
Bachelorarbeit

# Primal Dual Algorithmus zur Rauschminimierung

*Maximilian Friedl*
*Matrikelnummer: 77847*

betreut durch
Prof. Dr. Tomas Sauer

11. September 2018

# Kurzfassung

In dieser Arbeit wird, basierend auf der ROF-Arbeit, folgender Algorithmus hergeleitet, mit welchem man ein Bild entrauschen kann:

$$x_{k+1} = P_X(x_k + \tau_k \lambda A^T y_k)$$
$$y_{k+1} = (1 - \theta_k) y_k + \theta_k (z - \frac{1}{\lambda} A x_{k+1})$$

Dabei steht z und $y_0$ für das rauschende Bild, $x_0$ für den Nullvektor, $\lambda$ ist ein Parameter, der von der Standardabweichung des Rauschens abhängt und $\tau_k$ und $\theta_k$ für diverse Schrittweiten. $A$ ist so definiert, dass es mit $z$ multipliziert $\nabla z$ ergibt und $P_X$ für eine Projektion auf die Menge $X$. Diese kann mit dem Einheitskreis verglichen werden. Das letztendlich entrauschte Bild wird dann das konvergierte $y_k$ sein.
Anschließend wird die Dualitätstheorie erläutert und eine mögliche Implementierung in MATLAB vorgestellt. Abschließend wird der Algorithmus auf verschiedenen Beispielen getestet. Dabei zeigt sich, dass das entrauschte Bild nach etwa 15 Iterationen wiederhergestellt ist und bis zu einer Standardabweichung des Rauschens von $\sigma = 80$ sehr gut funktioniert. Probleme gibt es allerdings bei sehr feinen Strukturen wie zum Beispiel einer Wiese und großen Bildern, da die Matrix $A$ extrem viele Einträge besitzt und dementsprechend die Multiplikation mit $A$ sehr rechenaufwändig ist.

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

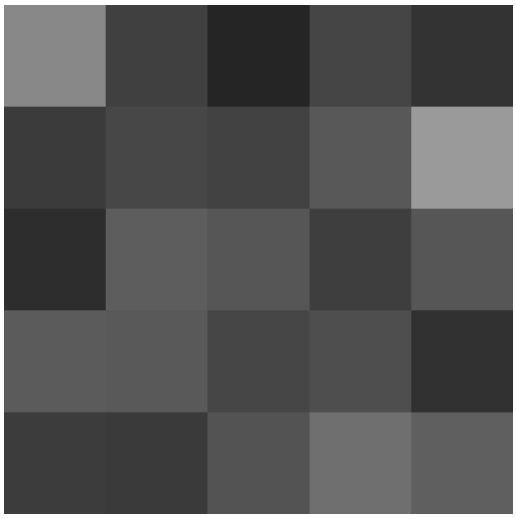
Diese Arbeit beschäftigt sich mit der Frage, wie man ein schwarz-weißes Bild entrauschen kann, sodass das eigentliche Bild gut zu erkennen ist. Hierzu wird ein Algorithmus hergeleitet und vorgestellt.
Oft kommt es vor, dass Bilder rauschen, also gestörte Pixel aufweisen. Dem kann unter anderem fehlerhafte Fotografie, schlechte Übertragung oder Speicherung des Bildes zu Grunde liegen.
Ein sogenanntes Graustufenbild - im herkömmlichen Sinne bestehend aus Pixeln - wird als Matrix gespeichert, wobei jeder Pixel durch den entsprechenden Eintrag an der selben Position repräsentiert wird. Die Werte nehmen jeweils eine ganze Zahl zwischen 0 und 255 an. Hierbei steht 0 für einen schwarzen Pixel und 255 für einen weißen. Alle dazwischenliegenden Werte stehen für verschiedene Graustufen in aufhellender Ordnung:


(a) Abbildung 1.1: Graustufen

Der Algorithmus wird basierend auf einem Ergebnis der wegbereitenden Arbeit von L. Rudin, S. Osher und E. Fatemi (ROF) „Nonlinear total variation based noise removal algorithms“(Physica D Vol.60, 1992) entwickelt. In dieser wird ein Problem formuliert, dessen Lösung das Entrauschte eines ursprünglichen Bildes ist. Dem Algorithmus liegt die primal-duale Formulierung des Problems und die Anwendung des Gradientenverfahrens zugrunde. Wie eine mögliche Implementierung davon aussieht und die Qualität und Grenzen da-

von werden ebenfalls ausfühlich behandelt. Als Ausgangslage dient dabei ein Bild, welches mit dem gaußschen Rauschen mit einer Standardabweichung der einzelnen Einträge zum Original von $\sigma$ versehen ist. Visuell bedeutet das für ein „Bild“der Größe $5 \times 5$ und $\sigma = 20$:

(b) Abbildung 1.2: Original



(c) Abbildung 1.3: Rauschen mit $\sigma = 20$

Als Matrix sehen die beiden Bilder folgendermaßen aus:

$$\begin{bmatrix} 80 & 80 & 80 & 80 & 80 \\ 80 & 80 & 80 & 80 & 80 \\ 80 & 80 & 80 & 80 & 80 \\ 80 & 80 & 80 & 80 & 80 \\ 80 & 80 & 80 & 80 & 80 \end{bmatrix},$$

Original

$$\begin{bmatrix} 136 & 64 & 37 & 69 & 51 \\ 59 & 71 & 66 & 88 & 154 \\ 45 & 93 & 86 & 62 & 86 \\ 91 & 89 & 70 & 78 & 49 \\ 60 & 58 & 83 & 111 & 95 \end{bmatrix}.$$

Rauschen

# Kapitel 2

# Der Algorithmus

Das folgende Kapitel basiert hauptsächlich auf den Inhalten von [ZC08], die hier in abgeänderter Form wiedergegeben werden.

## 2.1 Definitionen

Um einen genauen Überblick zu erhalten, beginnen wir damit alle Matritzen und Funktionen zu definieren, die im Verlauf dieses Kapitels benötigt werden.

Von hier an bezeichnen wir das zu entrauschende Bild als Matrix $B \in \mathbb{R}^{m \times n}$ und mit $N = m \cdot n$ die Anzahl der Pixel beziehungsweise der Einträge der Matrix $B$.
Den Vektor $z^{N \times 1}$ erhalten wir, indem die Spalten von $B$ von links nach rechts untereinander angeordnet werden, woduch für den $i$-ten Eintrag von $z$ gilt:

$$z_i = B_{((i-1) \bmod m)+1, \lceil \frac{i}{m} \rceil}.$$

Zur besseren Verständlichkeit der Definitionen führen wir ein kleines Beispielbild $B^{4 \times 2}$ ein, also $m = 4$, $n = 2$ und dementsprechend $N = 8$:

$$B = \begin{bmatrix} 2 & 3 \\ 1 & 8 \\ 8 & 9 \\ 5 & 4 \end{bmatrix} \Rightarrow z = \begin{bmatrix} 2 \\ 1 \\ 8 \\ 5 \\ 3 \\ 8 \\ 9 \\ 4 \end{bmatrix}$$

Mit $\partial_1 B \in \mathbb{R}^{m \times n}$ beschreiben wir die Differenz jedes Eintrages in $B$ zu seinem unterliegenden Nachbareintrag, also

$$(\partial_1 B)_{i,j} = \begin{cases} B_{i+1,j} - B_{i,j}, & \text{für } i < m, \\ 0, & \text{für } i = m, \text{ sprich für alle Pixel der letzten Zeile.} \end{cases}$$

Dazu wird $\partial_2 B \in \mathbb{R}^{m \times n}$ analog definiert. Nur mit dem Unterschied, dass es sich um die Differenz zu dem rechten Eintrag handelt. Somit gilt

$$(\partial_2 B)_{i,j} = \begin{cases} B_{i,j+1} - B_{i,j}, & \text{für } i < n, \\ 0, & \text{für } i = n, \text{ sprich für alle Pixel der letzten Spalte.} \end{cases}$$

Schlussendlich werden die beiden Matrizen $\partial_1 B$ und $\partial_2 B$ zum Vektor $\nabla B \in \mathbb{R}^{2N}$ kombiniert, indem wir die Einträge von $\partial_1 B$ und $\partial_2 B$ abwechselnd von oben nach unten und von links nach rechts durchlaufend untereinander anordnen:

$$\nabla B_i = \begin{cases} \partial_1 B_{((\frac{i+1}{2}-1) \bmod m)+1, \lceil \frac{(i+1):2}{m} \rceil} & \text{für } i \text{ ungerade,} \\ \partial_2 B_{((\frac{i}{2}-1) \bmod m)+1, \lceil \frac{i:2}{m} \rceil} & \text{für } i \text{ gerade.} \end{cases}$$

Für das Beispiel gilt also:

$$B = \begin{bmatrix} 2 & 3 \\ 1 & 8 \\ 8 & 9 \\ 5 & 4 \end{bmatrix} \Rightarrow \partial_1 B = \begin{bmatrix} -1 & 5 \\ 7 & 1 \\ -3 & -5 \\ 0 & 0 \end{bmatrix} \partial_2 B = \begin{bmatrix} 1 & 0 \\ 7 & 0 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \Rightarrow \nabla B = \begin{bmatrix} -1 \\ 1 \\ 7 \\ 7 \\ -3 \\ 1 \\ 0 \\ -1 \\ 5 \\ 0 \\ 1 \\ 0 \\ -5 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Mit der Funktion $P_X : \mathbb{R}^{2N} \to X$ projizieren wir den eingegebenen Vektor auf die Menge

$$X = \{x \in \mathbb{R}^{2N}, \left\|(x_{2i-1}, x_{2i})^T\right\|_2 \leq 1, i = 1, ..., N\}, \tag{2.1}$$

indem wir jeweils zwei Einträge zu einem Tupel zusammenfassen und falls dieses außerhalb des Einheitskreises liegt, durch dessen Norm teilen, also $x_i \in \mathbb{R}^2$ und

$$(P_X(x))_i = \frac{x_i}{\max\{\|X_i\|_2, 1\}}, \qquad i = 1, ..., N. \tag{2.2}$$

Zu guter Letzt werden die Matrizen $A_i \in \mathbb{R}^{N \times 2}$, $i = 1,...,N$, durch

$$A_i^T z = \begin{cases} (z_{i+1} - z_i, z_{i+m} - z_i)^T & \text{für } i \bmod m \neq 0 \text{ und } i \leqslant N - m, \\ (0, z_{i+m} - z_i)^T & \text{für } i \bmod m = 0 \text{ und } i \leqslant N - m, \\ (z_{i+1} - z_i, 0)^T & \text{für } i \bmod m \neq 0 \text{ und } i > N - m, \\ (0, 0)^T & \text{für } i \bmod m = 0 \text{ und } i > N - m \end{cases} \tag{2.3}$$

definiert. Dies entspricht exakt

$$A_i^T z = (\partial B_{2i-1}, \partial B_{2i})^T$$

und wir erhalten die Matrix $A \in \mathbb{R}^{N \times 2N}, A = [A_1|A_2|...|A_N]$, die eine sehr einfache Struktur besitzt. Da lediglich $A^T z = \nabla B$ gelten soll, muss kein kompliziertes lineares Gleichungssystem gelöst werden, sondern es kann - hier für das Beispiel mit $N = 8$ - folgende Matrix verwendet werden, die genau den Ansprüchen an $A$ genügt:

$$A = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ \underbrace{0 \quad 0}_{A_1} & & \underbrace{0 \quad 0}_{A_2} & & \underbrace{0 \quad 0}_{A_3} & & \underbrace{0 \quad 1}_{A_4} & & \underbrace{0 \quad 0}_{A_5} & & \underbrace{0 \quad 0}_{A_6} & & \underbrace{1 \quad 0}_{A_7} & & \underbrace{0 \quad 0}_{A_8} \end{bmatrix}.$$

## 2.2 Hintergrund

Rudin, Osher und Fatem entwickelten in ihrer Arbeit (ROF) unter anderem folgende Formel, mit der man das Rauschen eines Bildes entfernen kann, ohne vehemente Einbußen in der Qualität eingehen zu müssen:

$$\min_y \int_\Omega \|\nabla y\| \text{ so, dass } \|y - B\|^2 \leq |\Omega|\sigma^2. \tag{2.4}$$

Hier entspricht $\sigma$ der (geschätzten) Standardabweichung des Rauschens, $\Omega$ der Bilddomäne respektive $|\Omega|$ dem Bildbereich und $y$ dem entrauschten Bild, das von den Dimensionen her der Matrix $B$ gleicht. Fortan wird mit $\|\cdot\|$ die euklidische Standardnorm bezeichnet.
Bei der richtigen Wahl des Relaxationsfaktors $\lambda$ besitzt die obige Gleichung (2.4) die selbe Lösung wie

$$\min_y \int_\Omega \|\nabla y\| + \frac{\lambda}{2} \|y - B\|^2 . \tag{2.5}$$

Für den $\lambda$-Wert, welcher von der Standardabweichung $\sigma$ des Rauschens abhängt, gilt beim Standard gaußschen Rauschen $\sigma = 20$ für Lambda $\lambda \approx 0.05$. Weiter

haben $\lambda$ und $\sigma$ eine lineare inverse Relation, das bedeutet wenn $\sigma$ verdoppelt wird muss $\lambda$ halbiert werden, bzw. $\lambda = 0.05 \cdot \frac{20}{\sigma} = \frac{1}{\sigma}$.

An dieser Stelle kommt die Matrix $A$ ins Spiel. Diese ist, zur erneuten Verdeutlichung so definiert, dass sie mit $z$ multipliziert $\nabla B$ entspricht. Damit dies hier auf $y$ anwendbar ist, muss nun $y \in \mathbb{R}^N$ gelten. Um das Integral in der Gleichung zu vermeiden, wird ab jetzt die diskrete Version von (2.5) betrachtet. Deswegen wird das Integral $\int_\Omega \|\nabla y\|$ zu $\sum_{i=1}^{N} \left\|A_i^T y\right\|$ diskretisiert.

Da die Variable $y$ nun ein Vektor und nicht weiter eine Matrix ist, muss $B$, durch den im bereits letzten Abschnitt definierten Vektor $z$, ausgetauscht werden. Wir erhalten also das primale Problem

$$\min_{y \in \mathbb{R}^N} P(y) := \sum_{i=1}^{N} \left\|A_i^T y\right\| + \frac{\lambda}{2} \|y - z\|^2 . \tag{2.6}$$

Für den nächsten Schritt wird ein Hilfsmittel benötigt, um die Summe zu ersetzten:

**Cauchy-Schwarzsche Ungleichung**
Die Cauchy-Schwarzsche Ungleichung besagt, dass

$$|\langle x, y\rangle|^2 \leq \langle x, x\rangle \cdot \langle y, y\rangle,$$

wodurch im reellen Fall für die vom Skalarprodukt induzierte Norm $\|x\| := \sqrt{\langle x, x\rangle}$

$$\langle x, y\rangle \leq \|x\| \cdot \|y\|$$

folgt. Gleichheit gilt genau dann, wenn $x$ und $y$ linear abhänging sind. [Are13, 663]
Die Definition des Skalarprodukts lautet $\langle x, y\rangle := x^T y$. Dementsprechend ergibt sich für ein beliebiges $x \in \mathbb{R}^n$ und ein festes $0 \leq r < \infty$

$$\max_{y \in Y} \langle x, y\rangle = \max_{y \in Y} x^T y = \|x\| \cdot \|y\| ,$$
$$Y := \{y \in \mathbb{R}^n | \, \|y\| \leq r\}.$$

Daraus folgt die lineare Abhängigkeit von $x$ und dem durch das Maximum bestimmte $y$. Die Gleichung lässt sich weiter modifizieren, indem man $r = 1$ wählt und erhält schlussendlich

$$\max_{\|y\| \leq 1} x^T y = \|x\| \cdot \|y\| = \|x\| \cdot \|1\| = \|x\| . \tag{2.7}$$

Dies gilt, da die Norm von $y$ hier höchstens 1 ist.

Nun kann (2.7) auf (2.6) angewendet werden, indem zuerst die Summe folgendermaßen umgeschrieben wird:

$$\sum_{i=1}^{N} \left\| A_i^T y \right\| = \max_{\|x_i\| \leq 1} \sum_{i=1}^{N} (A_i^T y)^T x_i = \max_{\|x_i\| \leq 1} y^T \sum_{i=1}^{N} A_i x_i.$$

Wir erinnern uns an die Menge $X$ (2.1), wodurch weiter

$$\sum_{i=1}^{N} \left\| A_i^T y \right\| = \max_{x \in X} y^T A x$$

gilt und erhalten abschließend das zu lösende Problem

$$\min_{y \in \mathbb{R}^N} \max_{x \in X} \Phi(y, x) := y^T A x + \frac{\lambda}{2} \|y - z\|^2 . \tag{2.8}$$

Dabei entspricht $y$ dem entrauschten Bild.

## 2.3 Lösen des Problems

In diesem Abschnitt wird eine mögliche Lösung für (2.8) ermittelt. Dazu beginnen wir mit nachfolgendem Exkurs:

**Gradientenverfahren**
Liegt ein Problem der Form

$$\min_{x \in \mathbb{R}^n} f(x)$$

für eine differenzierbare Funktion $f : \mathbb{R}^n \to \mathbb{R}$ vor, kann das Gradientenverfahren genutzt werden, um sich der Lösung anzunähern. Dazu wird

folgende Iteration angewendet:

$$x_{i+1} = x_i + \alpha_i \cdot G_i,$$

wobei $x_i$ der vorherige Iterationsschritt ist und $G_i = -\nabla f(x_i)$ der negative Gradient von $f(x_i)$. Mit $\alpha_i$ wird die Schrittweite der $i$-ten Iteration bezeichnet, welche nötig ist, um die Konvergenz zur Lösung zu gewährleisten. Die Bestimmung der Schrittweiten wird in Kapitel 2.3.1 genauer behandelt. [Kar14, 653]

Unsere Funktion $\Phi(y, x)$ ist offensichtlich differenzierbar, weswegen das Gradientenverfahren angewendet werden kann. Da es sich um eine Funktion mit zwei Eingabeparametern handelt, beginnen wir damit

$$\max_{x \in X} \Phi(y, x)$$

für ein festes $y = y_k$ zu lösen. Das obige Problem ist äquivalent zu

$$\min_{x \in X} -\Phi(y_k, x)$$

und wir erhalten für $\nabla_{x_k} \Phi(y_k, x_k) = -y_k^T A$. Diese muss transponiert werden um keine Probleme mit den Dimensionen zu bekommen. Da außerdem $x_{k+1} \in X$ gelten soll, muss nach jedem Iterationsschritt das Ergebnis auf die Menge $X$ mit der bereits bei (2.2) definierten Funktion $P_X(x)$ projiziert werden. Wir erhalten schließlich den Algorithmus:

$$x_{k+1} = P_X(x_k + \tau_k \lambda A^T y_k). \tag{2.9}$$

Die Schrittweite wird hierbei durch $\tau_k \lambda$ repräsentiert. Damit der Algorithmus abhängig von verschiedenen Rauschintensitäten verwenden werden kann, wird $\lambda$ dabei mitverwendet,.

Nun lösen wir

$$\min_{y \in \mathbb{R}^N} \Phi(y, x)$$

für ein festes $x = x_{k+1}$ und erhalten unter Beachtung, dass $\frac{\partial}{\partial x} \|x\|^2 = 2x$, den Gradienten $\nabla_{y_k} \Phi(y_k, x_{k+1}) = A x_{k+1} + \lambda(y_k - z)$ und somit die Iteration:

$$y_{k+1} = y_k - \frac{\theta_k}{\lambda}(A x_{k+1} + \lambda(y_k - z)) = (1 - \theta_k) y_k + \theta_k (z - \frac{1}{\lambda} A x_{k+1}).$$

Diese hat die Schrittweite $\frac{\theta_k}{\lambda}$. $\lambda$ wird hier aus dem selben Grund wie bei (2.9) verwendet und wir haben den vollständigen Algorithmus um (2.8) zu lösen:

$$\begin{aligned} x_{k+1} &= P_X(x_k + \tau_k \lambda A^T y_k) \\ y_{k+1} &= (1-\theta_k)y_k + \theta_k(z - \frac{1}{\lambda} A x_{k+1}). \end{aligned} \tag{2.10}$$

Später wird sich zeigen, dass dieser recht schnell zur Lösung beziehungsweise zum entrauschten Bild konvergiert.

### 2.3.1 Bestimmen der Schrittweiten

Bevor der Algorithmus letztendlich implementiert werden kann, müssen noch die beiden Schrittweiten festgelegt werden. Dafür wäre es prinzipiell ausreichend $\tau_k = 2$ und $\theta_k = 0.2$ fest zu definieren. Um eine noch raschere Konvergenz gegen eine Lösung zu erhalten, bietet es sich allerdings an

$$\begin{aligned} \tau_k &= 0.2 + 0.08k \\ \theta_k &= (0.5 - \frac{5}{15+k})/\tau_k \end{aligned}$$

zu wählen.
Um dies zu zeigen, werden verschiedene $\tau$ und $\theta$ an einem Beispiel getestet und überprüft, nach wie vielen Iterationen eine Konvergenz vorliegt. Dazu nutzen wir eine Matrix $T^{5\times5}$, auf die wir das gaußsche Standardrauschen legen, wodurch wir $G^{5\times5}$ erhalten:

$$T = \begin{bmatrix} 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 \end{bmatrix}, G = \begin{bmatrix} 92 & 68 & 105 & 67 & 88 \\ 112 & 117 & 105 & 67 & 103 \\ 139 & 77 & 98 & 124 & 105 \\ 89 & 91 & 91 & 89 & 72 \\ 75 & 72 & 93 & 122 & 97 \end{bmatrix}.$$

Um zu bestimmen ob und wann eine Konvergenz erfolgt, wird nach jeder Iteration der Unterschied zwischen dem alten und dem neuen y-Vektor gebildet. Dazu berechnen wir die Summe $s$ über alle Absolutbeträge der Differenzen der einzelnen Einträge der y-Vektoren, sprich $s = \|y_k - y_{k-1}\|_1$. Der Algorithmus ist beendet, wenn $s < 1$ gilt, also nur noch ein minimaler Unterschied zwischen $y_k$ und $y_{k+1}$ auszumachen ist. Das Maximum an Durchläufen ist auf

100 festgelegt und mit $k$ wird der Index der aktuellen Iteration bezeichnet.

| $\tau$-Wert | $\theta$-Wert | Konvergenz nach · Durchläufen |
|---|---|---|
| 0.2 | 2 | - |
| 2 | 2 | - |
| 20 | 0.3 | - |
| k | $\frac{1}{k}$ | 71 |
| 3 | 0.1 | 31 |
| 5 | 0.02 | 29 |
| 4 | 0.1 | 27 |
| $0.5 + 0.08k$ | $(0.5 - \frac{5}{15+k})/\tau_k$ | 24 |
| 2 | 0.2 | 24 |
| $0.2 + 0.1k$ | $(0.5 - \frac{5}{15+k})/\tau_k$ | 19 |
| 17 | 0.1 | 18 |
| $0.2 + 0.08k$ | $(\frac{5}{15+k})/\tau_k$ | 18 |
| $0.2 + 0.08k$ | $(0.5 - \frac{5}{15+k})/\tau_k$ | 15 |

## 2.4 Dualität

Für diesen Abschnitt ziehen wir zusätzlich [Sau17, 51-57] zu Hilfe, um nun das duale Problem zu (2.6) zu formulieren. Dazu definieren wir zunächst zu einer Funktion $f(x) : M \to \mathbb{R}$ die Legendre-Fenchel Transformation

$$f^*(y) := \sup_{m \in M} (\langle m, y \rangle - f(m)).$$

Weiter betrachten wir für den allgemeinen Fall ein primales Minimierungsproblem der Form

$$\min_y f(Ky) + g(y), \tag{2.11}$$

wobei $f$ und $g$ konvexe Funktionen sind und $K$ ein linearer Operator.
Dieses soll zunächst in primal-dual Form gebracht werden. Da $f^{**} = f$ und

$$f^{**}(Ky) = \max_m \langle m, Ky \rangle - f^*(m)$$

gilt (Beweis siehe [Sau17, 52]), kann (2.11) folgendermaßen umgeformt werden, indem $f(Ky)$ durch $f^{**}(Ky)$ ersetzt wird:

$$\min_y \max_m \langle Ky, m \rangle + g(y) - f^*(m). \tag{2.12}$$

Nun wenden wir diese Definition auf unser primales Problem (2.6) an, wobei $K := A^T$, $f(x) := \sum_{i=1}^{n} \left\|(x_{2i-1}, x_{2i})^T\right\|, x \in \mathbb{R}^{2n}$ und $g(x) := \frac{\lambda}{2} \|x - z\|^2$ gesetzt wird. $A^T$ ist offensichtlich ein linearer Operator und die Funktionen $f$ und $g$ konvex, da:

$$\begin{aligned}
&\frac{\partial}{\partial x} g(x) = \lambda(x - z), \text{ monton nicht fallend } \Rightarrow g(x) \text{ konvex,} \\
&\frac{\partial}{\partial x} f(x) = \sum_{i=1}^{n} \frac{(x_{2i-1}, x_{2i})^T}{\|(x_{2i-1}, x_{2i})^T\|}, \\
&\frac{\partial^2}{\partial x^2} f(x) = \sum_{i=1}^{n} \frac{\left\|(x_{2i-1}, x_{2i})^T\right\| - \frac{\left\|(x_{2i-1}, x_{2i})^T\right\|^2}{\|(x_{2i-1}, x_{2i})^T\|}}{\left\|(x_{2i-1}, x_{2i})^T\right\|^2} = 0 \\
&\Rightarrow f(x) \text{ linear} \Rightarrow f(x) \text{ konvex (und konkav).}
\end{aligned}$$

Damit erfüllen $K$,$f$ und $g$ die Voraussetzungen für (2.12) und es ergibt sich das primal-duale Problem:

$$\begin{aligned}
&\min_y \max_x \langle A^T y, x\rangle + \frac{\lambda}{2} \|y - z\|^2 - f^*(x) = \\
&\min_y \max_x \langle A^T y, x\rangle + \frac{\lambda}{2} \|y - z\|^2 - \sup_r(\langle r, x\rangle - f(r)) = \\
&\min_y \max_x \langle A^T y, x\rangle + \frac{\lambda}{2} \|y - z\|^2 - \sup_r(\langle r, x\rangle - \sum_{i=1}^{n} \left\|(r_{2i-1}, r_{2i})^T\right\|), r \in \mathbb{R}^{2n}.
\end{aligned}$$

Bei genauerer Betrachtung und auf Grund einer Folge der Cauchy-Schwarzschen Ungleichung ($\langle x, y\rangle \leq \|x\| \cdot \|y\|$, siehe Kapitel 2.2) erkennt man, dass

$$\sup_r(\langle r, x\rangle - \sum_{i=1}^{n} \left\|(r_{2i-1}, r_{2i})^T\right\|) = \begin{cases} 0, \text{ falls, } x \in X \\ \infty, \text{ sonst} \end{cases}$$

gilt. Da $x$ die Formel maximieren soll, werden alle gültigen $x$ auf die bereits bekannte Menge $X$ beschränkt. Somit ist das primal-duale Problem zu (2.6)

$$\min_{y \in \mathbb{R}^n} \max_{x \in X} \langle A^T y, x\rangle + \frac{\lambda}{2} \|y - z\|^2,$$

was genau dem bereits errechneten Problem (2.8) entspricht.
Für die duale Version setzen wir wieder bei (2.12) an und machen folgende

Umformungen. Dabei beachten wir, dass hier min und max vertauscht werden können, da es sich um ein konvex-konkaves Problem handelt:

$$\begin{aligned}
&\min_y \max_m \langle Ky, m\rangle + g(y) - f^*(m) = \\
&\max_m \min_y \langle Ky, m\rangle + g(y) - f^*(m) = \\
&\max_m -f^*(m) + \min_y(\langle Ky, m\rangle + g(y)) = \\
&\max_m -f^*(m) - \underbrace{\max_y(\langle -K^*m, y\rangle - g(y))}_{=g^*(-K^*m)} = \\
&\max_m -(f^*(m) + g^*(-K^*m))
\end{aligned}$$

Wenden wir dies wieder auf das primale Problem (2.6) an, werden alle gültigen Werte erneut auf die Menge $X$ eingeschränkt, da die Formel maximiert werden soll:

$$\max_{x\in X} -(f^*(x) + g^*(-Ax)) = \max_{x\in X} -g^*(-Ax).$$

Weiter gilt nach Definition für $g^*(-Ax)$

$$g^*(-Ax) = \sup_r(\langle r, -Ax\rangle - \frac{\lambda}{2}\|r - z\|^2).$$

Um dieses Supremum zu lösen, leiten wir das Innere des Supremums nach $r$ ab und finden davon die Nullstelle:

$$\frac{\partial}{\partial r}(\langle r, -Ax\rangle - \frac{\lambda}{2}\|r - z\|^2) = -Ax - \lambda(r - z) = 0 \Leftrightarrow r = z - \frac{1}{\lambda}Ax.$$

Unter Betrachtung der zweiten Ableitung

$$\frac{\partial^2}{\partial r^2}(\langle r, -Ax\rangle - \frac{\lambda}{2}\|r - z\|^2) = -\lambda \leq 0$$

können wir darauf schließen, dass dies der Punkt ist, indem $g^*(-Ax)$ ihr

Maximum erreicht. Somit folgt schlussendlich

$$\begin{aligned}
&\max_{x\in X} -g^*(-Ax) = \\
&\max_{x\in X} -\langle z - \frac{1}{\lambda}Ax, -Ax\rangle + \frac{\lambda}{2}\left\|z - \frac{1}{\lambda}Ax - z\right\|^2 = \\
&\max_{x\in X} \langle z - \frac{1}{\lambda}Ax, Ax\rangle + \frac{1}{2\lambda}\langle Ax, Ax\rangle = \\
&\max_{x\in X} \langle z - \frac{1}{\lambda}Ax, Ax\rangle + \langle \frac{1}{2\lambda}Ax, Ax\rangle = \\
&\max_{x\in X} D(x) := \langle z - \frac{1}{2\lambda}Ax, Ax\rangle
\end{aligned}$$

für das duale Problem.
Die so genannte Duality Gap zu einem Paar $(y, x)$ bezeichnet die Differenz $G(y, x) := P(y) - D(x)$. Es gilt also $G(y, x) \geq 0$ und je näher $G(y, x)$ an 0 ist, desto näher ist $(y, x)$ an der primal-dualen Lösung, dem Sattelpunkt des Problems. In unserem Fall bedeutet das dementsprechend

$$G(y, x) = \sum_{i=1}^{N} \left\|A_i^T y\right\| + \frac{\lambda}{2}\|y - z\|^2 - \langle z - \frac{1}{2\lambda}Ax, Ax\rangle.$$

Da $\|\cdot\| \geq 0$ gilt, heißt das daher für den Sattelpunkt $(y, x)$ unseres Problems:

$$\sum_{i=1}^{N} \left\|A_i^T y\right\| + \frac{\lambda}{2}\|y - z\|^2 = \langle z - \frac{1}{2\lambda}Ax, Ax\rangle.$$

Allerdings ist sowohl diese Gleichung als auch das primale und das duale Problem schwer zu lösen und bedarf folglich extrem viel Rechenaufwand. Einfacher ist es die primal-duale Version (2.8) zu behandeln.

# Kapitel 3

# Implementierung

Nun können wir den Algorithmus in MATLAB implementieren. Dazu nutzen wir Hilfsfunktionen, um den Vektor $z$ und die Matrix $A$ zu berechnen. Außerdem wird eine Funktion, mit der ein Vektor auf die Menge $X$ projizieren werden kann, benötigt.
Für den Anfangswert $y_0$ wird logischerweise der Vektor $z$ verwendet, also das zu entrauschende Bild. $x_0$ kann theoretisch jeder beliebige Vektor $x \in \mathbb{R}^{2N}$ sein, zur Vereinfachung legen wir $x_0 = 0$ fest. [ZC08, 18]

## 3.1 Generieren des Vektors z

Wir beginnen mit der sehr einfachen Funktion „compZ“, die das Bild (img) in den Vektor $z^{N\times 1}$ umwandelt. Dazu wird der (:)-Befehl genutzt. Das Ergebnis wird noch zum Typ double gewandelt, da Bilder standartmäßig als uint8-Matrizen vorliegen und die Multiplikation dafür in MATLAB nicht implementiert ist.

```
function z = compZ(img)
  % cast uint8 matrix to matrix
  % (otherwise multiplication not implemented)
   z = img(:);
   z = double(y);
end
```

## 3.2 Erzeugen der Matrix A

Um die Matrix $A$ zu erhalten, folgen wir exakt der Definition (2.3). Zuerst wird eine Null-Matrix mit der richtigen Größe erstellt. Anschließend werden in jeweils zwei Spalten die 1 und -1 korrekt eingesetzt, da sich dann das Muster der von Null verschiedenen Einträge von $A$ stets wiederholt.

```
% m = height of the img
% n = width of the img
function A = compA(m,n)
  N = m*n;
  A = zeros(N,2*N); % preset A with right size

  for i = 1:N
  % one iterationstep for two columns of A,
  % therefore the doublings in the column-indices
  % for the row-indices see definition of A
    if (mod(i,m) != 0) % not the last row
      A(i + 1,2*i - 1) = 1;
      A(i,2*i - 1) = -1;
    end

    if (i <= N - m) % not the last column
      A(i + m,2*i) = 1;
      A(i,2*i) = -1;
    end
  end
end
```

## 3.3 Projektion auf die Menge X

Für die Funktion $P_X(x)$ wird der Eingabevektor $x \in \mathbb{R}^{2N}$ in Zweierpaare aufgeteilt. Dann werden die Tupel durch ihre Norm geteil, falls sie außerhalb des Einheitskreises liegen, um es auf die Länge 1 zu kürzen. Andernfalls liegt der Vektor bereits innerhalb des Einheitskreises und es wird durch 1 dividiert.

```
function x = px(t)
  x = [];
  % Split t to pairs of two and project the
  % vector into the unit circle
  for i = 1:size(t)/2
    h = t(i*2-1:i*2);
    length = norm(h);
    h2 = h/max(length,1);
    x = [x;h2];
  end
end
```

## 3.4 Hauptfunktion

Nun haben wir alle Hilfsmittel, die benötigt werden und können die Funktion zum Entrauschen eines Bildes implementieren. Als Eingabeparameter existieren neben dem Bild ein geschätzter Wert für die Standardabweichung des Rauschens o und die Anzahl der Iterationen, die der Algorithmus durchlaufen soll. Wie viele Durchläufe nötig sein werden, wird im nächsten Kapitel ausführlich behandelt.
Zuallererst bestimmen wir die Matrix $A$, sowie die Vektoren z, $y_0$ als y und $x_0$ als x und berechnen den Wert $\lambda$ als lambda.
Anschließend werden bei jedem Durchlauf zuerst die beiden Schrittweiten $\tau$ und $\theta$ als schrittweiteX beziehungsweise schrittweiteY berechnet und darauffolgend der Algorithmus gemäß Definition (2.10) auf $x$ und $y$ angewandt.
Abschließend wird der resultierte Vektor $y$ zu einer Matrix mit der Größe des ursprünglichen Bildes zurückgeformt und wieder zu einem unsignierten 8 Bit Integer gewandelt, damit MATLAB die Matrix als Bild darstellen kann.

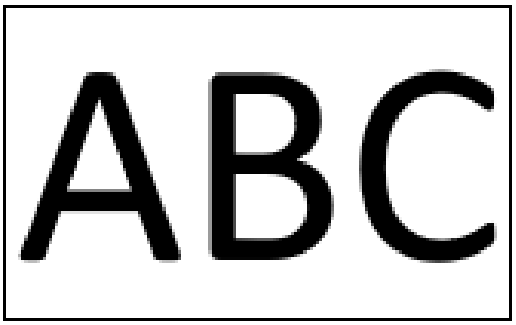
```
function img = denoise(noisedImg, nbrOfIterations, o)
  height = size(noisedImg)(1);
  width = size(noisedImg)(2);

  A = compA(height,width);
```

```
  z = compZ(noisedImg);
  y = z;
  x = zeros(size(z)*2,1);

  lambda = 1/o;

  % update stepsizes, x and y nbrOfIterations-times
  for i = 1:nbrOfIterations
    stepsizeX = 0.2 + 0.08*i;
    stepsizeY = (0.5 - (5/(15 + i)) ) / stepsizeX;

    x = px(x + stepsizeX*lambda*A'*y);
    y = (1-stepsizeY)*y
                + stepsizeY*(z-(1/lambda)*A*x);
  end

  % form the result to a matrix with the size
  % of the input image and cast back to uint8
  y = reshape(y,size(noisedImg));
  img = uint8(y);
end
```

# Kapitel 4

# Qualität

Jetzt können wir unseren Algorithmus testen. Dazu werden die beiden folgenden Bilder genutzt:

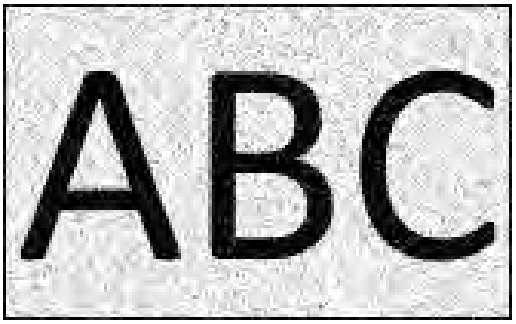



(a) Abbildung 4.1: ABC-Original

(b) Abbildung 4.2: Möwe-Original

Das erste Bild (Abb. 4.1) dient dazu, die Qualität des Algorithmus bei großen einfarbigen Flächen und klaren Kanten zu testen. Das Zweite (Abb. 4.2) wurde gewählt, um zu sehen, wie er sich im Entrauschen von komplexeren Strukturen verhält.
Aufgrund beschränkter Rechnerkapazität ist es an dieser Stelle nicht möglich, hochauflösendere Bilder als Tests zu verwenden. Die Qualität des Algorithmus kann trotzdem ohne Weiteres überprüft werden.

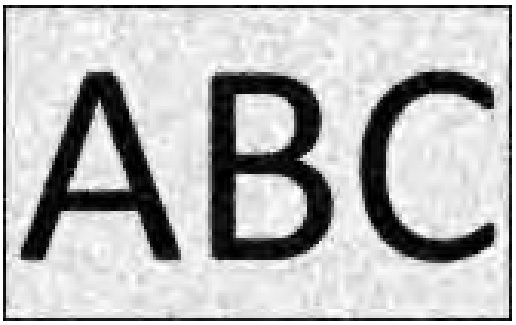
Wir legen über beide Bilder das Standard gaußsche Rauschen von MATLAB mit einem Mittel von 0 und einer Standardabweichung von 20 und erhalten:




(c) Abbildung 4.3: ABC mit Rauschen

(d) Abbildung 4.4: Möwe mit Rauschen

**Nach der ersten Iteration:**

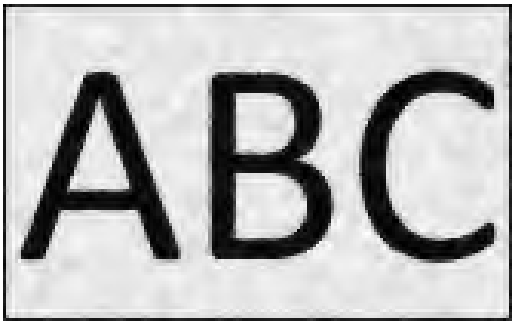



(e) Abbildung 4.5: ABC nach einer Iteration

(f) Abbildung 4.6: Möwe nach einer Iteration

Bereits nach dem ersten Durchlauf kann man eine klare Verbesserung der großen Flächen erkennen.

**Nach der vierten Iteration:**

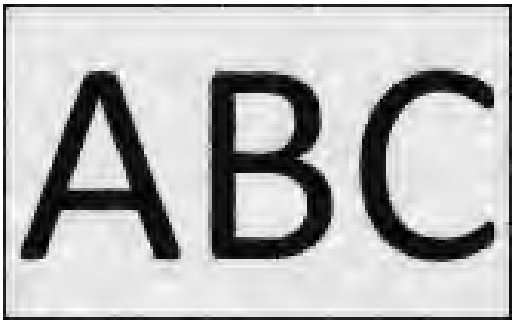



(g) Abbildung 4.7: ABC nach vier Iterationen

(h) Abbildung 4.8: Möwe nach vier Iterationen

Nach der Vierten haben sich die einfarbigen Stellen erneut stark gebessert und das ursprüngliche Rauschen auf diesen ähnelt nun lediglich Unreinheiten auf der Fläche. Auch auf dem Bild, das die Möwe zeigt (Abb. 4.8), ist kein direktes Rauschen mehr erkennbar.

**Nach der zehnten Iteration:**

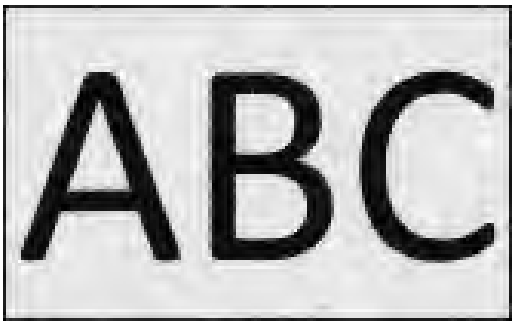



(i) Abbildung 4.9: ABC nach zehn Iteranen

(j) Abbildung 4.10: Möwe nach zehn Iterationen

Nach der zehnten Iteration kann man auf dem linken Bild (Abb. 4.9) nur noch bei genauerer Betrachtung minimale Unebenheiten erkennen. Was jedoch auffällt ist, dass sich der Hintergrund, welcher ursprünglich komplett weiß war, hellgrau gefärbt hat.
Beim rechten (Abb. 4.10) gilt selbiges. Der Hintergrund ist beinahe blank und der Vogel ist auch einwandfrei zu identifizieren.

**Nach der 15. Iteration:**




(k) Abbildung 4.11: ABC nach 15 Iterationen

(l) Abbildung 4.12: Möwe nach 15 Iterationen

Die einzigen wirklich mit dem Auge auszumachenden Unterschiede zwischen dem zehnten und dem 15. Durchlauf bestehen aus einer minimalen Verbesserung des Hintergrundes im rechten Bild (Abb. 4.12) und in der Klarheit des Farbwechsels von grau auf weiß am Kopf der Möwe beziehungsweiße an der Schattengrenze auf dieser. Deswegen und wegen Kapitel 2.3.1, in dem eine Konvergenz nach der 15. Durchlauf gezeigt wurde, ist es nicht notwendig weitere Iterationen durchzuführen. Der jetzige Stand wird dementsprechend als das Endergebnis betrachtet.

**Differenz zwischen Original und der 15. Interation:**

(m) Abbildung 4.13: ABC: Differenz zwischen Original und 15. Iteration

(n) Abbildung 4.14: Möwe: Differenz zwischen Original und 15. Iteration

Im Folgendem betrachten wir den Unterschied zwischen dem Original und dem Endergebnis. Dazu wurde die Differenz der beiden Bilder berechnet und ein Grau addiert, um die Fehler besser erkennen zu können. Als Wert wurde hierbei auf Grundlage der Graustufeneinteilung aus Kapitel 1 80 gewählt. Wären die Bilder identisch, wäre die Abbildung also komplett einfarbig grau anstatt weiß.
Da die Kanten in (Abb. 4.14) gut sichtbar sind und sich auch in (Abb. 4.13) vereinzelt sehr helle Stellen am Rand befinden, folgt, dass der Algorithmus diese nicht optimal wiederherstellt. Das wird allerdings nur bei direktem Vergleich des Originals mit dem Endergebnis erkennbar und fällt bei reiner Betrachtung der Ergebnisse nicht direkt negativ auf. Auch innerhalb des Vogels in (Abb. 4.14) sind helle Stellen auszumachen. Die feinen Strukturen im Original werden also nicht perfekt rekonstruiert und verschwimmen etwas.

Abschließend wird deutlich, dass der Algorithmus zum einen recht schnell konvergiert und zum anderen ein solides und zufriedenstellendes Ergebnis liefert.

# Kapitel 5

# Grenzen

## 5.1 Rechenaufwand

Ein grundlegendes Problem der Implementierung stellt die Matrix $A$ dar, welche beispielsweise bei einem Bild der Größe $200 \times 200$, also 40.000 Pixel, bereits 3.200.000.000 Einträge besitzt. Selbst bei der Benutzung einer Char-Matrix, bei der jeder Eintrag lediglich $1Byte = 8Bit$ beanspruchen würde, wäre in diesem Beispiel die Matrix $A$ bereits 3,2GB groß. Deswegen bildet der Arbeitsspeicher bei herkömmlichen Rechnern bereits sehr schnell einen limitierenden Faktor.
Bei so großen Matritzen ist auch die Matrixmultiplikation ein Problem und erfordert großen Rechenaufwand. Die Laufzeit für $E^{m \times n} \cdot F^{n \times o}$ beträgt $\mathcal{O}(m \cdot n \cdot o)$. In einem Durchlauf (2.10) wird einmal $A^T \cdot y$ und einmal $A \cdot x$ berechnet, was beides jeweils in $\mathcal{O}(2N^2)$ liegt und den kompletten Algorithmus sehr hardwareauslastend und somit auch zeitintensiv macht.

## 5.2 Stärkeres Rauschen

Bis jetzt wurden lediglich Bilder mit dem gaußschen Standardrauschen getestet. Im Folgenden werden wir sehen, ob der Algorithmus auch einen stärkeren Störeffekt entfernen kann. Dazu nutzen wir das bereits aus Kapitel 4 bekannte Bild der Möwe und betrachten die Ergebnisse bei einem Rauschen mit $\sigma = 40$, $\sigma = 80$, $\sigma = 120$, $\sigma = 200$ und $\sigma = 500$.

(a) Abbildung 5.1: Moewe: $\sigma = 40$

(b) Abbildung 5.2: Moewe: $\sigma = 40$ Ergebnis

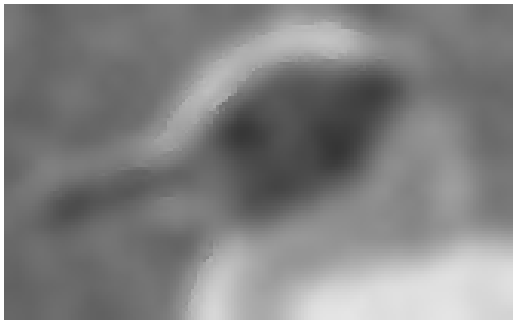
(c) Abbildung 5.3: Moewe: $\sigma = 80$

(d) Abbildung 5.4: Moewe: $\sigma = 80$ Ergebnis

(e) Abbildung 5.5: Moewe: $\sigma = 120$

(f) Abbildung 5.6: Moewe: $\sigma = 120$ Ergebnis

(g) Abbildung 5.7: Moewe: $\sigma = 200$



(h) Abbildung 5.8: Moewe: $\sigma = 200$ Ergebnis

(i) Abbildung 5.9: Moewe: $\sigma = 500$

(j) Abbildung 5.10: Moewe: $\sigma = 500$ Ergebnis

Daraus kann folgendes Fazit gezogen werden: Bis zu einem Rauschen mit Standardabweichung $\sigma = 80$ wird das ursprüngliche Bild noch gut erkennbar rekonstruiert. Bei allem was sich jenseits von $\sigma = 200$ befindet, kann man hingegen kaum noch das Original abschätzen, wobei dieses auch durch das starke Rauschen bereits nahezu unkenntlich gemacht wird.

# Kapitel 6

# Tests

Um dem im letzten Kapitel angesprochenen Problem des großen Speicheraufwandes aus dem Weg zu gehen, werden alle hier betrachteten Bilder in kleinere Quadrate unterteilt. Somit können auch größere, hochauflösendere Bilder betrachtet werden. Dies stört allerdings, wie folgendes Beispiel zeigt, bei guten Ergebnissen nur geringfügig. Im weiteren Verlauf des Kapitels wird sich herausstellen, dass die Unterteilung durchaus auffällt, wenn das Endergebnis an sich nur noch schlecht erkennbar ist.

(a) Abbildung 6.1: Möwe-Ergebnis ohne Unterteilung

(b) Abbildung 6.2: Möwe-Ergebnis mit Unterteilung

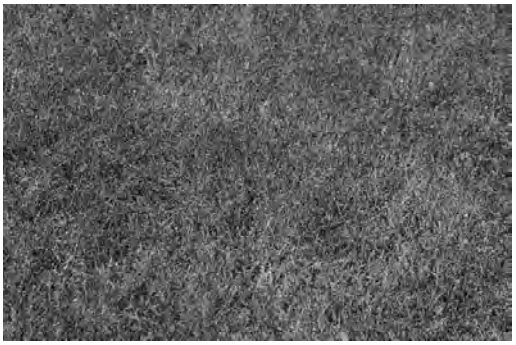
## 6.1 Wiese

Wir betrachten zuerst ein Stück Wiese, welches durch die vielen kleinen Grashalme eine sehr feine Struktur besitzt, die beim entrauschten Standardrauschen (Abb. 6.5) noch relativ gut erkennbar ist. Erhöht man den $\sigma$-Wert etwas, wird das Ergebnis allerdings sehr schnell sehr unkenntlich, was bei (Abb. 6.7) beziehungsweise (Abb. 6.9) deutlich wird. Dies liegt daran, dass der Algorithmus nicht eindeutig zwischen dem Gras und dem tatsächlichem Rauschen unterscheiden kann - auch mit bloßem Auge ist es schwer zu erkennen, um was es sich beim rauschenden Bild tatsächlich handelt, siehe (Abb. 6.6) beziehungsweise (Abb. 6.8). Hier fällt beim Ergebnis auch leicht die erwähnte Unterteilung des Bildes in kleinere Bereiche auf.

(c) Abbildung 6.3: Wiese: Original (Quelle: www.pixabay.com/de/gras-tapete-hintergrund-grün-natur-2119587)

(d) Abbildung 6.4: Wiese: Standardrauschen

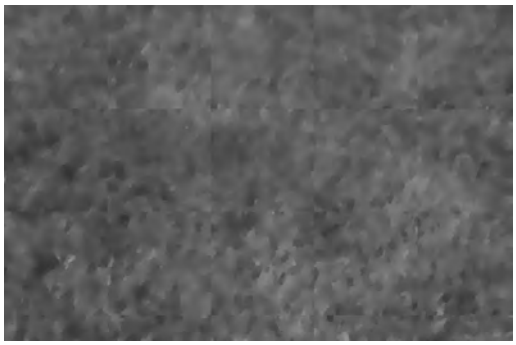
(e) Abbildung 6.5: Wiese: Standardrauschen Endergebnis



(f) Abbildung 6.6: Wiese: Rauschen mit $\sigma = 40$

(g) Abbildung 6.7: Wiese: $\sigma = 40$ Endergebnis

(h) Abbildung 6.8: Wiese: Rauschen mit $\sigma = 60$



(i) Abbildung 6.9: Wiese: $\sigma = 60$ Endergebnis

## 6.2 Fußabruck

Ein ähnliches Ergebnis würde man auch bei dem Fußabdruck im Sand (Abb. 6.10) erwarten, nämlich dass sowohl die vielen kleinen Unebenheiten im Sand als auch der Abdruck an sich, recht schnell verschwimmen. Dies ist allerdings nicht der Fall. Selbst beim Ergebnis nach sehr starkem Rauschen (Abb. 6.16) erkennt man das Original noch ohne Probleme. Zurückführen kann man das wahrscheinlich darauf, dass die Unebenheiten zu groß sind, um durch das Rauschen überdeckt zu werden.

(j) Abbildung 6.10: Abdruck: Original

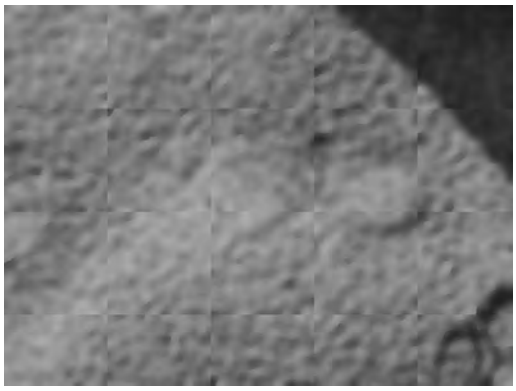
(k) Abbildung 6.11: Abdruck: Standardrauschen

(l) Abbildung 6.12: Abdruck: Standardrauschen Endergebnis

(m) Abbildung 6.13: Abdruck: Rauschen mit $\sigma = 80$

(n) Abbildung 6.14: Abdruck: $\sigma = 80$ Endergebnis

(o) Abbildung 6.15: Abdruck: Rauschen mit $\sigma = 160$



(p) Abbildung 6.16: Abdruck: $\sigma = 160$ Endergebnis

# Kapitel 7

# Fazit

Mit dem in dieser Arbeit hergeleiteten Algorithmus kann man größtenteils alle Bilder bis zu einem bestimmten Grad gut entrauschen. Probleme machen allerdings größere Bilder, da neben viel Rechenleistung auch viel Arbeitsspeicher benötigt wird. Die Unterteilung in kleinere Bilder hat zwar recht gut geklappt, ist aber keine perfekte Lösung hierfür, da sie durchaus auffallen kann. Deshalb ist es von Vorteil einen Rechner mit mehr Arbeitsspeicher für die Berechnungen zu verwenden, da so keine Aufteilung in kleinere Bilder notwendig ist.
Als weiterer Kritikpunkt sollen die willkürlich scheinenden Schrittweiten genannt werden. Da die Tests in Kapitel 2.3.1 allerdings zeigen, dass mit diesen eine vergleichbar sehr rasche Konvergenz erfolgt, wurde diese Frage nach hinten gestellt, um den Rahmen der Arbeit nicht zu sprengen.
Die Implementierung des Algorithmus war aufgrund vieler, anfangs oft undurchsichtigen, Variablen zunächst recht kompliziert. Nach Durchschauen des Aufbaus der Matrix $A$ konnte sie aber schlussendlich doch gut umgesetzt werden.

# Literaturverzeichnis


[Are13] Arens, Tilo: *Grundwissen Mathematikstudium: Analysis und Lineare Algebra mit Querverbindungen*. Berlin[u.a.] : Springer Spektrum, 2013

[Kar14] Karpfinger, Christian: *Höhere Mathematik in Rezepten: Begriffe, Sätze und zahlreiche Beispiele in kurzen Lerneinheiten*. Berlin[u.a.] : Springer Spektrum, 2014

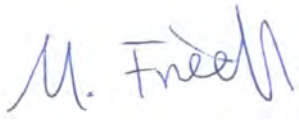
[Sau17] Sauer, Tomas: *Advanced Imaging*, 2017. – University of Passau

[ZC08] Zhu, Mingqiang ; Chan, Tony: *An Efficient Primal-Dual Hybrid Gradient Algorithm For Total Variation Image Restoration*, 2008

# Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Passau, 11. September 2018



Maximilian Friedl