

Das Thema "Apfelmännchen" gehört zum Oberthema

"Chaos und Ordnung in dynamischen Systemen" .

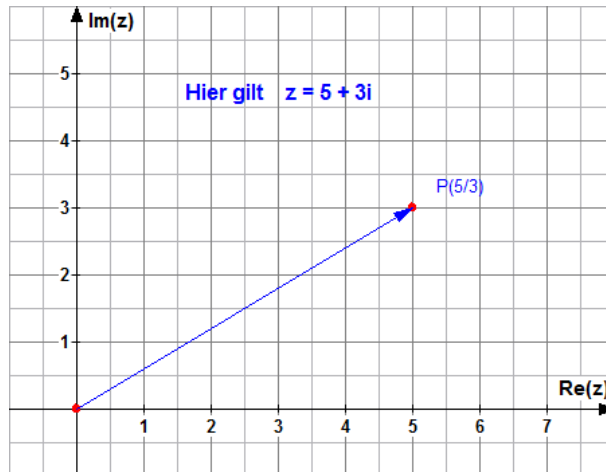
Es ist ein relativ neues Forschungsgebiete der Mathematik (ab ca. 1980).

Ausgangspunkt ist die komplexe Iterationsformel $z_{n+1} = z_n^2 + c := f(z_n)$.

Sowohl z als auch c sind komplexe Zahlen der Form $a+i*b$,
wobei **a der Realteil** und **b der Imaginärteil** der Zahl ist (s. Grafik).
Man schreibt hierfür auch: $a = \text{Re}(z)$ und $b = \text{Im}(z)$

Darstellung in der Gaußsche Zahlenebene

Beispiel $z = 5 + 3i$



Für jede komplexe Zahl $c=a+ib$ eines Ausschnitts aus der Gaußschen Zahlenebene wird die Zahlenfolge $\{z_n\}$ gemäß obiger Vorschrift ($z_{n+1} = z_n^2 + c$) auf Konvergenz geprüft, d.h. es wird untersucht, ob die Folge einen Grenzwert besitzt. Dies bedeutet, dass die Folge der Zahlen stets innerhalb eines endlichen Bereichs bleibt. Besitzt die Zahlenfolge jedoch keinen Grenzwert, so wächst sie „über alle Schranken“. Sie „divergiert“ .

Falls die Folge $\{z_n\}$ einen Grenzwert besitzt, so soll der Bildpunkt von c in der Gaußschen Zahlenebene gezeichnet werden. Divergiert die Folge, so wird kein Punkt gezeichnet. Das Gesamtbild aller derart gezeichneten Bildpunkte heißt Apfelmännchen bzw. **Mandelbrotmenge** ,eine sog. „selbstähnliche“ Figur (s.u.).

Da Konvergenz mit Hilfe des Computers schlecht feststellbar ist, begnügen wir uns mit folgender Forderung:

Die Folge sei konvergent, wenn ihr Betrag $|z|$ (das ist die Strecke vom Ursprung $(0;0)$ bis zur Zahl $z = x+i*y := (zx,zy)$) nach einer festen Zahl von Iterationsschritten (z.B. $n=500$ Schritte) die oben angegebene endliche Schranke 2 nicht überschreitet.

(Dass 2 als Schranke ausreicht lässt sich übrigens mathematisch beweisen.)

Anm.: Der Betrag berechnet sich zu $\sqrt{(zx^2+zy^2)}$

Rechenbeispiel für $c = -0,5 + 1,0 * i$ (also $\text{Re}(z) = -0,5$ $\text{Im}(z) = 1,0$):

Gestartet wird immer mit $z_0 = 0$.

1.Schritt: $z_1 = 0^2+(-0,5+i) = -0,5+i$

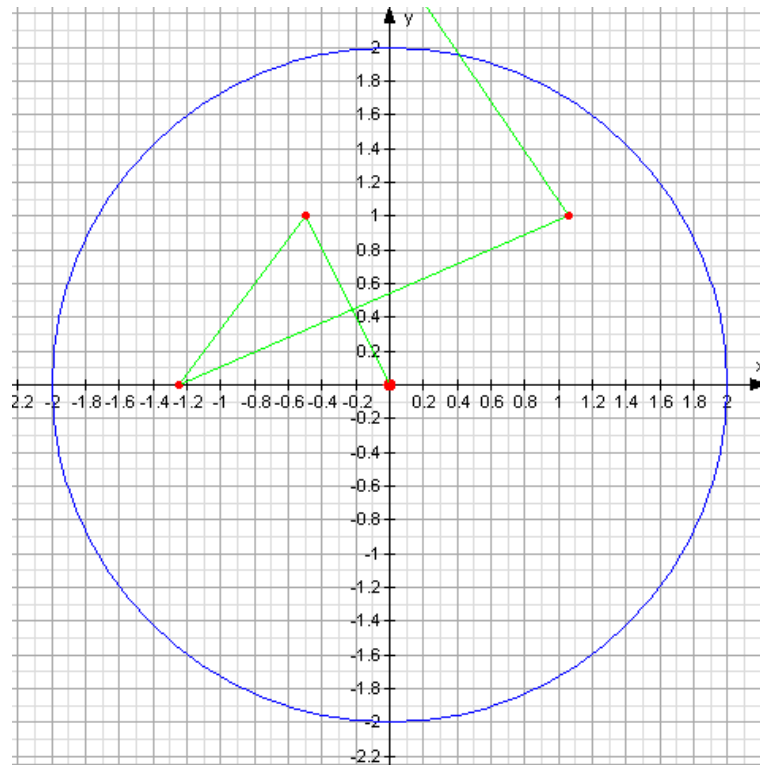
2.Schritt: $z_2 = (-0,5+i)^2+(-0,5+i) = 0,25-1-i-0,5+i = -1,25$

3.Schritt: $z_3 = (-1,25)^2+(-0,5+i) = 1,5625-0,5+i = 1,0625+i$

4.Schritt: $z_4 = (1,0625+i)^2+(-0,5+i) = 1,12890625-1+2,125i-0,5+i = -0,37109375+3,125i$

Wie man leicht prüft, ist der Betrag dieser Zahl ($\approx 3,1$) bereits größer als 2, und somit gehört der Punkt $(-0,5 ; 1,0)$ nicht zur Mandelbrotmenge.

Grafische Darstellung
der obigen Iteration:



Der vorläufige Algorithmus für die Iteration der Folge:

```
Vorgegeben ist die Iterationstiefe maxIt (z.B. 500)
z(0):=0   k:=0
wiederhole
  z(k+1):=z(k)2+c
  k:=k+1
bis |z|>2 oder k=maxIt
falls k=maxIt dann zeichne P(Re(c),Im(c))
```

Präzisierung des Algorithmus durch Verwendung der Regeln bei komplexen Zahlen:

Setzt man $z_x := \text{Re}(z)$, $z_y := \text{Im}(z)$, $c_x := \text{Re}(c)$, $c_y := \text{Im}(z)$, so erhält man:

$$z^2 = (z_x + i z_y)^2 = z_x^2 - z_y^2 + i 2 z_x z_y$$

$$z(k+1) = z_x^2 - z_y^2 + i 2 z_x z_y + c_x + i c_y, \text{ also } z(k+1) = z_x^2 - z_y^2 + c_x + i (2 z_x z_y + c_y)$$

Außerdem gilt in der Menge der komplexen Zahlen:

$$|z| = \sqrt{z_x^2 + z_y^2}, \text{ d.h. } |z|^2 = z_x^2 + z_y^2$$

Anmerkung:

Zur Darstellung der Mandelbrotmenge genügt als Ausschnitt aus der Gaußschen Zahlenebene z.B. $-2,05 \leq x \leq 0,75$; $-1,25 \leq y \leq 1,25$

Als Auflösung wählen wir **476 (x-Bereich) mal 425 (y-Bereich)**, womit wir eine verzerrungsfreie Darstellung erhalten (Verhältnis 112:100).

Für jeden Punkt (c_x, c_y) aus dem oben gewählten Ausschnitt wird dann iteriert.

Algorithmus (Pseudocode):

```
ImageBreite=476  ImageHoehe=425  maxIt=500
xa=-2.05  xe=0.75  ya=-1.25  ye=1.25
dx=(xe-xa)/ImageBreite  dy=(ye-ya)/ImageHoehe
cx=xa // von links nach rechts abarbeiten
für Sp=0 bis ImageBreite wiederhole
  cy=ye // Beginn am oberen Rand
  für z=0 bis ImageHoehe wiederhole
    zx=0  zy=0  zaehler=0
    wiederhole // iteriere
      tmp=zx*zx-zy*zy+cx  zy=2*zx*zy+cy  zx=tmp  zaehler=zaehler+1
    bis (zx*zx+zy*zy>4.0) oder (zaehler=maxIt)
    falls zaehler = maxIt dann setzePunkt(Sp,Ze)
    cy=cy-dy
  ende // für Ze
  cx=cx+dx
ende // für Sp
```

Erweiterungen und Hinweise:

1) Obiger Algorithmus ist insofern ungünstig, als er in der Regel den Wert $cy=0$ (x-Achse) übergeht ! Dies liegt an der Ungenauigkeit der Gleitpunktarithmetik von Programmiersprachen. Addiert man nämlich dy fortlaufend zu $cy = -1.2$, so ergibt sich nach 200 Iterationen nicht gleich 0, sondern z.B. $1.0390993621101074E-15$ (JAVA-Compiler).

Da aber $cy=0$ im Bereich $-2 \leq cx \leq 0$ sehr wohl zur Mandelbrotmenge gehört (dies ist rechnerisch sehr leicht zu zeigen), muss man bei der Iteration mit $cy=0$ beginnen und sich dann nach oben bzw. unten „vorarbeiten“.

Liegt die x-Achse jedoch nicht im zu untersuchenden Bereich, z.B. $y \in [0,1;1,4]$, dann muss eben doch der obige Algorithmus verwendet werden.

Modifizierter Algorithmus: $cy=0$ für $Ze=y0Pkt$ bis $ImageHoehe$ wiederhole ... $cy=cy-dy$
 $cy=0$ für $Ze=y0Pkt$ ab bis 0 wiederhole $cy=cy+dy$

2) Das Erstellen der Bilder dauert in der Regel mehrere Sekunden bei aktuellen Computern (ca. 2012), so dass sich Optimierungen des Algorithmus lohnen. Beispielsweise lassen sich die oben eingefärbten Iterationsanweisungen ersetzen durch eine optimierte Fassung (zxq, zyq sparen Rechnungen ein):

```
zx=0 zy=0 zxq=0 zyq=0 zaehler=0
wiederhole // iteriere
  zy=2*zx*zy+cy  zx=zxq-zyq+cx  zxq=zx*zx  zyq=zy*zy
  zaehler=zaehler+1
bis (zxq+zyq>4.0) oder (zaehler=maxIt)
```

Ein weitere Verbesserung besteht darin, vor der Iteration einen sog. Kreis-Zykliden-Test einzubauen:

```
zx=0.0 zy:=0.0 zxq=0.0 zyq=0.0 zaehler=0
r=cx*cx+cy*cy  s=wurzel(r-0.5*cx+0.0625)
falls (16*r*s > 5*s-4*cx+1) und ((cx+1)*(cx+1)+cy*cy > 0.0625)
  dann wiederhole
    zy=2*zx*zy+cy  zx=zxq-zyq+cx  zxq=zx*zx  zyq=zy*zy
    zaehler=zaehler+1
  bis (zxq+zyq>4.0) oder (zaehler=maxIt)
  sonst zaehler=maxIt
falls zaehler=maxIt dann .....
```

3) Untersucht man für divergente Folgen, wie nahe der Zähler der maximalen Iterationszahl $maxIt$ gekommen ist und zeichnet für entsprechende Bereiche des ermittelten Zählers (z.B. $[maxIt-10;maxIt-1]$, $[maxIt-20;maxIt-11]$ etc.) einen farbigen Punkt $(cx;cy)$ in die Gaußsche Zahlenebene, so erhält man eine interessante farbige Darstellung für das Apfelmännchen.

4) Setzt man den Startwert von z ungleich Null, so ergeben sich interessante Verformungen.

5) Die weiter oben erwähnte Selbstähnlichkeit zeigt sich, wenn man Bereiche aus der Mandelbrotmenge vergrößert. Man erhält dann zur Originalfigur ähnliche Figuren. Dies wird ganz unten anhand der Bilder gezeigt.

6) Gibt man die komplexe Zahl c konstant vor und betrachtet jetzt für jede komplexe Zahl z der Gaußschen Zahlenebene die Folge $\{z_n\}$ gemäß der Vorschrift $z_{n+1} = z_n^2 + c$, so entstehen Bilder, die man als Juliamengen bezeichnet !

Programmierung mit JAVA (auf einem Panel) / Programmauszug:

```
public int iter(double cx, double cy) {
    // Iteration mit Kreis- und Zykloidentest
    int zaehler = 0;
    double zx = 0.0, zy = 0.0, zxq = 0.0, zyq = 0.0, r, s;
    r = cx * cx + cy * cy;    s = Math.sqrt(r - 0.5 * cx + 0.0625);
    if ((16.0 * r * s > 5.0 * s - 4.0 * cx + 1)&&
        ((cx + 1.0) * (cx + 1.0) + cy * cy > 0.0625))
        do {
            zy = 2.0 * zx * zy + cy;    zx = zxq - zyq + cx;
            zxq = zx * zx;    zyq = zy * zy;
            zaehler++;
        } while ((zxq + zyq <= 4.0) && (zaehler < maxIt));
        else zaehler = maxIt;
    return zaehler;
}

public void zeichne(Graphics g) { // Apfelmännchen plotten
    // g ist die jeweilige Grafikumgebung von panelPlot oder bufImg
    int imageBreite = maxX, imageHoehe = maxY;
    int sp, ze, zaehler;
    int yHalbe = imageHoehe / 2, y0Pkt = (int) (ye / (ye - ya) * imageHoehe);
    double cx, cy;
    double dx = (xe - xa) / imageBreite, dy = (ye - ya) / imageHoehe;
    . . .

    g.setColor(Color.black); // nur Mandelbrotmenge
    for (sp = 0; sp <= imageBreite; sp++) {
        cy = 0.0; // auf der x-Achse
        for (ze = yHalbe; ze <= imageHoehe; ze++) {
            if (iter(cx, cy) == maxIt) {
                g.drawLine(sp, ze, sp, ze);
                g.drawLine(sp, imageHoehe - ze, sp, imageHoehe - ze);
            }
            cy = cy - dy;
        } // for ze
        cx = cx + dx;
    } // for Sp
}
```

JAVA-Screenshot:

Apfelmännchen (C) Ac 02-2012

Eingaben:

xa=-2.05 xe=0.75
ya=-1.25 ye=1.25

maximal 500 Iterationen

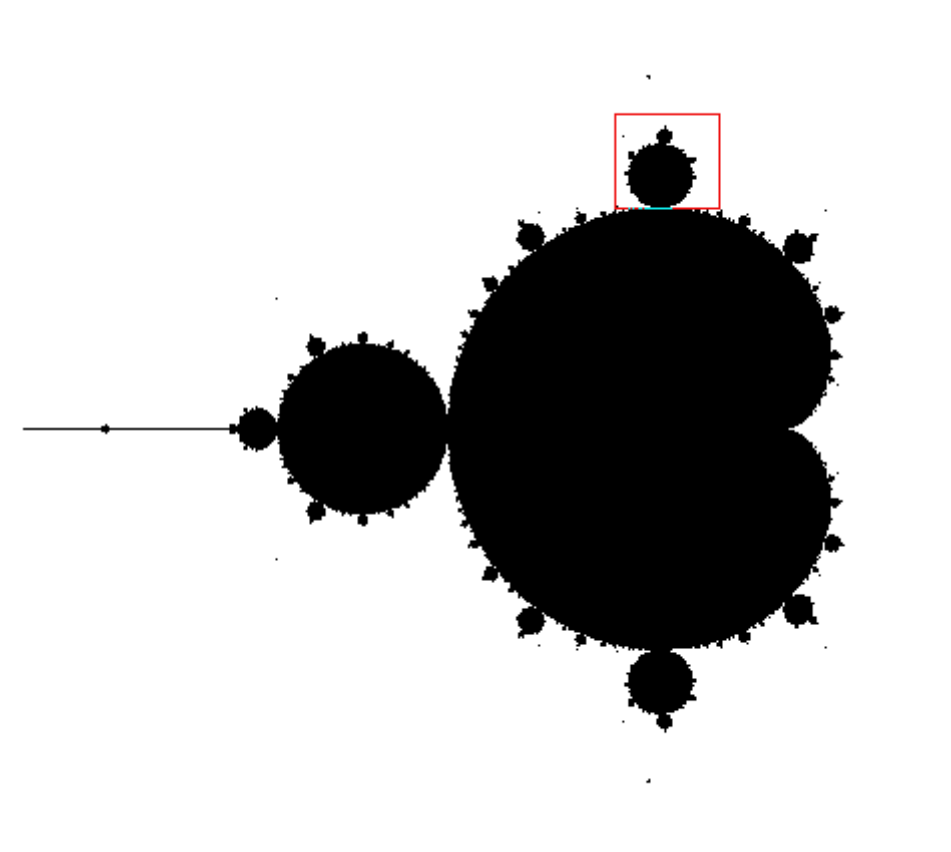
Darstellung
 schwarz farbig
 genauer viel genauer

Starten Achsen ein
 mit Rahmen Speichern
Reset als bmp

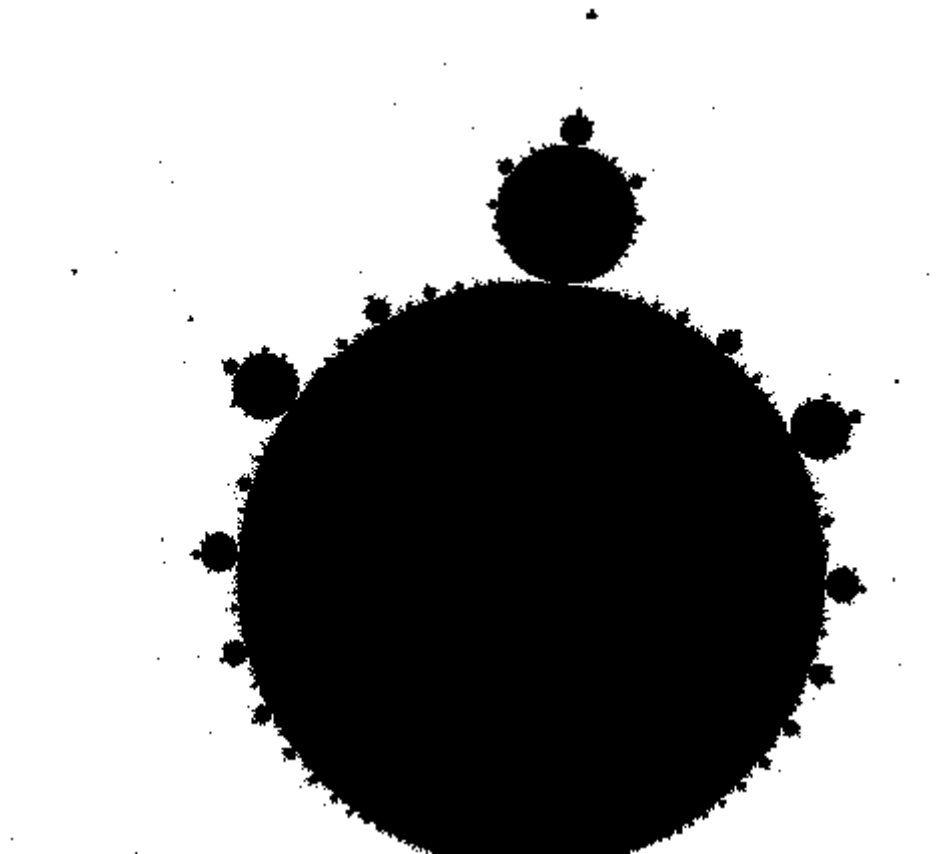
Beenden

Infos
Auflösung: 476*425 (202300Pixels)
davon 43694 (21%) schwarze Pixels
Farbe; x-Achsensymmetrie
Rechenzeit = 0,06 s
Mausposition: (-1,9852 | 0,8903)

Apfelmännchen „pur“ mit JAVA7:



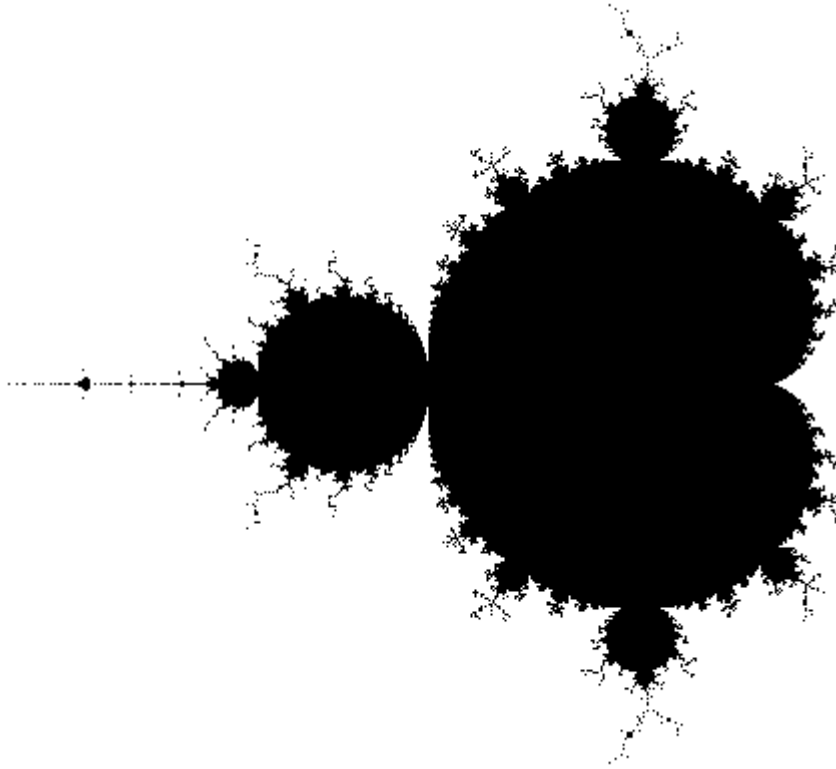
Der Ausschnitt mit $x \in [-0,29; 0,01]$, $y \in [0,65; 0,93]$ zeigt deutlich die Selbstähnlichkeit:



In den meisten Darstellungen wird das Apfelmännchen nur sehr ungenau dargestellt.
Daher erfolgt jetzt eine Darstellung mit ungleich präziserer Betrachtung, was natürlich
sehr viel Rechenzeit kostet:

Erweiterte Version mit größerer Genauigkeit (65^2 -fache Rechnung)

JAVA7: 44 sec



Erweiterte Version mit größerer Genauigkeit (513^2 -fache Rechnung)

JAVA7: 2670 sec

