International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

Image Restoration from Multiple Sources

Moltchanova, E.

IIASA Interim Report
October 2001

International Institute for
Applied Systems Analysis
Schlossplatz 1
A-2361 Laxenburg, Austria

Tel: +43 2236 807 342
Fax: +43 2236 71313
E-mail: publications@iiasa.ac.at
Web: www.iiasa.ac.at

Interim Report **IR-01-045**

Image Restoration from Multiple Sources

Elena Moltchanova (elena.moltchanova@ktl.fi)

Approved by

Sten Nilsson
Leader, Forestry Project

5 October 2001

Contents

Abstract

This paper proposes a new method of image restoration. The proposed method allows to combine information from several sources, taking the perceived credibility of each into account. It is applicable to both ordinal (e.g., gray level image) and non-ordinal (e.g., classified forest map) categorized images. The accuracy checks have shown the method to be robust with respect to the prior information and the accuracy of the sources. Two application examples are provided.

Acknowledgments

My profound gratitude extends to those who made my participation in the IIASA's 2001 YSSP possible. I am also grateful to the participants of the Forestry Project's Siberia-II study and to the members of the Institute of Surveying, Remote Sensing and Land Information of the University of Agricultural Sciences in Vienna for their many helpful comments and interest in my work. Finally, my sincere thanks to the YSSPers who made my summer at IIASA so enjoyable, to the members of the Forestry project for their support and the creative atmosphere, and my special thanks to Michael Obersteiner for his creative ideas, unwavering support, and helpful supervision.

About the Author

Elena Moltchanova received her M.Sc. degree from the University of Helsinki, Finland, in 2000. She is currently working on her Ph.D. thesis in the National Public Health Institute of Helsinki, applying Bayesian statistics in the field of spatial epidemiology.

Image Restoration from Multiple Sources

Elena V. Moltchanova

1 Introduction

Image recognition in general and the satellite image analysis in particular has been an area of intense interest for a long time. Currently, this is a wide field with many accepted approaches, and new methods and modifications being continuously suggested. Among the existing accepted approaches are neural networks (Krell *et al*., 1996), fuzzy logic (Mascarilla, 1997), semantic networks (Kunz *et al*., 1997), Bayesian CAR models (Besag *et al*., 1991), and Adaptive Weights Smoothing (Polzehl and Spokoiny, 2000; Divino *et al*., 1999; Kitano and Takagi, 1999). Most of the existing methods were developed to analyze a single observed pattern at one time. However, there are virtually no tools available that integrate multiple sources despite the fact that during the last decade increasingly large amounts of geo-referenced data have become available. In such a situation, it is certainly more efficient and economic to take all the available sources of information into account and to gather new information conditional on what is already available. The latter is of particular importance for the design of new satellite sensors. Of the methods mentioned above, Bayesian analysis (Gelman *et al*., 1995), which may be considered as a special case of the Dempster-Shafer theory (Shafer, 1976), is well suited for combining different data sources. The CAR model is currently widely applied in epidemiology, in particular, for the production of cancer atlases (Osnes and Aalen, 1999). However, this model tends to over-smooth and is thus not applicable, for example, to vegetation maps where the categories or gray levels may interchange abruptly introducing the edge estimation problem.

In this paper a new image recognition method is proposed, which is a combination of the two existing methods, namely (1) Bayesian analysis (henceforth denoted BR), and (2) Adaptive Weights Smoothing (AWS) or naïve smoothing (NS). It thus combines the superior ability of Bayesian inference to combine multiple information sources with the good performance of AWS or NS in image restoration and edge estimation.

This paper begins by describing BR, AWS and NS approaches and providing examples of their application in Section 2. Section 3 investigates the estimation accuracy of these methods. Since the corresponding accuracy functions are too complex to study analytically and in full, only some aspects are studied with the aid of simulations. These studies are sufficient to provide a general overview of the performance accuracy and sensitivity. The suggested new approach is described in Section 4 and examples of the applications are provided in Section 5. In conclusion, the possible significance of the proposed method is discussed together with the future work to be done (Section 6).

Since the method is computationally intensive and there is no ready-made software, a list of functions written in R-language along with their listings is provided in Appendices 1 and 2, respectively.

1.1 Notation

In order to make the report more readable a general notation adhered to throughout the paper is introduced here.

I = total number of cells in the analysis,
C = number of classes, $c = 1,2,\ldots,C$,
N = number of sources,
i = cell of grid, $i = 1,\ldots,I$,
X_i = true level at cell, i, and
Y_{ni} = observation at cell i by source n.

$$E_n = \begin{bmatrix} e_{n11} & e_{n12} & \cdots & e_{n1C} \\ e_{n21} & e_{n22} & \cdots & e_{n2C} \\ \vdots & \vdots & \ddots & \vdots \\ e_{nC1} & e_{nC2} & \cdots & e_{nCC} \end{bmatrix} = \text{error matrix of source } n \text{ s.t. } P(Y_{ni} = y \mid X_i = x) = e_{nxy}$$

$$Z_n = \begin{bmatrix} \zeta_{n11} & \zeta_{n12} & \cdots & \zeta_{n1C} \\ \zeta_{n21} & \zeta_{n22} & \cdots & \zeta_{n2C} \\ \vdots & \vdots & \ddots & \vdots \\ \zeta_{nC1} & \zeta_{nC2} & \cdots & \zeta_{nCC} \end{bmatrix} = \text{perceived error matrix of source } n \text{ (Bayesian MODEL)}$$

$P(X)$ = true distribution of X,
$p(X)$ = perceived distribution of X (Bayesian prior).

2 Existing Approaches: BR and AWS; Naïve Smoothing

2.1 Bayesian Approach (BR)

2.1.1 General notes on the Bayesian approach

Bayesian statistics was named after Rev. Thomas Bayes who, in his famous essay, has proposed a solution to the problem of estimating an unknown proportion. He thus arrived at the result that became known as the Total Probability Formula. For two events A and B and a partition of A such that $\bigcup_{i=1}^{I} A_i = A$ and $A_i \cap A_j = \mathrm{O} \forall i \neq j$

the following holds:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{i=1}^{I}\left[P(B|A_i)P(A_i)\right]} .$$

The formula can be extended to deal with probability distributions. Thus, if we denote the observations *x* and the parameter θ then the following is true:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} .$$

The above formula cannot always be solved analytically. However, with recent developments in computers this is no longer a problem, since Bayesian inference is mostly done using computer-intensive iterative MCMC techniques.

Bayesian statistics is not just the new way to make estimates or predictions — it is a way of thinking that is markedly different from that of classical statistics. Whereas in classical frequentist statistics parameters are viewed as fixed but unknown quantities and the data are considered to be the result of a repeatable experiment, in the Bayesian inference each experiment is thought to be unique. Conclusions about the parameters, on the other hand, are made in terms of probability statements. Thus, the parameters are assigned *a priori* distribution, which is supposedly based on the experience of the analyst or, perhaps, on the expert's opinion. In the case that no prior information is available at all, a non-informal prior may be assigned. The analysis results in *a posteriori* distribution for the parameters, on which the inference is then based.

The Bayesian inference still remains an object of controversy. One of the more notorious points of discussion is the supposed subjectivity of the analysis, which arises as a result of the prior distribution choice. However, it is often pointed out that this assignment is either done using the available information or the non-informative prior is then assigned. On the other hand, once such information is available, why should it be discarded? The Bayesian framework removes the need for restrictive classical assumptions of independent identical distribution and normality. Overall, it is an extremely flexible approach, which allows taking into account information from several different sources simultaneously and to make probabilistic statements about the estimates produced. It is this feature of allowing different sources to be accommodated that makes the Bayesian approach attractive in the solution of the particular problem of image restoration, when multiple sources of information on the same object are available.

2.1.2 Bayesian image restoration: Total probability formula

Let us now apply the Bayes' rule to the problem of image restoration. Using the notation described in the beginning and applying the Total Probability Formula we get the following:

$$P(X_i = x | Y_{i1} = y_1, Y_{i2} = y_2, \ldots, Y_{in} = y_n) = \frac{\prod_{n=1}^{N} \zeta_{n,x,y_1} p(x)}{\sum_{x=1}^{C} \left(\prod_{n=1}^{N} \zeta_{n,x,y_1} p(x) \right)} .$$

Using the mode estimation in each cell, we choose the value of *x* for which the posterior probability is highest. To illustrate:

Let,

C = 3,
$p(x)$ = (..5,.3,.2),
N = 2,

$$\mathrm{Z}_1 = \begin{bmatrix} .6 & .2 & .2 \\ .2 & .6 & .2 \\ .2 & .2 & .6 \end{bmatrix} \text{ and } \mathrm{Z}_2 = \begin{bmatrix} .8 & .1 & .1 \\ .3 & .7 & .0 \\ .4 & .0 & .6 \end{bmatrix} .$$

Assume further that in a single cell the values observed from the two sources are: $Y_1 = 2$ and $Y_2 = 3$. Then the estimation would proceed as follows. First, we would calculate the posterior probability distribution of *X*:

$$P(X = 1 | Y_1 = 2, Y_2 = 3) = \frac{\zeta_{1,1,2}\zeta_{2,1,3} p(1)}{\sum_{x=1}^{3} \zeta_{1,x,2}\zeta_{2,x,3} p(x)} = \frac{.2*.1*.5}{.2*.1*.5 + .6*.0*.3 + *.2*.6*.2} = \frac{.010}{.034} = .2941$$

$$P(X = 2 | Y_1 = 2, Y_2 = 3) = \frac{\zeta_{1,2,2}\zeta_{2,2,3} p(2)}{\sum_{x=1}^{3} \zeta_{1,x,2}\zeta_{2,x,3} p(x)} = \frac{.6*.0*.3}{.2*.1*.5 + .6*.0*.3 + *.2*.6*.2} = \frac{.000}{.034} = .0000$$

$$P(X = 3 | Y_1 = 2, Y_2 = 3) = \frac{\zeta_{1,3,2}\zeta_{2,3,3} p(3)}{\sum_{x=1}^{3} \zeta_{1,x,2}\zeta_{2,x,3} p(x)} = \frac{.2*.6*.2}{.2*.1*.5 + .6*.0*.3 + *.2*.6*.2} = \frac{.024}{.034} = .7059$$

Using this mode estimation *X=3* is the most likely estimate, whereas *X=2* is an impossible choice.

2.1.4 Example

To give a fuller impression of the described image restoration method, here is an example of the restoration of a 15x15 field with three sources and three classes each.

The parameters are:

$C = 3$

$N = 3$

$I = 15$

$$\mathrm{Z}_1 = \begin{bmatrix} .6 & .2 & .2 \\ .2 & .6 & .2 \\ .2 & .2 & .6 \end{bmatrix} (= E_1) \qquad \mathrm{Z}_2 = \begin{bmatrix} .8 & .1 & .1 \\ .1 & .8 & .1 \\ .1 & .1 & .8 \end{bmatrix} (= E_2) \qquad \mathrm{Z}_3 = \begin{bmatrix} .9 & .1 & .0 \\ .0 & .1 & .9 \\ .0 & .1 & .9 \end{bmatrix}$$

and the analysis is illustrated in Figure 1.

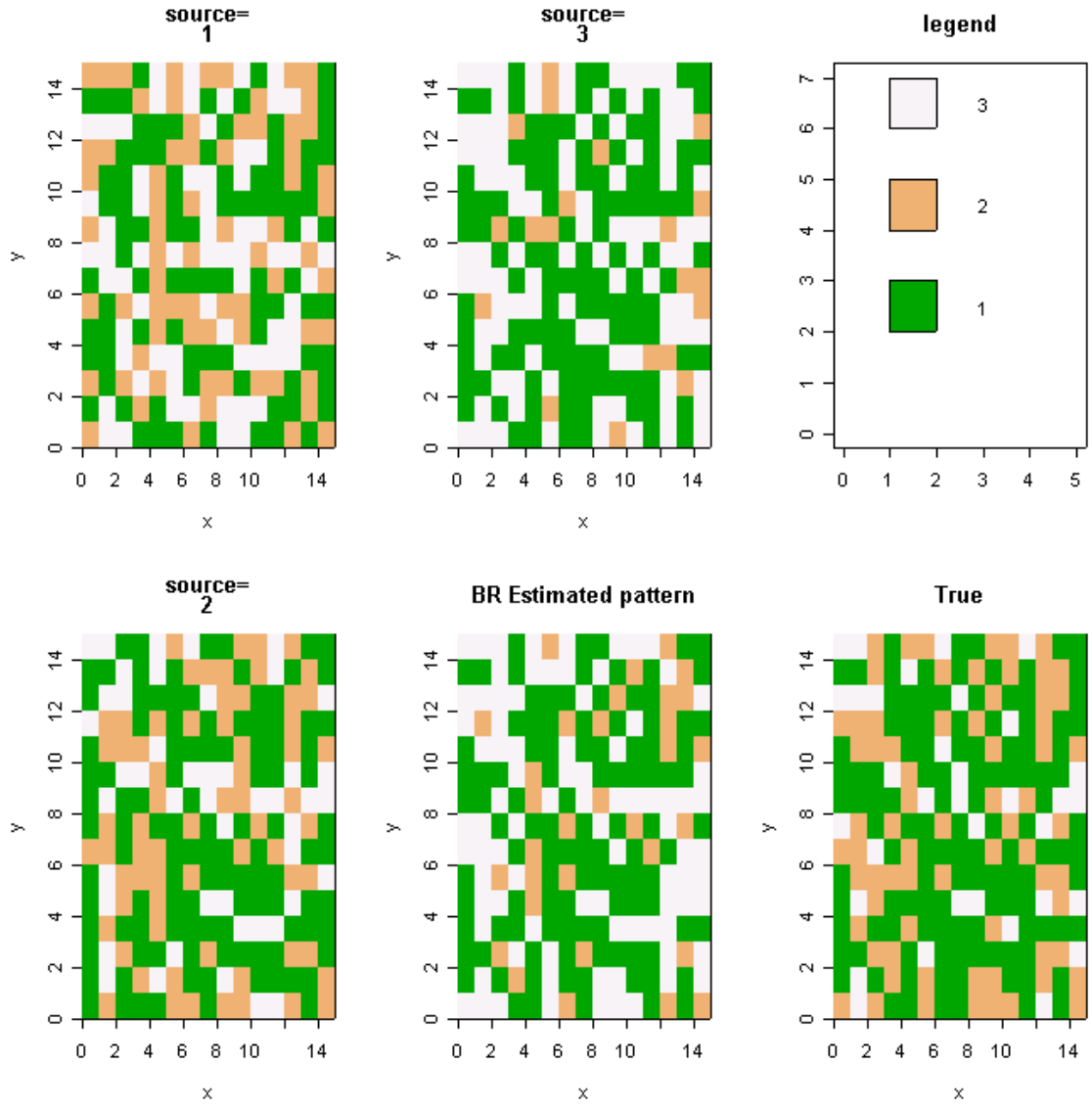


Figure 1: An example of Baysian image restoration.

2.1.5 Notes

The Bayesian total probability formula allows to take several different sources of observations as well as possible external information (prior) into account. However, as such, it makes no allowances for the existence of spatial correlation, which are certainly necessary in the image analysis. A Bayesian approach for image analysis has been developed by Besag *et al*. (1991). Since then, various developments have occurred. Currently, the CAR model is widely used in epidemiology, in particular in the creation of cancer atlases. However, the CAR model tends to over-smooth and is therefore not applicable to, for example, land-cover maps where a residential area can be abruptly changed by a river. In the next serction, I will describe a deterministic image restoration method, which deals with this so-called 'edge estimation' problem.

2.2 Adaptive Weights Smoothing (AWS)

2.2.1 General description

The AWS approach is described in detail by Polzehl and Spokoiny (2000). It is a method of non-parametric estimation, which is based on locally constant smoothing with an adaptive choice of weights for every pair of data points. Here, I will briefly describe the algorithm.

The model is described as:

$$Y_i = f(X_i) + \varepsilon_i, \quad X_i \in \Re^2, \; E(\varepsilon_i) = 0, \quad \mathrm{var}(\varepsilon_i) = \sigma^2) \; ,$$

where X are design points (e.g., cells of the grid) and the errors are assumed to be independent and identically distributed zero-mean random variables with unknown distribution.

The regression function f is the piecewise constant. This means that the spatial region (grid) can be split into disjoint regions $A_1, A_2, \ldots, A_M$, and

$$f(x) = \sum_{m=1}^{M} a_m 1\{x \in A_M\} \; ,$$

where $a_1, \ldots, a_M$ are some numbers; in our case they are the classes so that each $a_m \in \{1, \ldots, C\}$. The technical details of the procedure can then be described in the form of the iterative algorithm below.

We start by estimating the variance of residuals:

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} \hat{e}_i^2 \quad .$$

Another important element is the specification of an increasing sequence of neighborhoods around each design point. For each design point x, it is assumed that we

are given a sequence of neighborhoods $U_k(x), k = 0,1,\ldots,\infty$ with $U_k(x) \subset U_{k+1}(x)$ containing *x*. An example of such neighborhoods may be circles of increasing radii around each point.

We should also choose a univariate kernel K, which is a symmetric smooth function with the maximum at zero, non-increasing on the positive semi-axis, and integrable.

(a) *Initialization:* for each point X_i, the initial estimates of $f(X_i)$ and $\mathrm{var}\{\hat{f}(X_i)\}$ as:

$$\hat{f}_0(X_i) \frac{1}{N_0(X_i)} \sum_{X_j \in U_0(X_i)} Y_j$$

$$\hat{s}_0^2(X_i) = \frac{\hat{\sigma}^2}{N_0(X_i)}$$

set $k = 1$

(b) *Adaptation*: compute weights $w_k(X_i, X_j)$ as:

$$w_k(X_i, X_j) = K\left\{ \frac{\hat{f}_{k-1}(X_i) - \hat{f}_{k-1}(X_j)}{\lambda \hat{s}_{k-1}(X_i)} \right\}$$

for all points X_j in $U_k(X_i)$ and compute new estimates of $f_k(X_i)$ and $\mathrm{var}\left[\hat{f}_k(X_i)\right]$ as:

$$\hat{f}_k(X_i) = \frac{\sum_{X_j \in U_k(X_i)} w_k(X_i, X_j) Y_j}{\sum_{X_j \in U_k(X_i)} w_k(X_i, X_j)}$$

$$\hat{s}_k^2(X_i) = \frac{\hat{\sigma}^2 \sum_{X_j \in U_k(X_i)} w_k^2(X_i, X_j)}{\left\{ \sum_{X_j \in U_k(X_i)} w_k(X_i, X_j) \right\}^2}$$

for all X_i.

(c) *Control:* after the estimate $\hat{f}_k(X_i)$ has been computed we compare it with the previous estimates $\hat{f}_{k'}(X_i)$ at the same point X_i for all $k' < k$. If there is at least one index $k' < k$ such that:

$$\left|\hat{f}_k(X_i) - \hat{f}_{k'}(X_i)\right| > \eta \hat{s}_{k'}(X_i) \ ,$$

then we do not accept $\hat{f}_k(X_i)$ and keep the estimates $\hat{f}_{k-1}(X_i)$ from the preceding iteration.

(d) *Stopping*: stop if $k = k^*$ or if $\hat{f}_k(X_i) = \hat{f}_{k-1}(X_i)$ for all i; otherwise increase k by 1 and continue with the adaptation step.

For the purposes of our analyses the estimates of the stepwise regression function are rounded to the closest integers, because the permissible values are categories. As can be seen from the above description, the outcomes of the analysis are affected by four parameters:

λ affects the smoothness of the resulting image and so does k*, which describes how far the design points influence each other, i.e., the degree of locality.

η is involved in the control step, which prevents the algorithm from losing previously detected discontinuities.

Finally, the choice of *kernel* clearly affects the results as well.

The parameters of the procedure can be tuned using the bootstrap method. Another important point concerns the applicability of the method. Clearly, the region should be homogeneous enough. However, how can this homogeneity be measured and how much of it is needed? This question is addressed to some degree in Section 3 where the accuracy of the above method is discussed.

2.2.2 Notes

AWS is a good method for pattern restoration in the case of a reasonably homogeneous area. However, it only takes one source or one set of observations into account. It is also very computer intensive, since neighborhood and weight matrices as well as all the estimated values of the previous iterations should be kept in the memory.

2.3 Naïve Smoothing

AWS assumes that the data are aligned on an ordinal scale, i.e., the classes may be ordered so that class 1 < class 2 < class 3 and so on. Thus, roughly speaking, a point between classes 1 and 9 would be interpreted by AWS as class 5. However, the classified images do not always correspond to this assumption. For example, on a land-cover map there is no way to order “forest”, “river”, and “residence”. In such situations, a naïve smoothing method would be more appropriate. Here, for each cell, we find a statistical mode in its neighborhood (the neighborhood may include the cell itself) and assign the value to the cell. From here on, this method will be denoted NSr-i, where NS is naïve smoothing, *r* is the neighborhood radius, and *i* is the number of iterations. In the work associated with this paper NS1-1 has mainly been used.

We now move on to examine the accuracy of the described methods in more detail.

3 Accuracy of the Existing Solutions

In this section some aspects of the sensitivity of the two image restoration methods presented above, namely the Bayesian total probability formula (BR) and adaptive weights smoothing (AWS) as well as that of naïve smoothing (NS) are researched. Since the phenomena are too complex to be examined analytically, only the simulation results and some general guidelines are supplied.

3.1 Summarizing the Source Accuracy

As described at the outset, the accuracy of each source is described by a C*C matrix, where C is the number of categories in the classification. For the purposes of the analysis, it is convenient, however, to be able to describe the source error more concisely, e.g., through the expected value and variation of the overall accuracy. Using the standard notation of Part 0 we say that the observation is accurate if the value observed by the source is exactly the true value:

$$acc_{si} = 1\{Y_{si} = X_i\} ,$$

i.e., acc_s is a binary variable with the expected value and variance respectively:

$$E(acc_s) = \sum_{z=1}^{C} e_{zss} P(z) \text{ and } Var(acc_s) = \sum_{z=1}^{C} e_{szz} P(z) - \left[\sum_{z=1}^{C} e_{szz} P(z) \right]^2 .$$

In order to describe all the sources by two numbers weights can be used. Let w_s describe the weight assigned to the source s then we have:

$$E(Acc) = \sum_{s=1}^{S} w_s E(acc_s) \Big/ \sum_{s=1}^{S} w_s \text{ and } Var(Acc_i) = \sum_{s=1}^{S} \left(w_s \Big/ \sum_{s=1}^{S} w_s \right) Var\left(acc_i\right) .$$

3.2 BR Sensitivity

The expected accuracy of the BR may be calculated according to the definition as:

$$E(acc) = \sum_{x=1}^{C} \sum_{y_1=1}^{C} \cdots \sum_{y_N=1}^{C} 1\left\{ \frac{\prod_{n=1}^{N} \zeta_{n,x,y_n} p(x)}{\sum_{x=1}^{C} \left[\prod_{n=1}^{N} \zeta_{n,x,y_n} p(x) \right]} \geq .5 \right\} \prod_{n=1}^{N} e_{n,x,y_n} P(x) ,$$

i.e., the expected value depends on both the actual and the perceived source error matrices. Since each source error matrix for C classes is defined by $C(C-1)$ parameters and the multinomial vector is defined through $(C-1)$ parameters, the accuracy depends

on 2*(*N*C+1)*(C-1)* parameters. For example, for three sources and four multinomial classes this amounts to 2*(3*4+1)*(4-1)=78 parameters. Clearly, this is too complex a relationship to examine in detail. The investigation will thus be limited to two sources with binary outcomes. The expected accuracy is then a function of 10 parameters and can be written as:

$$E(acc) = \sum_{x=0}^{1}\sum_{y_1=0}^{1}\sum_{y_2=0}^{1} 1\left\{\frac{\zeta_{1,x,y_1}\zeta_{2,x,y_2}\, p^{x}(1-p)^{1-x}}{\zeta_{1,0,y_1}\zeta_{2,0,y_2}(1-p)+\zeta_{1,1,y_1}\zeta_{2,1,y_2}\, p} \geq .5\right\} e_{1,x,y_1} e_{2,x,y_2} P^{x}(1-P)^{1-x}$$

.

3.2.1 Accuracy as a function of p(x) and P(x)

To begin with, I will assume that the accuracy of the two sources is known perfectly $Z_s = E_s$ for $s = 1,2$ and examine the behavior of the expected accuracy for the given source error matrices as a function of P and p.

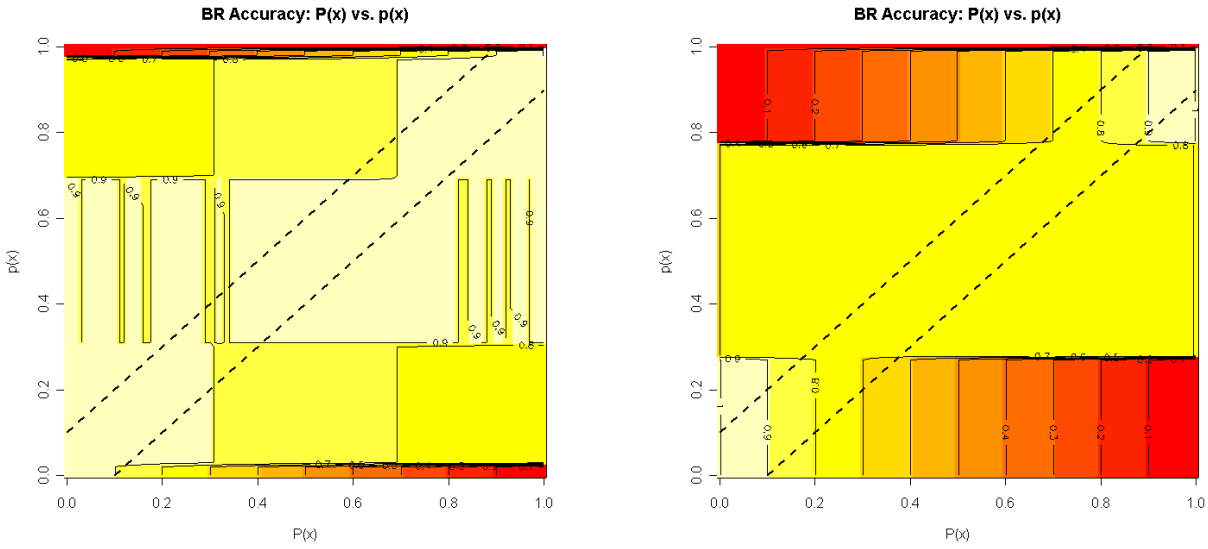


Figure 2: BR accuracy as a function of *P* and *p*. The maximum accuracy is reached on the line *P*=*p*. However, within a relatively narrow interval $p = P \pm .10$ the accuracy remains close to the optimal value. For the left-hand diagram, the overall expected source accuracy equals .85 with std .25, and for the right-hand diagram, the corresponding statistics are .625 and .33.

Firstly, it is a step function with five steps at most. The four jump points are calculated as:

$$\frac{\zeta_{10y_1}\zeta_{10y_2}}{\zeta_{10y_1}\zeta_{10y_2}+\zeta_{11y_1}\zeta_{11y_2}} \text{ for } y_1 = 0,1 \text{ and } y_2 = 0,1\ ,$$

whereas the levels of the stepwise constant function are calculated as various linear combinations of the terms:

$$E_{1xy_1}E_{2xy_2}P^x(1-P)^{1-x} \text{ for } x, y_1, y_2 \in \{0,1\} .$$

Thus, the true levels of the parameters influence the levels of the function and through them the maximum achievable accuracy whereas the modeling assumptions affect the variability of the accuracy. Two examples are shown in Figures 3a and b.

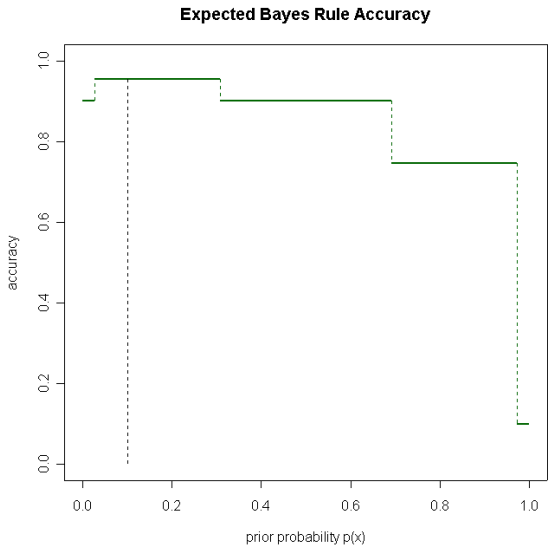


Figure 3a:

$$E_1 = \begin{bmatrix} .9 & .1 \\ .1 & .9 \end{bmatrix}, E_2 = \begin{bmatrix} .8 & .2 \\ .2 & .8 \end{bmatrix}, P = .1 .$$

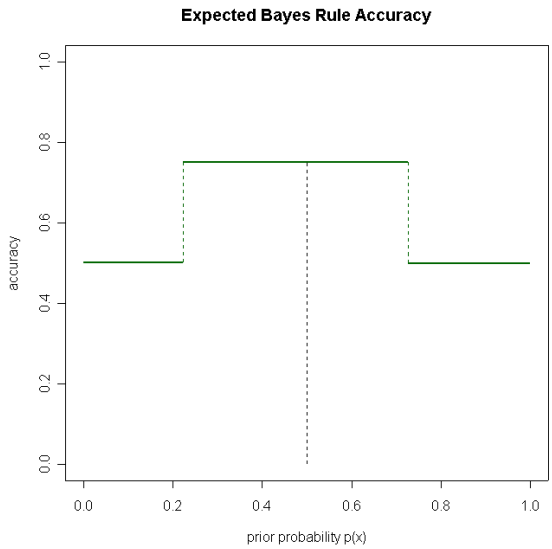


Figure 3b:

$$E_1 = \begin{bmatrix} .9 & .1 \\ .9 & .1 \end{bmatrix}, E_2 = \begin{bmatrix} .8 & .2 \\ .3 & .7 \end{bmatrix}, P = .5 .$$

The jump points are situated symmetrically around $p = .5$ if the source error matrices are symmetric. The function itself is symmetric around $p = .5$ if $P = .5$. As can be seen from the diagrams and also deduced intuitively, the maximum accuracy is reached when $p = P$.

3.2.2 Accuracy as a function of source errors E and Z

In order to investigate accuracy as a function of source error matrices E and Z, the simplest case will be considered, where:

$$E_1 = E_2 = \begin{bmatrix} e & 1-e \\ 1-e & e \end{bmatrix} \text{and } \mathrm{Z}_1 = \mathrm{Z}_2 = \begin{bmatrix} \zeta & 1-\zeta \\ 1-\zeta & \zeta \end{bmatrix} .$$

It will also be assumed that $p = P$, i.e., that we have perfect information regarding our prior.

In the case where modeling assumptions exactly correspond to the actual situation, i.e., $e = \zeta$, the maximum accuracy is reached by the method.

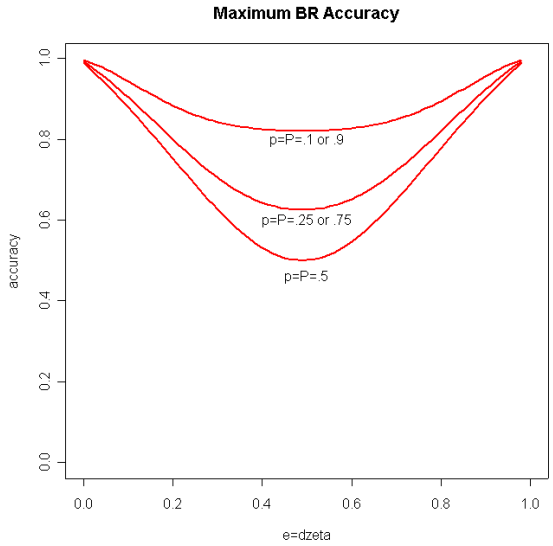


Figure 4: Maximum BR accuracy as a function of modeling assumptions. The potential maximum accuracy is higher for values of prior (*p* and *P*) and modeling (*e* and ζ) parameters which are further from *.50*.

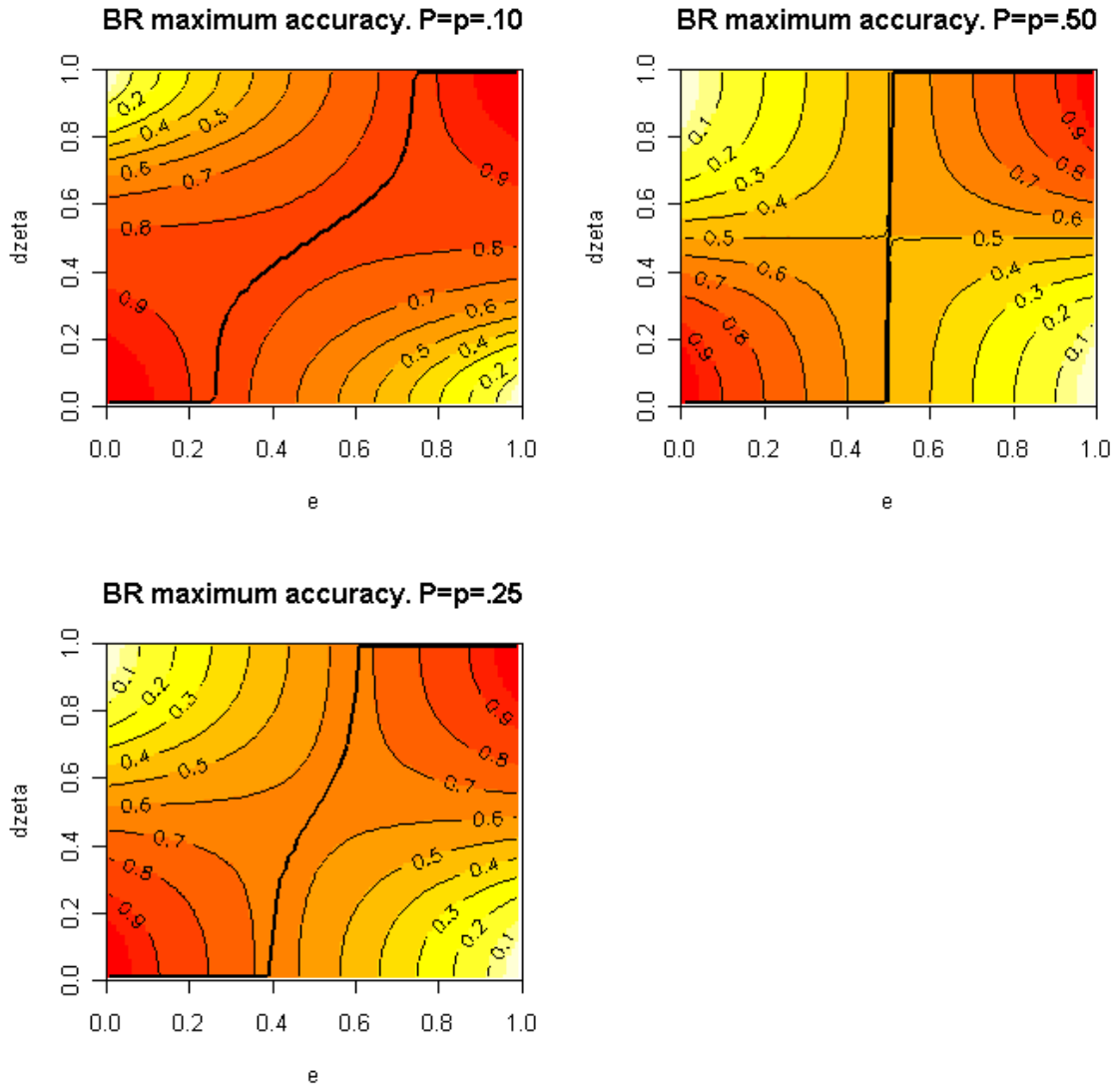


Figure 5: Maximum BR accuracy as a function of prior and modeling assumptions and parameters. The bold black line indicates the range of values over which the maximum accuracy is reached.

3.2.3 Prior elicitation

A question that also arises in Bayesian inference is the prior elicitation, i.e., how do we get information on E and P expressed in Z and p? In reality, the accuracy of the source will most likely be known quite precisely from technical specifications. Also, some information on the proportions of areas assigned to different categories will be known. There is, however, a theoretical formula allowing to make some inference about proper prior and modeling assumptions:

$$(P^{obs})^T = (P^{true})E \ .$$

For example, if we know that the source accuracy is $E = \begin{bmatrix} .7 & .2 & .1 \\ .2 & .6 & .2 \\ .1 & .1 & .8 \end{bmatrix}$ and the posterior distribution of the observations is $P^{obs} = [.53,.32,.15]$, then

$$P^{true} = \left(P^{obs}\right)^T E^{-1} = [.53 \quad .32 \quad .15] \begin{bmatrix} .7 & .2 & .1 \\ .2 & .6 & .2 \\ .1 & .1 & .8 \end{bmatrix}^{-1} = [.6655, .3069, .0276] .$$

3.3 AWS Sensitivity

The sensitivity of an iterative algorithm like AWS is too complex to be expressed analytically. However, the performance of the algorithm can still be assessed through simulations. The authors of the method have thoroughly tested it and come to the conclusion that the method performs very well for a piecewise constant image and provides good quality both within the homogeneous regions and near the edge (Polzehl and Spokoiny, 2000). It is also stable with respect to increasing noise level.

Generally, it is obvious that the accuracy of the method depends on homogeneity of the pattern as well as on the parameters of the algorithm, namely kernel function, smoothness parameters λ and k*,η, and the neighborhood structure. The optimal parameters can be found through the bootstrap method (Polzehl and Spokoiny, 2000).

Another important factor is the accuracy of the source. Since AWS makes no adjustments for perceived source inaccuracy however, all that can be said on this matter is the accuracy range for which it is applicable.

Since describing AWS sensitivity in detail is too big a task, I will only describe a few major features here, which should be of interest in my applications.

3.3.1 Moran's I

As mentioned above, the homogeneity of the pattern plays a big role in the accuracy of the estimation. Intuitively, homogeneous means that grid cells with similar values tend to stick together. Formally, various formulae exist to measure the homogeneity or spatial correlation. I will use Moran's I, which is defined by the formula:

$$I = \frac{I}{\sum_{i=1}^{I}\sum_{j=1}^{I} W_{ij}} * \frac{\sum_{i=1}^{I}\sum_{j=1}^{I} W_{ij}\left(x_i - \bar{x}\right)\left(x_j - \bar{x}\right)}{\sum_{i=1}^{I}\left(x_i - \bar{x}\right)^2},$$

where W = neiborhood matrix and x are the values in grid cells.

The values of Moran's I range from +1 meaning strong positive spatial autocorrelation (homogeneity) to 0 meaning a random pattern to –1 indicating a strong negative spatial autocorrelation.

3.3.2 Accuracy vs. homogeneity and source precision

In order to illustrate AWS accuracy, five patterns of different homogeneity, displayed in Figure 6, were distorted using three different source accuracies (*.95, .50, .05)* ten times. Afterwards, the patterns were estimated using AWS at optimal parameters (the estimation results were robust with respect to parameters of the algorithms). The results are shown in Table 1.

Table 1: AWS accuracy.

Source accuracy	**Pattern I**	**Pattern II**	**Pattern III**	**Pattern IV**	**Pattern V**
.95	.4191	.5902	.8770	.9605	.9996
.50	.5094	.5023	.4797	.5035	.6258
.05	.5840	.4109	.1168	.0324	.0012
Moran's I	-.0323	.0366	.4839	.8108	-

For the high precision observations, the accuracy of the estimation grows with the homogeneity of the source. However, for a low precision, the accuracy of the estimation actually gets worse as the pattern becomes more homogeneous. By way of general guidelines, it can be said that the AWS algorithm gives good results for patterns with Moran's I above .5 and for data observed with over 90% accuracy. In this case, the accuracy of the estimates will be over 80%.

3.4 NS Accuracy

The accuracy of NS1-1 was tested similarly. The five patterns in Figure 6 were distorted using the sources of different accuracy and were then restored using NS1-1. Ten simulations were run for each case. The results of the simulations are summarized in the Table 2.

Table 2: Naïve Smoothing (NS1-1) Accuracy.

Source accuracy	**Pattern I**	**Pattern II**	**Pattern III**	**Pattern IV**	**Pattern V**
.95	.7492	.8449	.9590	.9871	.9992
.50	.4922	.4949	.4820	.4543	.4520
.05	.2734	.1695	.0438	.0086	.0000
Moran's I	-.0323	.0366	.4839	.8108	-

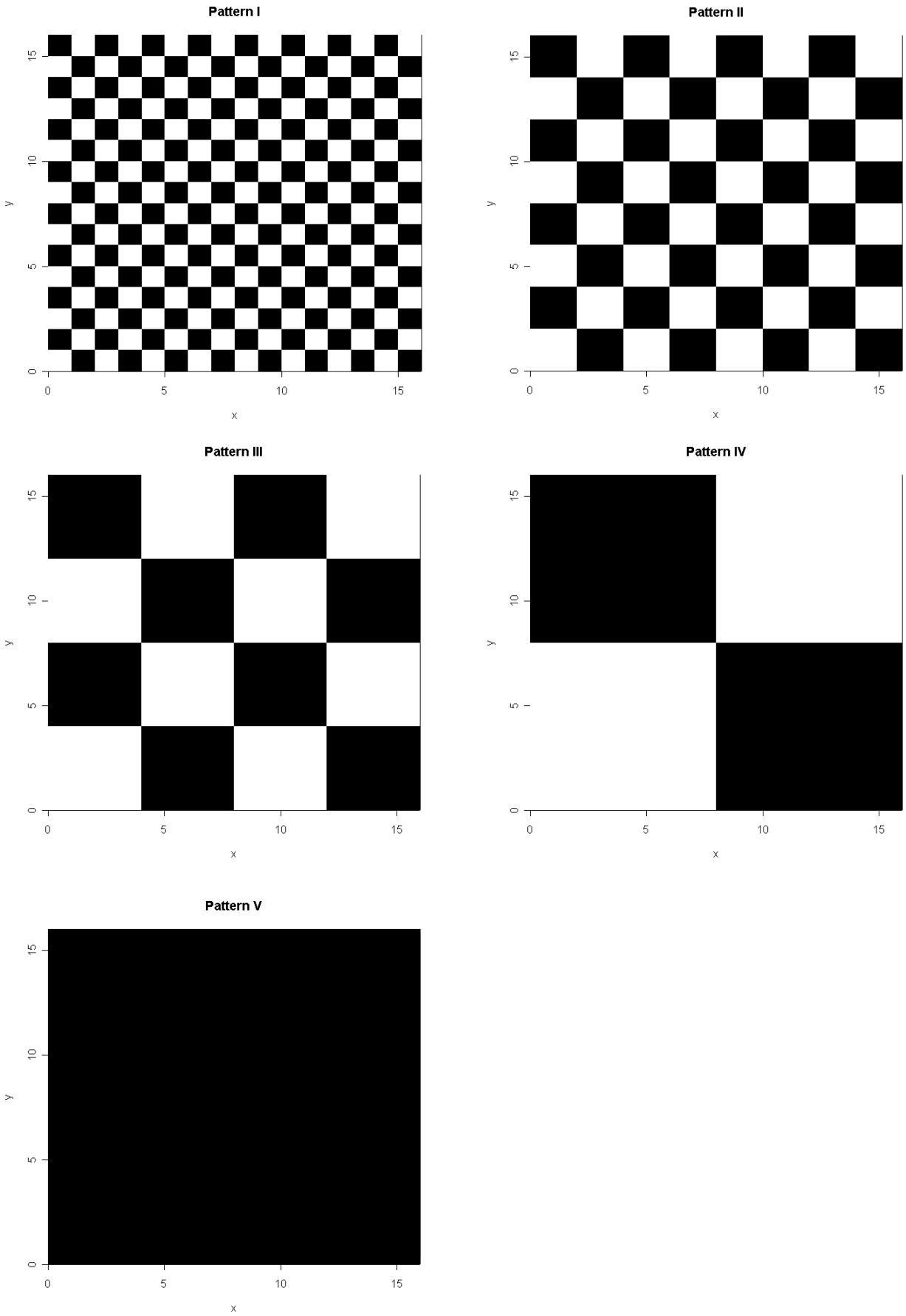


Figure 6: Five binary patterns of different heterogeneity.

4 Proposed Approach

So far, two of the existing approaches have been described and examined in terms of their accuracy and sensitivity. To recap, the BR is best to use when several sources of information have to be taken into account. It is especially accurate when both the general accuracy of the sources is far from 50% (guessing) and when proportions of the categories are unequal. Generally, accuracy is >80% and the results of the estimation are more accurate than the observed patterns as such. However, taking into account the information available on spatial correlation would improve this. On the other hand, AWS seems to be one of the best-performing tools to deal with spatial correlation. However, AWS cannot analyze several sources simultaneously. Its accuracy depends on the general homogeneity of the area, the algorithm parameters, which can be fine-tuned using the bootstrap method, and the accuracy of the source of observations. It is therefore reasonable to suggest that a combination of the two methods might yield better results than any of the methods taken separately.

4.1 AWS-BR and BR-AWS

Two logical possibilities follow. We can either apply the smoothing algorithm to all the S samples and then use the total probability rule (this combination will be denoted AWS-BR), or we can use the Bayesian method first and apply smoothing later (BR-AWS). There is no logical reason *a priori* to prefer one combination over the other. The conducted simulations have shown that BR-AWS gives consistently better results than AWS-BR. It is thus BR-AWS, which is suggested as a new improved method for image restoration when multiple sources of information are available.

Something needs to be said about the performance of the method as well as about its sensitivity to modeling and prior assumptions. Since it is a combination of the two methods previously considered in detail, it is reasonable to conclude that its accuracy follows the rules laid out earlier. Namely, its performance is better when source error is far from 50%. It improves with the homogeneity of the area and it can be improved by fine-tuning the parameters. However, its accuracy is still better than that of the above methods separately, because it simultaneously takes into account both spatial correlation and all the available information. This information includes not only the observed patterns, but also data on source accuracy, categories distribution, etc.

To illustrate, in the next section I provide two applications of the suggested approach and compare the results with those of the two traditional approaches described above as well as with the BR-AWS approach.

5 Applications

5.1 Application 1: Binary Data From Three Observation Sources

To illustrate the estimation processes and compare the results of different methods, I will first consider a case of a black and white image on a 16x16 regular square grid. The true pattern is shown in Figure 7.

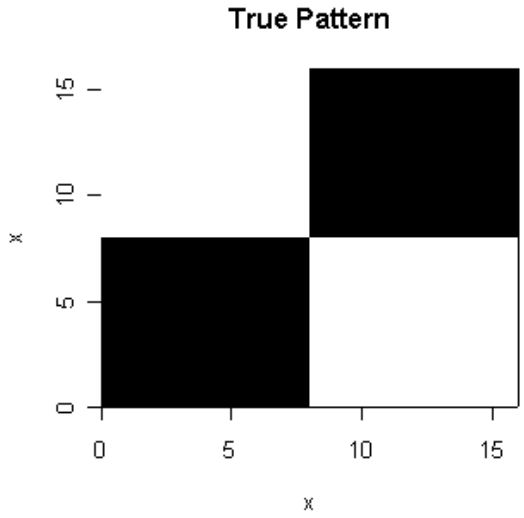


Figure 7: True pattern: black and white image on a 16x16 regular square grid.

Three sources with the accuracy matrices:

$$E_1 = \begin{bmatrix} .90 & .10 \\ .10 & .90 \end{bmatrix}, E_2 = \begin{bmatrix} .75 & .25 \\ .25 & .75 \end{bmatrix}, E_3 = \begin{bmatrix} .80 & .20 \\ .15 & .85 \end{bmatrix}$$

were involved. The simulated observed patterns are shown in Figure 8.

The results of AWS are shown in Figure 9. The optimal parameters $k^* = 3$ and $\lambda = 4$ were found using the bootstrap method described earlier. The Bayesian estimation, BR-AWS and AWS-BR are further shown in Figure 10. Thus, the most accurate results are achieved through BR-AWS and AWS-BR estimations.

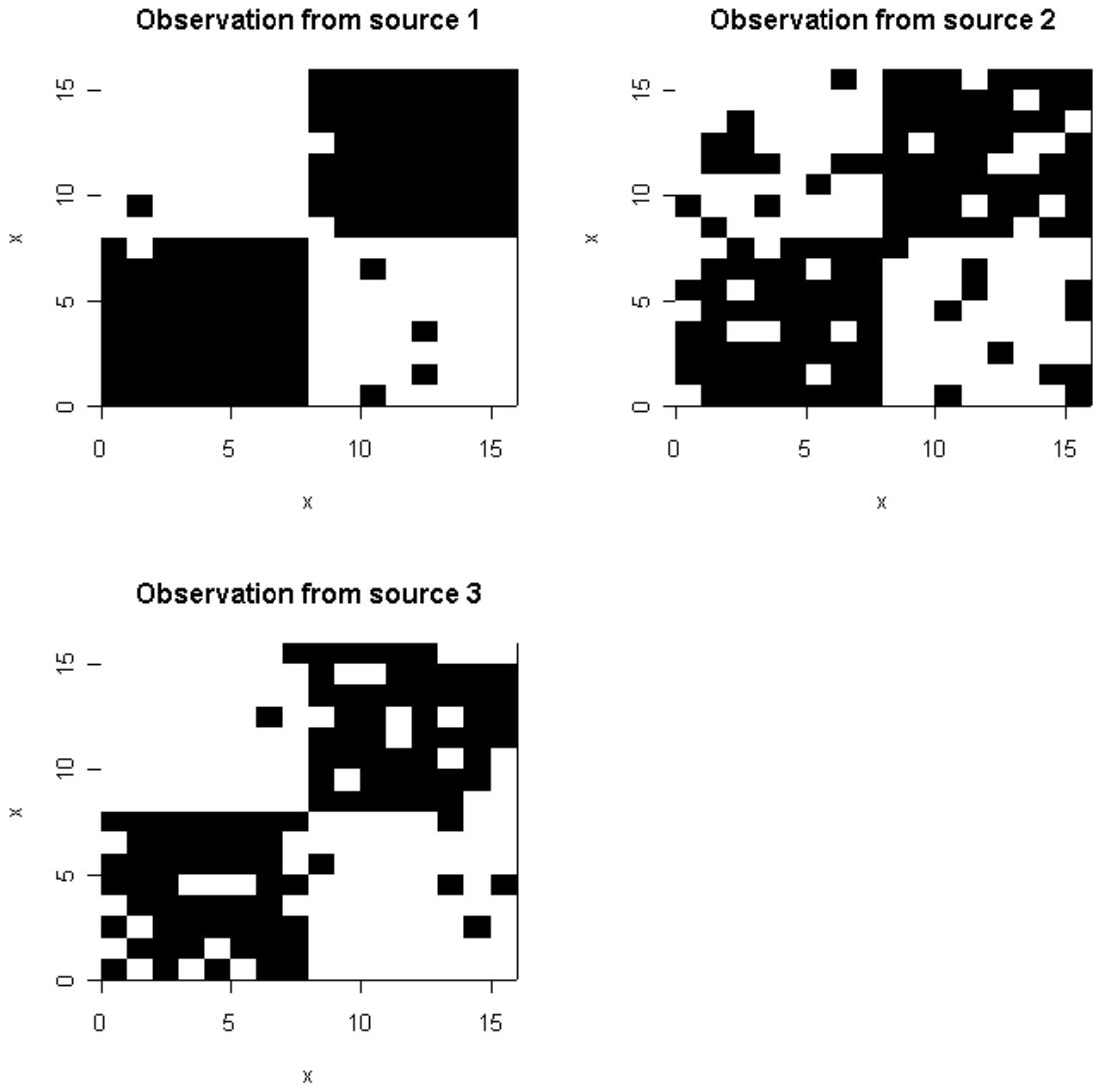


Figure 8: Observed patterns from sources with different sensitivities.

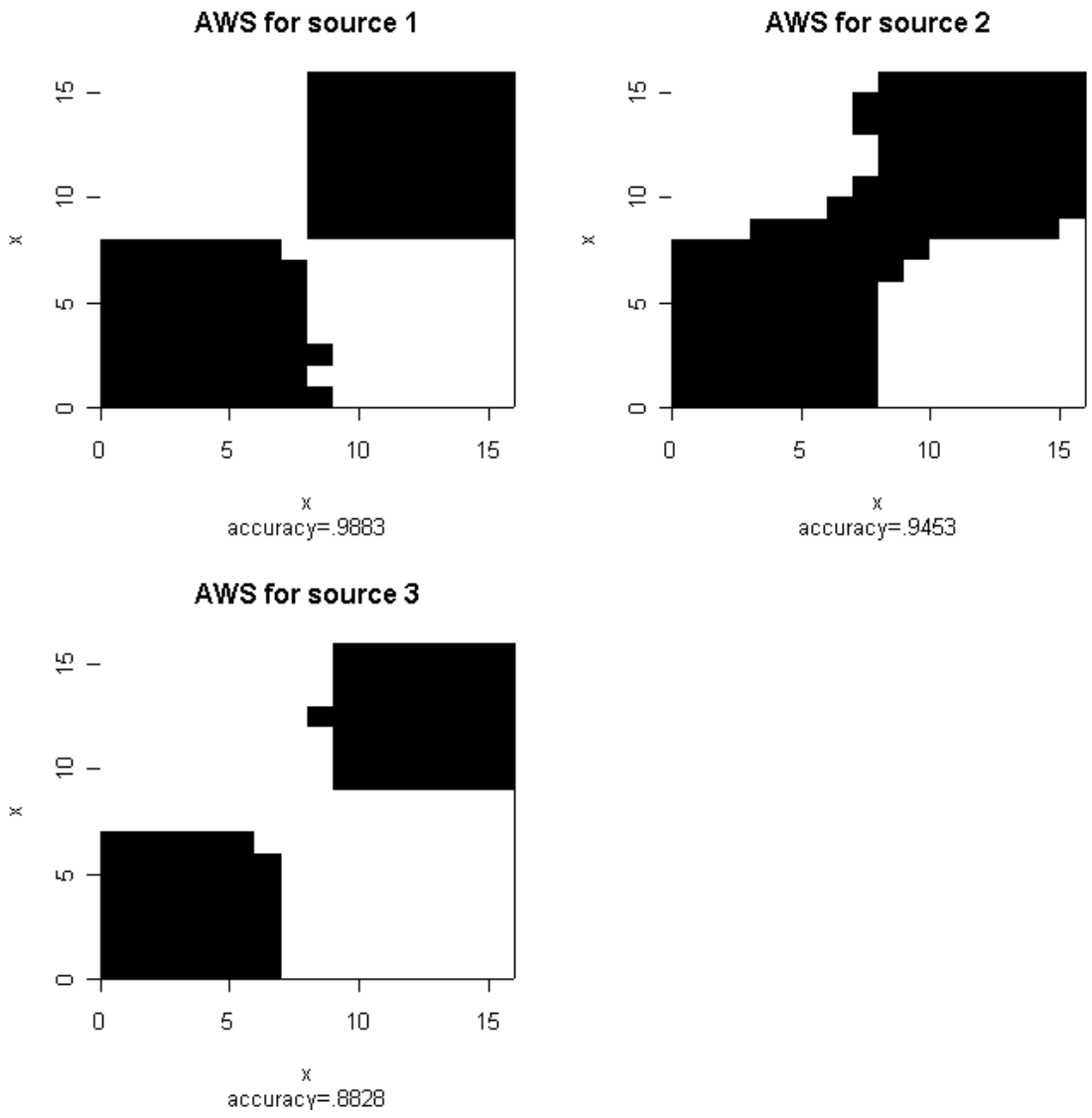


Figure 9: AWS for patterns observed by different sources.

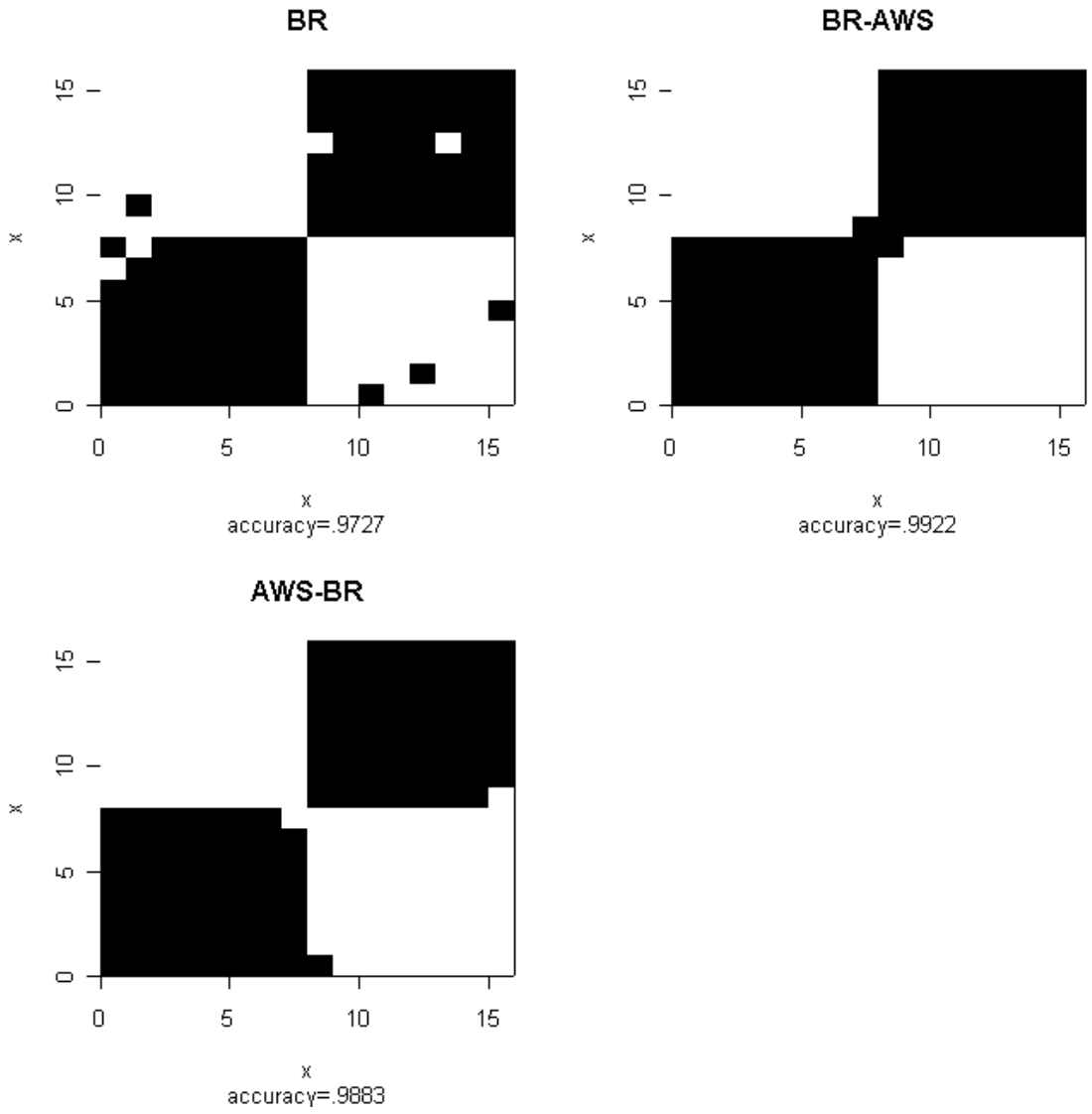


Figure 10: Results of BR, BR-AWS, and AWS-BR estimation and their accuracy.

Table 3: Accuracy of various image restoration methods in the first application.

Source accuracy matrix	Overall source accuracy	Observed source accuracy	AWS accuracy	BR	BR-AWS	AWS-BR
$E_1 = \begin{bmatrix} .90 & .10 \\ .10 & .90 \end{bmatrix}$	.9000	.9688	**.9883**	.9723	**.9922**	**.9883**
$E_2 = \begin{bmatrix} .75 & .25 \\ .25 & .75 \end{bmatrix}$	.7500	.8164	.9453			
$E_3 = \begin{bmatrix} .80 & .20 \\ .15 & .85 \end{bmatrix}$	.8250	.8594	.8828			

5.2 Application 2: Multi-class Image with Three Sources of Observation

An already classified 87x111 grid cells LANDsat image of the forest area in Siberia (Russia) has been used in this application. The true image is shown in Figure 11. The black color stands for the missing values and the other 32 colors denote different land-cover classes. To illustrate the attractiveness of the suggested method, three observed images were simulated. All three sources were given a similar error matrix:

$$E = \begin{bmatrix} .50 & .50/31 & \dots & .50/31 \\ .50/31 & .50 & \dots & .50/31 \\ \vdots & \vdots & \ddots & \vdots \\ .50/31 & .50/31 & \dots & .50 \end{bmatrix}.$$

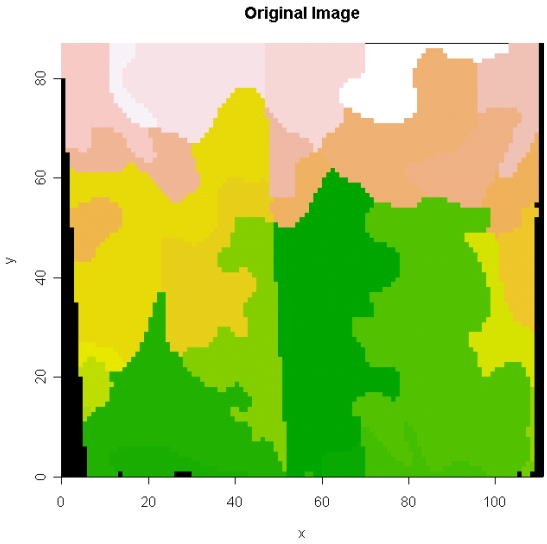


Figure 11.

Thus, the probability of identifying the class correctly was equal to 50% (equivalent to guessing) and the probability of erroneously identifying it with any other class was equal to *.50/31=.0161*. The three generated images are presented in Figure 12.

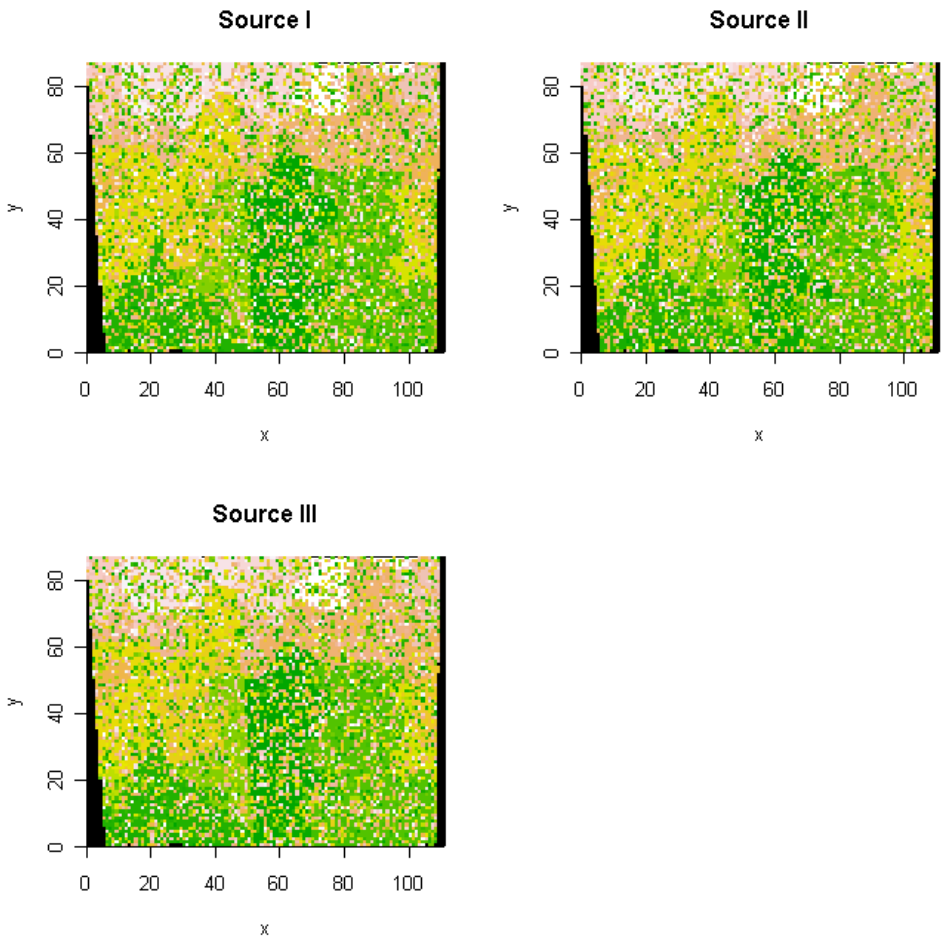


Figure 12: The observed (simulated) patterns.

The results of the BR application and the image resulting after naïve smoothing are shown further in Figures 13 and 14. Note that the BR is applied assuming perfect prior and modeling information. It is obvious that the naïve smoothing improves the estimate considerably. A second smoothing only slightly improves the estimate further. The accuracy of various sources and estimation methods is summarized in Table 4.

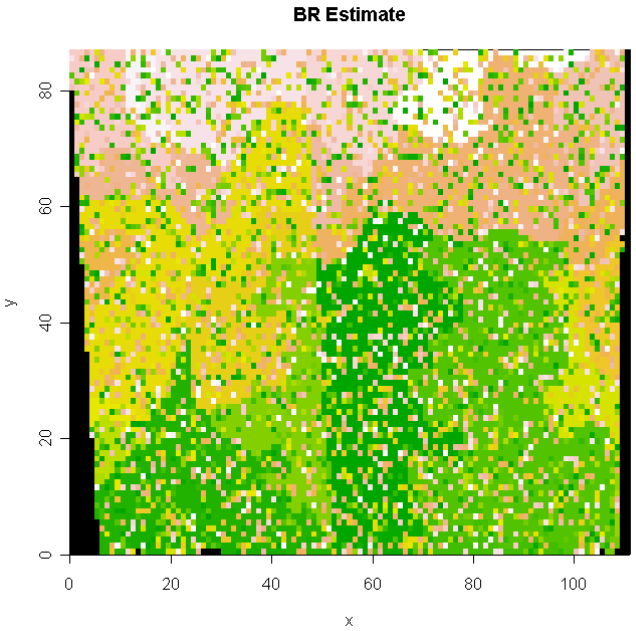


Figure 13.

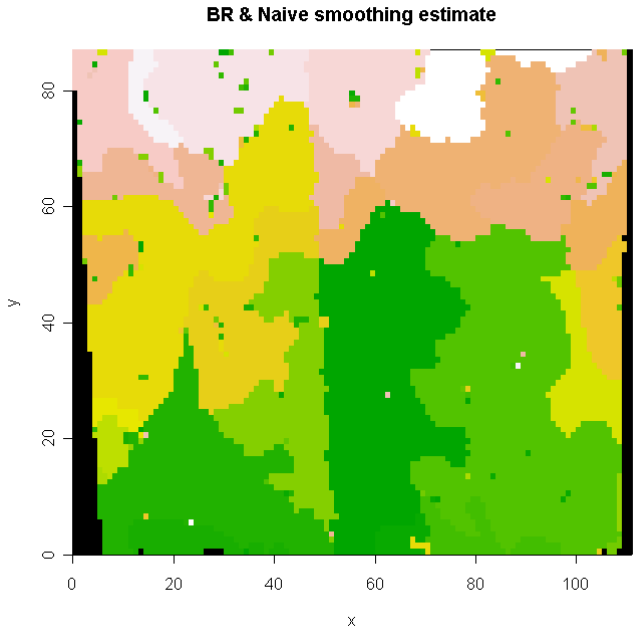


Figure 14.

Table 4: Accuracy of the various image restoration methods in application 2.

	Stated source accuracy	**Observed source accuracy**	**Naïve smoothing**	**BR**	**BR and Naïve smoothing once**	**BR and Naïve smoothing twice**
Source I	.5209	.5223	.9173	.6308	**.9522**	.9652
Source II	.5209	.5178	.9088			
Source III	.5209	.5159	.9090			

Thus, a combination of BR and naïve smoothing gives the best accuracy of *95%*. It is a significant improvement from the observed *50%* and a useful one over the naively smoothed *90%*.

6 Discussion

The suggested combination of BR and AWS performs better in image restoration than the methods separately when multiple data sources are available. It allows to take into account both the existence of spatial correlation and the multiple sources of differing quality. In this, it is different from existing methods. In this paper, however, only a rather small part of its possibilities and properties have been researched. What follows is a discussion of possible future directions of work.

Although the accuracy and the sensitivity of the method have been studied in some detail through simulations, more evaluation is needed. Analytical properties should be deduced if possible.

Although the classification method was assumed to be the same among sources, such an assumption is not necessarily realistic. More work therefore is needed on the question of classification compatibility. Another problem may arise if the spatial grids are different for different sources — the case of spatial misalignment. These questions of incompatibility should be addressed in order to make the method more practically applicable. It may also widen its applicability. So far, the possible applications concerned, e.g., combining satellite images made at different times to produce the most accurate map of vegetation, or to combine maps classified by different experts. In short, several observed samples on the same variable. But, another possible application would be to combine information on the different characteristics from different sources to produce a map or an image of some quantity, which is a function of those.

AWS can be described as a method of estimating a stepwise correlation function over a spatial field. Originally, the levels of the function are not limited to any particular set but, in order to make it applicable to the categorical situation, it has been modified to select the levels from the set $\{1,2,\ldots,C\}$ where C is the amount of categories. The bootstrap method for fine-tuning the AWS parameters has not been modified in any way. Perhaps, however, the modifications taking into account that we deal with categorical data may improve the method further.

As was shown, when the classification is not ordinal, the application of naïve smoothing gives good results. The questions of the applicable radius and the optimal number of iterations remain. It is suggested that the first may be deduced from the level of spatial correlation within the image. As to the number of iterations, the accuracy benefits of further smoothing will progressively become smaller and smaller and thus may arrive at some kind of convergence criteria.

Finally, the technical aspect should be mentioned. The AWS algorithm is computationally intensive. In the Appendices there is a description (Appendix 1) and listing (Appendix 2) of the functions, in R-language, used for the analysis described in this paper. In order to make the method more practical, it would be worthwhile to produce software on a lower level programming language such as C++.

To conclude, the suggested method AWS-BR is a promising solution to image restoration when data from multiple sources are available. Some research and

programming, however, is still required to make it applicable to field data, such as the data collected for the Forestry Project's Siberia II study.

References

Besag, J., J. York and A. Mollie (1991). Bayesian Image Restoration with Two Applications in Spatial Statistics. *Annals of the Institute of Mathematical Statistics*, 43: 1–59.

Divino, F., A. Frigessi and P.J. Green (1999). Penalized Pseudo-likelihood Inference in Spatial Interaction Models with Covariates. Downloaded from and available on the Internet: http://www.nr.no/~frigessi/Research.html.

Gelman, A., J.B. Carlin, H.S. Stern and D.B. Rubin (1995). *Bayesian Data Analysis*. Chapman and Hall.

Kitamoto, A. and M. Takagi (1999). Image Classification Using Probabilistic Models that Reflect the Internal Structure of Mixels. *Pattern Analysis and Applications*, 2: 31–43.

Krell, G., A. Herzog and B. Michaelis (1996). Real-Time Image Restoration with an Artificial Neural Network. In: Proceedings of the International Conference on Neural Networks (ICNN) '96, Washington, 3–6 June 1996, pp. 1552–1557. Downloaded from and available on the Internet: http://ipe.et.uni-magdeburg.de/TI/research1_publ.html.

Kunz, D., K.-J. Schilling and T. Vögtle (1997). A New Approach for Satellite Image Analysis by Means of a Semantic Network. In: SMATI 97, W. Förstner and L. Plümer (eds.), Birkhäuser, pp. 20–36. Downloaded from and available on the Internet: http:/www-ipf.bau-verm.uni-karlsruhe.de/.

Mascarilla, L. (1997). Fuzzy Rules Extraction and Redundancy Elimination: An Application to Remote Sensing Image Analysis. *International Journal of Intelligent Systems*, 12: 793–817.

Osnes, K. and O. Aalen (1999). Spatial Smoothing of Cancer Survival: A Bayesian Approach. *Statistics in Medicine*, 18: 2087–2099.

Polzehl, J. and V.G. Spokoiny (2000). Adaptive Weights Smoothing with Applications to Image Restoration. *Journal of Royal Statistical Society*, Series B, 335–354.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.

APPENDIX 1: R-functions Description

R-language was used in the analyses and simulations. Information on this language as well as the source code may be found and downloaded from http://www.R-project.org/.

LIST OF FUNCTIONS:

rmulti()

rmulti0()

gen.random()

br2bin.acc.px()

br()

neighbor1()

AWS.3d

source.error.sum()

AWS.3d.bootstrap()

moran.i()

neighbor()

naive1()

R-FUNCTION: **rmulti()**

DESCRIPTION: Generates a matrix of multinomial observations.

FILE: A:/rmulti_fun.txt

FORMAT: rmulti(N,m,C,P)

PARAMETERS:

N = number of variables (multinomial vectors) to be simulated.
m = size of each of the above vectors (can be either constant or a vector of length N).
C = number of classes: 1,2,…,C.
P = vector or matrix of multinomial probabilities (equal class probabilities *by default*).

OUTPUT: Y = a matrix of multinomial observations.

NOTES: Uses a more basic function rmulti0().

EFFICIENCY:

EXAMPLE:

```
> N
[1] 5
> m
[1]  2  3  5  7 20
> C
[1] 4
> P
     [,1] [,2] [,3] [,4]
[1,]  0.7  0.1  0.1  0.1
[2,]  0.4  0.2  0.2  0.2
[3,]  0.1  0.3  0.3  0.3
[4,]  0.5  0.5  0.0  0.0
[5,]  1.0  0.0  0.0  0.0
> rmulti(N,m,C,P)
     [,1] [,2] [,3] [,4]
[1,]    2    0    0    0
[2,]    2    0    1    0
[3,]    2    2    1    0
[4,]    3    4    0    0
[5,]   20    0    0    0
>
```

R-FUNCTION: **rmulti0()**

DESCRIPTION: Generates a vector of multinomial observations.

FILE: A:/rmulti0_fun.txt

FORMAT: rmulti0(m,C,P)

PARAMETERS:

m = size of each multinomial vector.
C = number of classes: 1,2,…,C.
P = vector of multinomial probabilities (equal class probabilities *by default*).

OUTPUT: Y = a vector of multinomial observations.

NOTES: rmulti0() is a simplified version of rmulti().

EFFICIENCY:

EXAMPLE:

```
> rmulti0(15,7,)
[1] 0 3 6 0 3 1 2
```

R-FUNCTION: **gen.random()**

DESCRIPTION: Generates multinomial random observations on a regular grid. Further generates source observations for the simulated grid according to the supplied source error matrices.

FILE: A:/gen_random_fun.txt

FORMAT: gen.random(size.x,size.y,C,N,E,P,pic)

PARAMETERS:

size.x = width of the grid.
size.y = height of the grid.
C = number of multinomial classes.
N = number of sources.
E = array of source error matrices.
P = true proportions of multinomial classes.
pic =T/F indicates whether a picture is to be drawn (*False by default).*

OUTPUT:

X = "true" situation (x,y).
Y = array of sources observations (x,y,source).
if pic=T then graphical output results.

NOTES:

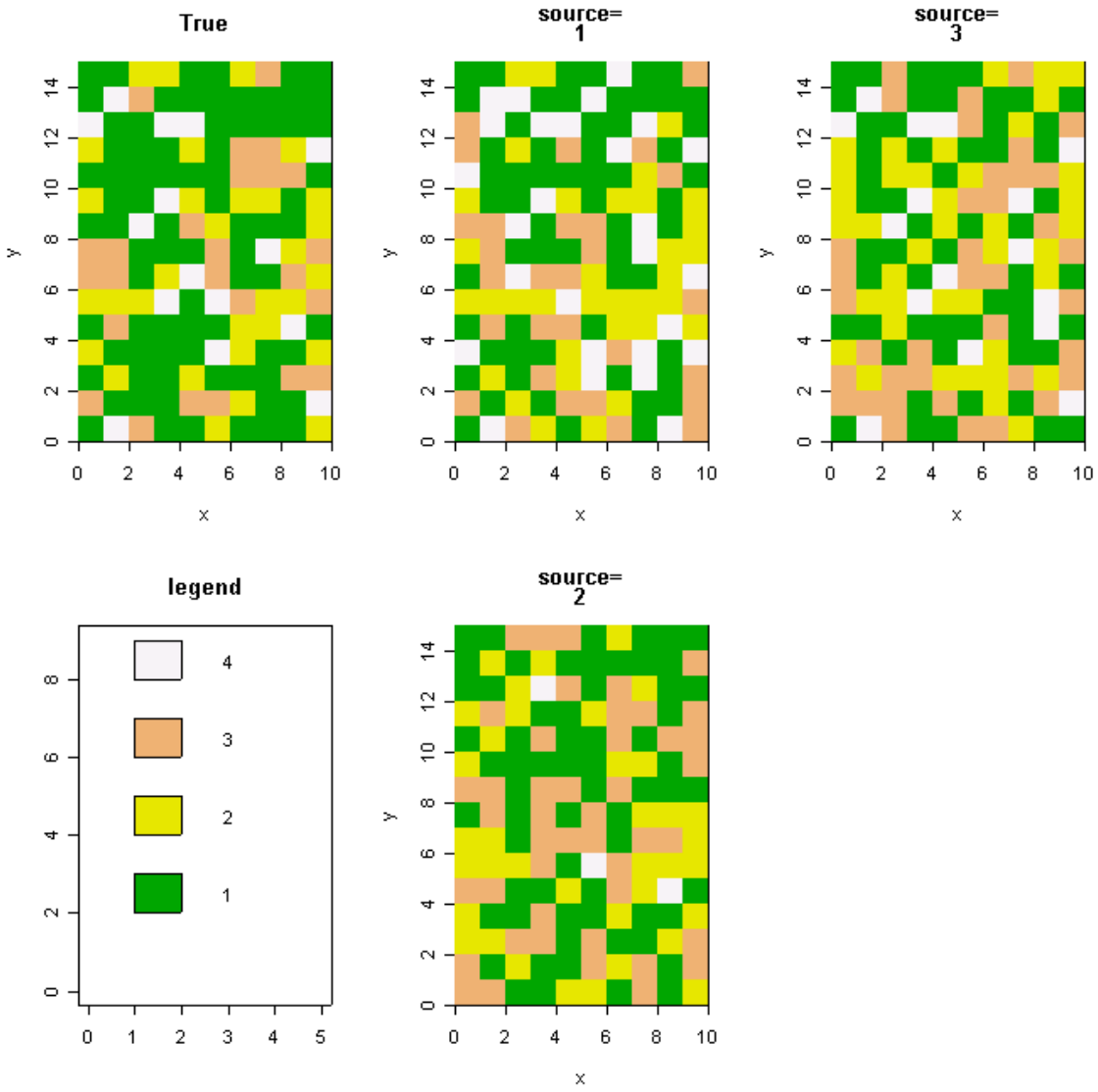
EFFICIENCY:

EXAMPLE:

```
> size.x<-10
> size.y<-15
> C<-4
> N<-3
>
> P<-c(.5,.2,.2,.1)
>
> E1<-array(c(.7,.1,.1,.1,.1,.7,.1,.1,.1,.1,.7,.1,.1,.1,.1,.7),dim=c(4,4))
> E2<-array(c(.6,.2,.2,.2,.2,.6,.2,.2,.2,.2,.6,.2,.2,.2,.2,.6),dim=c(4,4))
> E3<-array(c(.5,.3,.2,.0,.3,.4,.2,.1,.2,.2,.6,.0,.0,.1,.0,.9),dim=c(4,4))
>
> E<-array(cbind(E1,E2,E3),dim=c(4,4,3))
>
> E
```

```
, , 1

     [,1] [,2] [,3] [,4]
[1,]  0.7  0.1  0.1  0.1
[2,]  0.1  0.7  0.1  0.1
[3,]  0.1  0.1  0.7  0.1
[4,]  0.1  0.1  0.1  0.7

, , 2

     [,1] [,2] [,3] [,4]
[1,]  0.6  0.2  0.2  0.2
[2,]  0.2  0.6  0.2  0.2
[3,]  0.2  0.2  0.6  0.2
[4,]  0.2  0.2  0.2  0.6

, , 3

     [,1] [,2] [,3] [,4]
[1,]  0.5  0.3  0.2  0.0
[2,]  0.3  0.4  0.2  0.1
[3,]  0.2  0.2  0.6  0.0
[4,]  0.0  0.1  0.0  0.9
>
> gen.random(size.x,size.y,C,N,E,P,T)
…
```

GRAPHICAL OUTPUT:



R-FUNCTION: **br2bin.acc.px()**

DESCRIPTION: Calculates expected accuracy of the Bayesian Image Restoration applied to two sources in a binary case as a function of prior probability *p.*

FILE: H:/R-routines/br2bin_acc_px_fun.txt

FORMAT: br2bin.acc.px(E1,E2,e1,e2,P,pic)

PARAMETERS:

E1 = 2*2 primary source error matrix.
E2 = 2*2 secondary source error matrix.
e1 = 2*2 perceived primary source error matrix (*equal to E1 by default).*
e2 = 2*2 perceived secondary source error matrix (*equal to E2 by default).*
P = true 0-1 proportion.
pic =T/F depending on whether or not a graph is wanted (*False by default).*

OUTPUT:

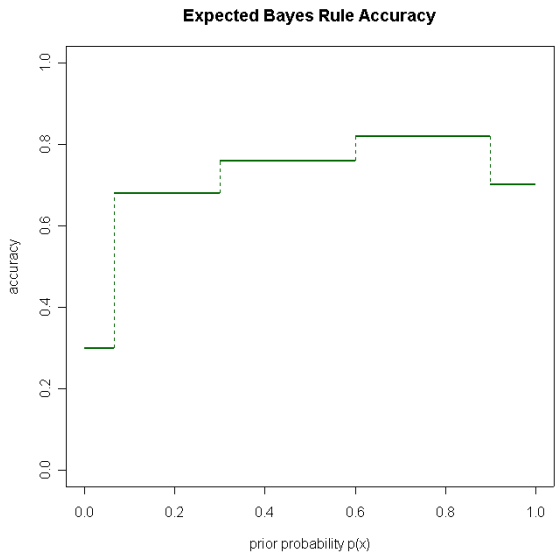
JP.order = jump points of the step function sorted in ascending order.
levelf = corresponding levels of the step function.
max.f = maximum expected accuracy.
Jlength.max = the length of the interval containing the maximum expected accuracy.

NOTES: The function is useful for preliminary investigation of BR accuracy.

EFFICIENCY:

EXAMPLE:

```
> E1
     [,1] [,2]
[1,]  0.4  0.6
[2,]  0.8  0.2
> E2
     [,1] [,2]
[1,]  0.9  0.1
[2,]  0.3  0.7
> br2bin.acc.px(E1,E2,,,P=.7,pic=T)
$JP.order
[1] 0.06666667 0.30000000 0.60000000 0.90000000

$levelf
[1] 0.30 0.68 0.76 0.82 0.70

$max.f
[1] 0.82

$Jlength.max
[1] 0.3

>
```



R-FUNCTION: **br()**

DESCRIPTION: Performs image restoration from a combination of different sources and prior information using Bayesian Total Probability formula and mode estimation.

FILE: A:/br_fun.txt

FORMAT: br(Y,C,e,p,pic)

PARAMETERS:

Y = array of observations from different sources (size.x, size.y, sources).
C = number of multinomial classes.
e = perceived source error matrices (C, C, sources).
p = prior information on multinomial probability vector.
pic =T/F depending on whether a graph is wanted.

OUTPUT:

Y.est = array of estimated values (size.x,size.y).
P.post = array of posterior multinomial probabilities for each grid cell (size.x, size.y, C).

NOTES: The estimation is done using the Bayesian total probability formula:

$$P(X_i = x | Y_{1i} = y_{1i} ..., Y_{Si} = y_{Si}) = \frac{\left\{\prod_{s=1}^{S}\left[P(Y_{si} = y_{si} | X_i = x\right]\right\}P(X_i = x)}{\sum_{x=1}^{C}\left\{\prod_{s=1}^{S}\left[P(Y_{si} = y_{si} | X_i = x\right]\right\}P(X_i = x)}$$

where S = the number of sources and C = the number of classes.

The mode estimation was used, i.e., the class with the largest posterior probability was chosen. In the case of multi-modality, one of the modes is chosen randomly.

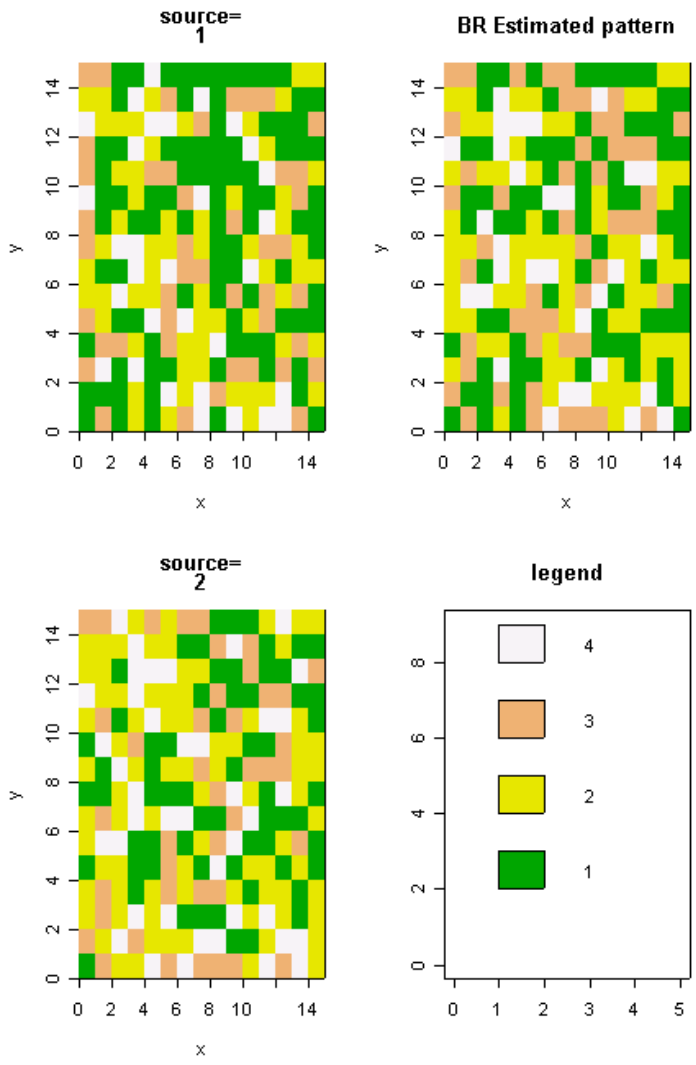
EFFICIENCY:

EXAMPLE:

```
> size.x<-15
> size.y<-15
> C<-4
> N<-2
> P<-c(.4,.3,.2,.1)
>
> E1<-array(c(.7,.1,.1,.1,.2,.6,.2,.2,.1,.3,.3,.3,.0,.0,.6,.4),dim=c(4,4))
> E2<-array(c(.3,.3,.2,.2,.4,.5,.1,.0,.0,.1,.6,.3,.3,.1,.1,.5),dim=c(4,4))
>
> E1
     [,1] [,2] [,3] [,4]
[1,]  0.7  0.2  0.1  0.0
[2,]  0.1  0.6  0.3  0.0
[3,]  0.1  0.2  0.3  0.6
[4,]  0.1  0.2  0.3  0.4
>
```

```
> E2
     [,1] [,2] [,3] [,4]
[1,]  0.3  0.4  0.0  0.3
[2,]  0.3  0.5  0.1  0.1
[3,]  0.2  0.1  0.6  0.1
[4,]  0.2  0.0  0.3  0.5
>
> E<-array(cbind(E1,E2),dim=c(4,4,2))
>
> source("H:/R-routines/gf/gen_random_fun.txt")
> GEN<-gen.random(size.x,size.y,C,N,E,P,F)
>
> source("H:/R-routines/gf/br_fun.txt")
> br(GEN$Y,C,E,P,pic=T)$Y.est
…
```

GRAPHICAL OUTPUT:



R-FUNCTION: **neighbor1()**

DESCRIPTION: Calculates neighborhood matrix of radius 1 for the user-defined grid according to the chosen method.

FILE: A:/neighbor1_fun.txt

FORMAT: neighbor(size.x, size.y, method="king")

PARAMETERS:

size.x = width of the grid.
size.y = height of the grid.
method = contiguity definition. There are three methods to choose from: "rook", "bishop", and "king" as follows:

Rook	**Bishop**	**King**
0 1 0	1 0 1	1 1 1
1 x 1	0 x 0	1 x 1
0 1 0	1 0 1	1 1 1

The default method is "king".

OUTPUT: A neighborhood binary matrix with 1 standing for neighborhood and 0 for lack thereof. The dimensions of the output matrix are (size.x*size.y,size.x*size.y).

NOTES:

EFFICIENCY:

EXAMPLE:

```
> source("H:/R-routines/gf/neighbor1_fun.txt")
> neighbor1(2,2,"rook")
     [,1] [,2] [,3] [,4]
[1,]    0    1    1    0
[2,]    1    0    0    1
[3,]    1    0    0    1
[4,]    0    1    1    0
>
```

R-FUNCTION: **AWS.3d()**

DESCRIPTION: Performs adaptive weights smoothing for categorical data on a 2D grid.

FILE: A:/AWS_3d_fun.txt

FORMAT: AWS.3d(Y,Cl,lambda,nu,k.max,kernel,method,U,res.m,sigma.m,pic)

PARAMETERS:

Y = 2D matrix of observations on the grid.
Cl = number of multinomial classes.
lambda = AWS algorithm smoothness parameter (see Section 2.2 for a detailed description).
k.max = maximum number of iterations.
nu = AWS algorithm control parameter (see Section 2.2 for a detailed description).
kernel = describes the type of kernel used in AWS. Three options currently implemented are:

"exp" exponential e^{-x^2}

"uni" uniform $1\{|x| \le 1\}$

"tri" triangular $\begin{cases} 0 & \text{if} \quad |x| < 2 \\ 1-.5x & \text{if} \quad |x| \ge 2 \end{cases}$

method = contiguity definition. Available options are:

"given", in which case the neighborhood matrix U should be supplied, and

"king"	"rook"	"bishop"
1 1 1	0 1 0	1 0 1
1 x 1	1 x 1	0 x 0
1 1 1	0 1 0	1 0 1

U = neighborhood matrix if available, in which case method = "given"
res.m = method by which pseudoresiduals are to be calculated:

method.r = 1: $\hat{e}_{i_1,i_2} = \left(2Y_{i_1,i_2} - Y_{i_1+1,i_2} - Y_{i_1,i_2+1}\right)/\sqrt{6}$

method.r = 2: $\hat{e}_{i_1,i_2} = \left(4Y_{i_1,i_2} - \left(Y_{i_1+1,i_2} + Y_{i_1-1,i_2} + Y_{i_1,i_2+1} + Y_{i_1,i_2-1}\right)\right)/\sqrt{20}$

It is set to 1 by default.

sigma.m = method by which noise variance estimate is to be obtained:

sigma.m = "s": $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} \hat{e}_i^2$

sigma.m = "t": $\hat{\sigma}^2 = (t_{75\%} - t_{25\%})/1.35$

It is set to "s" by default.

pic = T/F depending on whether the graphical output wanted or not. *F by default.*

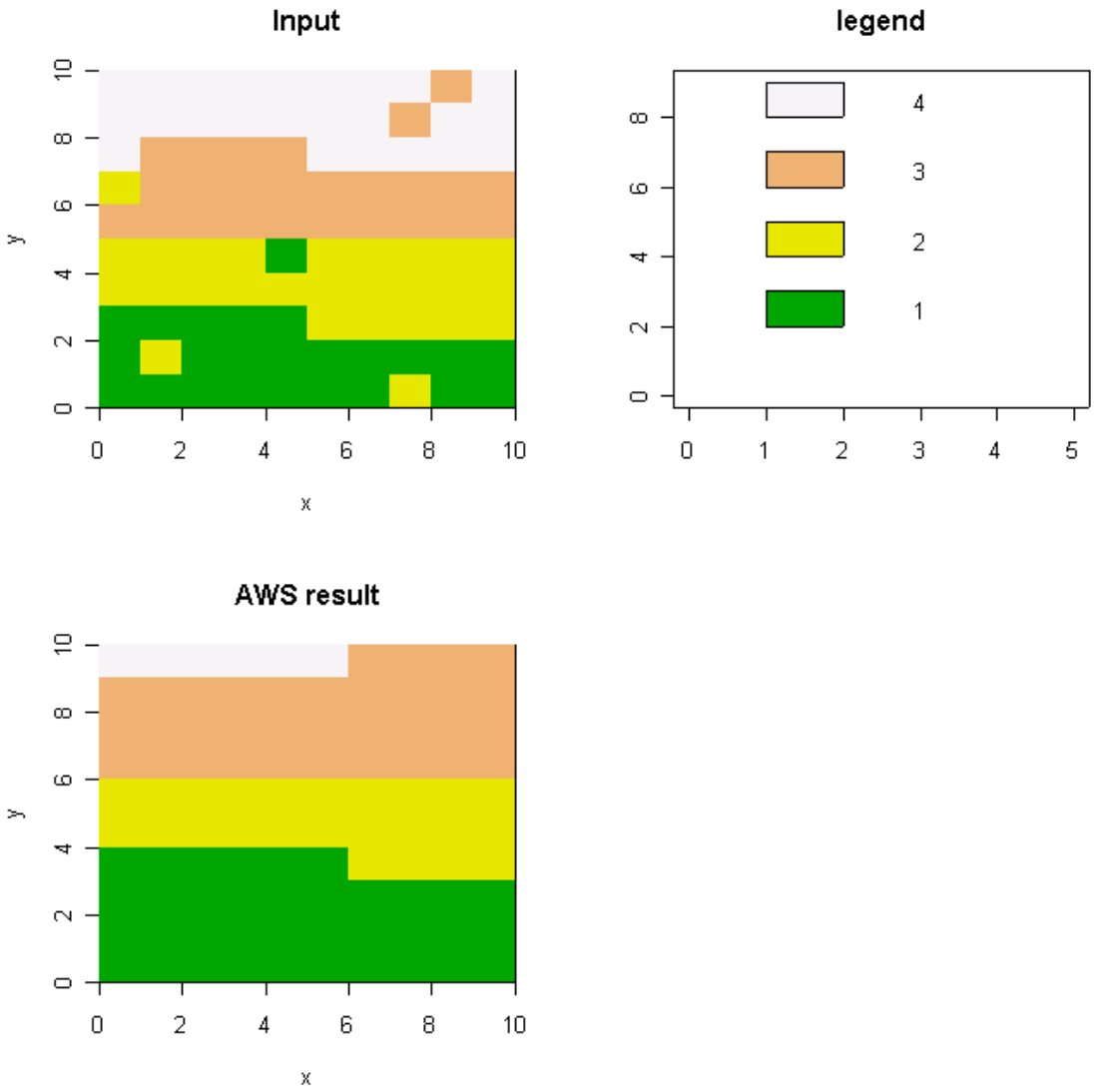
OUTPUT:

Y.est = the matrix of estimated values.
W = weights used in the final estimation. Used in bootstrap fine-tuning.

NOTES: The algorithm is very computationally intensive. It also uses large amounts of memory. For an array of size L with k* being the maximum amount of iterations, the amount of parameters grows to $2L(L+k^{*}+3)$.

EFFICIENCY:

EXAMPLE:



R-FUNCTION: **source.error.sum()**

DESCRIPTION: Calculates summary statistics (expected values and variances) for the source accuracy matrices.

FILE: A:/source_error_sum_fun.txt

FORMAT: source.error.sum(E,P,w)

PARAMETERS:

E = C*C*S array of source error matrices with C = number of categories and S = number of sources.
P = a multinomial probability vector of length C. *All equal probabilities by default*.
w = vector of (unnormalized) weights of length C. *All equal to 1 by default*.

OUTPUT:

E.acc.s = a vector of expected source accuracy for each source (length S).
V.acc.s = a vector of source accuracy variances for each source (length S).
E.acc.o = overall expected accuracy.
V.acc.o = variance of the overall accuracy.

NOTES:

EFFICIENCY:

EXAMPLE:

```
> E
, , 1

     [,1] [,2]
[1,]  0.9  0.1
[2,]  0.1  0.9

, , 2

     [,1] [,2]
[1,]  0.8  0.2
[2,]  0.2  0.8

> source.error.sum(E,c(.1,.9),)
$E.acc.s
[1] 0.9 0.8

$V.acc.s
[1] 0.09 0.16

$E.acc.o
[1] 0.85

$V.acc.o
[1] 0.0625

>
```

R-FUNCTION: ***AWS.3d.bootstrap()***

DESCRIPTION: A bootstrap method assisting in the choice of optimal parameters for the AWS procedure.

FILE: A:/AWS_3d_bootstrap_fun.txt

FORMAT: AWS.3d.bootstrap(Y,Cl,l.v,k.v,kernel,method,U,res.m,sigma.m,M,k.def,l.def).

PARAMETERS:

Y = a 2D array of observed levels.
Cl = number of levels.
l.v = vector of lambda values out of which the optimal value is to be chosen.
k.v = vector of k values out of which the optimal value is to be chosen.
kernel = type of kernel to use. “exp”, “uni” and “tri” types are implemented. See description of AWS.3d() function for details.
method = contiguity definition. See description of AWS.3d() for definition.
U = neighborhood matrix supplied if method = “given”.
res.m = residuals definition. See description of AWS.3d() for definition.
sigma.m = method of residual variance calculation. See description of AWS.3d() for definition.
M = number of bootstrap iterations.
k.def and l.def = values of k* and lambda to be used as default. I.e. the loss function is calculated with respect to the estimates for those parameters. k* = *19 and lambda = 4 by default.*

OUTPUT: A matrix of (quadratic) loss function values for different lambda*k combinations. The combination with the smallest values is recommended.

NOTES:

EFFICIENCY:

EXAMPLE:

R-FUNCTION: ***moran.i()***

DESCRIPTION: Calculates Moran's I, a spatial heterogeneity measure for a user defined process according to the desired method.

FILE: A:/moran_i_fun.txt

FORMAT: moran.i(Y,method,U)

PARAMETERS:

Y = the spatial process.
method = contiguity definition. "king" by default. If a neighborhood matrix can be provided, method = "given". See description of the **neighbor1()** for more information on available contiguity definitions.
U = neighborhood matrix if provided; undefined otherwise.

OUTPUT: Moran's I calculated for the given field and neighborhood structure. The value ranges from –1 to 0 to 1 indicating strong negative correlation, random pattern and strong positive correlation respectively.

NOTES: Supplied neighborhood matrix enables faster calculations, for example, in the case of loops.

EFFICIENCY:

EXAMPLE:

R-FUNCTION: ***neighbor()***

DESCRIPTION: Calculates neighborhood matrix for the given regular square grid with the specified measure of locality (radius).

FILE: A:/neighbor_fun.txt

FORMAT: neighbor(size.x,size.y,radius)

PARAMETERS:

size.x = width of the grid.
size.y = width of the grid.
radius = radius of neighborhood.

OUTPUT: Neighborhood matrix for the area of the dimensions (size.x*size.y,size.x*size.y).

NOTES:

EFFICIENCY:

EXAMPLE:

R-FUNCTION: ***naive1()***

DESCRIPTION: Performs naïve mode smoothing.

FILE: A:/naive1_fun.txt

FORMAT: neighbor(Y,mv,radius=1)

PARAMETERS:

Y = array, describing an image to smooth.
mv = missing value marker or class.
radius = radius of neighborhood, *equal to 1 by default. Varying radia not implemented yet!*

OUTPUT: Smoothed image: a matrix with the same dimensions as Y.

NOTES:

EFFICIENCY:

EXAMPLE:

Appendix 2: R-functions Listings

LIST OF FUNCTIONS:

rmulti()

rmulti0()

gen.random()

br2bin.acc.px()

br()

neighbor1()

AWS.3d

source.error.sum()

AWS.3d.bootstrap()

moran.i()

neighbor()

naive1

```
rmulti<-function(N,m,C,P=rep(1/C,C)){

# GENERATES OBSERVATIONS FROM MULTINOMIAL DISTRIBUTION
###############
# 02.08.2001 #
###############
# by Elena Moltchanova

#INPUT:
# N = size of the output array
# C = number of multinomial categories
# P = multinomial probability matrix (N*C) or vector (C)
# m = size of output vector (vector or scalar)

source("A:/rmulti0_fun.txt")

#setting up output array
Y<-array(dim=c(N,C))

if (length(m)==1){m<-rep(m,N)}
if (length(dim(P))==0){P<-t(array(rep(P,N),dim=c(C,N)))}

for (n in 1:N){
Y[n,]<-rmulti0(m[n],C,c(P[n,]))
}

return(Y)
}
```

```
rmulti0<-function(m,C,P=rep(1/C,C)){

# GENERATES SINGE OBSERVATION VECTOR FROM MULTINOMIAL DISTRIBUTION
###############
# 03.08.2001 #
###############
# by Elena Moltchanova

# INPUT:
# C = number of classes
# P = multinomial probability vector or matrix
# m = size of output vector

# calculating cumulative probabilities
cp<-round(cumsum(P),8)

#lower border
cp.low<-array(rep(c(0,cp[-length(cp)]),m),dim=c(C,m))
cp.upp<-array(rep(cp,m),dim=c(C,m))

#random numbers
rp<-array(rep(runif(m,0,1),rep(C,m)),dim=c(C,m))

# class belonging
Y<-apply((rp>cp.low)*(rp<=cp.upp),1,sum)

return(Y)
}
```

```
gen.random<-function(size.x,size.y,C,N,E=array(dim=c(C,C,N)),P=rep(1/C,C),pic=F){

# GENERATING RANDOM OBSERVATIONS ON A RECTANGULAR GRID
# C classes
# N sources
###############
# 02.08.2001 #
###############
# by Elena Moltchanova

# reading in additional functions

source("A:/rmulti_fun.txt")

#true category vector

X<-(rmulti(size.x*size.y,1,C,P))%*%c(1:C)

#sources

Y<-array(dim=c(size.x*size.y,N))

for (source in 1:N){

Y[,source]<-rmulti(size.x*size.y,1,C,t(E[,X,source]))%*%c(1:C)

}

X<-array(X,dim=c(size.x,size.y))
Y<-array(Y,dim=c(size.x,size.y,N))

if (pic==T){
#PLOT

#dividing the plotting area into a suitable number of screens:

x.fig<-ceiling(sqrt(N))
par(mfcol=c(x.fig,x.fig+1))

#plotting true source
image(c(0:size.x),c(0:size.y),X,main="True",xlab="x",ylab="y",col=terrain.colors(C))

#plotting legend
plot(0,0,xlim=c(0,5),ylim=c(0,(2*C+1)),xlab="",ylab="",main="legend",ty="n")
for (cl in 1:C){
polygon(c(1,2,2,1),c(2*cl,2*cl,2*cl+1,2*cl+1),col=terrain.colors(C)[cl])
text(3,2*cl+.5,cl)
}

#plotting observation sources
for (n in 1:N){
image(c(0:size.x),c(0:size.y),Y[,,n],main=c("source=",n),xlab="x",ylab="y",col=terrain.colors(C))
}
}

return(X,Y)
}
```

```
br2bin.acc.px<-function(E1=array(c(.5,.5,.5,.5),dim=c(2,2)),E2=array(c(.5,.5,.5,.5),
dim=c(2,2)),e1=E1,e2=E2,P=.5,pic=FALSE){

###############
# 24.07.2001 #
###############

#Evaluates theoretical accuracy of the Bayesian image restoration as a function of p

#calculate jump points

JP<-vector(length=4)
JP[1]<-e1[1,1]*e2[1,1]/(e1[1,1]*e2[1,1]+e1[2,1]*e2[2,1])
JP[2]<-e1[1,2]*e2[1,1]/(e1[1,2]*e2[1,1]+e1[2,2]*e2[2,1])
JP[3]<-e1[1,1]*e2[1,2]/(e1[1,1]*e2[1,2]+e1[2,1]*e2[2,2])
JP[4]<-e1[1,2]*e2[1,2]/(e1[1,2]*e2[1,2]+e1[2,2]*e2[2,2])

#eliminating doubles

JP<-JP[!is.na(JP)]

JP.order<-sort(JP)

#unit levels

U0<-vector(length=4)
U1<-vector(length=4)

U0[1]<-E1[1,1]*E2[1,1]*(1-P)
U0[2]<-E1[1,2]*E2[1,1]*(1-P)
U0[3]<-E1[1,1]*E2[1,2]*(1-P)
U0[4]<-E1[1,2]*E2[1,2]*(1-P)

U1[1]<-E1[2,1]*E2[2,1]*P
U1[2]<-E1[2,2]*E2[2,1]*P
U1[3]<-E1[2,1]*E2[2,2]*P
U1[4]<-E1[2,2]*E2[2,2]*P

#sorting U0 and U1 by JP

o<-order(JP)

U0<-U0[o]
U1<-U1[o]

levelf<-vector(length=5)
```

```
levelf[5]<-sum(U1[1:4])
levelf[4]<-sum(U1[1:3],U0[4])
levelf[3]<-sum(U1[1:2],U0[3:4])
levelf[2]<-sum(U1[1],U0[2:4])
levelf[1]<-sum(U0[1:4])

#maximum of the function

max.f<-max(levelf)

#the width of the interval for which maximum is reached

Jlength<-c(JP.order,1)-c(0,JP.order)

Jlength.max<-sum(Jlength[levelf==max(levelf)])

if(pic==TRUE){
#plot
plot(c(0:1),c(0:1),ty="n",xlim=c(0,1),ylim=c(0,1),xlab="prior probability p(x)",
ylab="accuracy", main="Expected Bayes Rule Accuracy")

segments(0,levelf[1],JP.order[1],levelf[1],lwd=2,col="darkgreen")
for (ss in 1:3){
segments(JP.order[ss],levelf[ss+1],JP.order[ss+1],levelf[ss+1],lwd=2,col="darkgreen")
}

segments(JP.order[4],levelf[5],1,levelf[5],lwd=2,col="darkgreen")

for (ss in 1:4){
segments(JP.order[ss],levelf[ss],JP.order[ss],levelf[ss+1],lwd=1,lty=2,col="darkgreen")
}}


return(JP.order,levelf,max.f,Jlength.max)
}
```

```
br<-function(Y1,C,e,p,pic=F){

###############
# 01.08.2001 #
###############

# by Elena Moltchanova

#######################################
#                                     #
# USING BAYESIAN TOTAL PROBABILITY FORMULA FOR IMAGE RESTORATION #
#######################################

size.x<-dim(Y1)[1]
size.y<-dim(Y1)[2]

Y<-array(Y1,dim=c(size.x*size.y,dim(Y1)[3]))

#############################################
# * NOTATION *                          #
# C=number of classes (1,...,C)           #
# N=number of sources                   #
# Y[1:L,1:N]=array of observations        #
# E[1:C,1:C,1:N]=array of source errors    #
# P[1:C,1:I]=vector of prior probabilities #
#############################################

#L number of observations
L<-dim(Y)[1]
N<-dim(Y)[2]

Ee<-array(dim=c(N,C,L))

for (n in 1:N){
Ee[n,,]<-E[,Y[,n],n]
}

Ee.sum<-exp(apply(log(Ee[1:N,,]),c(2,3),sum))

Ee.sumx<-apply(Ee.sum[1:C,],2,sum)

#array of posterior probabilities
P.post<-t(t(Ee.sum)/Ee.sumx)

#mode estimation
P.post.max<-t(t(P.post)==apply(P.post,2,max))
```

```
#dealing with multimodality

source("A:/rmulti_fun.txt")
PP<-t(P.post.max)/apply(P.post.max,2,sum)
Y.est<-rmulti(L,1,C,PP)%*%c(1:C)

Y.est<-array(Y.est,dim=dim(Y1)[1:2])

if (pic==TRUE){
#PLOT

#dividing the plotting area into a suitable number of screens:

x.fig<-ceiling(sqrt(N))
par(mfcol=c(x.fig,x.fig+1))

#plotting observation sources
for (n in 1:N){
image(c(0:size.x),c(0:size.y),Y1[,,n],main=c("source=",n),xlab="x",ylab="y",col=terrain.colors(C),
zlim=c(.5,C+.5))
}

#plotting estimated values
image(c(0:size.x),c(0:size.y),Y.est,main=c("BR Estimated pattern"),xlab="x",ylab="y",
col=terrain.colors(C),zlim=c(.5,C+.5))

#plotting legend
plot(0,0,xlim=c(0,5),ylim=c(0,(2*C+1)),xlab="",ylab="",main="legend",ty="n")
for (cl in 1:C){
polygon(c(1,2,2,1),c(2*cl,2*cl,2*cl+1,2*cl+1),col=terrain.colors(C)[cl])
text(3,2*cl+.5,cl)
}

}

P.post<-array(P.post,dim=c(dim(Y1)[1:2],C))

return(Y.est,P.post)

}
```

```
neighbor1<-function(size.x,size.y,method="king"){

###############
# 25.07.2001 #
#################################
# by Elena Moltchanova         #
# creating a neighborhood matrix #
#################################

x.coord<-rep(c(1:size.x),rep(size.y,size.x))
y.coord<-rep(c(1:size.y),size.x)

I<-size.x*size.y
U<-array(dim=c(I,I))

if (method=="king"){
U<-(abs(x.coord[col(U)]-x.coord[row(U)])<=1)*(abs(y.coord[col(U)]-y.coord[row(U)])<=1)*
(((x.coord[col(U)]==x.coord[row(U)])*(y.coord[col(U)]==y.coord[row(U)]))==0)
}

else if (method=="rook")
{
U<-(x.coord[col(U)]==x.coord[row(U)])*(abs(y.coord[col(U)]-y.coord[row(U)])==1)+
(abs(x.coord[col(U)]-x.coord[row(U)])==1)*(y.coord[col(U)]==y.coord[row(U)])

}
else if (method=="bishop")
{
U<-(abs(x.coord[col(U)]-x.coord[row(U)])==1)*(abs(y.coord[col(U)]-y.coord[row(U)])==1)
}

U<-array(U,dim=c(I,I))

return(U)
}
```

```
AWS.3d<-function(Y,Cl,lambda,nu,k.max,kernel,method,U,res.m=1,sigma.m="s",pic=F){

###############
# 06.08.2001 #
#########################
# by Elena Moltchanova #
########################################
# AWS FUNCTION FOR 3D MULTINOMIAL SPACE #
########################################

# GENERALIZED

size.x<-dim(Y)[1]
size.y<-dim(Y)[2]
L<-length(Y)

source("A:/neighbor_fun.txt")

k.max<-k.max+1

Y1<-c(Y)

#neighborhood

if (method!="given"){
source("A:/neighbor1_fun.txt")
U<-neighbor1(size.x,size.y,method)
}

#residuals:
res.w<-array(rep(0,size.x*size.y),dim=c(size.x,size.y))

if(res.m==1){
res<-c((2*Y-cbind(Y[,2:size.y],rep(0,size.x))-rbind(Y[2:size.x,],rep(0,size.y)))/
sqrt(res.w+6-(col(res.w)==size.x)-(col(res.w)==size.y)))
} else
{
res<-c((4*Y-(rbind(Y[2:size.x,],rep(0,size.y))+rbind(rep(0,size.y),Y[1:(size.x-1),])+
cbind(Y[,2:size.y],rep(0,size.x))+cbind(rep(0,size.x),Y[,1:(size.y-1)])))/
sqrt(res.w+20-(col(res.w)==1)-(col(res.w)==size.y)-(row(res.w)==1)-(row(res.w)==size.x)))
}

rm(res.w)

# sigma

if (sigma.m=="s"){
sigma<-c(res%*%res/length(res))
} else
{
# t-range estimation
```

```
sigma<-c((sort(res)[ceiling(length(res)*.75)]-sort(res)[floor(length(res)*.25)])/1.35)
}

# (a) Initialization

#weights matrics
W<-array(rep(1,L*L),dim=c(L,L))

f<-array(dim=c(k.max,L))
s<-array(dim=c(k.max,L))

f[1,]<-(U*W)%*%Y1/((U*W)%*%(rep(1,L)))

#rounding within the class boundaries
f[1,]<-apply(rbind(apply(rbind(round(f[1,]),rep(1,L)),2,max),rep(Cl,L)),2,min)

s[1,]<-sigma*(U*W)%*%t(W)%*%rep(1,L)/((U*W)%*%rep(1,L))^2

#correction for zero sigma
s[1,]<-s[1,]+.0001*(s[1,]==0)

#defining kernel function
if (kernel=="exp"){
kernel.fun<-function(x){
y<-exp(-x^2)
return(y)
}
} else{
if (kernel=="uni"){
kernel.fun<-function(x){
y<-(abs(x)<=1)
return(y)
}
} else{
if (kernel=="tri"){
kernel.fun<-function(x){
y<-(abs(x)<2)*(1-.5*x*sign(x))
return(y)
}}}}

if(k.max>1){
# (b) Adaptation

for (k in 1:(k.max-1)){

U<-neighbor(size.x,size.y,k+1)

#weights matrix
```

```
W<-array(kernel.fun((f[k,row(W)]-f[k,col(W)])/(lambda*sqrt(s[k,col(W)]))),dim=c(L,L))

f[k+1,]<-(U*W)%*%Y1/((U*W)%*%(rep(1,L)))
s[k+1,]<-sigma*(U*W)%*%t(W)%*%rep(1,L)/((U*W)%*%rep(1,L))^2

#note: estimates are rounded within class limits

f[k+1,]<-apply(rbind(apply(rbind(round(f[k+1,]),rep(1,L)),2,max),rep(Cl,L)),2,min)

#coorection for zero s
s[k+1,]<-s[k+1,]+.0001*(s[k+1,]==0)

# (c) Control

c.check<-sign(apply(t((-t(f[1:k,])+f[k+1,])/s[k+1,])>nu,2,sum))
f[k+1,]<-f[k,]*c.check+f[k+1,]*(1-c.check)

# (d) Stopping

if (sum(c.check[])==L){break}
}

#RESULTS

Y.est<-array(f[k+1,],dim=c(size.x,size.y))
}
else
{Y.est<-array(f[1,],dim=c(size.x,size.y))}
if (pic==T){
#PLOT
par(mfcol=c(2,2))

#plotting input data
image(c(0:size.x),c(0:size.y),Y,xlab="x",ylab="y",main="Input",col=terrain.colors(Cl),
zlim=c(.5,Cl+.5))

#plotting smoothed image
image(c(0:size.x),c(0:size.y),Y.est,xlab="x",ylab="y",main="AWS result",col=terrain.colors(Cl),
zlim=c(.5,Cl+.5))

#legend
plot(0,0,xlim=c(0,5),ylim=c(0,(2*Cl+1)),xlab="",ylab="",main="legend",ty="n")
for (cl in 1:Cl){
polygon(c(1,2,2,1),c(2*cl,2*cl,2*cl+1,2*cl+1),col=terrain.colors(Cl)[cl])
text(3,2*cl+.5,cl)
}}

return(Y.est,W)
}
```

```
source.error.sum<-function(E,P=rep(1/dim(E)[3],dim(E)[3]),w=rep(1,dim(E)[3]))
{

# 08.08.2001
# by Elena Moltchanova
# Summary statistics for source error matrices

#normalizing weights
w<-w/sum(w)

#calculating number of sources
S<-dim(E)[3]

#Calculating expected accuracy by source

E.acc.s<-vector(length=S)

for (s in 1:S){
E.acc.s[s]<-diag(E[,,s])%*%P[]
}

#variance by source
V.acc.s<-E.acc.s-E.acc.s^2

#overall expected source accuracy
E.acc.o<-sum(w*E.acc.s)
V.acc.o<-sum(w^2*V.acc.s)

return(E.acc.s,V.acc.s,E.acc.o,V.acc.o)
}
```

```
AWS.3d.bootstrap<-
function(Y,Cl,l.v,k.v,kernel,method,U,res.m=2,sigma.m="t",M,k.def=19,l.def=3){

###############
# 09.08.2001 #
###############
# By Elena Moltchanova

Y1<-c(Y)

# sources

source("A:/neighbor1_fun.txt")
source("A:/AWS_3d_fun.txt")

# neighborhood matrix

if(method!="given"){
U<-neighbor1(dim(Y)[1],dim(Y)[2],method)
}

# Fine-tunes AWS parameters through bootstrap

# (a) weights with default parameters
W<-AWS.3d(Y,Cl,l.def,4,k.def,kernel,method="given",U,res.m,sigma.m)$W
W1<-apply(W,1,sum)
W2<-apply(W^2,1,sum)

# (b) variance estimation on the basis of f

ff<-c(W%*%Y1/W1)
sig2<-c((Y1-ff)^2)%*%c(W1^2/(W2+W1^2-2*W1))/length(Y1)

# setting up arrays

f.boot<-array(dim=c(M,length(Y1)))
ksi<-array(dim=c(length(l.v),length(k.v)))

for (l.i in 1:length(l.v)){
for (k.i in 1:length(k.v)){
for (m in 1:M){
# (c) resampling

Y1<-ff+sqrt(sig2)*rnorm(ff,0,1)
Y1<-array(Y1,dim=dim(Y))

# (d) Parameter optimization

f.boot[m,]<-AWS.3d(Y,Cl,l.v[l.i],4,k.v[k.i],kernel,method="given",U,res.m,sigma.m)$Y.est
}

# loss function

ksi[l.i,k.i]<-sum((t(-t(f.boot[,])+ff))^2)/M
}}
return(ksi)
}
```

```
###############
# 23.07.2001 #
###############

# by Elena Moltchanova

# Calculating Moran's I to measure autocorrelation

moran.i<-function(Y,method="king",U){

# where
# Y = the data matrix, containing values
# method="king", "rook" or "bishop" describes the neighborhood
# method="given" allows to use externally calculated neighborhood matrix

#calculating the weight matrix:

if (method!="given"){
source("H:/r-routines/neighbor1_fun.txt")
U<-neighbor1(dim(Y)[1],dim(Y)[2],method)
}
x<-c(Y)-mean(Y)
W<-array(U,dim=c(length(Y),length(Y)))

I<-(t(x)%*%W%*%x)/(t(x)%*%x)*length(x)/sum(W)

return(I)
}
```

```
neighbor<-function(size.x,size.y,radius=1){

Y<-array(dim=c(size.x,size.y))

x.coord<-array(rep(c(col(Y)),size.x*size.y),dim=c(size.x*size.y,size.x*size.y))
y.coord<-array(rep(c(row(Y)),size.x*size.y),dim=c(size.x*size.y,size.x*size.y))

U<-(abs(x.coord-t(x.coord))<=radius)*(abs(y.coord-t(y.coord))<=radius)*(1-
(x.coord==t(x.coord))*
(y.coord==t(y.coord)))
return(U)
}
```

```
naive1<-function(Y,mv=1,radius=1){
# 27.08.2001
# by Elena Moltchanova

# naive smoothing
I<-dim(Y)[1]
J<-dim(Y)[2]

W<-array(rep(0,I*J),dim=dim(Y))
#statistical mode function:
statmod<-function(x){
statmod0 <- function(x) {
  z <- table(as.vector(x))
  names(z)[z == max(z)]
}
Z<-as.numeric(statmod0(x))

return(Z)
}

W<-array(dim=dim(W))

for (i in 2:(I-1)){
for (j in 2:(J-1)){
xx<-c(Y[(i-1):(i+1),(j-1):(j+1)])
if(length(xx[xx!=mv])>0){W[i,j]<-statmod(xx[xx!=mv])[1]}else{W[i,j]<-mv}
}}

#borders
for (i in 2:(I-1)){
xx<-c(Y[(i-1):(i+1),1:2])
if(length(xx[xx!=mv])>0){W[i,1]<-statmod(xx[xx!=mv])[1]}else{W[i,1]<-mv}
xx<-c(Y[(i-1):(i+1),(J-1):J])
if(length(xx[xx!=mv])>0){W[i,J]<-statmod(xx[xx!=mv])[1]}else{W[i,J]<-mv}
}

for (j in 2:(J-1)){
xx<-c(Y[1:2,(j-1):(j+1)])
if(length(xx[xx!=mv])>0){W[1,j]<-statmod(xx[xx!=mv])[1]}else{W[1,j]<-mv}
xx<-c(Y[(I-1):I,(j-1):(j+1)])
if(length(xx[xx!=mv])>0){W[I,j]<-statmod(xx[xx!=mv])[1]}else{W[I,j]<-mv}
}

#corners
xx<-c(Y[1:2,1:2])
if(length(xx[xx!=mv])>0){W[1,1]<-statmod(xx[xx!=mv])[1]}else{W[1,1]<-mv}
xx<-c(Y[(I-1):I,1:2])
if(length(xx[xx!=mv])>0){W[I,1]<-statmod(xx[xx!=mv])[1]}else{W[I,1]<-mv}
xx<-c(Y[1:2,(J-1):J])
```

```
if(length(xx[xx!=mv])>0){W[1,J]<-statmod(xx[xx!=mv])[1]}else{W[1,J]<-mv}
xx<-c(Y[(I-1):I,(J-1):J])
if(length(xx[xx!=mv])>0){W[I,J]<-statmod(xx[xx!=mv])[1]}else{W[I,J]<-mv}

#missing stay missing
W<-W*(Y!=mv)+mv*(Y==mv)

return(W)
}
```