Programmierpraktikum im WS 2009/2010

Denny Otten, V5-134, dotten@math.uni-bielefeld.de

Jens Rottmann–Matthes, V5-142, jrottman@math.uni-bielefeld.de

Pflichtaufgaben 2: Fraktale mit Matlab

Dieser Zettel besteht aus drei Teilen, es müssen zur erfolgreichen Teilnahme am Kurs **alle Pflichtaufgaben aus mindestens zwei der drei Teile** erfolgreich bearbeitet werden.

1 Lindenmayer-Systeme – Pflichtaufgaben Teil 1

Als erstes betrachten wir einfache *Lindenmayer-Systeme* zur Erzeugung von fraktaler Kurven. Dazu sei Ω ein Alphabet von Zeichen, $V \subset \Omega$ die Menge aller Variablen und $C \subset \Omega$ die Menge aller Konstanten, $\Omega = V \dot\cup C$. Ferner sei W die Menge aller Worte über Ω und P eine Abbildung $P : V \to W$ die *Produktionsregeln.* Mit diesen Bezeichnungen ist ein *Lindenmayer-System* gegeben als ein 4-Tupel $G = \{V, C, \omega, P\}$, wobei ω ein Wort aus W ist. Die Regeln werden iterativ auf jede Variable aus ω angewandt, wobei alle Produktionsregeln, gleichzeitig angewandt werden.

Beispiel. *Fibonacci-Zahlen:* $\Omega = \{A, B\}$, $V = \{A, B\}$, $C = \{\}$, $P(A) = B, P(B) = AB$, $\omega = A$, *damit:*
$n = 0$: A $\quad n = 1$: B $\quad n = 2$: AB
$n = 3$: BAB $\quad n = 4$: $ABBAB$ $\quad n = 5$: $BABABBAB$

Man kann Lindenmayer-Systeme mit Hilfe sogenannter Turtle-Grafik interpretieren. Eine abgespeckte, für unsere Zwecke völlig ausreichende Version liefert die folgende Tabelle:

Symbol	Aktion
F	Vorwärtsbewegung mit Zeichenstift
$+$	Linksdrehung um vorgegebenem Winkel δ
$-$	Rechtsdrehung um vorgegebenem Winkel δ

Implementierung in Matlab:

```
function [xdata,ydata]=tgr(txt,x0,y0,phi0,delta)
   N = length(find(txt=='F'));
   xdata = zeros(1,N); ydata = zeros(1,N);
   phi = phi0;
   xdata(1) = x0; ydata(1) = y0;
   n = 1;
   for j = txt
      switch j
      case '+'
         phi = phi+delta;
      case '-'
         phi = phi-delta;
      case 'F'
         xdata(n+1) = xdata(n)+cos(phi);
         ydata(n+1) = ydata(n)+sin(phi);
         n = n+1;
      otherwise
         error('Rule not specified.');
      end
   end
end
```

Beispiel (Koch Kurve)**.** *Die Koch Kurve hast Du schon auf dem dritten Blatt kennengelernt. Man kann sie auch mit dem Lindenmayer-System* (V, C, ω, P)*, gegeben durch*

$$V = \{F\}, \quad C = \{+, -\}, \quad P(F) = F + F - -F + F, \quad \omega = F, \quad \Omega = \{F, +, -\}$$

erzeugen. Dafür setzt man den Winkel δ *auf* $\frac{\pi}{3}$*, und erhält verschiedene Iterationsstufen der Kochkurve:*

```
function txt=lm(omega,produ,depth)
   txt = omega;
   pr = {'F',produ;'+','+';'-','-'};
   for j = 1:depth
      txtn = [];
      for t = txt
         txtn = [txtn,pr{strcmp(pr(:,1),t),2}];
      end
      txt = txtn;
   end
end
```

Dieses führt man dann wie folgt aus (dabei soll die Iterationstiefe 4 sein):

```
>> txt=lm('F','F+F--F+F',4);
>> [x,y]=tgr(txt,0,0,0,pi/3);
>> plot(x,y), axis equal;
```

Wie muss ω gewählt werden um die Koch-Schneeflocke zu bekommen?

Aufgabe 1 (**Pflichtaufgabe**). *Versuche die Programme* `tgr` *und* `lm` *zu verstehen und kommentiere den Programmcode. Zeichne die Minkowski-Kurve, die durch folgendes Lindenmayer-System gegeben ist: $\Omega = \{F, +, -\}$, $V = \{F\}$, $C = \{+, -\}$, $P(F) = F + F - F - FF + F + F - F$, $\omega = F$, der Winkel soll $\delta = \frac{\pi}{2}$ sein.*

Aufgabe 2 (**Pflichtaufgabe**). *Ändere die Programme so ab, dass Du auch die Drachenkurve, die durch das Lindenmayer-System mit $\Omega = \{F, G, +, -\}$, $V = \{F, G\}$, $C = \{+, -\}$, $P(F) = F + G +$, $P(G) = -F - G$, $\omega = F$ gegeben ist, erzeugen kannst. Der Winkel ist dabei $\delta = \frac{\pi}{2}$ und beim Zeichnen in der Turtle-Grafik soll G genau wie F behandelt werden! Erzeuge die Drachenkurve bis zur 12ten Iterationstiefe.*

2 Juliamengen und Apfelmännchen – Pflichtaufgaben Teil 2

Fraktale lassen sich nicht nur –wie oben gesehen– geometrisch erzeugen, sondern auch duch Iteration von Funktionen (d.h. wiederholte Anwendung einer Funktion).
Dazu startet man mit einem Punkt z_0 und führt die folgende Vorschrift aus

$$z_{n+1} = f(z_n), \quad n = 0, 1, 2, \ldots$$

Dabei kann man sich n als diskrete Zeit (etwa Jahre) vorstellen.
Besonders interessant ist es, wenn die Funktion f von $\mathbb{C} \to \mathbb{C}$ abbildet. Schon die quadratische Funktion

$$q_c : \mathbb{C} \to \mathbb{C},\ z \mapsto z^2 + c, \qquad c \in \mathbb{C} \text{ ein Parameter}$$

zeigt ein sehr überraschendes Verhalten. Durch Iteration mit dieser Funktion wird das berühmte Apfelmännchen-Fraktal erzeugt. Wir werden im Folgenden sehen, auf welche Weise das geschieht.

Aufgabe 3.

- *Wende die Abbildung q_c für $c = 0$, $c = -1.25$, und $c = 0.7i$ jeweils mehrfach auf die Punkte $z_0 = -0.1 + i$, $z_0 = -0.3 - 0.5i$, $z_0 = -0.7i$ an und stelle die Folge der Bilder $\{z_n\}_{n\in\mathbb{N}}$ mit der Funktion* `plot` *als Punkte in der komplexen Ebene dar. Verbinde die Punkte zusätzlich mit Linien um den Weg des Anfangspunktes unter der Abbildung q_c gut ablesen zu können.*
- *Schreibe eine reelle Version qr der Funktion q_c, d.h. definiere eine Funktion $qr_{cr,ci} : \mathbb{R}^2 \to \mathbb{R}^2$, die als Eingabe den Realteil und den Imaginärteil des Anfangswertes erhält und statt mit dem Parameter c mit dem Realteil $cr = Re(c)$ und dem Imaginärteil $ci = Im(c)$ arbeitet.*

Julia und Gefangenenmenge:

Alle Startpunkte z_0, für die die Folge $\{z_n\}_{n\in\mathbb{N}}$ mit

$$z_{n+1} = q_c(z_n) = z_n^2 + c, \quad n = 0, 1, 2, \ldots$$

für alle Zeiten beschränkt bleibt,bilden die *Gefangenenmenge* $G_c := \{z_0 : \limsup_{n\to\infty} |z_n| < \infty\}$. Der Rand dieser Menge $J_c := \partial G_c$ wird nach dem französischen Mathematiker Gaston Julia (1893–1978) *Juliamenge* genannt. Das Aussehen der Juliamenge bzw. der Gefangenenmenge ist je nach Wahl des Parameters $c \in \mathbb{C}$ sehr verschieden.

Aufgabe 4.
- *Definiere eine Funktion* `flucht`*, die für eine Zahl* $z_0 = x_0 + iy_0 \in \mathbb{C}$ *die Anzahl* n *der Iterationen ausgibt, nach denen für ein Folgenglied* z_n *gilt* $|z_n| > 2$*, (Man kann zeigen, dass die Folge* $\{z_n\}_{n\in\mathbb{N}}$ *für fast alle Punkte* z_0 *mit* $|z_0| > 2$ *divergiert).*
- *Eingabeparameter sollten der Realteil* x_0 *und Imaginärteil* y_0 *sein. Da man nicht unendlich viele Iterationsschritte testen kann, sollte die Iteration nach einer bestimmten Anzahl von Schritten abgebrochen werden (z.B. 25 Schritte). Die so definierte Funktion bildet also* $\mathbb{R} \times \mathbb{R}$ *auf* $\mathbb{N}$ *ab.*
- *Teste diese Funktion mit den in Aufgabe 3 gegebenen Startwerten* z_0 *und Parametern* c*.*

Das Bild der Gefangenenmenge erhält man, wenn man nicht nur einen einzigen Anfangswert $z_0 \in \mathbb{C}$ betrachtet, sondern für jeden solchen Anfangswert $z_0 = x_0 + iy_0$ aus einen bestimmten Gebiet das Ergebnis der Funktion `flucht` farblich kodiert zeichnet.

Aufgabe 5. *Benutze die Funktion* `pcolor`*,* `imagesc` *oder* `surf` *(*`pcolor(x,y, C), shading flat`*,* `imagesc(x,y,C)` *oder* `surf(x,y, C), shading flat`*) um die Funktion* ***`flucht`*** *in einem geeigneten Bereich* $x_0 = x_{\min}, ..., x_{\max}$*,* $y_0 = y_{\min}, ..., y_{\max}$ *in* $\mathbb{R} \times \mathbb{R}$ *zu zeichnen. Probiere dazu verschiedene Werte des Parameters* c *aus, etwa*

$$-1, \quad 0.3603 + 0.1i, \quad -0.122 + 0.745i, \quad i, \quad 0.113 - 0.6704i.$$

Welches ist die Gefangenenmenge G_c*?*

Aufgabe 6 (Pflichtaufgabe). *Schreibe eine Funktion* `flucht_reell`*, die das gleiche wie die Funktion* `flucht` *in reeller Arithmetik tut. Werte dazu die Operationen wie Multiplikation, Addition, etc. in Real- und Imaginärteil getrennt aus. Zeichne wie in Aufgabe 5.*
*(**Zusatz:** Vektorisiere die Funktionen* `flucht` *und* `flucht_reell` *und vergleiche die Laufzeiten.)*

Mandelbrotmenge:

Das *Apfelmännchen* bzw. die *Mandelbrotmenge*, benannt nach dem Mathematiker Benoît Mandelbrot (1924–), besteht aus den Parameterwerten $c \in \mathbb{C}$, für die der Startwert $z_0 = 0$ im Laufe der Iteration beschränkt bleibt. Man kann zeigen, dass die Mandelbrotmenge Teilmenge des Quadrates $\{c \in \mathbb{C} : |Re(c)| \leq 2, |Im(c)| \leq 2\}$ enthalten ist.
Das Apfelmännchen entsteht also ähnlich wie die Juliamenge:
Um zu bestimmen, ob ein Punkt $z \in \mathbb{C}$ in der Mandelbrotmenge liegt, startet man die Iteration $z_{n+1} = z_n^2 + c$ bei $z_0 = 0$ und rechnet solange, bis $|z_n| > 2$ ist. Jeder Punkt $c \in \mathbb{C}$ wird nach der Anzahl der Schritte eingefärbt, nach denen er den Kreis verlässt. Wenn der Punkt nach einer grossen Anzahl von Schritten (z.B. 25) den Kreis immer noch nicht verlassen hat, wird die Iteration abgebrochen.

Aufgabe 7 (Pflichtaufgabe). *Schreibe wie oben Funktionen* `mandelbrot` *und* `mandelbrot_reell`*, die für eine Zahl* $c \in \mathbb{C}$ *die Anzahl der Iterationen* n *ausgibt, nach denen* $|z_n| > 2$ *gilt. Benutze beide Funktionen um die Mandelbrotmenge (d.h. das Apfelmännchen) zu zeichnen.*
*(**Zusatz:** Vektorisiere die Funktionen* `mandelbrot` *und* `mandelbrot_reell`*.)*

3 Das Newtonfraktale – Pflichtaufgaben Teil 3

Das insbesondere in der numerischen Mathematik enorm wichtige Newton-Verfahren kann ebenfalls zur Erzeugung von Fraktalen benutzt werden.
Bei dem Newton-Verfahren handelt es sich um einen grundlegenden Algorithmus zur Bestimmung von Lösungen der Gleichung $f(x) = 0$ für $f : \mathbb{R}^m \to \mathbb{R}^m$. Das Verfahren besteht darin, dass man für einen geeigneten Startwert $x_0 \in \mathbb{R}^m$ die Iterationsvorschrift

$$x_{n+1} = x_n - Df(x_n)^{-1} f(x_n), \quad n = 0, 1, 2, ...$$

solange anwendet, bis der Defekt $\|f(x_n)\|$ unter einer vorgegebenen Toleranz liegt oder eine maximale Anzahl Iterationen erreicht ist. Der Index n bezeichnet hierbei den Iterationsindex und $Df(x)$ die Jacobimatrix $(\frac{\partial f_i(x)}{\partial x_j})_{i,j}$ an der Stelle $x \in \mathbb{R}^m$. Eine genauere Beschreibung findet sich in (vermutlich) jedem einführenden Buch in die numerische Mathematik, siehe auch `http://de.wikipedia.org/wiki/Newton_Verfahren`.
In Matlab kann man das mit Hilfe des Backslash Operators \ wie folgt implementieren:

```
tol     = 1e-8;          % Toleranz
maxsteps = 8;            % max. Anzahl Iterationen
x       = x0;            % Startwert der Iteration
defekt  = norm(f(x));    % vorgegebener Defekt
steps   = 0;             % Start im Nullten Schritt
while defekt > tol && steps < maxsteps
  xx = x - Df(x) \ f(x);
  x = xx;                % x aufdatieren
  defekt = norm(f(x));   % Defekt berechnen
  steps = steps + 1;     % Schritte weiterzaehlen
end
```

Beachte: `x = A\b` berechnet die Lösung x der Matrixgleichung $Ax = b$.

Aufgabe 8. *Erstelle aus obigem Code eine Funktion* `newton`*, die nach Eingabe der Funktionshandles für* `f`*,* `Df` *und des Startwertes* `x0` *die Newtoniteration durchführt und das Ergebnis der Iteration x_n (also den letzten x-Wert), sowie die benötigte Anzahl Schritte zurückgibt. Wähle für die Parameter* `tol` *und* `maxiter` *des Verfahrens die obigen Werte.*

Aufgabe 9. *Probiere die Funktion* `newton` *an den folgenden Beispielen aus*

$$f(x) = (x-2)(x+3), \qquad f(x) = \sin(x)e^x.$$

Starte mit verschiedenen Werten $x_0 \in \mathbb{R}$ und beobachte das Verhalten des Algorithmus (zum Beispiel im Debugger). Zeichne auch den Graphen von f und lokalisiere die Nullstellen.

Aufgabe 10 (Pflichtaufgabe). *Wende das Newtonverfahren auf die reelle Version der komplexen Funktion $f : \mathbb{C} \to \mathbb{C}, z \mapsto z^3 - 1$ an. Diese ist gegeben durch*

$$f(x,y) = \begin{pmatrix} x^3 - 3xy^2 - 1 \\ 3x^2y - y^3 \end{pmatrix}.$$

Die Jacobimatrix $Df(x,y)$ ist hier

$$Df(x,y) = \begin{pmatrix} 3(x^2-y^2) & -6xy \\ 6xy & 3(x^2-y^2) \end{pmatrix}.$$

Starte hierbei mit verschiedenen Werten (x_0, y_0) in der Nähe der drei Nullstellen

$$(x_1,y_1) = (1,0), \quad (x_2,y_2) = (-\frac{1}{2}, \frac{\sqrt{3}}{2}), \quad (x_3,y_3) = (-\frac{1}{2}, -\frac{\sqrt{3}}{2}).$$

Aufgabe 11 (Pflichtaufgabe). *Auch zum Newtonverfahren gibt es ein Fraktal, das sogenannte* ***Newton-Fraktal****, das wie folgt erzeugt wird: Führe das obige Newtonverfahren für verschiedene Anfangswerte in einem Bereich, der die Nullstellen umfasst durch. Färbe die Anfangswerte $(x_0, y_0) \in \mathbb{R}^2$ entweder danach,*

- *gegen welche der drei Nullstellen die Iteration konvergiert*

oder

- *nach wieviel Schritten die Iteration konvergiert (wie bei der Gefangenenmenge)*

und plotte das Ergebnis (z.B. wie in Aufgabe 5 mit `pcolor`*).*
Zeichne das Newton Fraktal für mindestens eins der folgenden Polynome

$$p_1(z) = (z+1)(z-1)(z+1+i)(z+1-i) \qquad p_2(z) = z(z-1)(z-i)(z-i+1)$$
$$p_3(z) = (z+1)(z-1)(z+1+i)(z+1+i)(z+1-i) \qquad p_4(z) = z^5 + 4z.$$