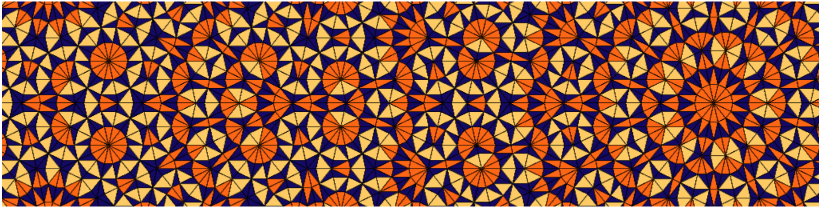
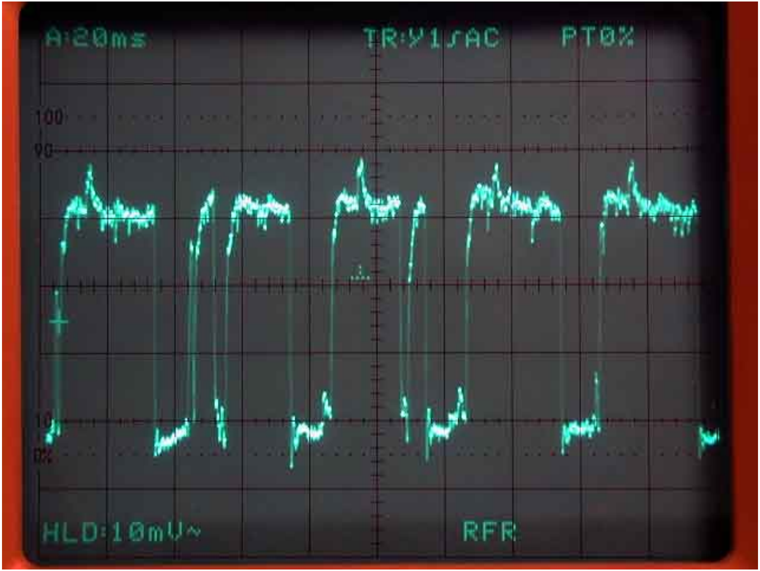
Panorama der Mathematik und Informatik

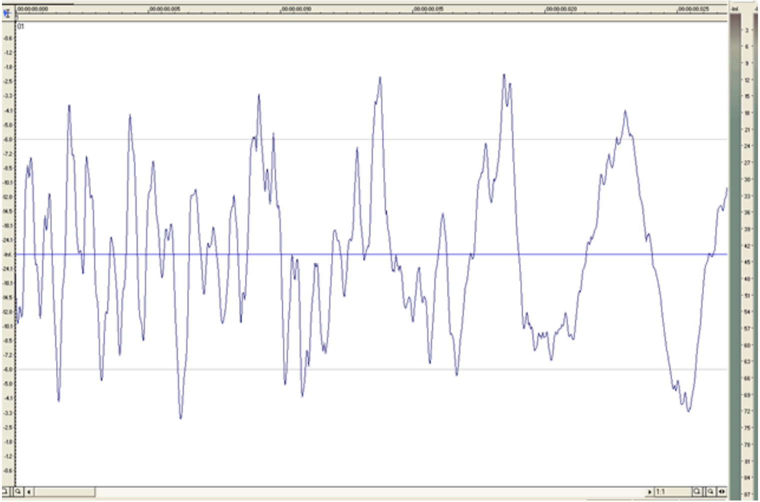
22: Algorithmen V: jpeg und Co

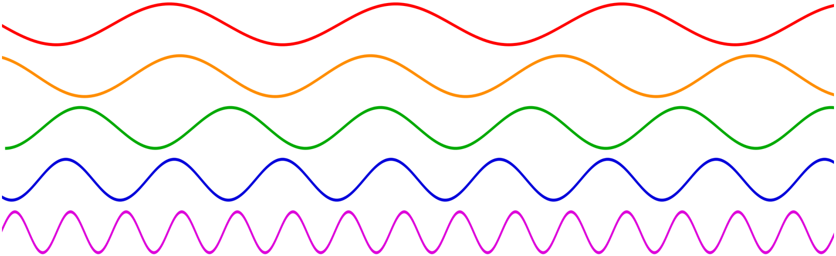
Dirk Frettlöh
Technische Fakultät / Richtig Einsteigen



Inspiration zu jpeg: Frequenzanalyse von Signalen.



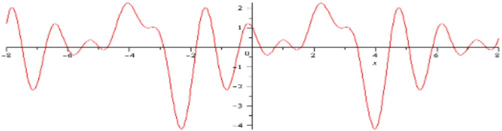




$\sin(x)$, $\sin(2x)$, $\sin(3x)$, $\cos(4x)$, $\sin(5x)$,

Ziel: Signal als Kombination "reiner" Töne darstellen. Reine Töne entsprechen Sinus- und Kosinusschwingungen $\sin(nx)$ bzw $\cos(nx)$ ($n \in \mathbb{N}$), also mit Periode $\frac{2\pi}{n}$,

Beispiel: Ein Signal als Kombination reiner Töne:



$$f(x) = \sin(x) + 0.1\sin(2x) - \sin(3x) + \cos(4x) + 0.7\sin(5x) + \cdots$$

Meist ist das Problem: Gegeben ein Signal f, wie bekommt man die Vorfaktoren? (hier: $1, 0.1, -1, 1, 0.7, \ldots$)

Allgemein ist das Problem also:

Wenn $f(x)$ eine gegebene Funktion ist, wie bestimmt man die a_n und b_n?

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos(nx) + b_n \sin(nx)\right)$$

(Vgl. Potenzreihen von Funktionen: $f(x) = \sum_{n=0}^{\infty} a_n x^n$,
da ist die Antwort: Taylorreihe (um a=0))

Dies hier heißt *Fourier-Reihe* von f.
(kommt in "Mathematische Methoden der Biowissenschaften III")
Antwort:

$$\boxed{a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(s) \cos(ns) ds \quad (n \in \mathbb{N}_0)}$$

$$\boxed{b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(s) \sin(ns) ds \quad (n \in \mathbb{N})}$$

(siehe H. Heuser: *Gewöhnliche Differentialgleichungen*)

Da kommt auch dran:

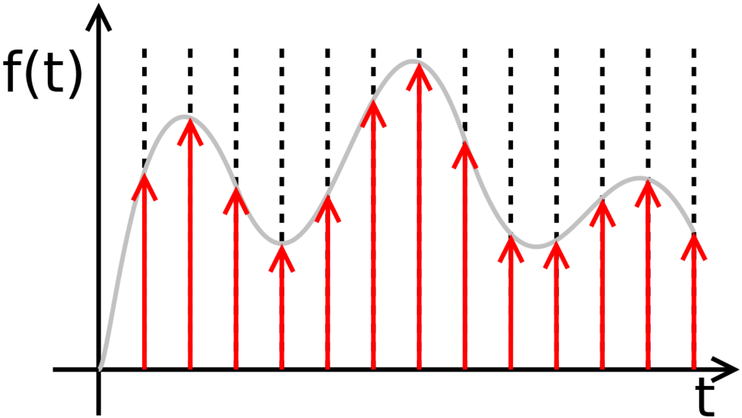
Riemann-Lebesgue-Lemma: Ist f stetig, dann gilt

$$\lim_{n\to\infty} a_n = 0, \quad \lim_{n\to\infty} b_n = 0.$$

D.h.: nur die ersten (paar) a_n und b_n sind wichtig! Die anderen werden klein oder verschwinden.

***

Computer kennen keine stetigen Funktionen und keine reellen Zahlen. Daher:



f ist beschrieben durch $f(t_0), f(t_1), f(t_2), \ldots, f(t_{N-1})$.

Diskrete Fouriertransformation (DFT)

Die *diskrete Fouriertransformation* (DFT) von $f = (f_0, f_1, \ldots, f_{N-1})$ ist $d = (d_0, d_1, \ldots, d_{N-1})$ mit

$$d_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j \cdot \big(\cos(-\frac{2\pi}{N} jk) + i \sin(-\frac{2\pi}{N} jk) \big) \quad (k = 0, 1, \ldots, N-1)$$

Das sind Vektoren!

Daher kann man die DFT mittels dieser Matrix ausrechnen (hier ist $\xi = e^{2\pi i/N} = \cos(\frac{2\pi}{N}) + i \sin(\frac{2\pi}{N})$).

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{N-1} \end{pmatrix} = \frac{1}{N} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \xi^{-1} & \xi^{-2} & \cdots & \xi^{-(N-1)} \\ 1 & \xi^{-2} & \xi^{-4} & \cdots & \xi^{-2(N-1)} \\ 1 & \xi^{-3} & \xi^{-6} & \cdots & \xi^{-3(N-1)} \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & \xi^{-(N-1)} & \xi^{-2(N-1)} & \cdots & \xi^{-(N-1)^2} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-1} \end{pmatrix}$$

Fast Fourier Transform (FFT)

Insbesondere falls $N = 2^k$ (oder $N \approx 2^k$) ist es günstig, die DFT nicht mittels der Matrix zu berechnen (Aufwand $O(n^2)$), sindern mit einem Divide-and-Conquer-Ansatz.

Theorem

Die DFTs von $(f_0, f_1, \ldots, f_{N-1})$ und $(f_N, f_{N+1}, \ldots, f_{2N-1})$ liefern die DFT von $(f_0, f_N, f_1, f_{N+1}, f_2, f_{N+2}, \ldots, f_{N-1}, f_{2N-1})$ (Länge 2N) so: Mit

$$d_k = \frac{1}{2}\big(DFT(f_0, f_1, \ldots, f_{N-1}) + e^{-\pi i k/N} DFT(f_N, f_{N+1}, \ldots, f_{2N-1})\big)_k$$

$$d_{N+k} = \frac{1}{2}\big(DFT(f_0, f_1, \ldots, f_{N-1}) - e^{-\pi i k/N} DFT(f_N, f_{N+1}, \ldots, f_{2N-1})\big)_k$$

(wobei $0 \leq k \leq N-1$) ist dann

$$DFT(f_0, f_N, f_1, f_{N+1}, f_2, f_{N+2}, \ldots, f_{N-1}, f_{2N-1}) = (d_0, d_1, \ldots, d_{2N-1})$$

.

Dieser Satz liefert einen Divide-and-Conquer-Algorithmums zum Berechnen der DFT in $O(n \log n)$ Schritten. (Hier nur für $N = 2^k$.) Dazu muss das Problem aufgeteilt werden in das Berechnen zweier DFTs, die dann wieder aufgeteilt werden in 2 mal 2 usw.; bis jeweils N Stück DFTs der Länge 1 berechnet werden müssen. Die müssen dann in der richtigen Reihenfolge kombiniert werden. Hier das Schema (für $N = 8$):

$$
\begin{tikzcd}[column sep=tiny]
f_0 \arrow[d, equal] & & f_4 \arrow[d, equal] & & f_2 \arrow[d, equal] & & f_6 \arrow[d, equal] & & f_1 \arrow[d, equal] & & f_5 \arrow[d, equal] & & f_3 \arrow[d, equal] & & f_7 \arrow[d, equal] \\
DFT(f_0) \arrow[dr] & & DFT(f_4) \arrow[dl] & & DFT(f_2) \arrow[dr] & & DFT(f_6) \arrow[dl] & & DFT(f_1) \arrow[dr] & & DFT(f_5) \arrow[dl] & & DFT(f_3) \arrow[dr] & & DFT(f_7) \arrow[dl] \\
& DFT(f_0, f_4) \arrow[drr] & & & & DFT(f_2, f_6) \arrow[dll] & & & & DFT(f_1, f_5) \arrow[drr] & & & & DFT(f_3, f_7) \arrow[dll] & \\
& & & DFT(f_0, f_2, f_4, f_6) \arrow[drrrr] & & & & & & & & DFT(f_1, f_3, f_5, f_7) \arrow[dllll] & & & \\
& & & & & & & DFT(f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7) & & & & & & &
\end{tikzcd}
$$

Das einzige Problem ist nun, wie bringen wir die f_n in die richtige Anfangsreihenfolge? Die richtige Reihenfolge erhalten wir einfach durch Bitumkehr:

$$000 \to 000,\ 001 \to 100,\ 010 \to 010,\ 011 \to 110,\ 100 \to 001,\ usw$$

Also (im Falle $N = 8$) muss an der 0-ten Stelle f_0 stehen, an der ersten Stelle f_4, an der zweiten f_2 usw. Natürlich hängt die Bitumkehr von $N = 2^k$ ab. Für $N = 16$ ergibt sich etwa

$$0000 \to 0000,\ 0001 \to 1000,\ 0010 \to 0100,\ 0011 \to 1100,\ 0100 \to 0010,\ usw$$

Algorithmus FFT:

Sei $N = 2^q$. Sei $g = (g_0, g_1, \ldots, g_{N-1})$ der Vektor, den man durch Umnummerierung mittels Bitumkehr aus $f = (f_0, f_1, \ldots, f_{N-1})$ erhält.
Starte mit den Vektoren der Länge 1: $d^{[0,j]} = (g_j)$ $(j = 0, \ldots, N-1)$. Berechne in Schritt r $(r \geq 1)$ die 2^{q-r} Vektoren der Länge 2^r

$$d^{[r,0]}, d^{[r,1]}, \ldots, d^{[r,2^{q-r}-1]}.$$

aus den Vektoren im $r-1$-ten Schritt gemäß

$$d_k^{[r,j]} = \frac{1}{2}\left(d_k^{[r-1,2j]} + e^{-\pi ik/2^{r-1}} d_k^{[r-1,2j+1]}\right) \qquad (1)$$

$$d_{2^{r-1}+k}^{[r,j]} = \frac{1}{2}\left(d_k^{[r-1,2j]} - e^{-\pi ik/2^{r-1}} d_k^{[r-1,2j+1]}\right) \qquad (2)$$

Dabei läuft (innerste Schleife) $k = 0, 1, \ldots, 2^{r-1} - 1$, und (zweitinnerste Schleife) $j = 0, 1, \ldots, 2^{q-r} - 1$, sowie $r = 1, 2, \ldots, q$.

Beispiel: Hier ein (sehr einfaches) Beispiel für $N = 4$: Wir berechnen die DFT für den Vektor (Datensatz) $f = (8, -4, -8, 16)$. Bitumkehr liefert

$$g_0 = f_0 = 8,\ g_1 = f_2 = -8,\ g_2 = f_1 = -4,\ g_3 = f_3 = 16$$

Das Schema ist

$$\begin{array}{ccccccc}
d^{[0,0]} & & d^{[0,1]} & & d^{[0,2]} & & d^{[0,3]} \\
& \searrow \ \swarrow & & & & \searrow \ \swarrow & \\
& d^{[1,0]} & & & & d^{[1,1]} & \\
& & \searrow & & \swarrow & & \\
& & & d^{[2,0]} & & &
\end{array}$$

Dabei sind $d^{[1,0]}$ und $d^{[1,1]}$ Vektoren der Länge 2 (also $d^{[1,0]} = (d_0^{[1,0]}, d_1^{[1,0]})$ usw) und $d^{[2,0]}$ ist ein Vektor der Länge 4.

Die konkreten Werte:

$$\begin{array}{ccccccc}
8 & & -8 & & -4 & & 16 \\
& \searrow \; \swarrow & & & & \searrow \; \swarrow & \\
& (0;8) & & & & (6;-10) & \\
& & \searrow & & \swarrow & & \\
& & & (3;4+5i;-3;4-5i) & & &
\end{array}$$

Dabei berechnet sich z.B. $d^{[1,0]} = (d_0^{[1,0]}, d_1^{[1,0]})$ so:

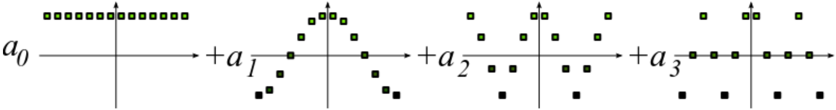
$$\begin{aligned}
d_0^{[1,0]} &= \frac{1}{2}(d_0^{[0,0]} + e^{-\pi i 0} d_0^{[0,1]}) = \frac{1}{2}(8-8) = 0 \\
d_1^{[1,0]} &= \frac{1}{2}(d_0^{[0,0]} - e^{-\pi i 0} d_0^{[0,1]}) = \frac{1}{2}(8-(-8)) = 8,
\end{aligned}$$

...und $d^{[2,0]} = (d_0^{[2,0]}, d_1^{[2,0]}, d_2^{[2,0]}, d_3^{[2,0]})$ so:

$$
\begin{aligned}
d_0^{[2,0]} &= \frac{1}{2}(d_0^{[1,0]} + e^{-\pi i 0} d_0^{[1,1]}) = \frac{1}{2}(0+6) = 3 \\
d_1^{[2,0]} &= \frac{1}{2}(d_1^{[1,0]} + e^{-\pi i/2} d_1^{[1,1]}) = \frac{1}{2}(8 - i(-10)) = 4 + 5i, \\
d_2^{[2,0]} &= \frac{1}{2}(d_0^{[1,0]} - e^{-\pi i 0} d_0^{[1,1]}) = \frac{1}{2}(0-6) = -3 \\
d_3^{[2,0]} &= \frac{1}{2}(d_1^{[1,0]} - e^{-\pi i/2} d_1^{[1,1]}) = \frac{1}{2}(8 + i(-10)) = 4 - 5i.
\end{aligned}
$$

Bildkompression

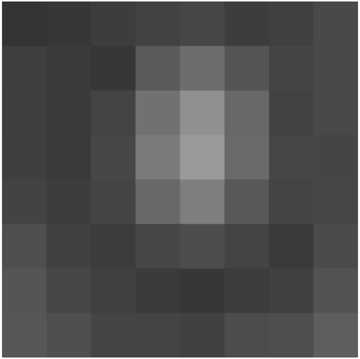
Gleiche Idee wie bei Fourierreihen: Diskretes Signal (Ton, Bildzeile...) darstellen als Summe von Vektoren (mit Kosinussen, "Diskrete Kosinustransformation" (DCT))

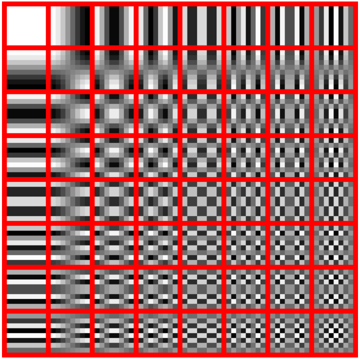


Die ersten paar (hier: 4) Terme liefern eine gute Näherung. Speichere also nur 4 Werte statt 14.

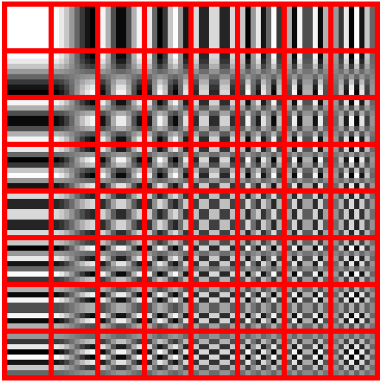
Funktionsweise jpeg:

- Bearbeite jeden der drei Farbwerte einzeln (RGB bzw. YCbCr)
- Unterteile das Bild in 8×8 Felder (u. links: ein solches Feld)
- 2D DCT für jedes einzelne Feld (Linearkombination der 64 8×8-Felder unten rechts)
- Quantisieren der DCT (übernächste Folie)
- Run-length coding, dann Huffman coding





DCT: Die Matrix rechts zeigt den Vorfaktor für das entsprechende Feld links ("wie viel" vom entsprechenden Feld links wir hinzuaddieren)



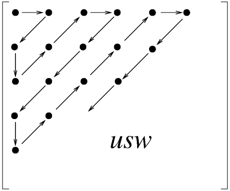
$$\begin{bmatrix} -415 & -30 & -61 & 27 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -3 & 4 & -1 \\ 0 & 0 & -1 & -1 & 0 & 1 & 2 \end{bmatrix}$$

Quantisieren:

$$\begin{bmatrix} -415 & -30 & -61 & 27 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -3 & 4 & -1 \\ 0 & 0 & -1 & -1 & 0 & 1 & 2 \end{bmatrix} \text{ mit } \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

$$\text{runden liefert } \begin{bmatrix} -26 & -3 & -6 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Speichere das in dieser Reihenfolge



Dann **Run-length-coding** (Blöcke von 0en!), dann **Huffman-coding** (häufige Zeichen mit wenig Bits)
Klingt kompliziert. Klappt sehr gut:

Komprimierungsfaktor $0,38$

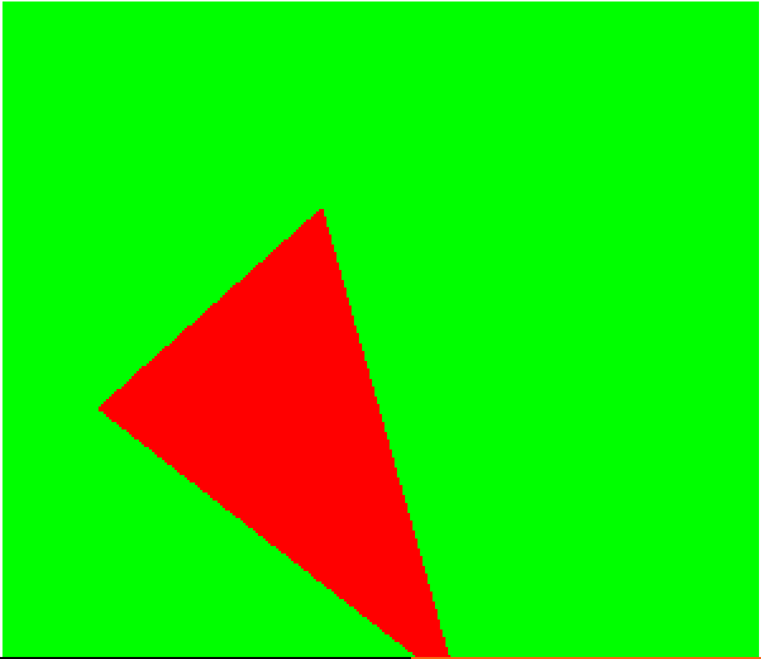
Komprimierungsfaktor $0,07$

Komprimierungsfaktor $0,04$

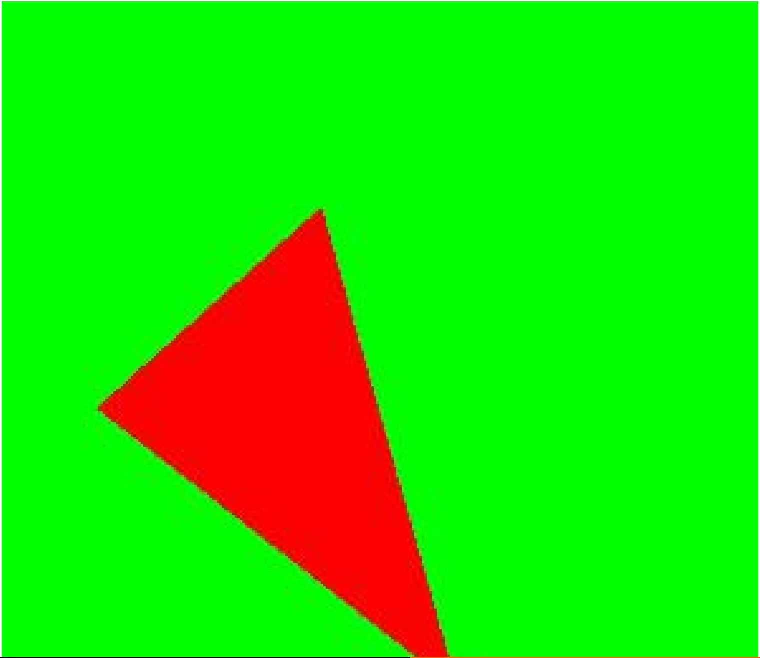
Komprimierungsfaktor $0,02$

Komprimierungsfaktor $0,007$

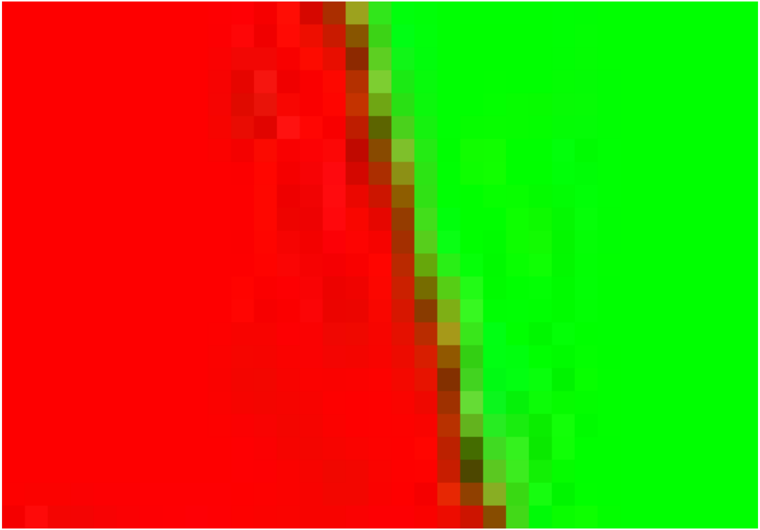
jpeg ist nicht gut bei scharfen Kanten und wenigen Farben. png:



Dasselbe Bild als jpeg:

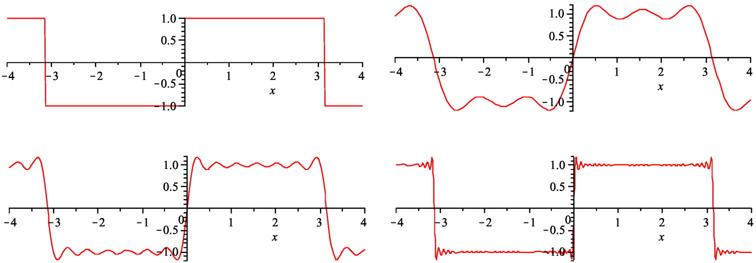


Ausschnittsvergrößerung desselben Bilds:



Gibbssches Phänomen:

Das kann man mittels der Theorie der Fourierreihen verstehen:



Oben links: Originalsignal f (Rechteckkurve), oben rechts: die ersten vier Terme der Fourierreihe.

Unten links: die ersten 16 Terme, unten rechts: die ersten 30 Terme.

Egal wie viele Terme man hinzunimmt, es gibt (beweisbar) immer einen Ausschlag um ca 15% zu weit nach oben.