Proseminar: Forschungserfahrungen im Bachelor

Verallgemeinerung von Tupper's self referential formula

Ausarbeitung

eingereicht von

Ellen Rudolph

Betreuerin: Prof. Dr. Anke Pohl

12. Mai 2021

Inhaltsverzeichnis

1 Einleitung

Tupper's self referential formula (zu deutsch auch *Tuppers Formel*) ist eine Formel, die zur Darstellung von Bildern im kartesischen Koordinatensystem benutzt werden kann. Sie erschien 2001 im Paper *Reliable Two-Dimensional Graphing Methods for Mathematical Formulae with Two Free Variables* von Jeff Tupper [Tup01]. In dem Paper gibt Tupper sie als eine der von ihm entwickelten Methoden an, die in dem Grafik-Programm GrafEq$^{\mathrm{TM}}$ 2.10 [Ped] verwendet werden können. Größere Aufmerksamkeit erlangte die Formel unter anderem durch das Video „The 'Everything' Formula - Numberphile“ [HP15], welches 2015 auf dem YouTube-Kanal *Numberphile* erschien. Das Video inspierte auch dieses „Forschungserfahrungen im Bachelor“-Projekt.

Tuppers Formel lautet

$$\frac{1}{2} < \left\lfloor \mathrm{mod}\left(\left\lfloor \frac{y}{17} \right\rfloor \cdot 2^{-17\lfloor x \rfloor - \mathrm{mod}\,(\lfloor y \rfloor, 17)}, 2\right)\right\rfloor. \tag{1}$$

Um zu verstehen, wie diese Ungleichung Bilder darstellt, kann zunächst festgestellt werden, dass $\left\lfloor \frac{y}{17} \right\rfloor = \left\lfloor \frac{\lfloor y \rfloor}{17} \right\rfloor$ für alle $y \in \mathbb{R}_{\geq 0}$ gilt und dass somit alle x und y, in die Ungleichung eingesetzt, als ganze Zahlen $\lfloor x \rfloor$ und $\lfloor y \rfloor$ verrechnet werden. Um mit der Ungleichung ein BIld im Koordinatensystem zu erstellen, setzt man ein Zahlenpaar (x, y) in die Formel ein und überprüft, ob der Wert auf der rechten Seite größer als $\frac{1}{2}$ ist oder nicht. Genau dann, wenn die Bedingung erfüllt ist, wird an der Stelle (x, y) ein schwarzer Punkt eingezeichnet, ansonsten ein weißer. Da die rechte Seite aufgrund der Abrundungen von x und y für alle Zahlenpaare $(x', y') \in [x, x+1) \times [y, y+1)$ denselben Wert wie für (x, y) hat, entstehen im Graphen schwarze und weiße Kästchen der Größe 1×1. Diese Kästchen können als Pixel eines Schwarz-Weiß-Bildes interpretiert werden, womit alle 17 Pixel hohen Schwarz-Weiß-Bilder im Graphen von (1) vorkommen können [Tup01, S. 7]. Die Formel wird als *self-referential* (selbstverweisend) bezeichnet, da für $(x, y) \in [0, 105] \times [k, k+16]$ mit einem bestimmten Parameter k die Formel selbst – kodiert als 17 Pixel hohes und 106 Pixel breites Bild – dargestellt wird.

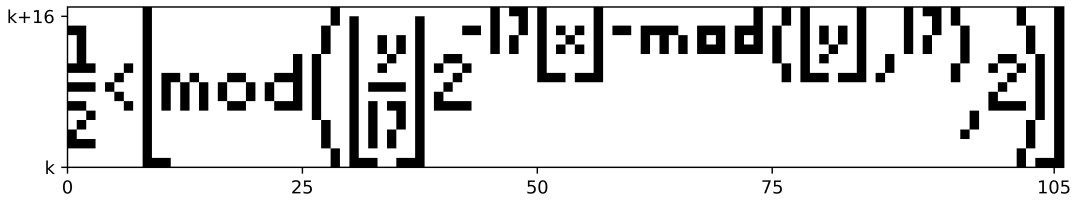

Abbildung 1: Tuppers Formel, Plot erstellt mit Python (siehe S. 12 ff.)

Dieses k lautet

$$
\begin{aligned}
k \;=\;& 4\,858\,450\,636\,189\,713\,423\,582\,095\,962\,494\,202\,044\,581\,400\,587\,983 \\
& 244\,549\,483\,093\,085\,061\,934\,704\,708\,809\,928\,450\,644\,769\,865\,524\,364 \\
& 849\,997\,247\,024\,915\,119\,110\,411\,605\,739\,177\,407\,856\,919\,754\,326\,571 \\
& 855\,442\,057\,210\,445\,735\,883\,681\,829\,823\,754\,139\,634\,338\,225\,199\,452 \\
& 191\,651\,284\,348\,332\,905\,131\,193\,199\,953\,502\,413\,758\,765\,239\,264\,874 \\
& 613\,394\,906\,870\,130\,562\,295\,813\,219\,481\,113\,685\,339\,535\,565\,290\,850 \\
& 023\,875\,092\,856\,892\,694\,555\,974\,281\,546\,386\,510\,730\,049\,106\,723\,058 \\
& 933\,586\,052\,544\,096\,664\,351\,265\,349\,363\,643\,957\,125\,565\,695\,936\,815 \\
& 184\,334\,857\,605\,266\,940\,161\,251\,266\,951\,421\,550\,539\,554\,519\,153\,785 \\
& 457\,525\,756\,590\,740\,540\,157\,929\,001\,765\,967\,965\,480\,064\,427\,829\,131 \\
& 488\,548\,259\,914\,721\,248\,506\,352\,686\,630\,476\,300 \\
\approx\;& 4,86 \cdot 10^{544}.
\end{aligned}
$$

Das Bild, das die Ungleichung an dieser Stelle im Koordinatensystem erzeugt, ist in Abbildung 1 zu sehen. Der Wert für k wurde hierbei nicht selbst ermittelt, sondern aus dem Blogeintrag [Shr11] übernommen.

In dieser Arbeit wird Tuppers Formel so verallgemeinert, dass mehrfarbige Bilder dargestellt werden können, die zudem beliebig hoch und breit sein dürfen. Außerdem wird gezeigt, wie für jedes Bild der entsprechende Definitionsbereich für die verallgemeinerte Formel ermittelt wird, sodass die Formel das Bild an dieser Stelle im Koordinatensystem erzeugt.

2 Verallgemeinerung der Formel

2.1 Vorbereitung des Satzes

Zunächst werden nach [MP11, S. 19] die modulo-Kongruenz definiert und dazugehörige Rechenregeln genannt.

Definition 1. Seien $a, b \in \mathbb{Z}$ und $n \in \mathbb{N}$. Dann ist a *kongruent* zu b modulo n, in Zeichen $a \equiv b \mod n$, falls $n|(a-b)$.
Falls a nicht kongruent zu b ist, schreibt man das als $a \not\equiv b \mod n$.
Insbesondere für nicht-ganzzahlige a und b wird auch der Ausdruck

$$b = \operatorname{mod}(a, n) = n \cdot \left(\frac{a}{n} - \left\lfloor \frac{a}{n} \right\rfloor \right) = a - n \cdot \left\lfloor \frac{a}{n} \right\rfloor$$

genutzt, wobei $\lfloor \cdot \rfloor$ die Abrundungsfunktion bezeichnet. Die Zahl b ist also der (nicht notwendigerweise ganzzahlige) Rest bei der Division von a mit n.

Lemma 1. *Seien* $a \equiv b \mod n$ *und* $c \equiv d \mod n$*. Dann gilt*

$$a + c \equiv b + d \mod n \quad \textit{und} \quad a \cdot c \equiv b \cdot d \mod n.$$

Weil die ursprüngliche Formel nur zwischen schwarzen und weißen Pixeln unterscheidet, reicht die Bedingung „$\frac{1}{2} < \lfloor \mathrm{mod}(\ldots, 2) \rfloor$“ aus, da $\lfloor \mathrm{mod}(\ldots, 2) \rfloor$ nur die Zahlen 0 und 1 annehmen kann. Um die Formel auf mehrere Farben anzupassen, wird die „wahr/falsch“-Notation nun geändert zu einer Funktion

$$\begin{aligned} f : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} &\to [0, c-1] \cap \mathbb{N}_0 \\ f(x, y) &= \lfloor \mathrm{mod}(\ldots, c) \rfloor, \end{aligned}$$

wobei c die Anzahl der verschiedenen Farben bezeichnet. Die Zahlen $0, \ldots, c-1$ entsprechen jeweils einem Farbwert im ursprünglichen Bild. Anstatt schwarzer Punkte bei Erfüllung der „<“-Bedingung, werden nun Punkte in der Farbe gezeichnet, die dem Funktionswert von f entspricht.

2.2 Verallgemeinerung der Formel

Die vorangegangenen Vorbereitungen führen nun zu folgendem Satz, dessen Formulierung und Beweis das Hauptergebnis dieses Projekts darstellt.

Satz 1 (Verallgemeinerung von Tuppers Formel)**.** *Sei A ein Bild mit einer Höhe von h und einer Breite von b Pixeln. Sei c die Anzahl der verschiedenen Farben des Bildes. Den Pixeln in A werden als Farbwerte die ganzen Zahlen in $[0, c-1]$ zugeordnet und das Bild als Matrix $(a_{ij}) \in [0, c-1]^{h \times b}$ aufgefasst.*
Dann existiert ein $k \in \mathbb{N}_0$ so, dass

$$\begin{aligned} f : [0, b-1] \times [k, k+h-1] &\to [0, c-1] \cap \mathbb{N}_0 \\ (x, y) &\mapsto \left\lfloor \mathrm{mod}\left(\left\lfloor \frac{y}{h} \right\rfloor \cdot c^{-h \cdot \lfloor x \rfloor - \mathrm{mod}\,(\lfloor y \rfloor, h)}, c \right) \right\rfloor \end{aligned}$$

das Bild A darstellt, also für $y = k + n$ gilt: $f(x, y) = a_{h-n, x+1}$ (schematisch dargestellt in Abb. 2).
Weiterhin wird k auf folgende Weise ermittelt:

1. *Die Farbwerte des Bildes werden, oben rechts beginnend, von oben nach unten und von rechts nach links notiert. Es entsteht also eine Zahlenfolge der folgenden Form:*

$$a_{1,b}\, a_{2,b}\, \ldots\, a_{h-1,b}\, a_{h,b}\, a_{1,b-1}\, \ldots\, a_{h,b-1}\, \ldots\, a_{1,1}\, a_{2,1}\, \ldots\, a_{h,1}.$$

2. *Die in Schritt 2 entstandene Zahlenfolge wird als Zahl k' in der c-adischen Zahlendarstellung aufgefasst und in die Dezimaldarstellung umgerechnet.*

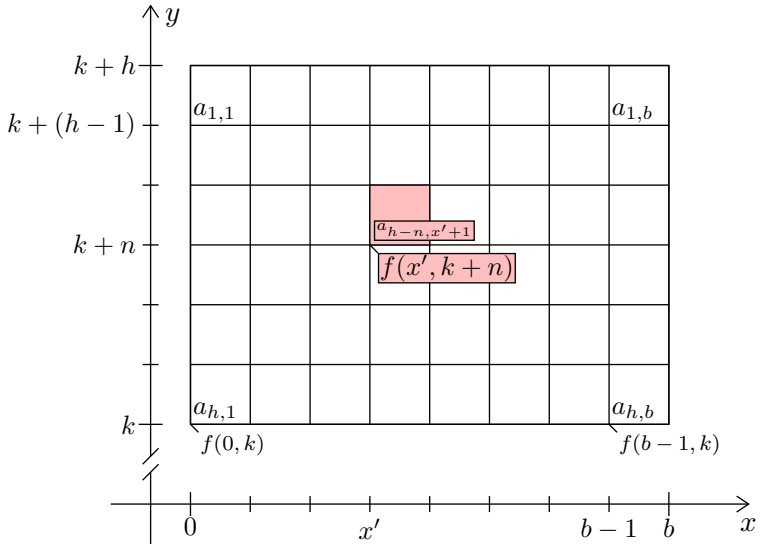

Abbildung 2: Zusammenhang Matrixeintrag – Funktionswert

3. k' wird mit der Höhe h multipliziert, um k zu erhalten.

Mathematisch ausgedrückt bedeutet dies

$$\begin{aligned} k' &= \sum_{j=0}^{b-1} \sum_{i=0}^{h-1} a_{h-i,j+1} \cdot c^{j \cdot h+i} \\ \Rightarrow \quad k &= h \cdot \sum_{j=0}^{b-1} \sum_{i=0}^{h-1} a_{h-i,j+1} \cdot c^{j \cdot h+i}. \end{aligned} \tag{2}$$

Beispiel 1 (Smiley)**.** Aus Abbildung 3 abzulesen ist die Höhe $h = 4$, die Breite $b = 5$ und die Anzahl der Farben $c = 3$. Den Farben weiß, rot und blau werden die Zahlen $0, 1, 2$ zugeordnet. Die Zuordnung der Zahl 0 zur Farbe weiß ist hierbei erfolgt, damit möglichst viele Koeffizienten der c-adischen Zahl k' wegfallen.

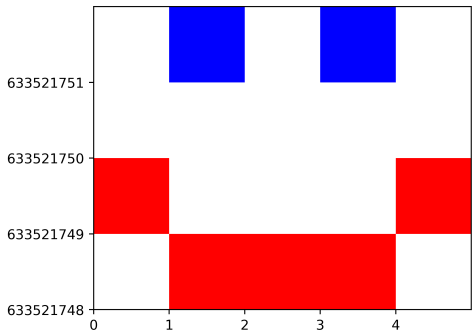

Abbildung 3: Smiley

Somit wird das Bild als Matrix A,

$$A = \begin{bmatrix} 0 & 2 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix},$$

interpretiert und kodiert zu

$$\begin{aligned} k' &= (0010\,2001\,0001\,2001\,0010)_3 \\ &= (1 \cdot 3^1 + 1 \cdot 3^4 + 2 \cdot 3^7 + 1 \cdot 3^8 + 1 \cdot 3^{12} + 2 \cdot 3^{15} + 1 \cdot 3^{17})_{10} \\ &= (158\,380\,437)_{10} = \frac{k}{4} \\ \Rightarrow \quad k &= 633\,521\,748. \end{aligned}$$

Im Definitionsbereich $[0, 4] \times [k, k+3]$ erzeugt f wieder das ursprüngliche Bild. Die Rechnung wird hier nun stichprobenartig für $(x, y) = (1, k+3)$ gezeigt.

$$\begin{aligned} f(1, k+3) &= \left\lfloor \operatorname{mod}\left(\left\lfloor \frac{k+3}{4} \right\rfloor \cdot 3^{-4\cdot 1 - \operatorname{mod}(k+3,4)},\ 3 \right) \right\rfloor \\ &= \left\lfloor \operatorname{mod}\left(\left\lfloor \frac{4k'+3}{4} \right\rfloor \cdot 3^{-4\cdot 1 - \operatorname{mod}(4k'+3,4)},\ 3 \right) \right\rfloor \\ &= \left\lfloor \operatorname{mod}\left(k' \cdot 3^{-4\cdot 1 - 3},\ 3 \right) \right\rfloor \\ &= \left\lfloor \operatorname{mod}\left(158380437 \cdot 3^{-7},\ 3 \right) \right\rfloor \\ &= \lfloor \operatorname{mod}(72419,03841,\ 3) \rfloor \\ &= \lfloor 2,03841 \rfloor \\ &= 2, \text{ was dem blauen Pixel des linken Smiley-Auges entspricht.} \end{aligned}$$

Die komplette Rechnung für Abbildung 3 ist mit Python erfolgt (siehe Anhang, S. 12).

Beispiel 2 („Super-Mario"-Blume)**.** Größere Bilder mit mehr Farben sind natürlich auch möglich, zum Beispiel eine Blume aus dem „Super Mario"-Universum, zu sehen in Abbildung 4.
Hier werden die Farben folgendermaßen übersetzt:

Farbe	weiß	schwarz	rot	orange	dunkelgrün	hellgrün
Zahl	0	1	2	3	4	5

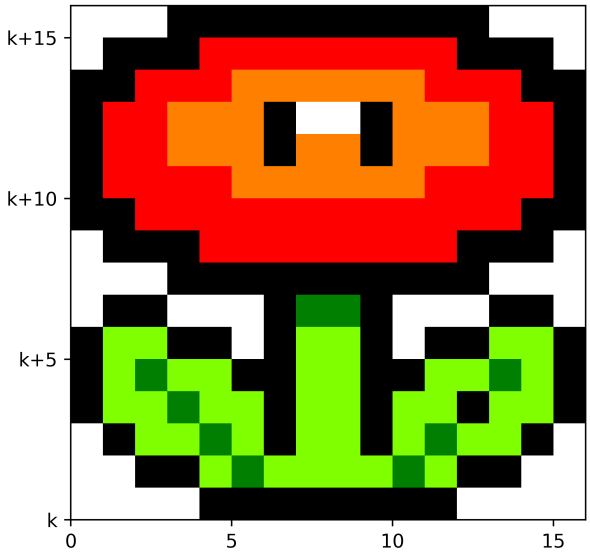

Abbildung 4: „Super-Mario“-Blume

Das Bild wird damit als Matrix B interpretiert,

$$B = \begin{bmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 0 \\
1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 2 & 2 & 1 & 1 \\
1 & 2 & 2 & 3 & 3 & 3 & 1 & 0 & 0 & 1 & 3 & 3 & 3 & 2 & 2 & 1 \\
1 & 2 & 2 & 3 & 3 & 3 & 1 & 3 & 3 & 1 & 3 & 3 & 3 & 2 & 2 & 1 \\
1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 2 & 2 & 2 & 1 \\
1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 \\
0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 4 & 4 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 5 & 5 & 1 & 1 & 0 & 1 & 5 & 5 & 1 & 0 & 1 & 1 & 5 & 5 & 1 \\
1 & 5 & 4 & 5 & 5 & 1 & 1 & 5 & 5 & 1 & 1 & 5 & 5 & 4 & 5 & 1 \\
1 & 5 & 5 & 4 & 5 & 5 & 1 & 5 & 5 & 1 & 5 & 5 & 1 & 5 & 5 & 1 \\
0 & 1 & 5 & 5 & 4 & 5 & 1 & 5 & 5 & 1 & 5 & 4 & 5 & 5 & 1 & 0 \\
0 & 0 & 1 & 1 & 5 & 4 & 5 & 5 & 5 & 5 & 4 & 5 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}.$$

Es gilt also $h = b = 16$ und $c = 6$. Mit der Formel (2) wird nun k berechnet:

$$\begin{aligned} k &= 1\,430\,579\,439\,309\,431\,169\,874\,125\,120\,864\,656\,533\,150\,313\,240\,707 \\ &\quad 517\,981\,188\,795\,239\,055\,383\,593\,686\,139\,218\,449\,399\,363\,713\,936\,545 \\ &\quad 797\,504\,249\,792\,103\,706\,704\,323\,732\,928\,013\,153\,590\,483\,749\,399\,854 \\ &\quad 915\,229\,085\,923\,328\,913\,778\,588\,781\,062\,627\,985\,119\,856\,387\,200 \\ &\approx 1,43 \cdot 10^{198}. \end{aligned}$$

Dieses k erzeugt dann mittels Python das Bild in Abbildung 4.

Nun folgt der Beweis für Satz 1.

Beweis. Zunächst einige Bemerkungen:

1. Wie bereits in der Einleitung für $h = 17$ erwähnt, gilt $\left\lfloor \frac{y}{h} \right\rfloor = \left\lfloor \frac{\lfloor y \rfloor}{h} \right\rfloor$. Somit kommen in der Abbildungsvorschrift von f die Zahlen x und y jeweils nur in der abgerundeten Form $\lfloor x \rfloor$ und $\lfloor y \rfloor$ vor. Der Einfachheit halber wird für diesen Beweis nun auf die $\lfloor\rfloor$-Notation verzichtet und x und y werden im Folgenden als nicht-negative ganze Zahlen verstanden.
2. Der Definitionsbereich für y ist $[k, k + h - 1]$. Somit lässt sich y darstellen als $y = k + n$ für ein $n \in [0, h - 1]$.
3. Es gilt
$$\left\lfloor \frac{y}{h} \right\rfloor = k',$$
denn
$$\begin{aligned} \left\lfloor \frac{y}{h} \right\rfloor &\overset{2}{=} \left\lfloor \frac{k+n}{h} \right\rfloor \\ &= \left\lfloor \frac{k}{h} + \frac{n}{h} \right\rfloor \\ &= \left\lfloor k' + \frac{n}{h} \right\rfloor \\ &= k', \quad \text{da } 0 \leq \frac{n}{h} < 1. \end{aligned}$$
4. Es gilt
$$\operatorname{mod}(y, h) = \operatorname{mod}(k + n, h) = n,$$
da $k \equiv 0 \mod h$ nach Konstruktion von k.
5. Sowohl a_{ij} als auch c nehmen nur nicht-negative Werte an, weshalb der Fall „< 0“ in keiner der folgenden Rechnungen berücksichtigt werden muss.

Mit diesen Bemerkungen gilt nun

$$\begin{aligned}
f(x,y) &= \left\lfloor \mathrm{mod}\left(\left\lfloor \frac{y}{h}\right\rfloor \cdot c^{-h\cdot\lfloor x\rfloor - \bmod(\lfloor y\rfloor,h)}, c\right)\right\rfloor \\
&\overset{\text{Bem. 1}}{=} \left\lfloor \mathrm{mod}\left(\left\lfloor \frac{y}{h}\right\rfloor \cdot c^{-h\cdot x - \bmod(y,h)}, c\right)\right\rfloor \\
&\overset{\text{Bem. 2}}{=} \left\lfloor \mathrm{mod}\left(\left\lfloor \frac{k+n}{h}\right\rfloor \cdot c^{-h\cdot x - \bmod(k+n,h)}, c\right)\right\rfloor \\
&\overset{\text{Bem. 3}}{=} \left\lfloor \mathrm{mod}\left(k' \cdot c^{-h\cdot x - \bmod(k+n,h)}, c\right)\right\rfloor \\
&\overset{\text{Bem. 4}}{=} \left\lfloor \mathrm{mod}\left(k' \cdot c^{-h\cdot x - n}, c\right)\right\rfloor \\
&\overset{\text{Formel (2)}}{=} \left\lfloor \mathrm{mod}\left(\left(\sum_{j=0}^{b-1}\sum_{i=0}^{h-1} a_{h-i,j+1}\cdot c^{jh+i}\right)\cdot c^{-hx-n},\ \ c\right)\right\rfloor
\end{aligned}$$

und es ist zu zeigen:

$$f(x,y) = \left\lfloor \mathrm{mod}\left(\left(\sum_{j=0}^{b-1}\sum_{i=0}^{h-1} a_{h-i,j+1}\cdot c^{jh+i}\right)\cdot c^{-hx-n},\ \ c\right)\right\rfloor = a_{h-n,x+1}.$$

Zunächst wird die linke Seite des modulo-Ausdrucks zusammengefasst.

$$\begin{aligned}
f(x,k+n) &= \left\lfloor \mathrm{mod}\left(\left(\sum_{j=0}^{b-1}\sum_{i=0}^{h-1} a_{h-i,j+1}\cdot c^{jh+i}\right)\cdot c^{-hx-n},\ \ c\right)\right\rfloor \\
&= \left\lfloor \mathrm{mod}\left(\sum_{j=0}^{b-1}\sum_{i=0}^{h-1} a_{h-i,j+1}\cdot c^{jh+i-hx-n},\ \ c\right)\right\rfloor \\
&= \left\lfloor \mathrm{mod}\left(\underbrace{\sum_{j=0}^{b-1}\sum_{i=0}^{h-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n}}_{=:A},\ \ c\right)\right\rfloor
\end{aligned}$$

Nun wird A weitestgehend vereinfacht, indem folgende Eigenschaft genutzt wird, die aus Lemma 1 folgt.

$$\begin{aligned}
& & c &\equiv 0 \mod c & & \\
\Rightarrow & & c^k &\equiv 0 \mod c & & \text{für } k \geq 1 \\
\Rightarrow & & a_{i,j}\cdot c^k &\equiv 0 \mod c & & \text{für alle } i,j \text{ und } k \geq 1 \\
\Rightarrow & \sum_j\sum_i a_{h-i,j+1}\cdot c^{(j-x)h+i-n} &&\equiv 0 \mod c, & & \text{wenn } (j-x)h+i-n \geq 1.
\end{aligned}$$

$$
\begin{aligned}
A = &\sum_{j=0}^{b-1}\left(\sum_{i=0}^{n-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n} + a_{h-n,j+1}\cdot c^{(j-x)h} + \sum_{i=n+1}^{h-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n}\right)\\
= &\sum_{j=0}^{x-1}\left(\sum_{i=0}^{n-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n} + a_{h-n,j+1}\cdot c^{(j-x)h} + \sum_{i=n+1}^{h-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n}\right)\\
&+\sum_{i=0}^{n-1} a_{h-i,x+1}\cdot c^{i-n} + a_{h-n,x+1}\cdot c^{0} + \sum_{i=n+1}^{h-1} a_{h-i,x+1}\cdot c^{i-n}\\
&+\sum_{j=x+1}^{b-1}\left(\sum_{i=0}^{n-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n} + a_{h-n,j+1}\cdot c^{(j-x)h} + \sum_{i=n+1}^{h-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n}\right)\\
= &\sum_{j=0}^{x-1}\sum_{i=0}^{n-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n} + \sum_{j=0}^{x-1} a_{h-n,j+1}\cdot c^{(j-x)h} + \sum_{j=0}^{x-1}\sum_{i=n+1}^{h-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n}\\
&+\sum_{i=0}^{n-1} a_{h-i,x+1}\cdot c^{i-n} + a_{h-n,x+1}\cdot c^{0} + \underbrace{\sum_{i=n+1}^{h-1} a_{h-i,x+1}\cdot c^{i-n}}_{\substack{\equiv 0 \mod c,\\ \text{da } i-n\geq 1}}\\
&+\underbrace{\sum_{j=x+1}^{b-1}\sum_{i=0}^{n-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n}}_{\substack{\equiv 0 \mod c,\\ \text{da } (j-x)h+i-n\geq h-n\geq 1}} + \underbrace{\sum_{j=x+1}^{b-1} a_{h-n,j+1}\cdot c^{(j-x)h}}_{\substack{\equiv 0 \mod c,\\ \text{da } (j-x)h\geq h\geq 1}} + \underbrace{\sum_{j=x+1}^{b-1}\sum_{i=n+1}^{h-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n}}_{\substack{\equiv 0 \mod c,\\ \text{da } (j-x)h+i-n\geq 1+1\geq 1}}\\
\equiv &\underbrace{\sum_{j=0}^{b-1}\sum_{i=0}^{h-1} a_{h-i,j+1}\cdot c^{(j-x)h+i-n}}_{=:B_1} + \underbrace{\sum_{i=0}^{n-1} a_{h-i,x+1}\cdot c^{i-n}}_{=:B_2} + a_{h-n,x+1}\cdot c^{0} \mod c.
\end{aligned}
$$

Als nächstes wird gezeigt, dass $B_1 + B_2 < 1$ gilt.

$$
\begin{aligned}
B_1 &= \sum_{j=0}^{b-1}\sum_{i=0}^{h-1} \underbrace{a_{h-i,j+1}}_{\leq c-1}\cdot c^{(j-x)h+i-n}\\
&\leq \sum_{j=0}^{b-1}\sum_{i=0}^{h-1}(c-1)\cdot c^{(j-x)h+i-n}\\
&= \sum_{j=0}^{b-1}\sum_{i=0}^{h-1}\left(c^{(j-x)h+i-n+1} - c^{(j-x)h+i-n}\right)\\
&= \sum_{j=0}^{b-1}\left(c^{(j-x)h-n+(h-1)+1} - c^{(j-x)h-n+0}\right) \qquad \text{(Teleskopsumme)}\\
&= \sum_{j=0}^{b-1}\left(c^{(j-x+1)h-n} - c^{(j-x)h-n}\right)\\
&= c^{((b-1)-x+1)h-n} - c^{(0-x)h-n} \qquad \text{(Teleskopsumme)}\\
&= c^{(b-x)h-n} - c^{-xh-n}.
\end{aligned}
$$

$$
\begin{aligned}
B_2 &= \sum_{i=0}^{n-1} \underbrace{a_{h-i,x+1}}_{\leq c-1} \cdot c^{i-n} \\
&\leq \sum_{i=0}^{n-1} (c-1) \cdot c^{i-n} \\
&= \sum_{i=0}^{n-1} c^{i-n+1} - c^{i-n} \\
&= c^{(n-1)-n+1} - c^{0-n} \qquad \text{(Teleskopsumme)} \\
&= c^0 - c^{-n}.
\end{aligned}
$$

Somit gilt

$$
\begin{aligned}
B_1 + B_2 \leq\ & c^{(b-x)h-n} - c^{-xh-n} + c^0 - c^{-n} \\
&= c^{-n}\left(c^{(b-x)h} - c^{-xh} - 1\right) + 1 \\
&= 1 + \frac{c^{(b-x)h} - c^{-xh} - 1}{c^n} \\
&= 1 - \frac{1 + c^{-xh} - c^{(b-x)h}}{c^n} \\
&= 1 - \frac{1 + c^{-xh}(1 - c^{bh})}{c^n} \\
&= 1 - \frac{1 + \frac{1-c^{bh}}{c^{xh}}}{c^n} \\
&< 1, \text{ wenn } 0 < 1 + \frac{1 - c^{bh}}{c^{xh}} < c^n.
\end{aligned}
$$

Angenommen $1 + \frac{1 - c^{bh}}{c^{xh}} \geq c^n$. Dann folgt durch Multiplikation mit c^{xh}

$$
\begin{aligned}
& c^{xh} + 1 - c^{bh} \geq c^{nxh} \\
\Rightarrow \qquad & 1 \geq c^{nxh} - c^{xh} + c^{bh} \\
& = c^h(c^{nx} - c^x + c^b) \\
& = \underbrace{c^h(\underbrace{c^x(c^n - 1)}_{\geq 0} + c^b)}_{\geq c^{hb} \geq 2^{hb} \geq 1} \qquad \lightning.
\end{aligned}
$$

Angenommen $1 + \frac{1 - c^{bh}}{c^{xh}} \leq 0$. Dann folgt durch Multiplikation mit c^{xh}

$$
\begin{aligned}
& c^{xh} + 1 - c^{bh} \leq 0 \\
\Rightarrow \qquad & 1 \leq c^{bh} - c^{xh} \\
& = \underbrace{c^h(\underbrace{c^b - c^x}_{\leq 0})}_{\leq 0} \qquad \lightning.
\end{aligned}
$$

Also gilt $B_1 + B_2 < 1$ und somit insgesamt

$$\begin{aligned} f(x, k+n) &= \left\lfloor \operatorname{mod}\left(\sum_{j=0}^{b-1} \sum_{i=0}^{h-1} a_{h-i,j+1} \cdot^{(j-x)h+i-n},\ c \right) \right\rfloor \\ &= \left\lfloor \operatorname{mod}\left(\sum_{j=0}^{b-1} \sum_{i=0}^{h-1} a_{h-i,j+1} \cdot c^{(j-x)h+i-n} \ +\ \sum_{i=0}^{n-1} a_{h-i,x+1} \cdot c^{i-n} \ +\ a_{h-n,x+1} \cdot c^0,\ c \right) \right\rfloor \\ &= \left\lfloor \sum_{j=0}^{b-1} \sum_{i=0}^{h-1} a_{h-i,j+1} \cdot c^{(j-x)h+i-n} \ +\ \sum_{i=0}^{n-1} a_{h-i,x+1} \cdot c^{i-n} \ +\ a_{h-n,x+1} \right\rfloor \\ &= a_{h-n,x+1}. \end{aligned}$$

□

3 Ausblick: Effiziente Codierung von Bildern

Weiterführend könnte noch untersucht werden, welche Möglichkeiten es gibt, ein Bild so zu kodieren, dass die Zahl k selbst sowie die Rechenzeit für ihre Berechnung möglichst gering ist. Dabei bieten sich vor allem folgende zwei Möglichkeiten an:

1. In Beispiel 1 wurde die Zahl 0 der Farbe weiß zugeordnet, damit möglichst viele Koeffizienten der c-adischen Zahl k' verschwinden. Dieses Vorgehen kann man weiterführen, indem der zweithäufigsten Farbe die Zahl 1 zugeordnet wird, der dritthäufigsten die Zahl 2 und so weiter. Das Vorgehen wurde in beiden Beispielen angewendet.
2. Die Farben werden so verteilt, dass die führende Ziffer von k' in c-adischer Darstellung - also die Farbe oben rechts im Bild - 0 wird. Somit fällt mindestens die größte Potenz von c weg, was die Zahl signifikant kleiner macht. Für die weiteren Zahlen kann man dann weiter so vorgehen: der nächsten Ziffer, die in der c-adischen Zahlenfolge von k' kommen würde, wird die Zahl 1 zugeordnet, dann 2, dann 3 usw.

Man könnte untersuchen, ob eine der beiden Möglichkeiten für alle möglichen Bilder effizienter ist, oder ob eine Kombination der Möglichkeiten oder sogar ein ganz anderes Vorgehen die besten Ergebnisse liefert. So könnte ermittelt werden, welche Möglichkeit der Kodierung unter Berücksichtigung von Speicherverbrauch und Rechenleistung die effizienteste ist.

Mit den Ergebnissen dieser Untersuchung könnte dann das Python-Skript aus dem Anhang so erweitert werden, dass ein Bild in einem gängigen Dateiformat für Bilder eingelesen werden kann und dieses zu einer Matrix konvertiert wird.

A Anhang

Es folgt der kommentierte Python-Code, der für die Berechnungen sowie die Erstellung der Bilder in dieser Ausarbeitung verwendet wurde.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# %% Funktionen

"""
Funktion die k aus dem Bild und der Anzahl der Farben generiert
(numerisch)
    Input:
        img - das Bild als Matrix
        num_colors - die Anzahl der verschiedenen Farben als int
    Output:
        k - das generierte k als int
"""
def get_k_num(img, num_colors):

    c = num_colors
    k_string = []

    """
    Die Matrix-Indizes werden rechts oben beginnend
    spaltenweise iteriert und die entsprechenden Eintraege
    in einem Array gespeichert
    """
    for j in range(1, img.shape[1]+1):
        for i in range(0, img.shape[0]):
            entry = img[i, -j]
            k_string.append(entry)

    """
    Die Zahlenkette wird als Zahl zur Basis c betrachtet und
    in Dezimaldarstellung umgerechnet
    """
    k = 0
    for l in range(1, len(k_string)+1):
        temp = k_string[-l]
        k = k + ( temp * (c**(l-1)) )
    k = k * img.shape[0]

    return int(k)

"""
Funktion die k aus dem Bild und der Anzahl der Farben generiert
(analytisch)
      Input:
          img - das Bild als Matrix
          num_colors - die Anzahl der verschiedenen Farben als int
    Output:
        k - das generierte k als int
"""
def get_k_ana(img, num_colors):

```

```
54      c = num_colors
55      (h,b) = img.shape
56      k = 0
57
58      """
59      k wird anhand der Formel im Satz berechnet
60      """
61      for j in range(b):
62          for i in range(h):
63              k += img[h-i-1, j] * c**(j*h+i)
64
65      return int(k*h)
66
67  """
68  Funktion die aus k, den Dimensionen des Bildes und der Anzahl
69  der Farben wieder das Bild erzeugt
70      Input:
71          height - Hoehe des Bildes
72          width - Breite des Bildes
73          num_colors - Anzahl der verschiedenen Farben
74          k - k aus dem Satz
75      Output:
76          img - Das Bild als Matrix
77  """
78  def make_img(height, width, num_colors, k):
79
80      img = np.zeros( (height, width), dtype='int')
81      k_small = k // height # k' aus dem Satz
82
83      for i in range(0, height):
84          for j in range(0, width):
85              """
86              Die ersten 2 Anmerkungen aus dem Beweis wurden
87              verwendet, um die Berechnung effizienter zu machen
88              """
89              temp = k_small // (num_colors**( height*j + i ))
90              img[i, j] = int(temp % num_colors)
91
92      return img
93
94  # %% Beispiel 1 (tuppers formula)
95
96  k_1 = 48584506361897134235820959624942020445814005879832445494830930850619347047080
97  """
98  k_1  =  48584506361897134235820959624942020445814005879832445494
99          30930850619347047088099284506447698655243648499972470249 1
100         51191104116057391774078569197543265718554420572104457358 8
101         36818298237541396343382251994521916512843483329051311931 9
102         99535024137587652392648746133949068701305622958132194811 1
103         36853395355652908500238750928568926945559742815463865107 3
104         00491067230589335860525440966643512653493636439571255656 9
105         59368151843348576052669401612512669514215505395545191537 8
106         54575257565907405401579290017659679654800644278291314885 4
107         82599147212485063526866304763 00
108 """
109 height_1 = 17
110 width_1 = 106
111 num_colors_1 = 2
```

```
112
113 """
114 erstellt das Bild aus den gegebenen Daten
115 """
116 pic_1 = make_img(height_1, width_1, num_colors_1, k_1)
117
118 """
119 plottet das rekonstruierte Bild und speichert dieses
120 """
121 fig = plt.figure(figsize=(10,40), dpi = 800)
122 ax = fig.add_subplot()
123 extent = (0, pic_1.shape[1], 0, pic_1.shape[0])
124 ax.imshow(pic_1, origin="lower", cmap = "Greys", extent=extent)
125 ax.set_xticks([0, 25, 50, 75, 105])
126 ax.set_yticks([0, 16])
127 ax.set_yticklabels(["k", "k+16"])
128 plt.show()
129 fig.savefig('tupper_plot.png', bbox_inches='tight')
130
131 # %% Besipiel 2 (smiley face)
132 """
133 Smiley-Bild als Matrix
134 """
135 orig_img_2 = [0,2,0,2,0,
136               0,0,0,0,0,
137               1,0,0,0,1,
138               0,1,1,1,0]
139 orig_img_2 = np.reshape(orig_img_2, (4,5))
140
141 (height_2, width_2) = orig_img_2.shape
142 num_colors_2 = 3
143 k_2 = get_k_num(orig_img_2, num_colors_2)
144
145 """
146 Rekonstruktion des Bildes
147 """
148 recon_img_2 = make_img(height_2, width_2, num_colors_2, k_2)
149
150 """
151 Definition der Farben als RGBA-Werte
152 """
153 blue = [0, 0, 1, 1]
154 red = [1, 0, 0, 1]
155 white = [1, 1, 1, 1]
156 my_colors_2 = [white, red, blue]
157 colors_2 = ListedColormap(my_colors_2)
158
159 """
160 plottet das rekonstruierte Bild und speichert dieses
161 """
162 fig = plt.figure(figsize=(5,5), dpi = 500)
163 ax = fig.add_subplot()
164 extent = (0, recon_img_2.shape[1], 0, recon_img_2.shape[0])
165 ax.imshow(recon_img_2, origin="lower", cmap=colors_2, extent=extent)
166 ax.set_xticks(np.arange(0, width_2, 1))
167 ax.set_yticks(np.arange(0, height_2, 1))
168 ax.set_xticklabels(np.arange(0, width_2, 1))
169 ax.set_yticklabels(np.arange(k_2, k_2 + height_2, 1))
```

```
plt.show()
fig.savefig('smiley.png', bbox_inches='tight')

# %% Beispielbild 3 (Pixel-Art von Super Mario Blume)
"""
Flower-Bild als Matrix
"""
orig_img_3 = [0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,
               0,1,1,1,2,2,2,2,2,2,2,2,1,1,1,0,
               1,1,2,2,2,3,3,3,3,3,3,2,2,2,1,1,
               1,2,2,3,3,3,1,0,0,1,3,3,3,2,2,1,
               1,2,2,3,3,3,1,3,3,1,3,3,3,2,2,1,
               1,2,2,2,2,3,3,3,3,3,3,2,2,2,2,1,
               1,1,2,2,2,2,2,2,2,2,2,2,2,2,1,1,
               0,1,1,1,2,2,2,2,2,2,2,2,1,1,1,0,
               0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,
               0,1,1,0,0,0,1,4,4,1,0,0,0,1,1,0,
               1,5,5,1,1,0,1,5,5,1,0,1,1,5,5,1,
               1,5,4,5,5,1,1,5,5,1,1,5,5,4,5,1,
               1,5,5,4,5,5,1,5,5,1,5,5,1,5,5,1,
               0,1,5,5,4,5,1,5,5,1,5,4,5,5,1,0,
               0,0,1,1,5,4,5,5,5,5,4,5,1,1,0,0,
               0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0]
orig_img_3 = np.reshape( orig_img_3, (16,16))

(height_3, width_3) = orig_img_3.shape
num_colors_3 = 6
k_3 = get_k_num(orig_img_3, num_colors_3)

"""
Rekonstruktion des Bildes
"""
recon_img_3 =  make_img(height_3, width_3, num_colors_3, k_3)

"""
Definition der zusaetzlichen Farben
"""
black = [0, 0, 0, 1]
orange = [1, .5, 0, 1]
light_green = [0.5, 1, 0, 1]
dark_green = [0, 0.5, 0, 1]
my_colors_3 = [white, black, red, orange, dark_green, light_green]
colors_3 = ListedColormap(my_colors_3)

"""
plottet das rekonstruierte Bild und speichert dieses
"""
fig = plt.figure(figsize=(5,5), dpi=(500))
ax = fig.add_subplot()
extent = (0, recon_img_3.shape[1], 0, recon_img_3.shape[0])
ax.imshow(recon_img_3, origin="lower", cmap=colors_3, extent=extent)
ax.set_xticks(np.arange(0, width_3, 5))
ax.set_yticks([0, 5, 10, height_3-1])
ax.set_xticklabels(np.arange(0, width_3, 5))
ax.set_yticklabels(["k", "k+5", "k+10", "k+" + str(height_3-1)])
plt.show()
fig.savefig('flower.png', bbox_inches='tight')
```

Quellen

[HP15] Brady Haran und Matt Parker. *The 'Everything' Formula - Numberphile*. 2015. URL: https://www.youtube.com/watch?v=_s5RFgd59ao&ab_channel=Numberphile (besucht am 27. 03. 2021).

[MP11] Stefan Müller-Stach und Jens Piontkowski. *Elementare und algebraische Zahlentheorie : ein moderner Zugang zu klassischen Themen*. 2., erw. Aufl. Algebra und Zahlentheorie. Wiesbaden: Vieweg + Teubner, 2011.

[Ped] Pedagoguery Software Inc. GrafEq^{TM}. URL: http://www.peda.com/grafeq/ (besucht am 20. 03. 2021).

[Shr11] Shreevatsa. *How does Tupper's self-referential formula work?* 12. Apr. 2011. URL: https://shreevatsa.wordpress.com/2011/04/12/how-does-tuppers-self-referential-formula-work/ (besucht am 10. 03. 2021).

[Tup01] Jeff Tupper. "Reliable Two-Dimensional Graphing Methods for Mathematical Formulae with Two Free Variables". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: Association for Computing Machinery, 2001.