad gedreht - an das Original gehängt. Die zweite Kopie rotiert man um -60 Grad und verschiebt sie zum Endpunkt des vorigen Zwischenergebnisses. Daran wird noch die um 2 nach rechts verschobene 3. Kopie gehängt. Die Figur wird um den Faktor 3 verkleinert - fertig.

$$Koch\_start := \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

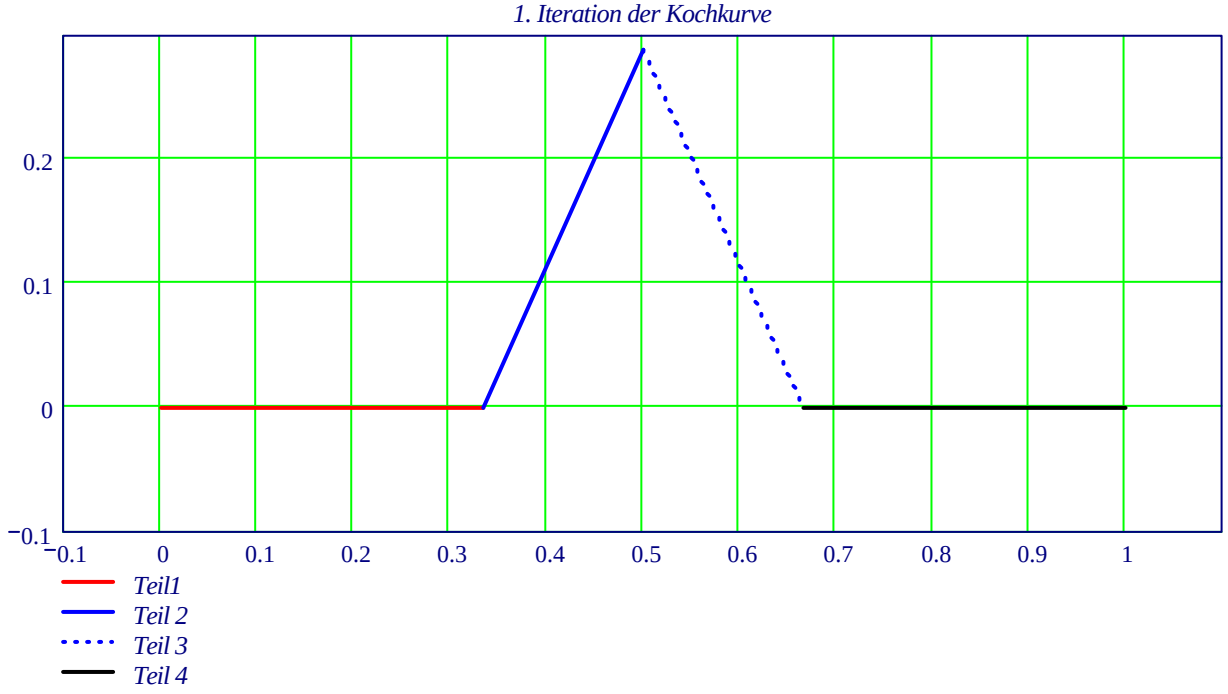


Nach links gedrehte Kopie: $links := Koch\_start \cdot e^{j\cdot\frac{\pi}{3}}$

Nach rechts gedrehte Kopie: $rechts := Koch\_start \cdot e^{-j\cdot\frac{\pi}{3}}$

Die richtig platzierten Kopien:

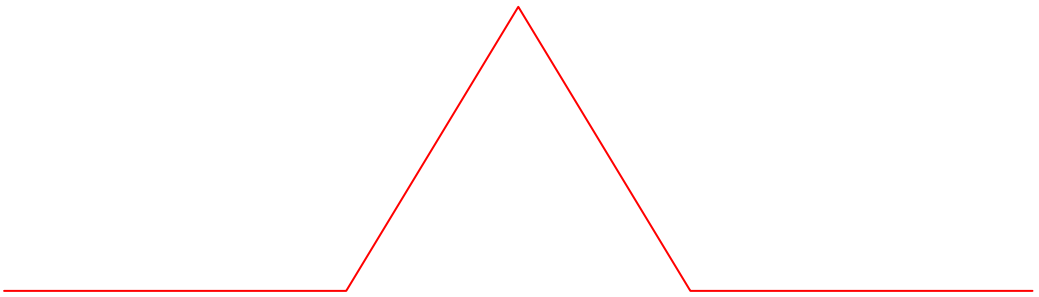
$Teil2 := links + 1$ $Teil3 := rechts + 1 + e^{j\cdot\frac{\pi}{3}}$ $Teil4 := Koch\_start + 2$



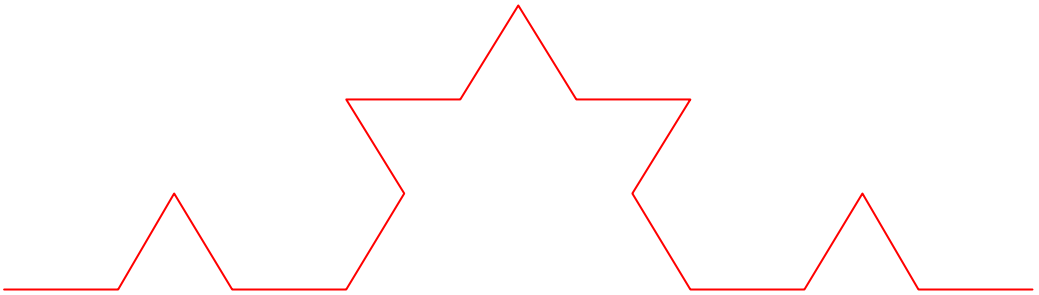
Der Algorithmus der MVKM ist im folgenden programmiert. Er lehnt sich so weit wie möglich an den Drachenalgorithmus an und sollte nach den Erklärungen von oben nachvollziehbar sein:

$$
\begin{aligned}
&Koch\_iteration(N) := \\
&\quad N \leftarrow max(0,\ floor(N)) \\
&\quad vektor \leftarrow Koch\_start \\
&\quad vektor \ \ if \ \ N = 0 \\
&\quad otherwise \\
&\qquad n \leftarrow 0 \\
&\qquad while \ \ n \le N - 1 \\
&\qquad\quad orig \leftarrow vektor \\
&\qquad\quad links \leftarrow vektor \cdot e^{j\cdot\frac{\pi}{3}} \\
&\qquad\quad rechts \leftarrow vektor \cdot e^{-j\cdot\frac{\pi}{3}} \\
&\qquad\quad L \leftarrow letzte(vektor) \\
&\qquad\quad short \leftarrow submatrix(vektor, 0, L-1, 0, 0) \\
&\qquad\quad vektor \leftarrow stapeln(short, links + 1) \\
&\qquad\quad L \leftarrow letzte(vektor) \\
&\qquad\quad short \leftarrow submatrix(vektor, 0, L-1, 0, 0) \\
&\qquad\quad vektor \leftarrow stapeln\left(short, rechts + 1 + e^{j\cdot\frac{\pi}{3}}\right) \\
&\qquad\quad L \leftarrow letzte(vektor) \\
&\qquad\quad short \leftarrow submatrix(vektor, 0, L-1, 0, 0) \\
&\qquad\quad vektor \leftarrow stapeln(short, orig + 2) \\
&\qquad\quad vektor \leftarrow \frac{vektor}{3} \\
&\qquad\quad n \leftarrow n + 1 \\
&\quad vektor
\end{aligned}
$$

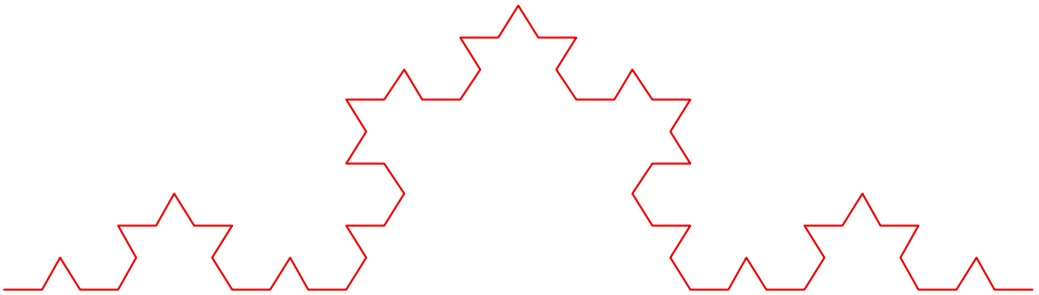
1. Iteration



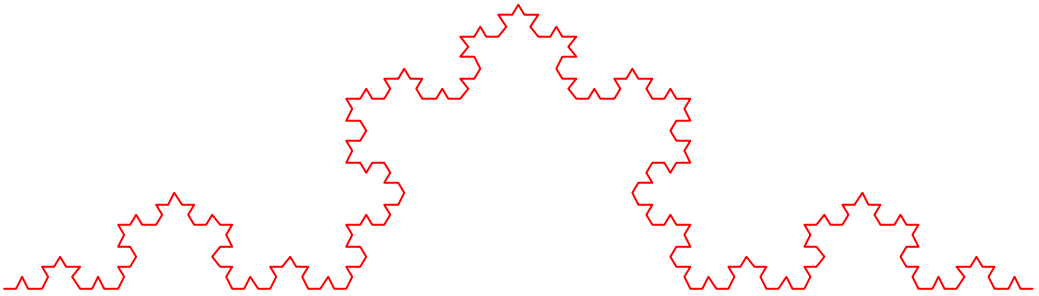
2. Iteration



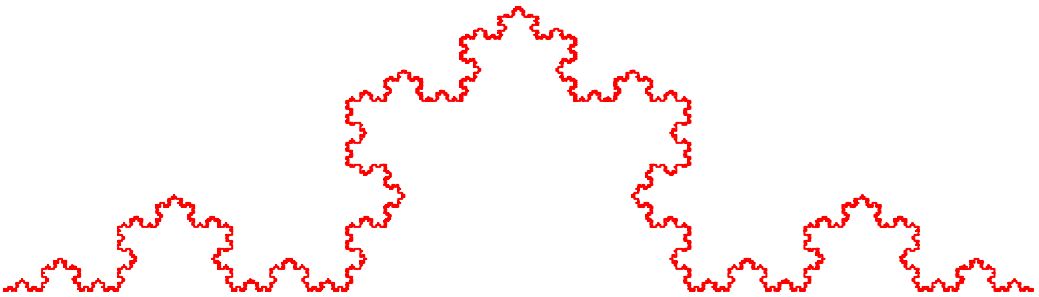
3. Iteration



4. Iteration



7. Iteration



Als letztes soll noch kurz

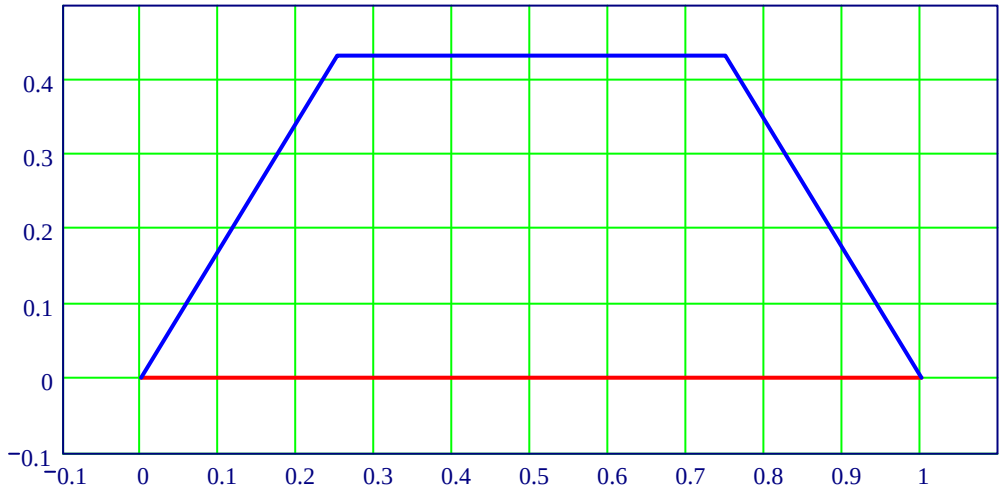
3. Die Sierpinski-Pfeilspitze

vorgestellt werden, die ein ähnliches Konstruktionsprinzip aufweist wie die Koch´sche Kurve und von derselben Grundfigur ausgeht.

$$Sierpinski\_start := \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad Sierpinski\_1 := \begin{pmatrix} 0 \\ \frac{1 + \sqrt{3} \cdot j}{4} \\ \frac{3 + \sqrt{3} \cdot j}{4} \\ 1 \end{pmatrix}$$

Das folgende Bild demonstriert, dass die Startfigur durch ein gleichschenkliges Trapez mit drei gleich langen Seiten ersetzt wird. Es müssen also zwei Kopien der vorhergehenden Strecke erzeugt und zusammen mit dem Original entsprechend angeordnet werden:

Startfigur und 1. Iteration

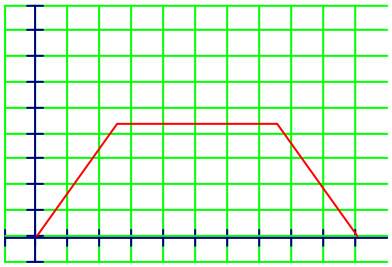


Die drei gleich langen Seiten werden jeweils wieder durch Trapeze ersetzt. Dabei ist zu beachten: die Trapeze, die auf den Seitenlinien aufsetzen, sind nach innen gerichtet, jenes auf der mittleren Linie nach außen.

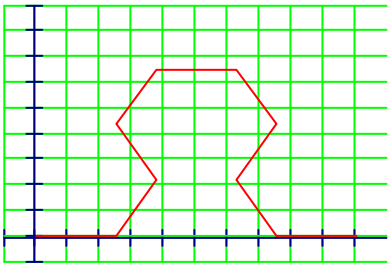
Die folgende Funktion enthält die Iterationsvorschrift. Anschließend werden die ersten paar Iterationsergebnisse gezeigt:

```
Sierpinski_iteration(N) := | N ← max(0, floor(N))
                           | vektor ← Sierpinski_start
                           | vektor if N = 0
                           | otherwise
                           |     n ← 0
                           |     while n ≤ N − 1
                           |         orig ← vektor
                           |         orig_conj ← conj(vektor)
                           |         teil1 ← orig_conj · e^(j·π/3)
                           |         teil3 ← orig_conj · e^(−j·π/3)
                           |         L ← letzte(teil1)
                           |         short ← submatrix(teil1, 0, L − 1, 0, 0)
                           |         vektor ← stapeln(short, orig + e^(j·π/3))
                           |         L ← letzte(vektor)
                           |         short ← submatrix(vektor, 0, L − 1, 0, 0)
                           |         vektor ← stapeln(short, teil3 + 1 + e^(j·π/3))
                           |         vektor ← vektor / 2
                           |         n ← n + 1
                           | vektor
```

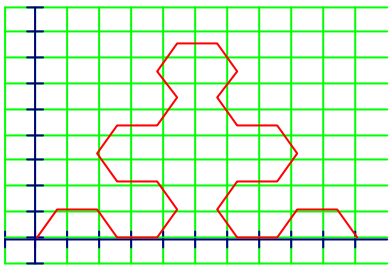
1. Iteration



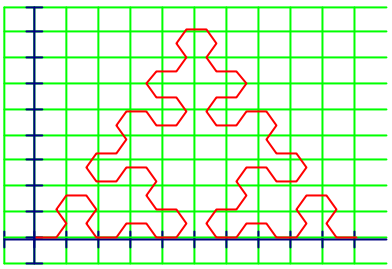
2. Iteration



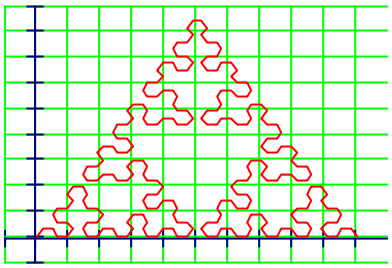
3. Iteration



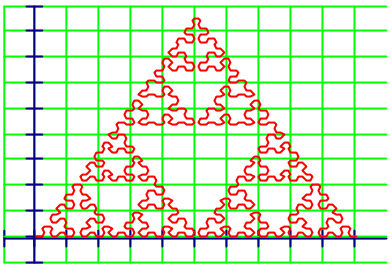
4. Iteration



5. Iteration

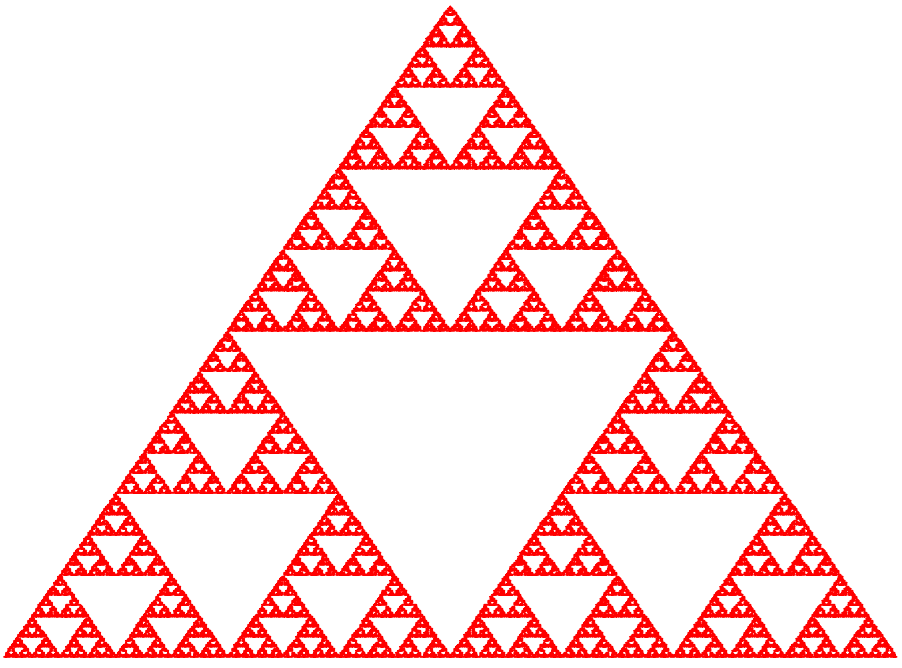


6. Iteration



Sierpinski := *Sierpinski_iteration*(10)

10. Iteration



Auch im Fall der Sierpinski-Pfeilspitze kann man seinem ästhetischen Empfinden freien Lauf lassen. Zwei Beispiele seien hier angeführt:

Anmerkung: Auch diese 2 Beispiele wurden aus Speicherplatzgründen aus der PDF-Datei entfernt. Bitte diese Beispiele gegebenenfalls den Mathcad-Files entnehmen!)