# Efficient Implementation of the Fuzzy $c$-Means Clustering Algorithms

ROBERT L. CANNON, JITENDRA V. DAVE, AND JAMES C. BEZDEK

*Abstract*—**This paper reports the results of a numerical comparison of two versions of the fuzzy $c$-means (FCM) clustering algorithms. In particular, we propose and exemplify an approximate fuzzy $c$-means (AFCM) implementation based upon replacing the necessary "exact" variates in the FCM equation with integer-valued or real-valued estimates. This approximation enables AFCM to exploit a lookup table approach for computing Euclidean distances and for exponentiation. The net effect of the proposed implementation is that CPU time during each iteration is reduced to approximately one sixth of the time required for a literal implementation of the algorithm, while apparently preserving the overall quality of terminal clusters produced. The two implementations are tested numerically on a nine-band digital image, and a pseudocode subroutine is given for the convenience of applications-oriented readers. Our results suggest that AFCM may be used to accelerate FCM processing whenever the feature space is comprised of tuples having a finite number of integer-valued coordinates.**

*Index Terms*—**Approximate FCM, cluster analysis, efficient implementation, fuzzy $c$-means, image processing, pattern recognition.**

## I. Introduction

CLUSTERING algorithms can be loosely categorized by principle (objective function, graph-theoretical, hierarchical) or by model type (deterministic, statistical, fuzzy). This paper concerns itself with an infinite family of fuzzy objective function clustering algorithms which are usually called the fuzzy $c$-means algorithms. For brevity, in the sequel we abbreviate fuzzy $c$-means as FCM. A special case of the FCM algorithm was first reported by Dunn [11] in 1972. Dunn's algorithm was subsequently generalized by Bezdek [3], Gustafson and Kessel [14], and Bezdek *et al.* [6]. Related algorithms and indirect generalizations of various kinds have been reported by, among others, Granath [13], Full *et al.* [12], and Anderson and Bezdek [1]. Applications of FCM to problems in clustering, feature selection, and classifier design have been reported in geological shape analysis [9], medical diagnosis [2], image analysis [10], irrigation design [8], and automatic target recognition [5]. These references manifest a nice progression of both theory and application of FCM over the last decade. Successes notwithstanding, many unanswered questions concerning these algorithms remain. Among these, one of the most frequent *operational* complaints about FCM is that it may consume—for large data sets—high amounts of CPU time (this holds, e.g., *in spite of* the fact that FCM is, for certain problems, itself several orders of magnitude faster than, say, maximum likelihood iteration [7]). With this as background, the scope of the present paper is stated simply: What means of accelerating computation time in the FCM loop can be tried? And how effective are these revised implementations?

Section II contains a brief description of the basic FCM algorithm. Section III presents a table lookup implementation of the basic FCM algorithm, called AFCM, and discusses the two approximations and six lookup tables that comprise AFCM. Experimental numerical results comparing a literal implementation, LFCM, to AFCM are given in Section IV. Section V contains a discussion of our conclusions.

Manuscript received September 6, 1984; revised August 30, 1985. The work of R. L. Cannon was supported in part by the National Science Foundation under Grant ECS-8217191 and by the IBM Palo Alto Scientific Center. The work of J. C. Bezdek was supported in part by the National Science Foundation under Grant IST-8407860.

R. L. Cannon and J. C. Bezdek are with the Department of Computer Science, University of South Carolina, Columbia, SC 29208.

J. V. Dave is with the IBM Palo Alto Scientific Center, Palo Alto, CA 94304.

IEEE Log Number 8406217.

## II. The FCM Algorithm

Let $\boldsymbol{R}$ be the set of reals, $\boldsymbol{R}^p$ the set of $p$ tuples of reals, $\boldsymbol{R}^+$ the set of nonnegative reals, and $W_{cn}$ the set of real $c \times n$ matrices. $\boldsymbol{R}^p$ will be called *feature space*, and elements $\boldsymbol{x} \in \boldsymbol{R}^p$ *feature vectors*; feature vector $\boldsymbol{x} = (x_1, x_2, \cdots, x_p)$ is composed of $p$ real numbers.

*Definition:* Let $X$ be a subset of $\boldsymbol{R}^p$. Every function $u : X \rightarrow [0, 1]$ is said to assign to each $\boldsymbol{x} \in X$ its *grade of membership* in the *fuzzy set* $u$. The function $u$ is called a *fuzzy subset* of $X$.

Note that there are infinitely many fuzzy sets associated with the set $X$. It is desired to "partition" $X$ by means of fuzzy sets. This is accomplished by defining several fuzzy sets on $X$ such that for each $\boldsymbol{x} \in X$, the sum of the fuzzy memberships of $\boldsymbol{x}$ in the fuzzy subsets is one.

*Definition:* Given a finite set $X \subseteq \boldsymbol{R}^p$, $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$ and an integer $c$, $2 \leq c \leq n$, a *fuzzy $c$ partition* of $X$ can be represented by a matrix $U \in W_{cn}$ whose entries satisfy the following conditions.

1) Row $i$ of $U$, say $U_i = (u_{i1}, u_{i2}, \cdots, u_{in})$ exhibits the $i$th membership function (or $i$th fuzzy subset) of $X$.

2) Column $j$ of $U$, say $U^j = (u_{1j}, u_{2j}, \cdots, u_{cj})$ exhibits the values of the $c$ membership functions of the $j$th datum in $X$.

3) $u_{ik}$ shall be interpreted as $u_i(\boldsymbol{x}_k)$, the value of the

membership function of the $i$th fuzzy subset for the $k$th datum.

4) The sum of the membership values for each $x_k$ is one (column sum $\Sigma_i u_{ik} = 1 \; \forall \; k$).

5) No fuzzy subset is empty (row sum $\Sigma_k u_{ik} > 0 \; \forall \; i$).

6) No fuzzy subset is all of $X$ (row sum $\Sigma_k u_{ik} < n \; \forall \; i$).

$M_{fc}$ will denote the set of fuzzy $c$ partitions of $X$. The special subset $M_c \subseteq M_{fc}$ of fuzzy $c$ partitions of $X$ wherein every $u_{ik}$ is 0 or 1 is the discrete set of "hard," i.e., non-fuzzy $c$ partitions of $X$. $M_c$ is the solution space for conventional clustering algorithms.

The fuzzy $c$-means algorithm uses iterative optimization to approximate minima of an objective function which is a member of a family of fuzzy $c$-means functionals using a particular inner product norm metric as a similarity measure on $\boldsymbol{R}^p \times \boldsymbol{R}^p$. The distinction between family members is the result of the application of a weighting exponent $m$ to the membership values used in the definition of the functional.

*Definition:* Let $U \in M_{fc}$ be a fuzzy $c$ partition of $X$, and let $\boldsymbol{v}$ be the $c$ tuple $(\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_c)$, $\boldsymbol{v}_i \in \boldsymbol{R}^p$. The *fuzzy c-means functional* $J_m : M_{fc} \times \boldsymbol{R}^{cp} \rightarrow \boldsymbol{R}^+$ is defined as

$$J_m(U, \boldsymbol{v}) = \sum_{k=1}^{n} \sum_{i=1}^{c} (u_{ik})^m \, (d_{ik})^2$$

where

$$U \in M_{fc}$$

is a fuzzy $c$ partition of $X$;

$$\boldsymbol{v} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_c) \in \boldsymbol{R}^{cp}$$

with $\boldsymbol{v}_i \in \boldsymbol{R}^p$ the cluster center or prototype of class $i$, $1 \leq i \leq c$, and

$$d_{ik}^2 = ||\boldsymbol{x}_k - \boldsymbol{v}_i||^2,$$

$||\cdot||$ any inner product norm metric, and $m \in [1, \infty)$.

Note the several parameters in the definition of the fuzzy $c$-means functional. The squared distance is weighted by the $m$th power of the membership of datum $\boldsymbol{x}$ in cluster $i$. Thus, $J_m$ is a squared error criterion, and its minimization produces fuzzy clusters (matrix $U$) that are optimal in a generalized least squared errors sense.

The FCM algorithm, via iterative optimization of $J_m$, produces a fuzzy $c$ partition of the data set $X = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$. The basic steps of the algorithm are given as follows (cf. [4] for the derivation).

1) Fix the number of clusters $c$, $2 \leq c < n$ where $n$ = number of data items. Fix $m$, $1 < m < \infty$. Choose any inner product induced norm metric $||\cdot||$, e.g.,

$$||\boldsymbol{x} - \boldsymbol{v}||_A^2 = (\boldsymbol{x} - \boldsymbol{v})^T A (\boldsymbol{x} - \boldsymbol{v}),$$

$A \in W_{pp}$ positive definite.

2) initialize the fuzzy $c$ partition $U^{(0)}$,

3) at step $b$, $b = 0, 1, 2, \cdots$,

4) calculate the $c$ cluster centers $\{\boldsymbol{v}_i^{(b)}\}$ with $U^{(b)}$ and the formula for the $i$th cluster center:

$$v_{il} = \frac{\sum_{k=1}^{n} (u_{ik})^m \, x_{kl}}{\sum_{k=1}^{n} (u_{ik})^m}, \qquad l = 1, 2, \cdots, p. \tag{1}$$

5) Update $U^{(b)}$: calculate the memberships in $U^{(b+1)}$ as follows. For $k = 1$ to $n$,

a) calculate $I_k$ and $\tilde{I}_k$:

$$I_k = \{i \,|\, 1 \leq i \leq c, \, d_{ik} = ||\boldsymbol{x}_k - \boldsymbol{v}_i|| = 0\},$$

$$\tilde{I}_k = \{1, 2, \cdots, c\} - I_k;$$

b) for data item $k$, compute new membership values:

i) if $I_k = \phi$,

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left( \frac{d_{ik}}{d_{jk}} \right)^{2/(m-1)}} \tag{2}$$

ii) else $u_{ik} = 0$ for all $i \in \tilde{I}_k$ and $\Sigma_{i \in I_k} u_{ik} = 1$;

next $k$.

6) Compare $U^{(b)}$ and $U^{(b+1)}$ in a convenient matrix norm; if $||U^{(b)} - U^{(b+1)}|| < \epsilon$, stop; otherwise, set $b = b + 1$, and go to step 4).

Use of the FCM algorithm requires determination of several parameters, i.e., $c$, $\epsilon$, $m$, the inner product norm $||\cdot||$, and a matrix norm. In addition, the set $U^{(0)}$ of initial cluster centers must be defined. Although no theoretical basis for choosing a good value of $m$ is available, $1.1 \leq m \leq 5$ is typically reported as the most useful range of values. Further details of computing protocols, empirical examples, and computational subtleties are summarized elsewhere, cf. [4]. The point of attack below is to reduce the computational burden imposed by iterative looping between (1) and (2) when $c$, $p$, and $n$ are large. It is this task to which we now turn.

## III. The AFCM Implementation

In this section, we develop AFCM as a subroutine to implement an approximation of the FCM algorithm. Six internal tables and approximations of several variables in the FCM algorithm are required. Section III-A describes the environment in which we use AFCM and the assumptions we have made about the tables passed to it. Section III-B describes the internal tables used. AFCM is presented as a subroutine in Section III-C.

### A. The External Tables

The developmental environment for the proposed implementation was that of interactive digital image processing. For this study, our images were $256 \times 256$ pixels, thus making $n = 65536$. Larger values of $n$, say 1 Mbyte, are easily encountered in imaging applications. Our values for $p$ have been as large as 11, and for $c$ as large as 16. In this context, $p$ is the number of (spectral) layers in multilayered digital data.

Each pixel vector $\boldsymbol{x}$ of such data has features that lie in the discrete set $J_{GL} = \{0, 1, 2, \cdots, n_{GL}\}$ where $n_{GL}$ is

the number of gray levels to which the data are resolved. For our application, $n_{GL} = 255$; thus, the data array $X$ can be represented as an array of $n \times p$ 1-byte integers.

In the discussion which follows, it is necessary to distinguish among "real" algorithmic triples $(\boldsymbol{v}, u, \{d_{ik}\})$, their *approximations* ($\tilde{\boldsymbol{v}}$, $\tilde{U}$, and $\{\tilde{d}_{ik}\}$), and their *realizations* as array elements ($[V]$, $[U]$, and $[DK]$) in implementations of the approximation.

In order to give AFCM the ability to adjust to cluster centers in a smooth manner and to converge to the same approximate values as a literal implementation of the FCM algorithm (LFCM), we have chosen to multiply them by 10 and round them to the nearest integer. Thus, the cluster centers may be represented by an array $[V]$ of $c \times p$ 2 byte integers. This approximation of the real $\boldsymbol{v}_i$ by $\tilde{\boldsymbol{v}}_i$ rounded to one decimal place is the first implementational deviation from FCM, and it demands the distinction made by calling our modified implementation AFCM. It should be noted that using these estimates for the $\boldsymbol{v}_i$ abrogates the *necessity* of (1) and (2); as we shall see, however, the approximation to $v_{il} \in \boldsymbol{R}$ by $\tilde{v}_{il}$ does not seriously affect the path of iterates traced through $M_{fc} \times R^{cp}$ by AFCM.

The fuzzy memberships $u_{ik}$ are in the interval [0.0, 1.0], so the values of the $u_{ik}$ are real. We choose to use a three decimal place approximation of $u_{ik}$ and multiply that value by 1000, so our approximations $\tilde{u}_{ik}$ of $u_{ik}$ are in [0, 1000]. Thus, we can represent the approximate fuzzy membership matrix $\tilde{U}$ by an array $[U]$ consisting of $c \times n$ 2-byte integers.

AFCM will be presented as a subroutine which has passed an $n \times p$ array $[X]$ of data, a $c \times p$ array $[V]$ of initial cluster centers, and a $c \times n$ array $[U]^{(0)}$. The subroutine updates the array $[V]$ with the cluster centers *after convergence* and also returns the array $[U]$ of fuzzy memberships of the data with respect to the final set of cluster centers.

To summarize, we have replaced a necessary pair $(U, \boldsymbol{v})$ for $J_m$ via (1) and (2) with an *approximately* necessary pair $(\tilde{U}, \tilde{\boldsymbol{v}})$. The net savings in storage and processing time are traded against the loss of true necessity. The loss of necessity is traded for economies of storage and CPU time. The numerical examples presented below imply that AFCM terminates at roughly the same place as LFCM, so these economies seem to be realized at no sacrifice in the quality of LFCM estimates.

### B. The Internal Tables

A literal implementation of the FCM algorithm can be very expensive computationally. As an example, in (1), for each value of $i$ and $l$, there are $n$ exponentiations in the denominator and $n$ exponentiations and products in the numerator. A straightforward transliteration of the algorithm would require $c \times p \times n$ uses of the exponentiation operators and $c \times p$ divisions. Because the algorithm is iterative, these $\boldsymbol{v}_i$ may be calculated repeatedly.

Three equations can be identified as making significant contributions to the overall time requirements of the algorithm. The first is the calculation of $d_{ik}$, the distance of $\boldsymbol{x}_k$ from $\boldsymbol{v}_i$. We have chosen as $||\cdot||$ the Euclidean norm ($A = I$, the $p \times p$ identity matrix). Thus,

$$d_{ik} = \sqrt{\sum_{l=1}^{p} (x_{kl} - v_{il})^2}. \tag{3}$$

Secondly, the updated value of the fuzzy membership matrix $[u_{ik}]$ is given in the general case by (2). The third is (1), the expression for the cluster centers.

The goal of our table lookup approach is to eliminate the use of exponentiation operators except in the initialization stage when the tables are constructed. Likewise, the number of divisions is reduced by subtracting logarithms which are obtained by referencing a table of logarithms followed by a reference to a table of exponentials for the antilogarithm. Of course, great amounts of space can be required for tables. We have made assumptions about the nature of the data and the accuracy of intermediate results such that we can contain the size of tables within reasonable limits. Of our six internal tables, two are 2-byte integers, two are 4 byte integers, and the other two are 4-byte reals. The use of lookup tables constitutes another major departure from LFCM.

Because the use of such tables may enhance implementation of the FCM algorithm in other environments (and possibly enhance other algorithms as well), we present here a description of the tables used by AFCM.

*1) $d_{ik}$:* In (3), the expression for $d_{ik}$, for each iteration there are $c \times n$ computations of the square root of a number obtained by $p$ differences, $p$ squares, and $p - 1$ additions. Two tables are used here. In defining the tables, we use a pseudolanguage in which we will present the algorithm.

Given that $V[i, l]$ is ten times the $l$th dimension (band) of the $i$th cluster center, define TABLEA as

$$\text{TABLEA}[i, l, y] = 0.01 \times (10 \times y - V[i, l])^2,$$

$1 \le i \le c$, $1 \le l \le p$, $0 \le y \le 255$. Thus, for each of the possible values of a datum $y$, TABLEA contains the square of the difference between that value $y$, which is an integer, and the coordinates of the cluster centers, which are reals. These values may be as large as $255^2$, so TABLEA consists of 4-byte reals. Also, as the cluster centers are updated on each iteration of AFCM, TABLEA must be recomputed on each iteration.

The reader has perhaps noted from (2) and (3) that (2) can be slightly modified to avoid repeated computation of the square root of real numbers. This strategy, although meaningful in a literal implementation of FCM, results in a very large increase in the size of the subsequent tables (from 255.0 to $p \times 255.0^2$ in our case). To reduce storage, therefore, it is advisable to obtain square roots and to work with the normalized form $\tilde{d}_{ik}$.

The second table is useful for computing the square root of the sum over $l$ of values from TABLEA. One decimal place accuracy will be achieved by multiplying square roots by 10 and storing them as integers. TABLEB is given as

$$\text{TABLEB}[y] = \text{round}(10\sqrt{y}),$$

$0 \le y \le 10\ 000$. If $y$ is in [0, 10 000], then $10\sqrt{y}$ is in [0, 1000], and can be represented by a 2 byte integer. The upper bound of 10 000 is determined somewhat arbitrarily because it is not advisable to keep a table of square roots for all integer values up to $255^2$ as this may lead to excessive paging in a virtual memory environment. For values larger than 10 000, square roots will be computed by a library subroutine; our assumption is that most distances will be less than 100.

In AFCM, we compute for each $k$, $1 \le k \le n$ the distance of the $k$th datum from each of the $i$ cluster centers and place it into the array $DK$, so we define

$$DK[i] = \text{TABLEB}\left[\frac{\sum_{l=1}^{p} \text{TABLEA}[i, l, X[k, l]]}{p}\right]$$

$$= \frac{10\sqrt{\sum_{l=1}^{p} (X[k, l] - V[i, l])^2}}{\sqrt{p}},$$

$$i = 1, 2, \cdots, c.$$

Thus, for a given value of $k$, say $k = q$, $DK[i] = 10\tilde{d}_{iq}/\sqrt{p}$. Because distances need to be calculated for only one datum at a time and $\tilde{d}_{ik}$ is in $[0.0, 255.0\sqrt{p}]$, $DK[i]$ is in [0, 2550], and can be a (one-dimensional) array of $c$ 2-byte integers.

*2) $u_{ik}$:* Two tables are used in the calculation of $u_{ik}$, given in the general case by (2). The first is given as

$$\text{TABLEC}[y] = \text{round}((20\ 000/(m - 1)) \times \log(0.1 \times y)),$$

$0 \le y \le 2550$. Values from TABLEC give logarithms which can be subtracted in order to perform the division of $\tilde{d}_{ik}$ by $\tilde{d}_{jk}$. Then we can define

$$\text{TABLED}[y] = 10^{(0.0001 \times y)}.$$

The bounds for $y$ will be discussed after further motivation. Thus,

$$U[i, k] = \frac{1000}{\sum_{j=1}^{c} \text{TABLED}[\text{TABLEC}[DK[i]] - \text{TABLEC}[DK[j]]]}$$

$$= \frac{1000}{\sum_{j=1}^{c} 10^{0.0001(\log((0.1 \times DK[i]/0.1 \times DK[j])^{20000/m-1}))}}$$

$$= 1000 \frac{1}{\sum_{j=1}^{c} \left(\frac{0.1 \times DK[i]}{0.1 \times DK[j]}\right)^{2/(m-1)}}.$$

Thus, $U[i, k] = 1000 \times \tilde{u}_{ik}$, $1 \le i \le c$, $1 \le k \le n$.

If we assume $m \ge 1.01$, then TABLEC will contain a minimum value of $-2\ 000\ 000$ and a maximum value of 4 813 030. Thus, TABLEC must be composed of 4-byte integers. The aforementioned range of TABLEC suggests an extremely large ($\approx 10^6$) range for the argument of TABLED. However, our ultimate goal in these computations is the evaluation of the sum in the denominator of (2). If the argument of TABLED is very small, i.e., a very large negative number, that particular term contributes very little to the sum. Assuming that a contribution of less than 0.001 is too small to deserve any further consideration, the lowest limit for TABLED is set at $10^{-3}$, i.e., the lowest permissible value of $y$ is taken as $-30\ 000$. If the argument of TABLED is very large, the sum of all the terms of the series will also be large, leading to a very small ($<0.001$) value of $\tilde{u}_{ik}$. Thus, if a condition arises where the argument of TABLED is greater than 30 000, the corresponding value of $\tilde{u}_{ik}$ can be set to zero without any further calculation. Based upon these arguments, we restrict $y$ to the range $-30\ 000 \le y \le 30\ 000$ or $-10\ 000 \log(1000) \le y \le 10\ 000 \log(1000)$. Furthermore, TABLED values should be stored as reals, as they are to be used for the generation of a floating-point sum. Although not shown in Fig. 1, adequate safeguards must be taken such that references to TABLED are within the proper array bounds, as assumed earlier.

*3) $v_i$:* TABLEE and TABLEF are used for calculation of $v_i$, as given by (1). $(\tilde{u}_{ik})^m$ is represented by a logarithm in TABLEE and $x_{kl}$ by a logarithm in TABLEF.

TABLEE is defined as

$$\text{TABLEE}[y] = \text{round}((10\ 000 \times m) \times \log(0.001 \times y)),$$

$1 \le y \le 1000$. TABLEF is defined by

$$\text{TABLEF}[y] = \text{round}(10\ 000) \times \log(y)),$$

$1 \le y \le 255$. In the program, we compute

$$V[i, l] = \frac{10 \sum_{k=1}^{n} \text{TABLED}[\text{TABLEE}[U[i, k]] + \text{TABLEF}[X[k, l]]]}{\sum_{k=1}^{n} \text{TABLED}[\text{TABLEE}[U[i, k]]]}$$

$$= \frac{10 \sum_{k=1}^{n} 10^{0.0001(\log((0.001 \times U[i,k])^{10000m}) + \log(X[k,l]^{10000}))}}{\sum_{k=1}^{n} 10^{0.0001(\log((0.001 \times U[i,k])^{10000m}))}}$$

$$= \frac{10 \sum_{k=1}^{n} (0.001 \times U[i, k])^m X[k, l]}{\sum_{k=1}^{n} (0.001 \times U[i, k])^m}.$$

But, as $U[i, k] = 1000 \times \tilde{u}_{ik}$, then

```
procedure afcm(X: array of data; var V: array of cluster centers;
    var U: array of fuzzy memberships; m: real; n, p, c, maxiter,
        epsilon: integer)
begin
    initialize tables B, C, D, E, F;
    unorm := 1000;
    iter := 1;
    repeat
        {update the U matrix}
        initialize TABLEA for current set of cluster centers;
        for k := 1 to n do begin
            numik := 0;     {number of elements in I_k}
            for i := 1 to c do begin
                DK[i] := TABLEB[round(sum(TABLEA[i, l, X[k, l]],
                        1≤l≤p) / p)];
                if DK[i] = 0 then numik := numik+1
                end;
            if numik ≠ 0 then for i := 1 to c do begin
                if DK[i] = 0 then U[i,k] := round(1000/numik);
                else U[i,k] := 0;
                update unorm
                end
            else for i := 1 to c do begin
                U[i,k] :=
                    round(1000/sum(TABLED[TABLEC[DK[i]]
                            -TABLEC[DK[j]]], 1≤j≤c));
                update unorm
                end
            end;
        {compute cluster centers on all but first iteration}
        if iter > 1 then begin
            for i := 1 to c do begin
                sumu := TABLED[sum(TABLEE[U[i,k]], 1≤k≤n)];
                for l := 1 to p do begin
                    sumux := TABLED[sum(TABLEE[U[i,k]]
                            + TABLEF[X[k,l]], 1≤k≤n)];
                    V[i,l] := round(10.0*summux)/sumu);
                    end
                end;
    iter := iter+1
    until (unorm < epsilon) or (iter > maxiter)
end;
```

Fig. 1. The AFCM implementation.

$$V[i, l] = \frac{10 \sum_{k=1}^{n} (\tilde{u}_{ik})^m x_{kl}}{\sum_{k=1}^{n} (\tilde{u}_{ik})^m} = 10 \times \tilde{v}_{il}.$$

The values of TABLEE are integers in $[-30\ 000m, 0]$. The values of TABLEF are integers in [0, 24 065]. TABLEE must be represented by 4-byte integers, while TABLEF can be represented by 2-byte integers.

### C. The Subroutine

Fig. 1 presents AFCM in pseudocode. Translation to languages such as Fortran and C should not be difficult. It is assumed that the user has determined some initial set of cluster centers and has initialized the cluster center array before invoking the subroutine.

Since the matrix norm for convergence testing must be specified, we have chosen to use the sup norm on $W_{cn}$, viz.

$$||\tilde{U}^{(b)} - \tilde{U}^{(b+1)}|| = \max_{i,k} \{|\tilde{u}_{ik}^{(b)} - \tilde{u}_{ik}^{(b+1)}|\}.$$

The value of $\epsilon$ such that

$$||\tilde{U}^{(b)} - \tilde{U}^{(b+1)}|| < \epsilon \tag{4}$$

is supplied by the user as a parameter to the subroutine.

We note that a substantial savings in both storage and computation time can be effected by comparing cluster centers instead of partitions. Thus,

$$||\tilde{\boldsymbol{v}}^{(b)} - \tilde{\boldsymbol{v}}^{(b+1)}|| = \max_{i,l} \{|\tilde{v}_{il}^{(b)} - \tilde{v}_{il}^{(b+1)}|\} < \epsilon$$

reduces the overall calculations and storage needed in (4) from $c \times n$ to $c \times p$. This is an effective reduction of roughly $n:p \approx 64\ 000:16 = 4000:1$ for data on the order of one 16-band 256 × 256 image. This economy, however, may lead to termination at different fixed points than does (4). Since the relative accuracy of these termination criteria has not been reported, we leave this point to a future investigation.

## IV. Experimental Results

Two versions of the fuzzy $c$-means algorithm were coded and tested with test data in order to determine speed of execution, number of iterations to convergence, and accuracy of results. One implementation, LFCM, was a literal transliteration of the algorithm. The other, AFCM, was the table lookup approach described in Section III. Both programs were written in Fortran and were identical except in their implementation of the FCM equations as delineated above.

The three-screen experimental image processing environment of DIMAPSAR [16] allowed interactive display of the original image and the results of each iteration (cluster centers, fuzzy-membership-matrix histogram and image for every cluster, segmented image, etc.). The host processor was an IBM 3081 with 16 Mbytes of memory. Virtual machines of 5 and 7 Mbytes were required for execution of AFCM and LFCM, respectively.

### A. Test Data

The test data sets were nine-band aircraft flight data showing a region of Oklahoma. A 256 × 256 pixel region was selected as the subject. AFCM was used to segment the region into ten clusters, using the brightness values as feature vectors. Each pixel was assigned to the "class" associated with the cluster in which its fuzzy membership was maximal. Thus, ten spectral classes were defined. The mean $\mu_{il}$ and standard deviation $\sigma_{il}$ of the brightness values for each class $i$, $1 \le i \le 10$ and band $l$, $1 \le l \le 9$ were determined from observations of the data. A "ground truth" data set with each pixel in class $i$ and band $l$ was assigned a brightness value from a Gaussian distribution [15] with mean $\mu_{il}$ and standard deviation $0.25\sigma_{il}$. The smaller standard deviation was used for two reasons. 1) Based upon actual observations, the noise is not all Gaussian; thus, because of natural variability, $\sigma_{il}$

is expected to be much greater than would result if all noise were Gaussian. 2) Random noise was to be added for our experiments. This data set will be called *okla*0.

Two "noisy" data sets, *okla*5 and *okla*10, were created by the addition of Gaussian noise to data in *okla*0. Let gauss($\mu$, $\sigma$) be a function which returns values having a Gaussian distribution with mean $\mu$ and standard deviation $\sigma$. For each $k$, $1 \leq k \leq 65536$, let $y_k$ = gauss(0, $\sigma$). For each $l$, $1 \leq l \leq 9$, let $z_l$ = gauss(0, $0.25\sigma$). Then let $x'_{kl} = x_{kl} + \text{round}(y_k) + \text{round}(z_l)$ where $x_{kl}$ is the original datum from the simulated noise-free image *okla*0. In generating *okla*5, we let $\alpha = 5$, and in generating *okla*10, $\sigma = 10$. Thus, each of *okla*5 and *okla*10 has Gaussian noise which varies from datum to datum in a given band, with band-to-band correlation of Gaussian noise for a given datum. (For $\sigma = 10$, it was no longer possible to distinguish between any class in any band for, in effect, the histograms had merged. A higher value of $\sigma$ was investigated but discarded after comparing the resulting histograms to those for the natural scene.)

*B.*

Using the interactive capabilities of DIMAPSAR [16], three bands of the nine-band *okla*0 image were displayed on a color monitor. Ten spatial positions in the image were selected on the basis of their visual characteristics to be initial cluster centers and were identified to the system by a cursor. For each of the ten positions thus identified as a cluster center, the brightness value for each band of the cluster center was the average of the brightness values in a $5 \times 5$ spatial neighborhood of the datum in that band.

The set of cluster centers defined by the aforementioned procedure was used as a benchmark for all of our experiments to enable comparisons of all aspects of the performance of LFCM and AFCM. Values of $m = 1.5$ and $\epsilon = 0.001$ were fixed in all the experiments.

### C. Discussion

The tables which follow show the results of applying the two implementations of the fuzzy *c*-means algorithm to the *okla* data. LFCM used all real numbers and performed all divisions and exponentiations as required in a literal coding of the algorithm. AFCM used the approach presented in Section III.

Table I shows the number of iterations required by AFCM and LFCM to converge when using each of *okla*0, *okla*5, and *okla*10 as data. As the amount of noise in the data increases, LFCM requires slightly more iterations than AFCM to converge. As mentioned earlier, in our implementation, we have used $V[i, l]$ to an accuracy of one decimal place. LFCM requires a few more iterations to converge because of the more precise representation of the cluster centers.

Table II shows the time required to compute new cluster centers and to update the fuzzy membership matrix. The time required by AFCM to compute the cluster centers is taken as unity. All other timings are measured in this unit. There was no essential difference between corresponding times for the different data sets. From the results presented in Table II, it appears that, on average, AFCM is about six times more efficient (or faster) than LFCM.

TABLE I
NUMBER OF ITERATIONS FOR CONVERGENCE OF AFCM AND LFCM FOR THE THREE TEST DATA SETS

| | *okla*0 | *okla*5 | *okla*10 |
|---|---|---|---|
| AFCM | 5 | 18 | 75 |
| LFCM | 4 | 19 | 81 |

TABLE II
NORMALIZED AVERAGE TIME FOR COMPUTATION OF NEW CLUSTER CENTERS AND UPDATED FUZZY MEMBERSHIP MATRIX

| | $\{v_i\}$ | $\{u_{ij}\}$ |
|---|---|---|
| AFCM | 1.00 | 3.23 |
| LFCM | 11.07 | 13.72 |

TABLE III
MAXIMUM OF THE ABSOLUTE VALUE OF THE DIFFERENCE BETWEEN BAND COORDINATES FOR LFCM AND AFCM FOR EACH OF THE TEN CLUSTERS OF THE TEST DATA SETS

| Cluster | *okla*0 | *okla*5 | *okla*10 |
|---|---|---|---|
| 1 | 0.1 | 0.1 | 0.9 |
| 2 | 0.1 | 0.1 | 0.3 |
| 3 | 0.1 | 0.1 | 1.7 |
| 4 | 0.2 | 0.1 | 0.3 |
| 5 | 0.1 | 0.1 | 0.2 |
| 6 | 0.1 | 0.1 | 0.3 |
| 7 | 0.1 | 0.1 | 1.7 |
| 8 | 0.1 | 0.1 | 1.5 |
| 9 | 0.2 | 0.1 | 0.8 |
| 10 | 0.3 | 0.1 | 0.5 |

The results shown in Tables I and II together are very significant for us, as we have developed AFCM for interactive use. The time required for each iteration of LFCM is significantly greater than for AFCM, and the number of iterations to convergence is slighty fewer. Analysis of *okla*10 by AFCM required a session of several hours. Analysis of the same data set by LFCM required an entire day! We would never attempt serious interactive work on digital images with LFCM, given our successful experience with AFCM.

Table III reports the maximum of the absolute values of the differences between band coordinates of the terminal cluster centers obtained with LFCM and AFCM for each of the three data sets. For the data sets with either none or moderate amounts of random noise, viz. *okla*0 and *okla*5, the maximum differences are negligible for all practical purposes. For *okla*10 with large amounts of random noise, the maximum differences are on the order of several units for three clusters. Even then, considering the accuracy of the original data, we feel very comfortable with the results of AFCM. In a pilot version of AFCM, the $\tilde{v}_i$ were represented by integers instead of reals to the

accuracy of one place after the decimal point, and tables $C$ and $D$ were computed to three places after the decimal point instead of four. The pilot version resulted in convergence in about one fourth as many iterations of *okla*10. However, the cluster centers were found to diverge by as much as six units from those found by LFCM. Thus, we conclude that more precise approximations of the intermediate values result in cluster centers which are in better agreement with the "correct" cluster centers found by LFCM.

Lastly, we show the degree to which LFCM and AFCM produce results which agree with the ground truth from which all of the data sets were generated.

*Definition:* For $X \in \boldsymbol{R}^p$ and $u$ a fuzzy set, define the $\alpha$-level set of $u$ to be

$$u_\alpha(X) = \{x \in X | u(x) > \alpha\}.$$

Viewing the values of $u$ as indicants of membership, we let $\alpha$ be a threshold such that for a cluster membership greater than $\alpha$, we assign a hard class membership. By this means, fuzzy partitions can be converted to hard ones. In particular, we can, for a value of $\alpha$, find the $\alpha$-level sets of the ten clusters we have determined for *okla*0, *okla*5, and *okla*10. Data set *okla*0 is the benchmark, and so exhibits a perfect classification. Tables IV and V show more interesting cases. For the data sets *okla*5 and *okla*10, the ten clusters determined by LFCM and AFCM have been converted into classes by applying a threshold of $\alpha = 0.5$. These classes were matched with the classes of the original data from which the data sets were constructed. The fraction of data classified correctly (A), misclassified (B), and unclassified (below the $\alpha$ level for any class) were determined. Additionally, an overall figure of merit is calculated by subtracting B from A. (the number of data classified correctly less the number classified incorrectly). From the tables, we see that AFCM performed essentially as well as LFCM in both cases.

Although Table V appears to make AFCM the implementation of choice, we do not believe that these experiments serve to validate any claim to that effect. Rather these results seem to corroborate the intuition that loss of the true necessity of (1) and (2) does not, for the approximation proposed, cause a significant change in iterate sequences generated by LFCM and AFCM. Due to the truncations used in the approximate representation, we may be, in effect, using a slightly different value of $m$ at each iteration. Since $(U, v)$ are continuous functions of $m$, it may be that $(U(m), v(m)) \approx ((\tilde{U}(m + \delta), \tilde{v}(m + \delta))$ for some small $\delta > 0$. An investigation is currently underway to substantiate this conjecture.

The results presented in this section are for a weighting exponent of $m = 1.5$. For image applications, $1.1 \leq m < 2.5$ has proved to be adequate for all practical purposes [10]. Problems are eventually to be encountered, however, with higher values of $m$, at first with AFCM, and for yet higher values of $m$ with LFCM because of the limitations of computer hardware. As $m$ becomes larger, the

TABLE IV
RESULTS FOR *okla*5 AND $\alpha = 0.5$ FOR COMPARISON OF CLASSIFICATIONS BY EACH OF AFCM AND LFCM WITH GROUND TRUTH FROM WHICH *okla*5 WAS CONSTRUCTED

| | Classified Correctly (A) | Classified Incorrectly (B) | Unclassified | (A-B) |
|---|---|---|---|---|
| AFCM | 0.890 | 0.093 | 0.017 | 0.797 |
| LFCM | 0.889 | 0.092 | 0.018 | 0.797 |

TABLE V
RESULTS FOR *okla*10 AND $\alpha = 0.5$ FOR COMPARISON OF CLASSIFICATIONS BY EACH OF AFCM AND LFCM WITH GROUND TRUTH FROM WHICH *okla*10 WAS CONSTRUCTED

| | Classified Correctly (A) | Classified Incorrectly (B) | Unclassified | (A-B) |
|---|---|---|---|---|
| AFCM | 0.605 | 0.342 | 0.053 | 0.262 |
| LFCM | 0.595 | 0.347 | 0.058 | 0.248 |

$u_{ik}$ become smaller because the data spread their fuzzy memberships among a greater number of clusters. Thus, in (1), the denominator will be a sum of higher powers of smaller numbers. Moreover, the denominator of (1) is affected more severely than the numerator because the terms in the numerator sum are multiplied by the $x_{kl}$. Thus, for $m \geq 3$, (1) may generate cluster centers outside [0–255], especially if the number of clusters is greater than five. If necessary, the current AFCM may be modified easily to represent $\tilde{u}_{ik}$ with four digits of precision, i.e., [0–10 000] without any significant increase in the size of the associated tables.

## V. SUMMARY

We have compared a literal coding of the fuzzy $c$-means algorithm (LFCM) with a table-driven approach (AFCM). Results of the performance comparison of LFCM and AFCM using multiband test image data indicate that AFCM requires about one sixth less computer time, while yielding practically the same accuracy as the literal implementation. Our data were limited to integers in [0,255], and many of the proposed tables were constructed on that basis. These approximations and the general method of using lookup tables are seen to have far greater utility in image analysis than the present experiments suggest. In view of the approximate nature of the *application* of fuzzy $c$-means, it may be that AFCM suffices—at a considerable savings in CPU time—whenever feature space is confined to a tuple of finite integers.

While the examples above show that AFCM does indeed run faster than LFCM, a number of theoretical issues were put aside in our exposition of the proposed methodology. We conclude this paper with a short list of important questions concerning theoretical relationships between LFCM and AFCM.

First, as previously noted in Section IV-C, there is no guarantee that $J_m$ descends on iterates generated by AFCM. This probably precludes any type of convergence

theory for AFCM, and there is no reason to assume that AFCM will always converge. On the other hand, although LFCM is known to generate sequences that always contain a subsequence convergent to either a local minimum or saddle point of $J_m$, there is no assurance that LFCM attains a local minimum *in practice*. Indeed, iterative convergence of LFCM to a local minimum of $J_m$ is guaranteed only by starting "close enough" to a solution; oscillatory behavior of LFCM is not precluded by its convergence theory. Against the possibility that AFCM may not converge, we can offer only our computational experience: in all of the computations described above, as well as several other examples detailed in [17], AFCM has always converged. Nonetheless, convergence theory for AFCM is at this point an open question.

A second point of great interest is whether AFCM and LFCM—when convergent—always terminate at (roughly) the same fixed point of the FCM operator. Because AFCM is not driven by the FCM penalty function, it is certainly possible that iterate sequences generated by the two algorithms diverge from each other. This has yet to happen in practice, but there is no doubt that it can occur. Whether AFCM always follows LFCM along a "parallel" path through $M_{fc}$ is unknown.

Finally, we emphasize again that the weighting exponent $m$ varies from iteration to iteration—indeed, perhaps from term to term—in AFCM, so it is very difficult to envision a fixed objective function underlying AFCM. In summary, the relationship of AFCM to LFCM is analogous to, e.g., the acceleration of Newton's method by computing the inverse of the Jacobian every $k$th time instead of at every iteration. From a theoretical viewpoint, both shortcuts are disturbing; from a practical view, both are well justified. We acknowledge the importance of these theoretical issues, and hope to make them the basis of a future study.

## References


[1] I. Anderson and J. C. Bezdek, "Curvature and tangential deflection of discrete arcs," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 27–40, 1984.

[2] J. C. Bezdek, "Feature selection for binary data—Medical diagnosis with fuzzy sets," in *Proc. Nat. Comput. Conf.* AFIPS Press, 1972, pp. 1057–1068.

[3] ——, "Fuzzy mathematics in pattern classification," Ph.D. dissertation, Cornell Univ., Ithaca, NY, 1973.

[4] ——, *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum, 1981.

[5] ——, "Statistical pattern recognition systems," Boeing Aerospace, Kent, WA, Tech. Rep. TR-SC83001-004, 1983.

[6] J. C. Bezdek, C. Coray, R. Gunderson, and J. Watson, "Detection and characterization of cluster substructure," *SIAM J. Appl. Math.*, vol. 40, pp. 339–372, 1981.

[7] J. C. Bezdek, R. Hathaway, and V. Huggins, "Parametric estimation for normal mixtures," *Pattern Recognition Lett.*, vol. 3, pp. 79–84, 1984.

[8] J. C. Bezdek and K. Solomon, "Simulation of implicit numerical characteristics using small samples," in *Proc. ICASRC 6.* New York: Pergamon, 1981, pp. 2773–2784.

[9] J. C. Bezdek, M. Trivedi, R. Ehrlich, and W. E. Full, "Fuzzy clustering: A new approach for geostatistical analysis," *Int. J. Syst., Meas, Decision*, vol. 1, pp. 13–23, 1981.

[10] R. L. Cannon and C. Jacobs, "Multispectral pixel classification with fuzzy objective functions," Cen. for Automation Res., Univ. Maryland, College Park, Tech. Rep. CAR-TR-51, 1984.

[11] J. C. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact, well-separated clusters," *J. Cybern.*, vol. 3, pp. 32–57, 1973.

[12] W. E. Full, R. Ehrlich, and J. C. Bezdek, "FUZZY QMODEL—A new approach to linear unmixing," *Math Geol.*, vol. 14, pp. 259–270, 1982.

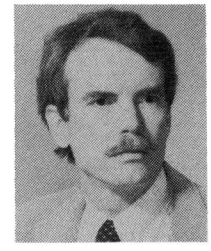
[13] G. Granath, "Application of fuzzy clustering and fuzzy classification to evaluate provenance of glacial till," *Math Geol.*, vol. 16, pp. 283–301, 1984.

[14] E. E. Gustafson and W. C. Kessel, "Fuzzy clustering with a fuzzy covariance matrix," in *Proc. IEEE CDC*, San Diego, CA, 1979, pp. 761–766.

[15] T. H. Naylor, *Computer Simulation Experiments with Models of Economic Systems*. New York: Wiley, 1971.

[16] M. A. Wiley and J. V. Dave, DIMAPSAR—An interactive image interpretation system for petroleum and minerals exploration," in *Proc. 8th IBM Univ. Study Conf.*, Raleigh, NC, Oct. 17–19, IBM Acad. Inform. Syst. Stamford, CT, 1983.

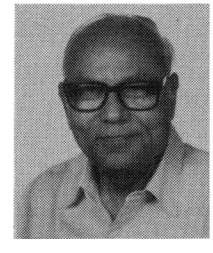
[17] R. L. Cannon, J. V. Dave, J. C. Bezdek, and M. M. Trivedi, "Segmentation of thematic mapper image data using fuzzy *c*-means clustering," in *Proc. 1985 IEEE Workshop on Languages for Automation*, IEEE Computer Society, 1985, pp. 93–97.

**Robert L. Cannon** received the B.S. degree in mathematics from the University of North Carolina, Chapel Hill, the M.S. degree in mathematics from the University of Wisconsin, Madison, and the Ph.D. degree in computer science from the University of North Carolina, Chapel Hill.

He has been a member of the faculty of the University of South Carolina, Columbia, since 1973, and has held visiting appointments with the University of Maryland and the IBM Corporation. His interests include image processing applications in remote sensing and geology and fuzzy mathematical applications in artificial intelligence.

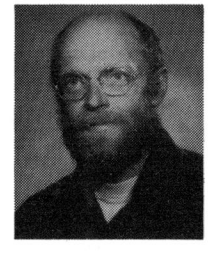
Dr. Cannon is a member of the IEEE Computer Society, the Association for Computing Machinery, the Pattern Recognition Society, the International Fuzzy Systems Association, and the North American Fuzzy Information Processing Society.



**Jitendra V. Dave** received the Ph.D. degree in physics from Gujarat University, India, in 1957.

He is a Senior Technical Staff Member at the IBM Palo Alto Scientific Center, Palo Alto, CA. He has worked on problems of atmospheric physics and radiative transfer, remote sensing of ozone and aerosols, the effect of particulate contamination on climate, photovoltaic harvesting of solar energy, image analysis and display applications in oil and mineral explorations, and large-scale scientific computing through Fortran. He has published about 60 technical papers in various scientific journals.

Dr. Dave is a Fellow of the American Physical Society.



**James C. Bezdek** received the B.S. degree in civil engineering from the University of Nevada, Reno, in 1969, and the Ph.D. degree in applied mathematics from Cornell University, Ithaca, NY, in 1973.

He is currently Professor and Chairman of the Department of Computer Science, University of South Carolina, Columbia. His interests include research in pattern recognition, computer vision, information retrieval, and numerical optimization.

Dr. Bezdek is currently President of NAFIPS (North American Fuzzy Information Processing Society) and President-Elect of IFSA (International Fuzzy Systems Association). He is a member of the IEEE Computer Society the Classification Society, and the Pattern Recognition Society.