Das Potts-Modell zum Entrauschen digitaler Mammographien

Projektbericht

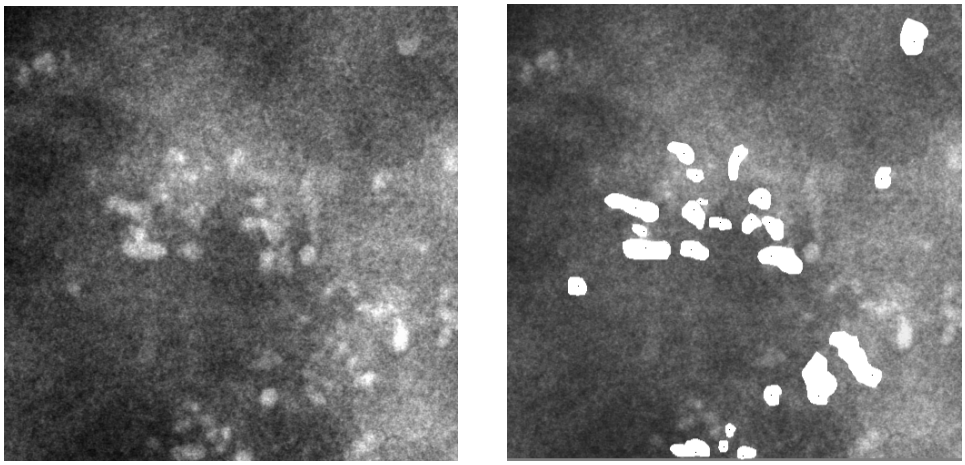
Katrin Wicker

Technische Universität München
Fakultät für Mathematik

Inhaltsverzeichnis

1 Problemstellung

Anwendungsbezogenes Ziel dieses Projektes ist die Entrauschung von digitalen Bilddaten, die aus Röntgenaufnahmen der weiblichen Brust entstanden sind.
Die Mammographie ist ein Werkzeug zur Früherkennung von Brustkrebs. Dabei haben sich als spezielle Merkmale für das Auftreten von Krebs sog. Mikrokalke etabliert: Kalkeinschlüsse, die auf Röntgenaufnahmen als kleine helle Flecken mit Durchmessern von ungefähr 0.2 - 1 mm zu erkennen sind. Die meisten maschinellen Werkzeuge zur Nachbearbeitung bzw. Analyse solcher digitalen Bilder versuchen das Auffinden solcher speziellen Intensitätsmaxima zu vereinfachen oder sogar vollständig zu automatisieren. Bestehende Verfahren zielen auf eine Bildverbesserung sowie auf Bereitstellung eines „second reader“ zur Unterstützung der Diagnose durch den Radiologen; vollautomatische Diagnose ist nicht angestrebt!



Ein Ausschnitt aus einer Mammographie, rechts wurden Mikrokalke von einem Radiologen eingefärbt.

Die Restauration, die in dieser Arbeit mit Hilfe des Potts Modells realisiert wird, soll nun nur einen allerersten Schritt für die weitere Analyse der Daten mit anderen Methoden darstellen. Ziel ist eine möglichst unspezifische Vereinfachung des Signals: Die Daten sollen einerseits geglättet, andererseits sollen aber Sprünge erhalten werden. Weitere Features, besonders Eigenschaften die sich aus der Art der Bildgewinnung ableiten, sollen in diesen ersten Schritt ausdrücklich nicht mit einbezogen werden.

Ziele des Projektes sind: Verständnis und Implementation eines 2-dimensionalen Potts-Minimierers und Untersuchung auf Eignung als Entrauschungsalgorithmus für Mammographien.

2 Das Potts Modell

An erster Stelle soll das Potts-Modell kurz in der Form angesprochen werden, in der es ursprünglich formuliert war. [1]
Zum Rekonstruieren verrauschter Daten gibt es im allgemeinen viele verschiedene Methoden. Wir wollen hier etwas formlos kurz die Bayes'sche Methode ansprechen, um im nächsten Abschnitt davon auch schon wieder Abstand zu nehmen.
Das Bayes'sche Paradigma unterstellt den zu beobachtenden, meist jedoch unbekannten Daten X eine gewisse *a priori* Verteilung π. Außerdem ist ein Verrauschungsprozeß in Form eines Übergangskerns Π vorgesehen, so dass die letztlich beobachteten Daten $Y = y$ einer gewissen Verteilung μ genügen:

$$\mu(y) = \mathbb{P}(Y = y) = \pi(X = x)\Pi(x, y).$$

So könnten die Beobachtungen y aus den Daten x beispielsweise durch additives Rauschen hervorgegangen sein:

$$y = x + r,$$

wobei r hier eine stochastische Größe, etwa eine normalverteilte Zuvallsvariable darstellt.
Mit dem Satz von Bayes lässt sich nun auch die Verteilung der (unbekannten) Daten X bei bekannten Beobachtungen $Y = y$ angeben. Aufgrund dieser a priori Verteilung $\pi(X = x|Y = y)$ lassen sich nun wiederum Schätzer für X angeben. Einer davon ist der *Maximum a Posteriori Schätzer* (MAP), nämlich ein Wert von x, so dass $\pi(X = x|Y = y)$ bei gegebenem y maximal wird. Wenn $\mathbb{P}(Y = y) = \pi(X = x)\Pi(X = x)$ in exponentieller Form

$$\mathbb{P}(Y = y) = exp(-H_y(x))$$

mit einer reellen Funktion $H_y(x)$ angegeben werden kann ist eine solche Maximierung äquivalent mit der Minimierung der sogenannten *Energie* $H_y(x)$.
Bevor das Potts-Modell angegeben werden kann, werden einige Notationen benötigt:

[1] Bayes Ansatz und stoch. Minimierung in Winkler (1995)

Notation 2.1 *Sei $d \in \mathbb{N}$. Im folgenden wird durch S eine endliche Teilmenge von $\mathbb{Z}^d$ (Indexmenge) und durch E eine endliche Teilmenge von $\mathbb{R}$ (Pixelmenge) bezeichnet. Die Menge*

$$E^S := \{(X_{i,i\in S}), X_i \in E\}$$

heißt im weiteren Text Zustandsraum.
Zwei Elemente $i, j \in S$ werden als benachbart bezeichnet, in Zeichen

$$i \sim j,$$

wenn gilt $||i - j|| = 1$, wobei hier $||\cdot||$ den euklidischen Abstand bezeichnet.

Sei nun $x \in E^S$ und $\gamma \in \mathbb{R}$, die Energie des Potts-Modells ist dann gegeben durch

$$H(X = x) = \gamma \sum_{i\sim j\in S} \chi_{\{x_i \neq x_j\}}.$$

Außerdem wollen wir hier zusätzlich einen quadratischen Datenterm vorsehen, nämlich

$$H(Y = y|X = x) = \sum_{i\in S} (x_i - y_i)^2.$$

Wir verstehen $\exp(H(x))$ als die (a-priori) Wahrscheinlichkeit für das Auftreten von x und $\exp(H(Y = y|X = x))$ als die Wahrscheinlichkeit für Auftreten von y bei gegebenem x. Die Berechnung eines MAP-Schätzers für X ist dann äquivalent zur Minimierung der Funktion

$$H_y(x) := H(X = x) + H(Y = y|X = x) = \gamma \sum_{i\sim j\in S} \chi_{\{x_i \neq x_j\}} + \sum_{i\in S} (y_i - x_i)^2.$$

in x.
Eine solche Minimierung kann wahrscheinlichkeitstheoretisch approximativ mit Simulated Annealing und ähnlichen stochastischen Algorithmen vorgenommen werden. [2] Diese haben jedoch einige Nachteile: Zum einen können die Laufzeiten solcher Algorithmen sehr lang sein. (auch abhängig von der Anzahl $|S|$ der Graustufen, wir haben hier 65000 Graustufen, das sind verhältnismäßig viele.) Die berechneten Minima können (nach endlicher Laufzeit) zum anderen nicht mit Sicherheit als globale Minima identifiziert werden.

[2] siehe z.B. Winkler and Liebscher (2002)

Außerdem sind aufgrund der stochastischen Natur solcher Algorithmen Ergebnisse meist nicht reproduzierbar und vergleichbar.
Daher werden in dieser Arbeit exakte Minimierungsalgorithmen angegeben, dafür wird im zweidimensionalen eine Variante des Potts-Modells vorgeschlagen.

3 Eindimensionale Minimierung

In diesem Abschnitt wird ein Algorithmus für die exakte Minimierung des o.a. Funktionals $H_y(x)$ im eindimensionalen Fall hergeleitet und angegeben. [3]

Für die eindimensionale Betrachtung des Potts Modell sei die Menge S nun gegeben durch

$$S := \{1, ..., N\}.$$

Benachbart heißen zwei Punkte s und t aus S also, wenn

$$|s - t| = 1 \qquad (s \sim t).$$

Die Abweichung der Daten y von x wird in einer Funktion $D(y, x)$ durch die quadratische Abweichung gemessen:

$$D(y, x) = \sum_{s \in S} (x_s - y_s)^2.$$

Darüber hinaus beschreibt eine weitere Funktion $G(x)$ die Regularität des Bildes x durch die Anzahl der benachbarten ungleichen Pixel,

$$G(x) = \gamma \sum_{s,t \in S : s \sim t} \chi_{\{x_s \neq x_t\}} = \gamma \cdot |\{s \sim t : x_s \neq x_t\}|.$$

Die beiden Funktionen $D(y, x)$ und $G(x)$ werden nun dem Ansatz aus Abschnitt 2 folgend in einer Funktion $H_y(x)$ zusammengesetzt.

$$H_y(x) = D(y, x) + G(x)$$

Die Konstante $\gamma \in \mathbb{R}$ steuert hier das Gewicht von Datentreue gegenüber Regularität. Gesucht wird eine Approximation

$$\hat{x} = (\hat{x}_s)_{s \in S} \in \mathbb{R}^S$$

[3] Dieser Algorithmus wurde in Winkler and Liebscher (2002) vorgestellt. Siehe auch Kempe (2003). Herleitung und Beweise wurden hier neu ausgearbeitet.

, welche einerseits die Abweichung der Daten y zu dem Bild x möglichst klein hält (Datenterm D) und andererseits die Anzahl der Sprünge (Regularisierung G) minimiert. Somit ist also nun das Ziel, die Funktion $H_y(x)$ in x bei festem (bekanntem) y und festem Parameter γ zu minimieren.
x induziert auf eindeutige Weise eine Darstellung

$$P = (I, \mu), \qquad I = \{I_1, \ldots, I_n\}, \qquad \mu = \{\mu_1, \ldots, \mu_n\}$$

von (x, S), d.h. eine Zerlegung von S in n Intervalle I_j, auf denen x jeweils den konstanten Wert μ_j besitzt:

$$I_j \subset S \quad j = 1, \ldots, n, \qquad I_1 \cup \ldots \cup I_n = S, \qquad I_j \cap I_k = \emptyset \quad \forall j \neq k.$$

$|P|$ ist die Anzahl der von x induzierten Intervalle I. Im folgenden bezeichnet eine Darstellung P eine Zerlegung des Indexraumes S in paarweise disjunkte Intervalle I_j, die zusätzlich mit *Gewichten* μ_j versehen sind.
Mit einer solchen Zerlegung definieren wir

$$g(P) := \gamma[|P| - 1]$$

und darüber hinaus

$$d(y, P) := \sum_{i=1}^{n} \sum_{j \in I_i} (\mu_i - y_j)^2$$

Somit lässt sich die Funktion $H_y(x)$ auffassen als eine Funktion von P, sofern $\mu_i \neq \mu_j$ für alle $i \sim j$. Dann gilt nämlich

$$H_y(x) = h_y(P).$$

Für den Fall, dass P eine Partition des Intervalles $\{m_1, \ldots, m_2\}$ mit $1 \leq m_1 \leq m_2 \leq N$ ist, sei

$$h_y(P) := h_{y|\{m_1,\ldots,m_2\}}(P).$$

Die Minimierung von $H_y(x)$ bzgl. x ist also gleichbedeutend mit der Konstruktion einer *optimalen* (d.h. $h_y(P)$ minimierenden) Darstellung P, sofern benachbarte Intervalle verschiedene Werte annehmen. [4] (Der Minimierer ist aber nicht notwendigerweise eindeutig.) Dies ist der entscheidende Punkt sowohl für die hier besprochene eindimensionale Minimierung als auch für die zweidimensionale Variante (s.u.).

[4] Aufgrund dieser Restriktion beziehen sich die weiteren Berechnungen auf eine Minimierung von $h_y(P)$, die Äquivalenz mit der Minimierung von $H_y(x)$ wird danach gezeigt.

Notation 3.1 *Sei A eine beliebige Teilmenge der Indexmenge S. Dann bezeichne*

$$\hat{\mu}(A) := \frac{1}{|A|} \sum_{i \in A} y_i, \qquad \hat{\sigma}(A) := \sum_{i \in A} (\hat{\mu}(A) - y_i)^2$$

In dem Intervall $Q_l^k := \{l, \ldots, k\}$ mit $1 \leq l \leq k \leq N$ bezeichnet

$$\mu_l^k := \hat{\mu}(Q_l^k)$$

den Mittelwert der Daten y, und

$$\sigma_l^k := \hat{\sigma}(Q_l^k)$$

die quadratische Abweichung des Vektors x von den Daten y jeweils im Intervall Q_l^k.

Satz 3.1 *Gegeben sei eine Zerlegung von $\{1, \ldots, N\}$ in Intervalle $I = (I_1, \ldots, I_n)$, $n \leq N$. Der Minimierer der Funktion $h_y(P)$ unter allen auf den Intervallen $I_1, \ldots, I_n$ konstanten Vektoren μ ist gegeben durch*

$$\hat{\mu}(P) = (\hat{\mu}(I_1), \ldots, \hat{\mu}(I_n)).$$

Beweis Für festgelegte Intervalle ist die Funktion $h_y(P)$ gegeben durch

$$h_y(P) = g(P) + d(y, P) = \gamma |P| + \sum_{i=1}^{n} \sum_{j \in I_i} (\mu_i - y_j)^2.$$

Da dann $g(P)$ konstant ist, ist die Minimierung von $h_y(P)$ somit gleichbedeutend mit der Minimierung des zweiten Terms $d(y, P) = \sum_{i=1}^{n} \sum_{j \in I_i} (\mu(I_i) - y_j)^2$. Dieser ist dann minimal wenn jede einzelne Summe $\sum_{j \in I_i} (\mu(I_i) - y_j)^2$ für jedes Intervall I minimal ist. Somit ist jeweils der optimale Wert das empirische Mittel $\hat{\mu}(I_i)$.

■

Satz 3.2 *Es existiert ein Minimierer für die Funktion $h_y(P)$.*

Beweis Nach Satz 3.1 ist für jede feste Partition der Minimierer bekannt, deshalb muss nur noch über die endliche Menge von Partitionen von $\{1, \ldots, N\}$ minimiert werden. Somit ist die Existenz klar. ■
Für ein $k \in \{1, \ldots, N\}$ bezeichne $\hat{P}^k$ eine optimale Darstellung zu den Daten $y_1, \ldots, y_k$, weiter sei $\hat{P}_0 = \emptyset$ und $\hat{P}^1 = (I_1^1, \mu_1^1)$.

Lemma 3.1 *Seien P_1 und P_2 Partitionen von $\{1,\ldots,m\}$ bzw. $\{m+1,\ldots,N\}$ mit $1 \le m \le N$. Dann gilt*

$$h_y(P_1,P_2) = h_y(P_1) + h_y(P_2) + \gamma.$$

Insbesondere gilt

$$h_y(P_1, Q_{m+1}^N) = h_y(\hat{P}_1) + \gamma + \sigma_{m+1}^N.$$

Beweis

$$\begin{aligned}
h_y(P_1,P_2) &= g(P_1,P_2) + d(y,(P_1,P_2)) \\
&= \gamma(|P_1,P_2|-1) + \sum_{I\in(P_1,P_2)} \sum_{j\in I} (\hat{\mu}(I) - y_j)^2 \\
&= \gamma(|P_1|-1) + \gamma(|P_2|-1) + \gamma \\
&\quad + \sum_{I\in P_1} \sum_{j\in I} (\hat{\mu}(I) - y_j)^2 + \sum_{I\in P_2} \sum_{j\in I} (\hat{\mu}(I) - y_j)^2 \\
&= g(P_1) + g(P_2) + \gamma + d(y,P_1) + d(y,P_2) \\
&= h_y(P_1) + h_y(P_2) + \gamma
\end{aligned}$$

$$\begin{aligned}
h_y(P_1, Q_{m+1}^N) &= h_y(P_1) + h_y(Q_{m+1}^N) + \gamma \\
&= h_y(P_1) + \gamma + \sigma_{m+1}^N
\end{aligned}$$

■

Kern des Algorithmus für die eindimensionale Minimierung ist nun folgende Beobachtung:

Satz 3.3 *Sei $k,l \in \{1,\ldots,N\}$ mit $l < k$ und*

$$P_l^k = (\hat{P}^l, Q_{l+1}^k),$$

Sei $\hat{l}$ ein Minimierer von $h_y(P_l^k)$ in l, also

$$\hat{l} \in \{l \in \{0,\ldots,k-1\} : h_y(P_l^k) \text{ ist minimal }\},$$

dann ist $P_{\hat{l}}^k$ ein Minimierer der Funktion $h_y(P^k)$, d.h. es gilt

$$h_y(P_{\hat{l}}^k) = h_y(\hat{P}^k), \quad k > 1.$$

Beweis Sei $J(\{1,\ldots,N\})$ die Menge der Partitionen von $\{1,\ldots,N\}$. Aus Satz 3.1 und Lemma 3.1 folgt für eine optimale Darstellung

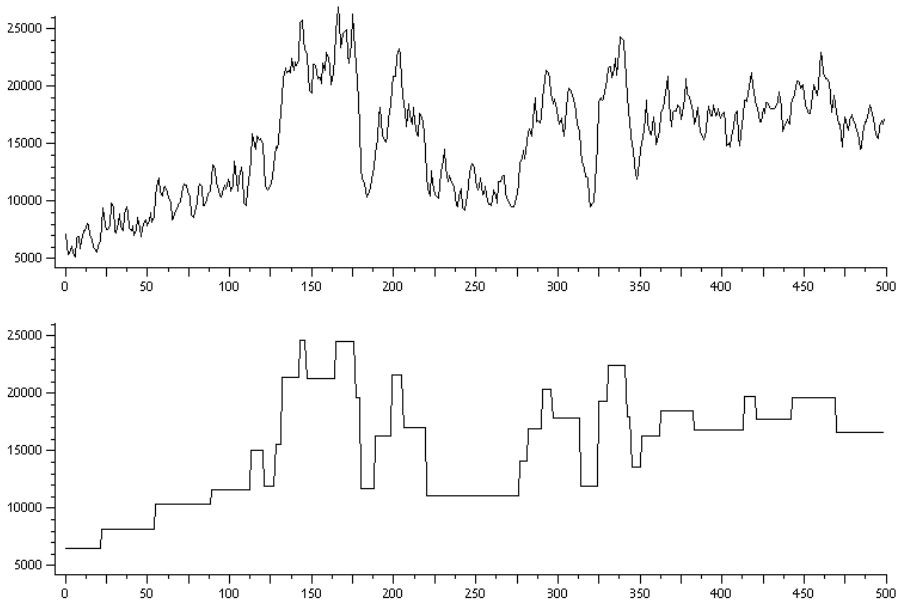
$$\begin{aligned}
h_y(\hat{P}^N) &= \min_{\tilde{P}\in J(\{1,\ldots,N\})} h_y(\tilde{P}) \\
&= \min_{k\in\{1,\ldots,N\}} \min_{\tilde{P}\in J(\{1,\ldots,k\})} \{h_y(\tilde{P}, I_k^N)\} \\
&= \min_{k\in\{1,\ldots,N\}} \min_{\tilde{P}\in J(\{1,\ldots,k\})} (h_y(\tilde{P}) + \gamma + \sigma_k^N) \\
&= \min_{k\in\{1,\ldots,N\}} \left[\left(\min_{\tilde{P}\in J(\{1,\ldots,k\})} h_y(\tilde{P}) \right) + \gamma + \sigma_k^N \right] \\
&= \min_{k\in\{1,\ldots,N\}} (h_y(\hat{P}^k) + \gamma + \sigma_k^N) \\
&= h_y(P_{\hat{k}}^N)
\end{aligned}$$

Der Schluss des Beweises erfolgt per Induktion. ∎

Satz 3.4 *Ein Minimierer von h_y ist gleichzeitig Minimierer von H_y.*

Beweis Zu zeigen ist nur, dass $\mu_i \neq \mu_j$ für benachbarte Intervalle I_i und I_j gilt. Das ist aber aus dem Beweis zu Satz 3.3 sofort ersichtlich. ∎
Somit lässt sich ein Algorithmus zur Minimierung von $h_y(P)$ nun angeben, er läuft auch induktiv:

- VAR $\hat{l}$: ARRAY $N+1$ OF INT; h: ARRAY $N+1$ OF REAL;
- $h(0) = h(1) := 0$, $l(0) = l(1) := 0$, $k := 2$
- WHILE($k \leq N$) DO
  $\hat{l}(k) := \arg\min_{l\in\{0,\ldots,k-1\}}(h(l) + \gamma + \sigma_{l+1}^k)$
  $h(k) := \min_{l\in\{0,\ldots,k-1\}}(h(l) + \gamma + \sigma_{l+1}^k)$
  $k := k + 1$
  END;
- $k := N$, $\hat{P}^k = P_{\hat{l}(k)}^k$ (rekursiv)



Original (Zeile 270 aus obiger Mamographie) und entrauschtes Signal, $\gamma = 0.1 median^2$

4 Zweidimensionale Variante des Potts Modells

In diesem Abschnitt sei nun $S \subset \mathbb{Z}^2$, $S = \{1, \ldots, N_1\} \times \{1, \ldots, N_2\}$. Im zweidimensionalen Fall lautet das Potts-Modell

$$\begin{aligned} H_y(x) &= D(y,x) + G(x) \\ &= \sum_{i \in S} (x_i - y_i)^2 + \gamma \sum_{i \sim j} 1_{\{x_i \neq x_j\}} \\ &= \sum_{i \in S} (x_i - y_i) + \gamma |\{i \sim j : x_i \neq x_j\}| \end{aligned}$$

Der Algorithmus aus dem ersten Teil kann hier nicht angewendet werden, da bei dem Übergang von dem eindimensionalen Fall zum zweidimensionalen Fall eine Bedingung der Form $h_y(P_l^k) = h_y(P^{\hat{l}}) + h_y(I_l^k) + c$ nicht übertragen werden kann. Es ist scheinbar nicht möglich, eine Unterteilung für $\mathbb{Z}^2$ zu finden, so dass dieses Problem eine exakte Lösung besitzt. Im folgenden wird daher das Potts-Modell etwas modifiziert, um sowohl einen exakten als auch

einen relativ zeit-unkritischen Algorithmus zu finden.

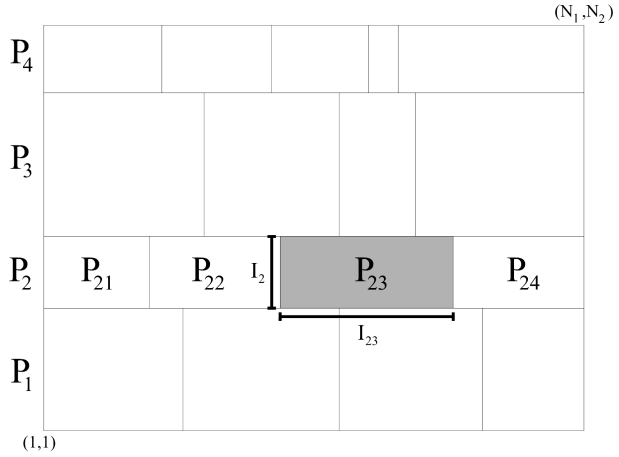
In der Indexmenge S wird zunächst in vertikaler Richtung eine Partition P induziert, und anschließend wird für jedes Intervall der vertikalen Partition eine Unterteilung in Intervalle in horizontaler Richtung bestimmt. Es werden also Partitionen der Form

$$P = \bigcup_{i=1}^{n} P_i, \qquad I := \{I_1, \ldots, I_n\}$$

mit

$$P_i = \{P_{i1}, \ldots, P_{im_i}\}, \qquad P_{ij} = I_i \times I_{ij}$$

bestimmt, wobei I die Partition in vertikaler Richtung von $\{1, \ldots, N_1\}$ ist, und die P_i sind die Partitionen in horizontaler Richtung von $\{1, \ldots, N_2\}$ für jedes Intervall $I_i (1 \leq j \leq n)$ der vertikalen Partition.
$|P|$ ist die Anzahl der Intervalle der Partition P.



2-d Partitionen P

Die betrachtete Energiefunktion lautet nun

$$\begin{aligned} h_y(P) &= d(y,P) + g(P) \\ &= \gamma|P| + \sum_{J\in P}\sum_{j\in J}(\mu(J) - y_j)^2 \\ &= \gamma\sum_{i=1}^{n}|P_i| + \sum_{i=1}^{n}\sum_{J\in P_i}\sum_{j\in J}(\mu(J) - y_j)^2 \end{aligned}$$

$$= \sum_{i=1}^{n} \underbrace{\left[\gamma |P_i| + \sum_{J \in P_i} \sum_{j \in J} (\mu(J) - y_j)^2 \right]}_{h_y^*(P_i)}$$

Das Ziel ist nun, wieder eine Minimierung für diese Funktion $h_y(P)$ zu finden. Beachte: Hier lässt sich ein Ergebnis aber nicht auf das eigentliche Potts-Modell übertragen; die Analogie, die im eindimensionalen Fall aufgetreten ist lässt sich leider nicht auf diesen Fall übertragen.
Analog zu Satz 3.1 wird $h_y(P)$ bei festgelegter Partition $P = (P_1, \ldots, P_n)$ in μ wiederum durch

$$\mu(J) = (\hat{\mu}(J)) \quad \forall J \in P_I$$

minimiert.

Bemerkung 1 Zur Vereinfachung der Notation wird jede Partition mit optimalen Gewichten $\hat{\mu}(P_{ij})$ versehen und diese nicht mehr gesondert angegeben.

Satz 4.1 *Für jedes Intervall $I \subset \{1, \ldots, N_1\}$ lässt sich $h_y^*(P_i)$ in P_i analog zu Satz 3.3 minimieren.*

Beweis Sei $P_i = (p, q)$. Dann ist

$$h_y^*(P_i) = h_y^*(p, q) = h_y^*(p) + h_y^*(q).$$

Somit lässt sich hier die Struktur des Beweises von Satz 3.3 übernehmen. ■
Wie oben sei $\hat{P}_l$ die optimale Partition, hier diejenige in vertikaler Richtung, darüberhinaus sei noch $\hat{Q}_l^k$ der Minimierer von $h_y^*(Q_l^k)$ mit $Q_l^k = ((J \times J_1), \ldots, (J \times J_m))$, wobei J das vertikale Interval $\{l, \ldots, k\}$ ist, und $(J_1, \ldots, J_m)$ die dazugehörige Partition von $(1, \ldots, N_2)$ in horizontaler Richtung ist.

$$\{I_{l+1,1}, \ldots, I_{l+1,m_{l+1}}\}.$$

Satz 4.2 *Sei $k, l \in \{1, \ldots, N_1\}$ mit $l < k$ und sei*

$$P_l^k = (\hat{P}^l, \hat{Q}_l^k),$$

Sei $\hat{l}$ ein Minimierer von $h_y(P_l^k)$ in l, also

$$\hat{l} \in \{l \in \{0, \ldots, k-1\} : h_y(P_l^k) \text{ ist minimal }\},$$

dann ist $P_{\hat{l}}^k$ ein Minimierer der Funktion $h_y(P^k)$, d.h. es gilt

$$h_y(P_{\hat{l}}^k) = h_y(\hat{P}^k), \quad k > 1.$$

Beweis Sei $P = (P_1, \ldots, P_n)$ und $P^1 = (P_1, \ldots, P_k)$, $P^2 = (P_{k+1}, \ldots, P_n)$, mit $1 \leq k < n$. Dann ist

$$h_y(P^1, P^2) = h_y(P^1) + h_y(P^2).$$

Der Beweis lässt sich also analog wie der von Satz 3.3 zu Ende führen. ∎

Bemerkung 4.1 *Selbst wenn man statt der Anzahl der Partitionen auch noch deren Kantenlänge bestraft, also*

$$h_y(P) = \sum_{I=1}^{|P|} \gamma(N_1 + |I||P_i|) + \sum_{i=1}^{|P|} \sum_{J \in P_i} \sum_{j \in J} (\mu(J) - y_j)^2$$

setzt, lassen sich die beiden letzten Sätze noch beweisen und eine Minimierung ist auf die gleiche Weise möglich. Man beachte aber wiederum, dass dies nicht die Minimierung des ursprünglichen Potts-Funktional bedeutet.

Der Algorithmus lautet nun:

- VAR $\hat{l}$: ARRAY $N_1 + 1$ OF INT; h: ARRAY $N_1 + 1$ OF REAL;
- $h(0) = h(1) := 0$, $l(0) = l(1) := 0$ $k := 2$
- WHILE($k \leq N$) DO
  $\hat{l}(k) := \arg\min_{l \in \{0,\ldots,k-1\}}(h(l) + \text{getmin}(l, k))$
  $h(k) := \min_{l \in \{0,\ldots,k-1\}}(h(l) + \text{getmin}(l, k))$
  $k := k + 1$
  END;
- $k := N_1$, $\hat{P}^k = P^k_{\hat{l}(k)}$ (rekursiv)
- **g**etmin(from,to)

- ⋆ VAR $\hat{l}$: ARRAY $N_2 + 1$ OF INT; h: ARRAY $N_2 + 1$ OF REAL;
- ⋆ $h(0) = h(1) := 0$, $l(0) = l(1) := 0$ $k := 2$
- ⋆ WHILE($k \leq N$) DO
  $\hat{l}(k) := \arg\min_{l \in \{0,\ldots,k-1\}}(h(l) + \hat{\sigma}(\{\text{from}, \ldots, \text{to}\} \times \{l, \ldots, j\}) + \gamma)$
  $h(k) := \min_{l \in \{0,\ldots,k-1\}}(h(l) + \hat{\sigma}(\{\text{from}, \ldots, \text{to}\} \times \{l, \ldots, j\}) + \gamma)$
  $k := k + 1$
  END;

END;

Notation 4.1 *Sei $l, k \in \mathbb{Z}^2$. Es heisst $l \leq k$ genau dann wenn $l_1 \leq k_1$und $l_2 \leq k_2$ ist. Die Intervalle in $\mathbb{Z}^2$ von S werden bezeichnet mit*

$$I_l^k := \{j \in \mathbb{Z}^2 : l \leq j \leq k\},$$

I_1^1 bezeichnet das Intervall $\{(1,1)\}$. Analog zu dem eindimensionalen Fall bezeichnet im folgenden

$$\mu_l^k := \hat{\mu}(I_l^k)$$

den Mittelwert und

$$\sigma_l^k := \hat{\sigma}(I_l^k)$$

die quadratische Abweichung der Daten x von dem Mittelwert μ_l^k im Intervall I_l^k.

Um die Partitionen in $O(N_2^2)$ bestimmen zu können, müssen in den Intervallen der Mittelwert und die Summe der Quadrate in einer festen Anzahl von Schritten berechenbar sein.
Der Mittelwert in einem Rechteck des Intervalls I_l^k:

$$\mu_l^k = \frac{1}{|I_l^k|} \sum_{i \in I_l^k} y_i$$

Die Summe der Quadrate in dem Intervall I_l^k lässt sich folgendermassen berechnen:

$$\sigma_l^k = \sum_{i \in I_l^k} (y_i - \mu_l^k)^2$$

$$
\begin{aligned}
&= \sum_{i\in I_l^k}^{k} y_i^2 - 2\sum_{i\in I_l^k}^{k} y_i\mu_l^k + \sum_{i\in I_l^k}(\mu_l^k)^2 \\
&= \sum_{i\in I_l^k}^{k} y_i^2 - 2\mu_l^k \sum_{i\in I_l^k}^{k} y_i + |I_l^k|(\mu_l^k)^2 \\
&= \sum_{i\in I_l^k}^{k} y_i^2 - 2\frac{1}{|I_l^k|}(\sum_{i\in I_l^k} y_i)^2 + |I_l^k|\frac{1}{|I_l^k|^2}(\sum_{i\in I_l^k} y_i)^2 \\
&= \sum_{i\in I_l^k} y_i^2 - \frac{1}{|I_l^k|}(\sum_{i\in I_l^k} y_i)^2
\end{aligned}
$$

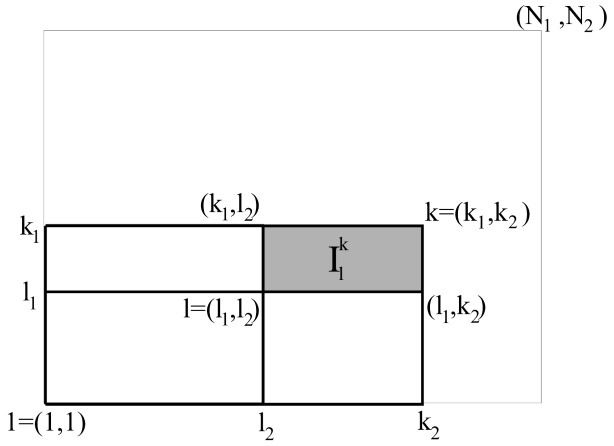
Das bedeutet, dass nur Quadratsummen und Summen über beliebige Rechtecke benötigt werden. Mit Hilfe der Matrix der kumulativen Summen ist das möglich.
Die Matrix B der kumulativen Summen wird berechnet, indem im ersten Schritt die Zeilen der Matrix $Y = (y_s)_{s\in S} = (y_{ij})$ kumulativ aufsummiert werden, im zweiten Schritt dann die Spalten: $B = (b_{ij})$ mit $b_{ij} = \sum_{\substack{i\le N_1\\ j\le N_2}} y_{i,j}$.
Für die Quadratsummen werden die Elemente der Matrix Y zuvor quadriert und dann kumulativ aufsummiert.

$$
Y = \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1N_2} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N_1 1} & y_{N_1 2} & \cdots & y_{N_1 N_2} \end{pmatrix}
$$

$$
B = (b_{ij}) = \begin{pmatrix} y_{11} & \sum_{j=1}^{2} y_{1j} & \cdots & \sum_{j=1}^{N_2} y_{1j} \\ \sum_{i=1}^{2} y_{i1} & \sum_{j=1}^{2}\sum_{i=1}^{2} y_{ij} & \cdots & \sum_{j=1}^{N_2}\sum_{i=1}^{N_1} y_{ij} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{N_1} y_{i1} & \sum_{j=1}^{2}\sum_{i=1}^{N_1} y_{ij} & \cdots & \sum_{j=1}^{N_2}\sum_{i=1}^{N_1} y_{ij} \end{pmatrix}
$$

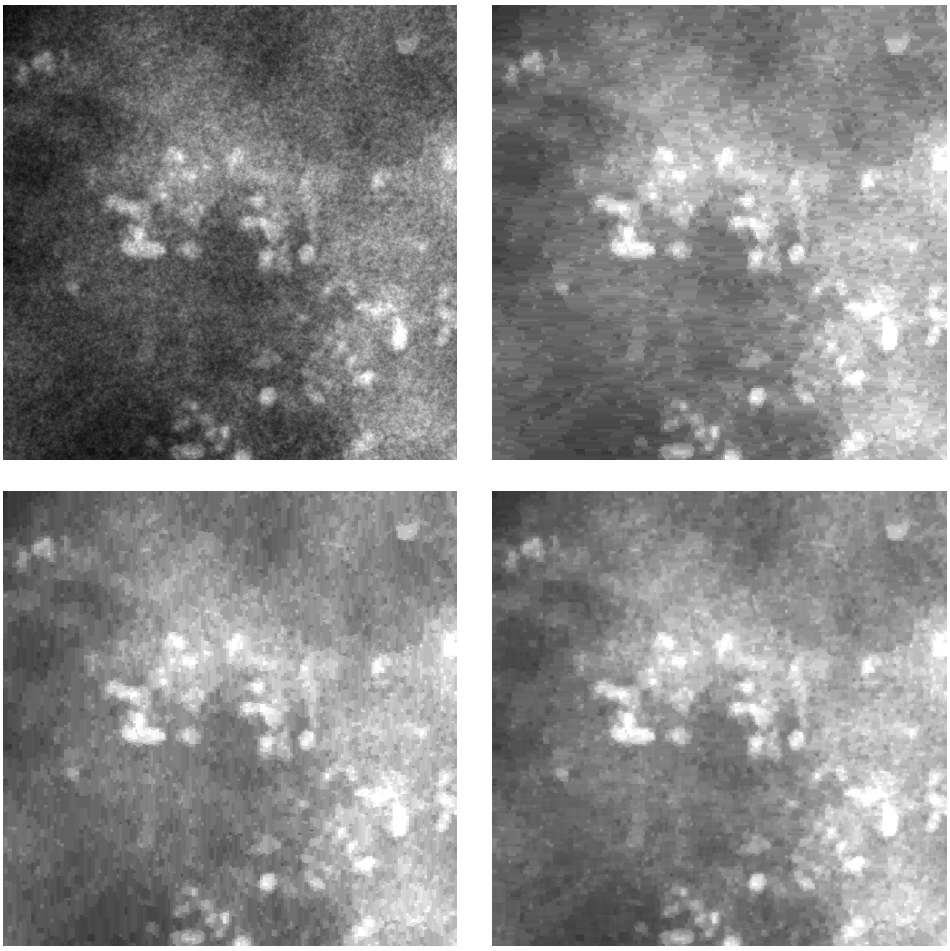
Die benötigten Summen $\sum_{i\in I_l^k} y_i = \sum_{\substack{l_1<i\le k_1\\ l_2<j\le k_2}} y_{ij}$ bzw. Quadratsummen $(\sum_{i\in I_l^k} y_i)^2 = (\sum_{\substack{l_1<i\le k_1\\ l_2<j\le k_2}} y_{ij})^2$ für die Berechnung des Mittelwertes und der Varianz in dem Intervall I_l^k werden aus der Matrix B folgendermassen berechnet:

$$
\begin{aligned}
\sum_{i\in I_l^k} y_i = \sum_{\substack{l_1<i\le k_1\\ l_2<j\le k_2}} y_{ij} &= \sum_{\substack{0<i\le k_1\\ 0<j\le k_2}} y_{ij} - \sum_{\substack{0<i\le k_1\\ 0<j\le l_2}} y_{ij} - \sum_{\substack{0<i\le l_1\\ 0<j\le k_2}} y_{ij} + \sum_{\substack{0<i\le l_1\\ 0<j\le l_2}} y_{ij} \\
&= b_{k_1k_2} - b_{k_1l_2} - b_{l_1k_2} + b_{l_1l_2}
\end{aligned}
$$



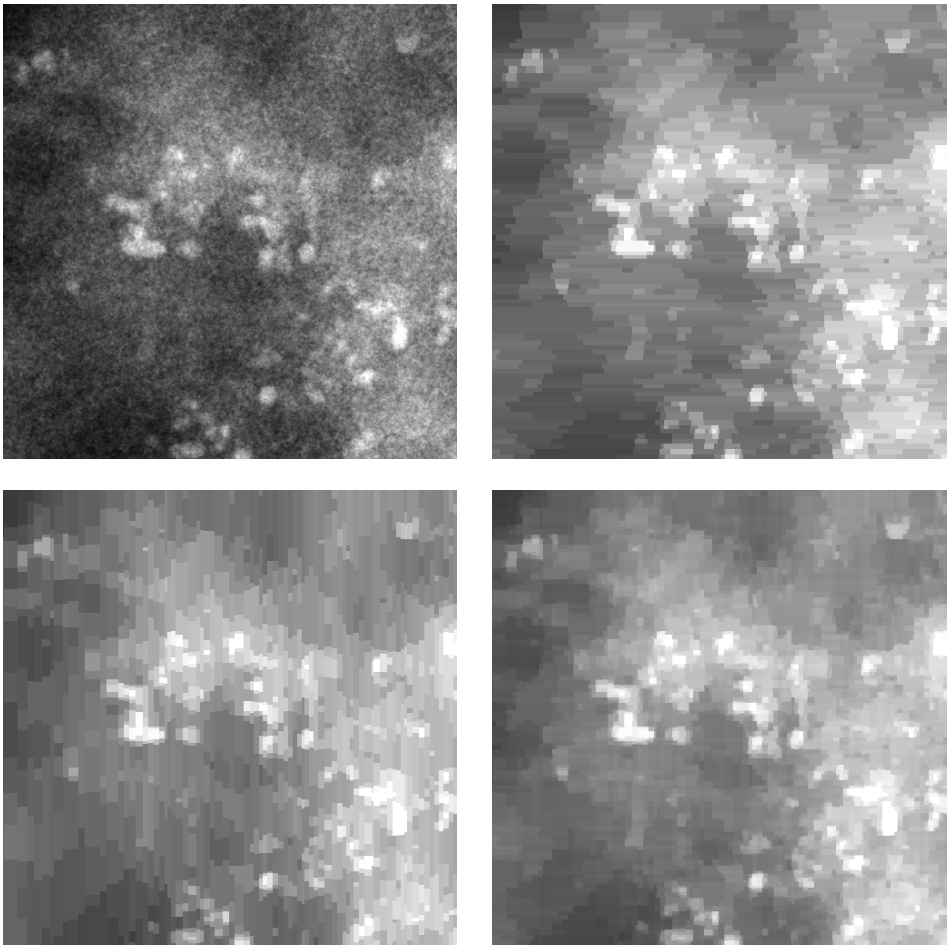
Verallgemeinertes Intervall I_l^k

5 Beispiele



Anwendung des zweidimensionalem Algorithmus auf den Ausschnitt aus einer Mammographie. $\gamma = 0.1$

Original, horizontale und vertikale Rekonstruktion sowie Summe aus diesen

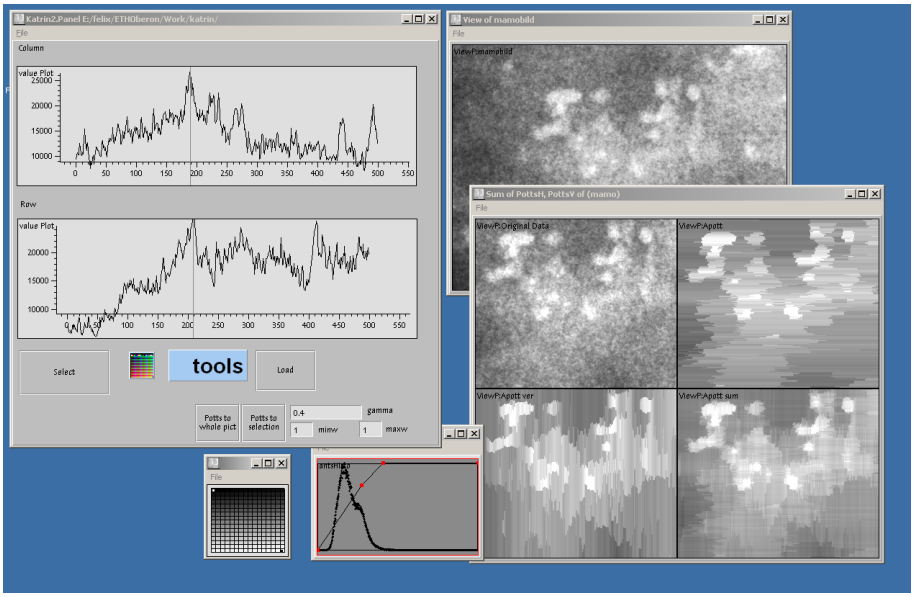


Anwendung des zweidimensionalem Algorithmus auf den Ausschnitt aus einer Mammographie. $\gamma = 0.5$

Original, horizontale und vertikale Rekonstruktion sowie Summe aus diesen

6 Zur Software

Die Algorithmen wurden in Oberon[5] geschrieben, dabei wurden Routinen von Ants[6] und Voyager[7] verwendet.



Screenshot: Oberon / Ants / Voyager und Potts-Minimierung

Eine Minimierungsroutine für den eindimensionalen Fall war vorhanden[8], diese wurde für generelle Zwecke angepaßt. Für den eindimensionalen Fall wurden die Routinen zur Generierung der Statistik optimiert (Provisional means etc.), um sie auf große Bilder in akzeptabler Laufzeit anwenden zu können. Für den zweidimensionalen Fall wurden entsprechende Prozeduren für die Mittelwerte und Varianzen in Intervallen auf Bildern erstellt. Außerdem wurden die Energiefunktionale im zweidimensionalen Fall so variabel gestaltet, dass zum einen eine Begrenzung der Intervallbreite, zum anderen die Verwendung der Kantenlängen einbezogen werden kann. Die Minimierungsroutinen können interaktiv per Mausklick auf Bilder angewendet werden oder per Kommando auf ganzen Verzeichnissen von Bildern operieren.

[5] Betriebssystem und Programmiersprache von der ETH Zürich
[6] F.Friedrich, IBB, GSF
[7] 1993-2001, G.Sawitzki, F.Friedrich, M.Diller et al.
[8] A.Kempe, F.Friedrich

Hier soll nun beispielhaft die zentrale Minimierungsroutine angegeben werden, der restliche Code befindet sich im Anhang.

```
PROCEDURE Bellman* (VAR rstern: ARRAY OF INTEGER; VAR bellman: ARRAY OF LONGREAL;   n, minweite, maxweite: LONGINT; p: Function);
VAR i, minpos: LONGINT;  min,current: LONGREAL; from,to: LONGINT;
BEGIN
    min := MAX(LONGREAL);  from := Max(0,n-maxweite);  to := Min(n-1,n-minweite);
    FOR i := from TO to DO  (*Bestimmen des Min fuer die jeweilige Partiton*)

          current :=  bellman[i]+ p(i, n-1);
          IF current < min THEN  min := current;   minpos := i END;
    END;
    bellman[n] := min;
    rstern[n] := SHORT(minpos);
END Bellman;

PROCEDURE SetBellmanFunction* (VAR rstern: IVect;   VAR bellman: PVect;  N, minweite, maxweite: LONGINT;  p: Function; initvalue: LONGREAL);
VAR i: LONGINT;
BEGIN
    IF (bellman = NIL) OR (LEN(bellman) # N+1) THEN
        NEW(bellman,N+1);                          NEW(rstern,N+1);
    END;
    bellman[0] := initvalue;  bellman[1] := 0;
    FOR i := 2 TO (N-1) DO (* vorsicht: Index-Verschiebung durch den Startwert der Bellman-Fkt *)
        Bellman(rstern^, bellman^, i, minweite, maxweite,p);
    END;
    Bellman(rstern^, bellman^, N, 0, maxweite,p);

    rstern[0] := MIN(INTEGER);                       rstern[1] := 0;
END SetBellmanFunction;
```

7 Literatur

Literatur

A. Kempe. *Statistical analysis of the Potts model and applications in medical imaging*. PhD thesis, Institute of Biomathematikcs and Biometry, National Research Center for Environment and Health, Munich, Germany, 2003.

G. Winkler. *Image Analysis, Random Fields and Dynamic Monte Carlo Methods*, volume 27 of *Applications of Mathematics*. Springer Verlag, Berlin, Heidelberg, New York, 1995.

G. Winkler and V. Liebscher. Smoothers for discontinuous signals. *J. Nonpar. Statist.*, 14(1-2):203–222, 2002.

8 Code

8.1 Modul katrinPotts

```
MODULE katrinPotts;                          (** kjw  **)


IMPORT vyBase, vyLongrealVec, antsToolbox, antsArrayObjects, Strings := vyHostStrings, antsCommands, Out, vyName,  antsImageViews,
antsPictureViews, aM := antsMessages, Input, Objects, vyPort, Oberon, angieAIC, vyLongreal, antsRegions,   antsSecurity, katrinBellman,
katrinMatrix, antsReadWrite, FileDir, antsTrapper, antsPlots;

TYPE
    PVect = angieAIC.PVect;
    PMatr = angieAIC.PMatr;
    IVect = angieAIC.IVect;

    Statistic = angieAIC.Statistic;

VAR
    globalStat: Statistic;
    globalAkum, globalAkumSq: antsArrayObjects.tLONGREALArray;
    globalGamma: LONGREAL;
    globaly1, globaly2: LONGINT;
    globalRstern: IVect;            globalBellman: PVect;
    globalMinWeite, globalMaxWeite: LONGINT;
    inDirectory, outDirectory: ARRAY 256 OF CHAR;


PROCEDURE OpenMammo*;
VAR name, file: ARRAY 256 OF CHAR; o: antsArrayObjects.tINTEGERObject;
BEGIN
    antsCommands.InitScan();
    IF antsCommands.ExpectName( "name",  name) & antsCommands.ExpectString( "file",  file) THEN
        Out.String(name);           Out.Ln;
        Out.String(file);           Out.Ln;
        antsArrayObjects.ReadPGM(o, file);
        vyName.RegisterNamed(o, name)
    END
END OpenMammo;

(* Interaktive Routine zum Anzeigen der Zeilen und Spalten eines 2d-Bildes *)
PROCEDURE TakeVectors* (row, col: vyBase.tVector;r, c: vyBase.Object);
VAR msg: antsToolbox.GetCoordMsg; x, y: LONGINT; o: antsArrayObjects.tObject; xm, ym: INTEGER; keys: SET;
layer : Objects.Object;coords: ARRAY 64 OF CHAR;
BEGIN
    x := MAX(LONGINT);          y := MAX(LONGINT);
    msg.layer := NIL;           msg.update := TRUE;
    antsToolbox.InteractiveMsg(msg);            Out.Ln;         Out.String("sent first im");
    layer := msg.layer;
    Input.Mouse(keys, xm, ym);          msg.x := MAX(INTEGER);          msg.y := MAX(INTEGER);
    WHILE(keys # {}) DO
        IF (msg.x # xm) OR (msg.y # ym) THEN
            msg.x := xm;            msg.y := ym;
            msg.update := FALSE;            msg.layer := NIL;
            Out.Ln;         Out.String("Sending msg to typed layer");
            antsToolbox.MsgToTypedXY(xm, ym, msg, "antsViews.tLayerDesc");
            Out.Ln;         Out.String("sent msg, layer found ?");
            IF msg.layer # layer THEN msg.update := TRUE; antsToolbox.MsgToTypedXY(xm, ym, msg, "antsViews.tLayerDesc"); layer := msg.layer END;
            coords := "coords: ";
            IF msg.layer # NIL THEN
                Out.Ln;      Out.String("msg.layer # NIL");
                IF msg.layer IS antsImageViews.tLayer  THEN
                    x := ENTIER(msg.xr);                        y := ENTIER(msg.yr);
                    o := msg.layer(antsImageViews.tLayer).E.A;
                    IF (0 <= x) & (0 <= y) & (x < o.w) & (y < o.h) THEN
                        antsArrayObjects.GetRowCol(x, y, o, col, row)
                    END;
                    vyBase.Update(row);         vyBase.Update(col);
                    IF r # NIL THEN r(vyLongrealVec.tLONGREAL).x := y;          vyBase.Update(r) END;
                    IF c # NIL THEN c(vyLongrealVec.tLONGREAL).x := x;          vyBase.Update(c) END;
                    coords := "coords: x= ";Strings.AppendInt(x, 0, coords);
                    Strings.Append(coords, "; y= "); Strings.AppendInt(y, 0, coords)
                ELSIF
                    msg.layer IS antsPictureViews.tLayer  THEN
                    x := ENTIER(msg.xr);            y := ENTIER(msg.yr);
```

```
                    o := msg.layer(antsPictureViews.tLayer).E.A;
                    IF (0 <= x) & (0 <= y) & (x < o.w) & (y < o.h) THEN
                        antsArrayObjects.GetRowCol(x, y, o, col, row)
                    END;
                    vyBase.Update(row);         vyBase.Update(col);
                    IF r # NIL THEN r(vyLongrealVec.tLONGREAL).x := y; vyBase.Update(r) END;
                    IF c # NIL THEN c(vyLongrealVec.tLONGREAL).x := x; vyBase.Update(c) END;coords := "coords:x=";
                ELSE
                    coords := " no layer "
                END
            ELSE
                coords := " no image view "
            END;
            antsToolbox.EndDisplayHelp();
            antsToolbox.StartDisplayHelp(coords)
        END;
        Input.Mouse(keys, xm, ym)
    END;
    antsToolbox.EndDisplayHelp();
    IF y # MAX(LONGINT) THEN
        aM.SISI("Row = ", y, "Col = ", x)
    ELSE
        aM.S("Hmm? Nothing selected.")
    END
END TakeVectors;

(* Interaktive Routine um Array per Mausklick auszuw\"{a}hlen*)
PROCEDURE GetArrayObject* (r: antsRegions.tRegion): antsArrayObjects.tObject;
VAR msg: antsToolbox.GetCoordMsg;           layer: vyPort.tLayer;
BEGIN
    msg.layer := NIL;           msg.update := TRUE;
    IF r = NIL THEN
        antsToolbox.InteractiveMsg(msg);
        antsToolbox.MsgToTypedXY(msg.x, msg.y, msg, "antsViews.tLayerDesc");
        IF msg.layer # NIL THEN
            layer := msg.layer;
            WITH layer: antsPictureViews.tLayer DO RETURN  layer.E.A ;
                |layer: antsImageViews.tLayer DO  RETURN  layer.E.A
            ELSE RETURN NIL
            END
        ELSE RETURN NIL
        END
    ELSE
        antsRegions.DoSelectRegion(r);
        IF r.view # NIL THEN
            layer := r.view;
            WITH layer: antsPictureViews.tLayer DO RETURN  layer.E.A ;
            |layer: antsImageViews.tLayer DO  RETURN  layer.E.A
            ELSE RETURN NIL
            END
        ELSE
            RETURN NIL
        END
    END
END GetArrayObject;

(* Interaktive Routine zum Anwenden des Minimierers auf Array oder Teil davon per Mausklick *)
PROCEDURE Testget*;
VAR         A: antsArrayObjects.tObject;            r: antsRegions.tRegion;         gamma: LONGREAL;
    w, h, minweit, maxweit,             this, last: LONGINT;
    Aorig, Aorigtr, APott1, Apotttr, APott2, APott3, APott4, Apotttr2: antsArrayObjects.tLONGREALArray;
    lengths: BOOLEAN;
BEGIN
    antsCommands.InitScan();
    IF  antsCommands.ExpectReal( "gamma", gamma) &
        antsCommands.ExpectNumber( "minWeite",  minweit) &
        antsCommands.ExpectNumber( "maxWeite",  maxweit)

        THEN
    IF antsCommands.Find( "region") THEN NEW(r) ELSE r := NIL END;
    lengths := antsCommands.Find("lengths");
        A := GetArrayObject(r);
        IF A # NIL THEN
            WITH A: antsArrayObjects.tINTEGERObject DO
                IF r = NIL THEN
                    antsArrayObjects.CopyUnsignedIR(A.A, Aorig, 0, 0, A.w, A.h, 0, 0)
                ELSE
```

```
            IF (r.w < 2)OR (r.h < 2) THEN RETURN END;
            antsArrayObjects.CopyUnsignedIR(A.A, Aorig, r.x, r.y, r.w, r.h, 0, 0)
        END;
        w := LEN(Aorig[0]);         h := LEN(Aorig);

        (* Skalieren gamma mit Median*Median*)
        ScaleMatrix(Aorig^, gamma);
        globalGamma := gamma;

        (* Potts2 auf Matrix Orig mit ssq und median aus Matrix mit kumulativen Summen*)
        (* Originalbild*)
        antsPlots.ShowArray(antsArrayObjects.IntObjR(Aorig,A.depth),"Original Data",FALSE,FALSE);

        antsToolbox.NameOpenedWId("Original Data ", A);

        NEW(Aorigtr, w, h);
        NEW(APott1, h, w);
        NEW(Apotttr, h, w);
        NEW(APott2, w, h);

        last := Oberon.Time();

        MatrixPottsLongreal(Aorig^, APott1^, minweit, maxweit,lengths);
        (* zeigt Potts2 horizontal*)
        antsPlots.ShowArray(antsArrayObjects.IntObjR(APott1,A.depth),"Apott",TRUE,TRUE);

        (*...auf  transponierte Matrix Aorig*)
        Aorigtr := katrinMatrix.Transpose(Aorig^);
        MatrixPottsLongreal(Aorigtr^, APott2^, minweit, maxweit,lengths);
        Apotttr := katrinMatrix.Transpose(APott2^);

        this := Oberon.Time();
        Out.Ln;    Out.Ln;
        Out.String("Zeit fuer MatrixPottsLongreal (mit Cum):");
        Out.LongReal((this - last) / Input.TimeUnit, 10);           Out.Ln();

        (* zeigt Potts2 vertikal*)
        antsPlots.ShowArray(antsArrayObjects.IntObjR(Apotttr,A.depth),"Apott ver",TRUE,TRUE);

        antsToolbox.NameOpenedWId("rows Potts", A);

        (*Summe von Potts2 horizontal und Potts2 vertikal*)
        antsPlots.ShowArray(antsArrayObjects.IntObjR((APott1 + Apotttr) / (2),A.depth),"Apott sum",TRUE,TRUE);
        antsToolbox.NameOpenedWId("Sum of PottsH, PottsV of", A);

        (* ab hier nur f\"{u}r Zeitvergleich *)
        (*
        (* PottstatLongreal, Potts auf Matrix Aorig1 mit statistic (ssq, median)*)
        (*Originalbild*)
        antsPlots.ShowArray(antsArrayObjects.IntObjR(Aorig,A.depth),"Original Data",FALSE,FALSE);
        (*NEW(Aorigtr2, w,h);*)
        NEW(APott3, h, w);
        NEW(Apotttr2, h, w);
        NEW(APott4, w, h);

        last := Oberon.Time();
        PottsStatLongreal(Aorig^, APott3^);
        (*Potts horizontal*)
        antsPlots.ShowArray(antsArrayObjects.IntObjR(APott3,A.depth),"Potts hor",TRUE,TRUE);
        (*fuer transponiert Aorig*)
        PottsStatLongreal(Aorigtr^, APott4^);
        Apotttr2 := katrinMatrix.Transpose(APott4^);

        this := Oberon.Time();
        Out.Ln;    Out.Ln;
        Out.String("Zeit fuer PottsStatLongreal (mit Stat):");
        Out.LongReal((this - last) / Input.TimeUnit, 10);           Out.Ln;       Out.Ln;

        antsPlots.ShowArray(antsArrayObjects.IntObjR(Apotttr2,A.depth),"APottr2",TRUE,TRUE);
        (*Potts vertikal*)
        antsToolbox.NameOpenedWId("rows Potts2", A);
        antsPlots.ShowArray(antsArrayObjects.IntObjR((APott3 + Apotttr2) / (2),A.depth),"sum",TRUE,TRUE);
        (*Summe von Potts horizontal und Potts vertikal*)
        *)
    END
  END
END
```

```
END Testget;

(* Routine um Minimierer auf ganze Verzeichnisse anzuwenden *)
PROCEDURE PottsFile (in, out: ARRAY OF CHAR;            minWeite, maxWeite: LONGINT);
VAR A: antsArrayObjects.tLONGREALArray; o: antsArrayObjects.tINTEGERObject; AL: antsArrayObjects.tLONGINTArray;  w, h: LONGINT;
    Aorigtr, APott1, Apotttr, APott2: antsArrayObjects.tLONGREALArray; gamma: LONGREAL;
BEGIN
    antsArrayObjects.ReadPGM(o, in);
    IF o = NIL THEN HALT(100) END;
    antsArrayObjects.CopyIR(o.A, A, 0, 0, o.w, o.h, 0, 0);

    (* Potts2 berechnen (wie Testget)... *)
    w := LEN(A[0]);         h := LEN(A);

    NEW(Aorigtr, w, h);
    NEW(APott1, h, w);
    NEW(Apotttr, h, w);
    NEW(APott2, w, h);

    Aorigtr := katrinMatrix.Transpose(A^);
    gamma := globalGamma;
    ScaleMatrix(A^, globalGamma);

    MatrixPottsLongreal(A^, APott1^, minWeite, maxWeite,FALSE);
    MatrixPottsLongreal(Aorigtr^, APott2^,  minWeite, maxWeite,FALSE);

    globalGamma := gamma;      (* r\"{u}cksetzen, da n\"{a}chstes File denselben Wert benutzt *)

    Apotttr := katrinMatrix.Transpose(APott2^);

    antsArrayObjects.CopyRL((APott1 + Apotttr) / (2), AL, 0, 0, o.w, o.h, 0, 0);
    antsReadWrite.StorePGM(AL, out, o.depth);   (*speichert die Summe von Potts2 horizontal und Potts2 vertikal in out*)
END PottsFile;


PROCEDURE FileEnum (path, name: ARRAY OF CHAR; time, date, size: LONGINT; attrs: SET);
VAR outname, outlongname: ARRAY 256 OF CHAR;
BEGIN
    (* zusammensetzen: path/name*)
    COPY(name, outname);
    COPY(path, name);       Strings.Append(name, "/");
    Strings.Append(name, outname);
    outname[Strings.Length(outname) - 4] := 0X;
    (* zusammensetzen: outDirectory/name.Potts.PGM*)
    Strings.Append(outname, ".Potts.PGM");
    COPY(outDirectory, outlongname);
    Strings.Append(outlongname, "/");
    Strings.Append(outlongname, outname);
    Out.Ln;
    Out.String("Applying algorithm to ");       Out.String(name);       (* = path/name*)
    Out.String(" => ");       Out.String(outlongname);      (* = outDirectory/name.Potts.PGM*)
    PottsFile(name, outlongname, globalMinWeite, globalMaxWeite);
    Out.Ln;       Out.String("done.")
END FileEnum;

(* Kommando um Minimierer auf ganze Verzeichnisse von Bildern des Typs PGM anzuwenden *)
PROCEDURE OnDirectory*;
VAR       done: BOOLEAN;       path: ARRAY 256 OF CHAR;
BEGIN
    antsCommands.InitScan();
    IF antsCommands.ExpectString("inDirectory", inDirectory) &
        antsCommands.ExpectString("outDirectory",  outDirectory) &
        antsCommands.ExpectReal( "gamma", globalGamma) &
        antsCommands.ExpectNumber( "minWeite", globalMinWeite) &
        antsCommands.ExpectNumber("maxWeite",  globalMaxWeite) THEN
        FileDir.GetWorkingDirectory(path);
        FileDir.ChangeDirectory(inDirectory, done);
        IF ~done THEN antsTrapper.cause := "in Directory not found"; HALT(100) END;
        FileDir.ChangeDirectory(path, done);
        FileDir.ChangeDirectory(outDirectory, done);
        IF ~done THEN antsTrapper.cause := "out Directory not found"; HALT(100) END;
        FileDir.ChangeDirectory(path, done);
        FileDir.EnumerateFiles(inDirectory, "*.pgm", FALSE, FileEnum)
    END
END OnDirectory;
```

```
(* Interaktives Kommando, benutzt TakeVectors, s.o. *)
PROCEDURE Take*;
VAR         row, col: vyBase.Object;        r, c: vyBase.Object;
BEGIN
    antsCommands.InitScan();
    IF antsCommands.ExpectObject("row", row) &
        antsCommands.ExpectObject("col",  col) THEN
        IF antsCommands.GetObject("r",  r) &
            antsCommands.GetObject("c", c) THEN
        ELSE
            r := NIL;           c := NIL
        END;
        TakeVectors(row(vyBase.tVector), col(vyBase.tVector), r, c);
        vyBase.Update(row);         vyBase.Update(col)
    END
END Take;


(* F\"{u}r 1d-Minimierung: Initialisiere eine Statistik f\"{u}r Mittelwert und empirische Varianz auf Intervallen des Signals*)
PROCEDURE DoInitStatistic* (VAR statistic: Statistic;           N: LONGINT);
VAR i: LONGINT;
BEGIN
    IF statistic = NIL THEN NEW(statistic) END;
    IF (statistic.Mean = NIL) OR (statistic.N # N) THEN
        NEW(statistic.Mean, N, N);          NEW(statistic.Ssq, N, N);           statistic.N := N
    END;
    statistic.n := 0;
    FOR i := 0 TO N - 1 DO
        vyLongreal.Fill(N - i, statistic.Mean[i], i, 0);(* Init of upper triangle only *)
        vyLongreal.Fill(N - i, statistic.Ssq[i], i, 0)  (* Init of upper triangle only *)
    END
END DoInitStatistic;

(* F\"{u}r 1d-Minimierung:  Statistik f\"{u}r Mittelwert und empirische Varianz auf Intervallen des Signals*)
PROCEDURE UpdateStatistic* (VAR A: ARRAY OF LONGREAL; VAR stat: Statistic);
VAR i, j, n, nm1, len, N: LONGINT;          factor, provmean, provssq, diff, x: LONGREAL; ssq, mean: PMatr;
BEGIN
    IF stat = NIL THEN DoInitStatistic(stat, LEN(A)) END;
    INC(stat.n);
    ssq := stat.Ssq; mean := stat.Mean; n := stat.n; nm1 := n - 1; factor := nm1 / n; N := stat.N;
    FOR j := 0 TO N - 1 DO
        i := j;         len := 1;           provssq := 0;           provmean := 0;
        WHILE(i >=  0) DO
            x := A[i];
            diff := (x - provmean);
            provmean := provmean + diff / len;
            provssq := provssq + diff * (x - provmean);
            (* hier haben wir also: Mittelwert und Varianz an der Stelle i *)
            mean[i, j] := mean[i, j] * factor + provmean / n;
            ssq[i, j] := ssq[i, j] + provssq;
            DEC(i);         INC(len)
        END
    END
END UpdateStatistic;


(* berechnet aus der Matrix Aorig die Matrix der kumulierten Summen (Akum) und die Matrix der Summe der Quadrate (AkumSq)*)
PROCEDURE CumSum (VAR Aorig, Akum, AkumSq: ARRAY OF ARRAY OF LONGREAL);
VAR i, j, h, w: LONGINT;
BEGIN
    (* Matrix AKum, AkumSq sind um 1 nach oben und rechts verschoben, in Zeile 0 und Spalte 0 steht nur 0 *)
    h := LEN(Aorig);                w := LEN(Aorig[0]);
    (* Quadrieren jedes Elementes von Aorig in AkumSq *)
    FOR i := 1 TO h DO
        FOR j := 1 TO w DO
            AkumSq[i, j] := Aorig[i - 1, j - 1] * Aorig[i - 1, j - 1]
        END
    END;
    vyLongreal.Copy(w, Aorig[0], 0, Akum[1], 1); (* kopieren der 1.Zeile (=0) von Aorig in die erste Zeile (=1) von Akum*)
    (* Aufaddieren von Zeile i-1 auf Zeile i von Zeile 2 bis h = Anzahl der Zeilen*)
    FOR i := 2 TO h DO
        vyLongreal.Copy(w, Aorig[i - 1], 0, Akum[i], 1 );
        vyLongreal.Add(Akum[i - 1], Akum[i], Akum[i], w + 1);
        vyLongreal.Add(AkumSq[i - 1], AkumSq[i], AkumSq[i], w + 1)
    END;
    (*Aufaddieren der Spalten*)
    FOR i := 2 TO w DO
```

```
        FOR j :=  0 TO h DO
            Akum[j, i] := Akum[j, i - 1] + Akum[j, i];
            AkumSq[j, i] := AkumSq[j, i - 1] + AkumSq[j, i]
        END
    END
END CumSum;

    (* berechnet den Mittelwert aus der Matrix der kumulativen Summen (Akum)*)
PROCEDURE MeanCum (VAR Akum: ARRAY OF ARRAY OF LONGREAL;                fromx, fromy, tox, toy: LONGINT): LONGREAL;
VAR factor, mean: LONGREAL;
BEGIN
    INC(tox);           INC(toy);           factor := (tox - fromx) * (toy - fromy);
    mean := (Akum[toy, tox] -  Akum[fromy, tox] -  Akum[toy, fromx] + Akum[fromy, fromx]) / factor;
    RETURN mean
END MeanCum;


    (* berechnet SSQ aus der Matrix der kumulierten Summen (Akum) und aus der Matrix der Summe der Quadrate(AkumSq)*)
PROCEDURE SsqCum (VAR Akum, AkumSq: ARRAY OF ARRAY OF LONGREAL; fromx, fromy, tox, toy: LONGINT): LONGREAL;
VAR Sum, SumSq, L, factor: LONGREAL;
BEGIN
    INC(tox);           INC(toy);           factor := ( tox - fromx) * (toy - fromy);
    SumSq := AkumSq[toy, tox] -  AkumSq[fromy, tox] -  AkumSq[toy, fromx] + AkumSq[fromy, fromx];
    Sum :=  Akum[toy, tox] -  Akum[fromy, tox] -  Akum[toy, fromx] + Akum[fromy, fromx];
    L := Sum * Sum;
    RETURN SumSq - L / factor
END SsqCum;


    (* gamma + ssq (aus statistic)*)
PROCEDURE Function1d (from, to: LONGINT): LONGREAL;
BEGIN
    RETURN globalStat.Ssq[from, to] + globalGamma
END Function1d;


    (* gamma + ssq (aus Matrix der kumulativen Summen)*)
PROCEDURE CumFunction (from, to: LONGINT): LONGREAL;
BEGIN
    RETURN globalGamma + SsqCum(globalAkum^, globalAkumSq^, from, globaly1, to, globaly2)
END CumFunction;


PROCEDURE MatrixFunction (from, to: LONGINT): LONGREAL;
BEGIN
    globaly1 := from;           globaly2 := to;
    antsSecurity.SetProgress(SHORT(ENTIER(to / LEN(globalAkum) * 100)));
    katrinBellman.SetBellmanFunction(globalRstern, globalBellman, SHORT(LEN(globalAkum[0])) - 1, 0, MAX(LONGINT), CumFunction, - globalGamma);
    RETURN globalBellman[LEN(globalAkum[0]) - 1]
END MatrixFunction;

PROCEDURE CumFunction2 (from, to: LONGINT): LONGREAL;
BEGIN
    RETURN globalGamma * (globaly2-globaly1+1)+ SsqCum(globalAkum^, globalAkumSq^, from, globaly1, to, globaly2)
END CumFunction2;


PROCEDURE MatrixFunction2 (from, to: LONGINT): LONGREAL;
BEGIN
    globaly1 := from;           globaly2 := to;
    antsSecurity.SetProgress(SHORT(ENTIER(to / LEN(globalAkum) * 100)));
    katrinBellman.SetBellmanFunction(globalRstern, globalBellman, SHORT(LEN(globalAkum[0])) - 1, 0, MAX(LONGINT), CumFunction2, - globalGamma);
    RETURN globalBellman[LEN(globalAkum[0]) - 1]+ globalGamma * LEN(globalAkum[0]);
END MatrixFunction2;


    (*Berechnen der besten Partition fuer einen Vektor mit Mittelwert aus Statistic*)
PROCEDURE WritePartition* (VAR parti: ARRAY OF LONGREAL; rstern: IVect; s: Statistic; N: LONGINT);
VAR k, next: LONGINT;
BEGIN
    k := N;
    WHILE(k  >  0) DO
        next := rstern[k];
        vyLongreal.Fill(k - next, parti, next, s.Mean[next, k - 1]);
        k := next
    END
END WritePartition;
```

```
(* Berechnen der besten Partition fuer Matrix mit Mittelwert aus den kumulativen Summen*)
PROCEDURE CumWritePartition* (VAR  Akum, Parti: ARRAY OF ARRAY OF LONGREAL; rstern: IVect; y1, y2: LONGINT);
VAR k, next, y: LONGINT; mean: LONGREAL;
BEGIN
    k := LEN(Akum[0]) - 1;
    WHILE(k > 0) DO
        next := rstern[k];
        mean := MeanCum(Akum, next, y1, k - 1, y2);            (*Mittelwert aus der Matrix der kumulativen Summen*)
        FOR y := y1 TO y2 DO
            vyLongreal.Fill(k - next, Parti[y], next, mean)
        END;
        k := next
    END
END CumWritePartition;


PROCEDURE MatrixWritePartition* (VAR  Akum, Parti: ARRAY OF ARRAY OF LONGREAL;          rstern: IVect);
VAR k, next: LONGINT;                   bellmanneu: PVect;                  rsternneu : IVect;
BEGIN
    k := LEN(Akum) - 1;
    WHILE(k > 0) DO
        next := rstern[k];
        globaly1 := next;               globaly2 := k - 1;
        (* fuer jedes Intervall der besten zeilenweise Partition nochmal bellmann...*)
        katrinBellman.SetBellmanFunction(rsternneu, bellmanneu, LEN(Akum[0]) - 1, 0, MAX(LONGINT), CumFunction, - globalGamma);
        (* und die beste Partition*)
        CumWritePartition(Akum, Parti, rsternneu, globaly1, globaly2);
        k := next
    END
END MatrixWritePartition;


    (* Skalieren von gamma  mit Min und Max von einem Vektor orig*)
PROCEDURE Scale* (VAR orig: ARRAY OF LONGREAL;                  VAR gamma: LONGREAL);
VAR max, min: LONGREAL;                 na, minpos, maxpos: LONGINT;
BEGIN
    vyLongreal.MinMax(LEN(orig), orig, 0, min, minpos, max, maxpos, na);
    gamma := gamma * (max - min) * (max - min)
END Scale;


    (* Skalieren von gamma mit Median von Matrix S*)
PROCEDURE ScaleMatrix* (VAR S: ARRAY OF ARRAY OF LONGREAL;              VAR gamma: LONGREAL);
VAR   median: LONGREAL;
BEGIN
    median := katrinMatrix.Median(S);                   (* Median von S*)
    gamma := gamma * median * median
END ScaleMatrix;


    (* Potts auf Matrix Orig mit statistic (ssq, median)*)
PROCEDURE PottsStatLongreal* (VAR Orig, Pott: ARRAY OF ARRAY OF LONGREAL);
VAR i, h, w: LONGINT;           this, last: LONGINT;
BEGIN
    h := LEN(Orig);         w := LEN(Orig[0]);
    FOR i := 0 TO h - 1 DO
        vyLongreal.Copy(w, Orig[i], 0, Pott[i], 0)
    END;
    last := Oberon.Time();

    FOR i :=  0 TO h - 1 DO
        DoInitStatistic(globalStat, w);
        UpdateStatistic(Pott[i], globalStat);

        katrinBellman.SetBellmanFunction(globalRstern, globalBellman, SHORT(w), 0, MAX(LONGINT), Function1d, - globalGamma);
        WritePartition(Pott[i], globalRstern, globalStat, w)

    END;

    this := Oberon.Time();
    Out.Ln();    Out.Ln();
    Out.String("Zeit fuer Bellmann und und WritePartition:");
    Out.LongReal((this - last) / Input.TimeUnit, 10);           Out.Ln()

END PottsStatLongreal;
```

```
    (* Potts2 auf Matrix Orig mit ssq und median aus Matrix mit kumulativen Summen*)
PROCEDURE MatrixPottsLongreal* (VAR Orig, Pott: ARRAY OF ARRAY OF LONGREAL; minweit, maxweit: LONGINT; lengths: BOOLEAN);
VAR i, h, w: LONGINT;                       Akum, AkumSq: antsArrayObjects.tLONGREALArray; this, last: LONGINT;
    Rstern: IVect;        Bellman: PVect;
BEGIN
    h := LEN(Orig);                        w := LEN(Orig[0]);
    FOR i := 0 TO h - 1 DO
        vyLongreal.Copy(w, Orig[i], 0, Pott[i], 0)
    END;

    (* Berechnen der Matrix mit kumulativen Summen (Akum) bzw. mit kumulativen Summe der Quadrate (AkumSq)*)
    NEW(Akum, h + 1, w + 1);
    NEW(AkumSq, h + 1, w + 1);

    CumSum(Pott, Akum^, AkumSq^);

    (* globale Variablen*)
    globalAkum := Akum;                globalAkumSq := AkumSq;

    NEW(globalRstern, w + 1);           NEW(globalBellman, w + 1);

    last := Oberon.Time();

    IF lengths THEN
    katrinBellman.SetBellmanFunction(Rstern, Bellman, SHORT(h), minweit, maxweit, MatrixFunction2, - globalGamma);
    ELSE
    katrinBellman.SetBellmanFunction(Rstern, Bellman, SHORT(h), minweit, maxweit, MatrixFunction, - globalGamma);
    END;
    MatrixWritePartition(Akum^, Pott, Rstern);

    this := Oberon.Time();
    Out.Ln();    Out.Ln();
    Out.String("Zeit fuer Setbellmann und MatrixWritePartition:");
    Out.LongReal((this - last) / Input.TimeUnit, 10);           Out.Ln()

END MatrixPottsLongreal;


    (* berechnet einen Vektor maxima mit Eintrag 1 fuer Max bei Vektor orig (sonst 0)*)
PROCEDURE FindMaxima (orig: ARRAY OF LONGREAL;                  VAR maxima: ARRAY OF LONGREAL );
VAR i, N, lastuppos: LONGINT;                   lastv, thisv: LONGREAL;                up: BOOLEAN;
BEGIN
    N := LEN(orig);
    vyLongreal.Fill(N, maxima, 0, 0);
    lastv := MIN(LONGREAL);                 up := TRUE;                lastuppos := 0;
    FOR i := 0 TO N - 1 DO
        thisv := orig[i];
        IF thisv = lastv THEN
        ELSIF thisv > lastv THEN
            up := TRUE;                lastuppos := i
        ELSE
            IF up THEN
                vyLongreal.Fill(i - lastuppos, maxima, lastuppos, 1);
                up := FALSE
            END
        END;
        lastv := thisv
    END;
    IF up THEN  vyLongreal.Fill(N - lastuppos, maxima, lastuppos, 1)          END
END FindMaxima;


    (* berechnet eine Matrix Maxima mit Eintrag 1 an den Stellen Max bei Matrix Orig*)
PROCEDURE FindMaximaMatrix (VAR Orig, Maxima: ARRAY OF ARRAY OF LONGREAL);
VAR  i, h: LONGINT;
BEGIN
    h := LEN(Orig);
    FOR i := 0 TO h - 1 DO
        FindMaxima(Orig[i], Maxima[i])
    END
END FindMaximaMatrix;


    (*berechnet eine Matrix Result, mit Eintrag 1 an den Stellen, an denen Max1 und Max2 ebenfalls Eintrag 1 haben*)
PROCEDURE AndMax* (VAR Max1, Max2, Result: ARRAY OF ARRAY OF LONGREAL);
```

```
VAR  i, j, h, w: LONGINT;
BEGIN
    h := LEN(Max1);             w := LEN(Max1[0]);
    FOR i := 0 TO h - 1 DO
        vyLongreal.Fill(w, Result[i], 0, 0);
        FOR j := 0 TO w - 1 DO
            IF (Max1[i, j] =  1) & (Max2[i, j] = 1) THEN  Result[i, j] := 1         END
        END
    END
END AndMax;

    (* Test fuer Potts auf vektoren*)
PROCEDURE PottsToVec*;
VAR  obj1, obj2, obj3: vyBase.Object; vec1, vec2, vec3: vyLongrealVec.tVector;
    i, N, this, last: LONGINT; rstern: IVect; bellman, maxima, maxima2, data: PVect; gamma: LONGREAL;
BEGIN
    antsCommands.InitScan();
    IF antsCommands.ExpectObject( "meinVektor",  obj1) &
        antsCommands.ExpectObject( "zweiterVektor",  obj2) &
        antsCommands.ExpectObject( "dritterVektor",  obj3) &
        antsCommands.ExpectReal( "gamma",  gamma)
        THEN

        last := Oberon.Time();

        vec1 := obj1 (vyLongrealVec.tVector);
        vec2 := obj2(vyLongrealVec.tVector);
        vec3 := obj3(vyLongrealVec.tVector);

        IF antsCommands.Find( "init") THEN
            (* vektoren mit Eintrag 1*)
            FOR i :=  0 TO  vyLongrealVec.Len(vec1) - 1 DO
                vyLongrealVec.Set(1, vec1, i)
            END;
            FOR i :=  0 TO  vyLongrealVec.Len(vec2) - 1 DO
                vyLongrealVec.Set(1, vec2, i)
            END
        END;

        N := vyLongrealVec.Len(vec1);
        NEW(data, N);
        vyLongrealVec.CopyToArray(vec1, data^);

        DoInitStatistic(globalStat, N);
        UpdateStatistic(data^, globalStat);

        Scale(data^, gamma);

        globalGamma := gamma;
        katrinBellman.SetBellmanFunction(rstern, bellman, SHORT(N), 0, MAX(LONGINT), Function1d, - globalGamma);

        this := Oberon.Time();
        IF antsCommands.Find( "time") THEN
            Out.Ln; Out.String("Katrin, time ellapsed :");
            Out.LongReal((this - last) / Input.TimeUnit, 10); Out.String(" sec")
        END;

        WritePartition(data^, rstern, globalStat, N);
        NEW(maxima, N);
        NEW(maxima2, N);

        FindMaxima(data^, maxima^);
        FindMaxima(data^, maxima2^);

        vyLongrealVec.SetLen(vec2, N);
        vyLongrealVec.CopyToVec(data^, vec2, N, 0, 0);

        vyLongrealVec.SetLen(vec3, N);
        vyLongrealVec.CopyToVec(maxima^, vec3, N, 0, 0);

        vyBase.Update(vec1);
        vyBase.Update(vec2);
        vyBase.Update(vec3)
    END
END  PottsToVec;

BEGIN
```

```
END katrinPotts.
```

8.2 Modul katrinBellman

```
MODULE katrinBellman;   (** kjw  **)


IMPORT angieAIC,Out;
TYPE

    PVect = angieAIC.PVect;
    IVect = angieAIC.IVect;

TYPE Function = PROCEDURE(from,to: LONGINT): LONGREAL;

PROCEDURE Max(i,j: LONGINT): LONGINT;
BEGIN
IF i > j THEN RETURN i ELSE RETURN j END;
END Max;

PROCEDURE Min(i,j: LONGINT): LONGINT;
BEGIN
IF i < j THEN RETURN i ELSE RETURN j END;
END Min;


PROCEDURE Bellman* (VAR rstern: ARRAY OF INTEGER;  VAR bellman: ARRAY OF LONGREAL;   n, minweite, maxweite: LONGINT;  p: Function);
VAR i, minpos: LONGINT;  min,current: LONGREAL;      from,to: LONGINT;
BEGIN
    min := MAX(LONGREAL);  from := Max(0,n-maxweite);  to := Min(n-1,n-minweite);
    (*from := 0;  to := n-1;  *)

    FOR i := from TO to DO  (*Bestimmen des Min fuer die jeweilige Partiton*)
        (* IF ((n-i) >= minweite) & ((n-i) <= maxweite) THEN  *)
            current :=  bellman[i]+ p(i, n-1);
            IF current < min THEN  min := current;   minpos := i END;
        (* END;   *)
    END;
    bellman[n] := min;
    rstern[n] := SHORT(minpos);
END Bellman;

PROCEDURE SetBellmanFunction*(VAR rstern: IVect; VAR bellman: PVect; N, minweite, maxweite: LONGINT; p: Function; initvalue: LONGREAL);
VAR i: LONGINT;
BEGIN
    IF (bellman = NIL) OR (LEN(bellman) # N+1) THEN
        NEW(bellman,N+1);                           NEW(rstern,N+1);
    END;
    bellman[0] := initvalue;  bellman[1] := 0;
    FOR i := 2 TO (N-1) DO (* vorsicht: Index-Verschiebung durch den Startwert der Bellman-Fkt *)
        Bellman(rstern^, bellman^, i, minweite, maxweite,p);
    END;
    Bellman(rstern^, bellman^, N, 0, maxweite,p);

    rstern[0] := MIN(INTEGER); stern[1] := 0;
END SetBellmanFunction;

END katrinBellman.
```

8.3 Modul katrinMatrix

```
MODULE katrinMatrix;    (** kjw  **)

IMPORT vyLongreal,Out,antsArrayObjects;

TYPE
(* berechnet Median aus den Eintraegen der Matrix S*)
PROCEDURE Median*(VAR S: ARRAY OF ARRAY OF LONGREAL): LONGREAL;
VAR values: POINTER TO ARRAY OF LONGREAL;   w,h,i: LONGINT;
```

```
BEGIN
    h := LEN(S);          w := LEN(S[0]);
    NEW(values,w*h);
    FOR i := 0 TO h-1 DO
        vyLongreal.Copy(w, S[i], 0, values^, i*w);
    END;
    vyLongreal.Sort(w*h, values^, 0);
    RETURN values [w*h DIV 2];
END Median;

(*bestimmt den min und max Eintrag in der Matrix S (und deren Position)*)
PROCEDURE MinMax*(VAR S: ARRAY OF ARRAY OF LONGREAL;   VAR min,max: LONGREAL);
VAR h, i, minpos, currentminpos, maxpos, currentmaxpos, na: LONGINT;      currentmax, currentmin: LONGREAL;
BEGIN
    h:= LEN(S);
    min := MAX(LONGREAL);       max := MIN(LONGREAL);
    FOR i:= 0 TO h-1 DO
        vyLongreal.MinMax(LEN(S[i]), S[i], 0, currentmin, currentminpos, currentmax, currentmaxpos, na);
        IF currentmin < min THEN  min := currentmin;             minpos := currentminpos END;
        IF currentmax > max THEN max:= currentmax;           maxpos:= currentmaxpos  END;
    END;
END MinMax;

PROCEDURE OutMatrix*(A: ARRAY OF ARRAY OF LONGREAL);
VAR i,j: LONGINT;
BEGIN
    FOR i := 0 TO LEN(A)-1 DO
        Out.Ln;
        FOR j := 0 TO LEN(A[0])-1 DO
            Out.LongReal(A[i,j], 20);
        END;
    END;
    Out.Ln;
END OutMatrix;

(* transponiert Matrix A zu Atr*)
PROCEDURE  Transpose*(VAR A: ARRAY OF ARRAY OF LONGREAL): antsArrayObjects.tLONGREALArray;
VAR h, w, i, j: LONGINT;       Atr: antsArrayObjects.tLONGREALArray;
BEGIN
    h:= LEN(A);             w:=LEN(A[0]);
    NEW(Atr, w, h);
    FOR i := 0 TO h-1 DO
        FOR j := 0 TO w-1 DO
            Atr[j, i]  := A[i, j];
        END;
    END;
    RETURN Atr;
END Transpose;

END katrinMatrix.
```