VLA COMPUTER MEMORANDUM NO. 174

More on Image Compression

FRED SCHWAB

February 15, 1985

ABSTRACT. A new method of image compression (i.e., a method to reduce the number of bits needed to encode the essential information in a digital image) was developed *ab ovo* and described by Arnold Rots in an earlier memorandum in this series (VLA Computer Memo. No. 173). Since a great many data compression methods have already been described in published literature, I felt that it might be worthwhile to consider a couple of the methods that receive prominent mention there. One such method is based on the so-called singular value decomposition (SVD) of linear algebra. Another is Huffman coding, a minimum-redundancy scheme for error-free data encoding; a sub-optimal variant of this method, called Huffman shift coding, might be preferred on technical grounds.

INTRODUCTION

A new method of image compression—i.e., a method to reduce the number of bits needed to encode the essential information in a digital image—is described by Arnold Rots in a recent VLA Computer Memorandum [18]. Many data compression methods (and image compression methods, in particular) have already been described in the published literature (see [5],[8],[17],[20]), so I felt that it would be worthwhile to examine a couple of the methods that receive prominent mention there. (Our aim is to find a suitable scheme to transmit pictures, over a slow communications link, to the remote users of a supercomputer.)

One interesting, and somewhat promising method is based on the so-called singular value decomposition (SVD), a matrix factorization technique of linear algebra. This method is the only one with which I was familiar when I set out to write a memorandum; consequently, it is the first method that I describe. But, except perhaps if one is interested in transmitting high precision, floating-point digital imagery—rather than coarsely quantized images—it is not obvious that the SVD method would be the method of choice. This question is highly intertwined with the degree of quantization, and therefore the issue is somewhat difficult to resolve. The SVD method might be useful, however, for *archival storage* of floating-point images: for example, rather than quantizing 32-bit floating-point images to 16 bit integer, one could, in most cases, achieve much higher accuracy via factor-of-two floating-point SVD compression.

Huffman coding is an optimal means of encoding data that have been quantized into a finite number of levels. In transmitting a digital image by Huffman coding, the average number of bits per pixel which is required never exceeds the entropy of the image by more than one bit. Usually this excess, which is called the code redundancy, is much smaller than one bit. Huffman coding employs variable-length codewords, the shorter words being assigned to the more frequently occurring quantization levels. With standard Huffman coding, a decoding table whose length is equal to the number of (occupied) quantization levels is required. Especially when there is a large number of quantization levels, it may be preferable to use a variant of Huffman coding, such as the so-called Huffman shift coding technique. This increases the code redundancy by a small (but usually a tolerable) amount.

Given its simplicity and near-optimality, I believe that Huffman (shift) coding might be a sensible choice for transmission of quantized digital imagery to the remote users of a supercomputer facility. Other methods might be preferred in cases when one can tolerate some degree of degradation of the data that is not precisely known. One manner of working would be to quantize the data to the coarsest degree thought to preserve all of the desired information, and then to employ an error-free coding method such as Huffman coding.

In this discussion I am assuming that one relies on some commercial package of networking software to achieve data transmission and error correction. If not, then one would need to consider the subject of error correction together with that of data compression, and more careful thought would be required.

The SVD Compression Technique

The idea of using the SVD to achieve image compression has been around since at least the early 70's, when it caught the fancy of the war research community (see [2],[3],[4], and references cited therein). [2] is a good expository article; textbook accounts appear in [5] and [16]. The SVD, itself, has been known since at least 1890—J. J. Sylvester[1] gives a one page description in [19].

Preliminaries. An arbitrary real $m \times n$ matrix (or "image") A of rank r, with $m \geq n$,[2] can be written in the form

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T, \tag{1}$$

where U is an $m \times m$ orthogonal matrix, V is $n \times n$ orthogonal, and

$$\Sigma = \begin{pmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_n \end{pmatrix}$$

is an $n \times n$ diagonal matrix with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. When A is of less than full rank, $\sigma_{r+1} = \cdots = \sigma_n = 0$. This representation of A is called its ***singular value decomposition (SVD).*** The scalars $\sigma_1, \ldots, \sigma_n$ (which are the nonnegative square roots of the eigenvalues of $A^T A$) are called the ***singular values*** of A, the columns $\mathbf{u}_1, \ldots, \mathbf{u}_m$ of U are called the ***left singular vectors***, and the columns $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of V are called the ***right singular vectors.*** The $\mathbf{u}_i$ are the orthonormalized eigenvectors of AA^T, and the $\mathbf{v}_i$ are those of $A^T A$. The SVD is unique, up to trivial reorderings of any singular vectors that are associated with identical singular values.

The SVD of A can be written as a linear combination of "outer products" of pairs of vectors; i.e., as a sum of rank 1 matrices:

$$A = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T . \tag{2}$$

[1]Sylvester stayed for a while in Charlottesville. In a preface to Sylvester's *Collected Works*, H. F. Baker writes: "Leaving University College in the session of 1840–41, he proceeded as Professor of Mathematics across the Atlantic, to the University of Virginia, founded in 1824 at Charlottesville, Albemarle Co., where his colleague, Key, of University College, had previously occupied the chair of Mathematics. Such a considerable change deserved a better fate than befell; in Virginia at this time the question of slavery was a subject of bitter contention, and Sylvester had a horror of slavery. The outcome was his almost immediate return; apparently he had intervened vigorously in a quarrel between two of his students. ... On his return from America Sylvester seems to have abandoned mathematics for a time."

[2]Recall that rank $A = r$ implies that A has r, but not $r+1$, linearly independent rows (or columns). The rank of an $m \times n$ matrix ($m \geq n$) representing a noisy (i.e., a measured) digital image normally equals n. (Possible exceptions are cases of contrived, or model data, and cases of very coarsely quantized data.) The restriction $m \geq n$ is not essential; it just permits the use of simpler notation (one can work with A^T when the image has more columns than rows). (Superscript-T denotes matrix transpose.)

Here, let us denote the k^{th} partial sum as A_k; i.e., $A_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$. A_k is (in the least-squares sense) the best rank k approximation to A. The (squared) Euclidean norm[3] of the difference between A and A_k is given by the sum of the squares of the singular values corresponding to the omitted terms; i.e.,

$$\|A_k - A\|_2^2 = \sum_{l=k+1}^{r} \sigma_l^2 , \tag{3}$$

and so the root mean square error of approximation is

$$\text{r.m.s. error} = \sqrt{\frac{1}{mn} \sum_{l=k+1}^{r} \sigma_l^2} .$$

If the singular values σ_i decrease rapidly, then, for relatively small values of k, A_k is close to A.

An efficient and numerically stable algorithm for computing the singular value decomposition is given by Golub and Reinsch in [11]. Portable Fortran implementations of their algorithm are available in the LINPACK package [9]. The LINPACK subroutine DSVDC was used to obtain the test results that are presented below.

Use of the SVD for Compression of Digital Images. Expressing Eq. 3 in words, the orthogonality constraints on U and V ensure that each of the rank 1 matrices $\mathbf{u}_i \mathbf{v}_i^T$, $i \leq r$, has the same Euclidean norm; thus, the importance (in the mean-squared error sense) of the ith term in the right-hand side of Eq. 2 is measured by σ_i. Assuming that A_k is a good enough approximation to A, the "image" can be faithfully represented by $(m+n+1)k$ real numbers (or, for a square $(n \times n)$ matrix, by $(2n+1)k$ numbers). For image transmission, if the data are transmitted (in packets of $m+n+1$ numbers) in the order $(\sigma_1, \mathbf{u}_1, \mathbf{v}_1)$, $(\sigma_2, \mathbf{u}_2, \mathbf{v}_2)$, $\ldots$, then the received image can be built up in "real-time." If the singular values (there are only n of them) are transmitted first, then, on the receiving end, one can determine (by plotting them) how much image compression can be achieved and what value of k is appropriate.

Examples given in [2]–[5] and [16] illustrate that digitized photographs of natural scenery (satellites, tanks, and women) often can be economically represented by truncated SVD's. Examples given below illustrate that the same can be said for radio maps.

$O(mn^2)$ arithmetic operations are needed to compute the SVD of an $m \times n$ matrix, by the Golub and Reinsch algorithm (see [9, ch. 11] for the exact operation counts); thus the decomposition is more computationally burdensome than, say, an FFT of the same sized matrix. In addition, reconstructing the matrix or "image" requires $k-1$ additions and about k multiplications, per matrix element or "pixel." By these considerations, to use the SVD to achieve economical image transmission, one would partition the matrix representing a large image into a set of smaller submatrices. The question of choosing a sensible partition of the image is addressed below.

Transmitting the full SVD of an $m \times n$ matrix requires sending more numbers ($(m+n+1)n$ numbers) than the quantity which is present in the image itself ($\sim$ twice the amount in the case of a square matrix). In all instances, the actual compression ratio achieved depends on whether one chooses to quantize the singular vectors prior to transmission (and on how coarsely they are quantized)—a bit more on this below.

[3]i.e., the square root of the sum of the squares of the matrix elements.

Examples.

Example 1: A "high dynamic range" map. A 256×256 VLA map of the jet in NGC 6251 is shown in Fig. 1. This is a "high dynamic range" image produced by the CLEAN deconvolution algorithm. The intensity of the unresolved "core" at the tip of the jet is 0.7 Jy/beam, the features along the jet are at a level of a few mJy, and the r.m.s. level (the noise) in areas away from the jet is $\sim$ 80 μJy. A plot of the singular values (normalized to σ_1) is shown in Fig. 2a, and a plot of the r.m.s. error of the k^{th} truncated SVD approximation to this image (normalized to the peak in the original image) is shown in Fig. 2b. Selected rank 1 images, $\mathbf{u}_i\mathbf{v}_i^T$, comprising the SVD of this image are shown in Fig. 3. Most of the energy of the compact core is represented by the first two singular values. Selected truncated SVD representations A_k are shown in Fig. 4. The r.m.s. errors of the approximations by A_{40} and A_{50} are 55 μJy and 44 μJy, respectively—well below the r.m.s. noise level in the original image.

Example 2: A "low dynamic range" map. Selected truncated SVD representations of a "low dynamic range" 20 cm. VLA (CLEAN) map are shown in Fig. 5. This is a map of a region containing a large number of point-like sources. In this example the r.m.s. noise level is a few percent of the map peak. No reproduction of the original map is included here, as it would be indistinguishable from the picture of A_{40} that is shown. The singular values, and the r.m.s. errors of the approximations by the A_k, are shown in Figs. 2c and 2d, normalized as in Example 1. The singular value plot would bear a strong resemblance to the plot in Fig. 2a if the spike were removed from the latter. It is interesting to note (from Fig. 5) that various ones of the point sources seem to pop out as k increases—and it is fairly obvious why this should be so.

Example 3: A "dirty beam." Truncated SVD representations of a "dirty beam" (corresponding to the VLA observation of Example 2) are shown in Fig. 6; the singular value profile and an error plot are shown in Figs. 2e and 2f. A_{80} furnishes an adequate pictorial display of the original image, so a picture of the dirty beam itself is not included here. The dirty beam looks much more "complex" than the deconvolved maps used in the previous two examples, and, indeed, the singular values decrease much less rapidly in this instance—here there is not a lot of potential for image compression. If a few iterations of the CLEAN algorithm were applied to this image, then a spike (much like the spike in Fig. 2a, which is due to the presence of a compact "core") would appear on the leftmost portion of the singular value profile. The depth of the spike would depend on the number of CLEAN iterations, and its width would depend (roughly) on the width of the CLEAN "restoring beam."

All three of the singular value plots have various "knees" or bends. Those in Figs. 2a and 2c both have inverted knees in the range k = 20–30. Empirically (and intuitively) it seems that the horizontal location of this knee should depend (in the typical radio map of a fairly empty region) on the depth of "cleaning".

Figs. 2a and 2c also have fairly sharp bends in the range k = 140–170, whereas in Fig. 2e there is a gradual bending in the neighborhood of k = 180. Both of the CLEAN maps (Examples 1 and 2) were converted to 16-bit integer format before the singular value analysis was performed, but the dirty beam (of Example 3) was kept in 32-bit floating-point format. These shallow bends in the CLEAN maps appear to be due to truncation (i.e., to the relatively coarse quantization that was applied). The knee in the dirty beam is of greater depth and of more gently sloping character. Its ordinate ($\sim 10^{-8}$–10^{-7}) is of the same order of magnitude as the unit-roundoff characteristic of 32-bit floating-point arithmetic. And the benign properties of floating-point arithmetic with rounding probably account for the gentleness of the bend.

Remarks. Choosing a sensible partioning of the image is not very difficult. First of all, the Golub and Reinsch algorithm is computationally expensive enough (about $7mn^2 + 11n^3/3$ arithmetic operations are required) that the decomposition of blocks larger than, say, 256×256 probably oughtn't to be considered. Second, given that the map is to be reconstructed on a modest-sized computer rather than on a supercomputer, it would be unreasonable to expend more than a hundred or so arithmetic operations per pixel in doing so. The vector registers on a supercomputer such as the Cray usually are best suited to vector operations on vectors of length 64, or so. Thus, 64×64 might be a reasonable block size for partitioning. However, nothing should prevent one from going smaller; 16×16 is mentioned in the textbook by Pratt [16] as having been used in practical applications.

Some further economy can be achieved by this partitioning of the image. Namely, in fairly empty regions of the map, relatively fewer singular values and singular vectors need to be retained than in other regions (assuming that one is aiming for the same r.m.s. error level in each area). Although this extra degree of sophistication entails greater programming effort, it is clear that there would be a substantial payoff in many instances.

HUFFMAN CODING

For a message (read *map*) composed from the letters (read *pixel values*) of a finite-length alphabet (read *set of quantization levels*) binary Huffman coding [14] is an optimal encoding technique, in the sense that the number of bits in the encoded message is the minimum number which is required (by any such binary substitution code) to allow a unique decoding of the message. Huffman coding employs variable-length codewords. As in Morse code, the shorter codewords are assigned to the more frequently occurring letters of the alphabet. The code is called a prefix code because no codeword is a prefix of (or the beginning of) another codeword.

Let p_i denote the relative frequency of occurrence, within the message, of the ith letter of the n-letter alphabet (i.e., in our case, p_i is the occupancy rate of the ith quantization level, and $n = 2^N$). Huffman's method for constructing the code based on $\{p_1, \ldots, p_n\}$ can be described as follows: Set $k = n$. For $k > 1$, let p_i and p_j denote the smallest and second smallest elements of $\{p_1, \ldots, p_k\}$. Replace p_i and p_j by $(p_i + p_j)$, and, retaining all parentheses, repeat the construction on the remaining $k-1$ elements, until $k = 1$. For example, if $n = 7$ and $\{p_1, \ldots, p_7\} = \{.4, .08, .08, .2, .12, .08, .04\}$ the construction is

$$\begin{array}{rl}
\{.4, .08, .08, .2, .12, .08, .04\} & k = 7 \\
\{.4, .08, .08, .2, .12, (.08 + .04)\} & k = 6 \\
\{.4, (.08 + .08), .2, .12, (.08 + .04)\} & k = 5 \\
\{.4, (.08 + .08), .2, (.12 + (.08 + .04))\} & k = 4 \\
\{.4, ((.08 + .08) + .2), (.12 + (.08 + .04))\} & k = 3 \\
\{.4, (((.08 + .08) + .2) + (.12 + (.08 + .04)))\} & k = 2 \\
\{(.4 + (((.08 + .08) + .2) + (.12 + (.08 + .04))))\} & k = 1.
\end{array}$$

The length l_i of the codeword for the ith letter is given by the corresponding depth of parenthesis nesting; in this case $(l_1, \ldots, l_7) = (1,4,4,3,3,4,4)$. The codewords themselves are determined by the order in which the parenthesizing occurs; in this example, the codewords are (1,0111,0110,010,001,0001,0000). This description is borrowed from Knuth [15]; the better known graphical construction of Huffman is illustrated in [12] and [17].

The amount by which the quantity $l_{\text{av}} = \sum_i l_i p_i$, the average number of bits used to encode a letter of the message, exceeds the binary entropy of the message is called the code redundancy r; i.e., $r = \sum_i l_i p_i + \sum_i p_i \log_2 p_i$. With Huffman coding, r is minimized, and r never exceeds unity. The code has two additional special properties: the maximum codeword length $\max_i l_i$ is minimized, as is the total length $\sum_i l_i$ of the encoding table. The ratio $\eta = \dfrac{-\sum_i p_i \log_2 p_i}{\sum_i l_i p_i}$ is called the code efficiency.

Gallager [10] shows that the redundancy of a binary Huffman code satisfies the upper bound $r \leq p_{\max} + (\ln 2 - \ln \ln 2 - 1)/\ln 2 \approx p_{\max} + 0.0861$, where $p_{\max} \equiv \max_i p_i$ denotes the probability of occurrence of the most likely letter (and that for $p_{\max} \geq \frac{1}{2}$, $r \leq p_{\max}$). Thus, without actually constructing any Huffman codes, it's possible to get a good idea of the degree of image compression that is achievable with the technique (simply by tabulating image statistics—entropy, in particular). Nevertheless, I decided to work out some real-life examples of Huffman encoding, so that it would appear that I had done some real work. Although Gallager's bound is quite a tight bound, in the sense that the redundancy can be almost as large as his formula allows, in all of my examples his upper bound considerably overestimates r. (It's strictly tight in the limit $p_{\max} \to 0$.)

Examples. An integer programming algorithm for construction of minimum-redundancy codes is given by Hu and Tucker in [13]. Yohe [21] has written an Algol language implementation of their algorithm. My own Fortran translation of Yohe's subroutine was used to obtain the test results reported here; a listing appears in the Appendix to this memorandum.

My test results are summarized in Tables 1–3, which correspond, respectively, to the Examples 1–3 presented above. Each of the three test maps was quantized to 2^N levels equispaced between the map extrema (i.e., a linear "transfer function" was used), for $N = 8$, 12, and 16. In addition to an analysis of the basic maps themselves (labeled "Original" in the Tables), results are presented for maps obtained by performing differencing operations (the original maps are recoverable from these differences). The "1-D differenced" map corresponding to the image (f_{ij}) is formed according to the formula

$$g_{ij} = \begin{cases} f_{ij} - f_{i,j-1}\,, & j > 1\,, \\ f_{ij}\,, & j = 1\,, \end{cases}$$

and the "2-D differenced" map by

$$h_{ij} = \begin{cases} f_{ij} - f_{i-1,j} - f_{i,j-1} + f_{i-1,j-1}\,, & i > 1,\ j > 1\,, \\ f_{ij}\,, & \text{otherwise.} \end{cases}$$

Although 2^{N+1} quantization levels are needed to represent g and 2^{N+2} to represent h, the differencing operations typically yield maps with smaller numbers of *occupied* quantization levels and smaller entropy—hence, greater potential for compression. This differencing technique is advocated in the textbooks by Gonzalez and Wintz [12] and Rosenfeld and Kak [17]. (A similar differencing technique—along with a "pyramidal" transmission scheme, allowing progressive data transmission—is considered in Rots' memorandum. Huffman coding could be used in transmission of these differences, but possibly to no great advantage, because a different encoding might be desired at each level of the pyramid, in which case a number of different decoding tables would have to be transmitted.)

Tables 1–3 are, for the most part, self-explanatory. In most cases—except when the entropy is less than 1—the code redundancy is only around several hundredths of a bit, and the code efficiency often exceeds 99%. It's interesting to note how small is the fraction of occupied quantization levels in the case of a high dynamic range image. And in general, the more highly processed an image is, the less information it contains; e.g., "dirty maps" have a higher information content than maps from which the point source response has been deconvolved. This is a fortunate situation, for it will generally be the more highly processed images that one will want to transmit from the supercomputer to a remote station. The differencing schemes generally yield a significant improvement in the achieved data compression ratio.

Disadvantages. Although Huffman coding is conceptually very simple, (and perhaps also *for* this reason) it has a number of obvious disadvantages.

First, a possibly very long decoding table is required. In the case of image transmission, the number of entries in the table is equal to the number of occupied quantization levels. (More precisely, one has the choice of transmitting either a one-column table of 2^N entries or a two-column table of length equal to the number of occupied quantization levels.) The need for such a long table can be overcome by the use of a variant of Huffman coding, called Huffman shift coding, which is described below.

Since the Huffman codewords are of variable-length, special effort is required in coding and in decoding, and the data generally must be packed, for transmission, into fixed-length records. The encoding, the packing, and the de-packing and decoding are awkward to program in, say, Fortran (in PLI it would be easy, though), so one would probably resort to doing this part of this task in assembly language. This would require the services of a competent programmer (or possibly two in our case—one for the supercomputer and one for the computer on the receiving end).

A consequence of the need for a decoding table and of the variable-length property of the code is that the time required for decoding varies from record to record and is not predictable. Sometimes so-called "alphabetic coding" is used in lieu of Huffman coding. With alphabetic coding, the numerical binary order of the codewords corresponds to the alphabetic order of the encoded "letters" (in our case, to the numerical order of the quantization levels). This results in a minimum mean search time for decoding, but also, generally, in slightly higher code redundancy. Still, the decoding effort remains variable. The subroutine shown in the Appendix can be used to compute alphabetic codes as well as Huffman codes.

The amount of effort needed to compute the code is another consideration. While the computational complexity of the Hu–Tucker algorithm, employed in the subroutine I used, is $O(n^2)$, where n is the number of occupied quantization levels, there does exist an $O(n \log n)$ algorithm—I just don't know where it is written down. Nevertheless, our supercomputer presumably would be capable of the task. And, if Huffman shift coding is employed (see below) the computational burden of the task of code construction can be greatly diminished.

Huffman coding is not widely used in the computer industry, especially so in applications where special-purpose hardware is used for encoding and decoding. Instead, so-called algebraic encoding is employed. Algebraic codes (BCH codes, Reed–Solomon codes, quadratic residue codes, Golay codes, Hamming codes, etc.) in fact do not require any decoding table at all; instead, algorithms are used to decode the bit patterns. On the other hand, these codes are ones which often are designed specifically with built-in redundancy, to be used for automatic error detection and error correction rather than for data compression.

There are, however, scattered published results on the use of algebraic codes for data compression, in cases where some data degradation is allowed. For example, the quadratic residue code of type $(23, 12)$ over the finite field $GF(2)$ can be used to encode 23-bit data in 12-bit codewords—one 12-bit codeword per 23-bit datum (see [6] for details). This entails occasional decoding errors, in at most three binary digits out of the 23.[4] According to [6], Shannon gave similar results in 1959 for the Hamming codes. Such schemes might be attractive for our needs, assuming that the decoding errors occur sufficiently infrequently; for, in visual display of astronomical data, isolated overbright pixels stick out like a sore thumb (and thus cannot be mistaken for reality), and isolated underbright ones generally go unnoticed. I intend to pursue this matter further—I think that further results are given in [8]. This reference, which the U. Va. library has just put on hold for me, has so far been unobtainable.

Scattered results such as the one just mentioned form the body of a discipline known as rate distortion theory. Shannon formulated the major problem of rate distortion study in 1959: Given a measure D of data distortion (r.m.s. error, or whatever), what is the minimum information rate $R(D)$, in bits per source sample, required to transmit a given message over a communications channel with an average distortion no greater than D, and what encoding algorithm(s) can be used to encode (and decode) the data and achieve this rate? Some early references on the subject are [1] and [7]. The image processing textbooks do not describe any recent results. I plan to search for an up-to-date bibliography on the subject.

Huffman shift coding. Huffman shift coding is described by Gonzalez and Wintz in [12] and is further discussed by Rosenfeld and Kak in [17]. The objective is to modify Huffman coding so that a smaller decoding table is required; this lessens the burden of decoding. Assuming that n, the number of quantization levels is divisible by m, so that $n = mq$, one can get by with a decoding table of length $m + 2$. (Here we have to take $n = 2^N$ to be the total number of quantization levels, rather than the number of occupied levels.) The histogram of quantization levels is divided into q equal parts, each of length m. Two extra codewords are employed: one, the "shift-down" word, is used to tell the decoder that the next symbol to be decoded occurs within the next lower section of the histogram (unless the next word too is a shift-down word, in which case the symbol is in an even lower section); similarly, the "shift-up" word says to move to the next higher section of the histogram.[5]

A number of comparisons of Huffman shift coding with standard Huffman coding are given in [17]. Typically, in the examples given there, the code redundancy is increased by about 0.2 bits—occasionally the increase is as much as one-half bit (sometimes they've taken m as small as 14, which seems to me to be overdoing it). Generally this increase in redundancy is equivalent to a few percent increase in the encoded data bulk, and it probably would have been smaller if Gallager's method had been employed there. I haven't experimented with this technique, but it could certainly be used to alleviate any excessive burden incurred in the data decoding or entailed by the table transmission requirement.

DISCUSSION

The main advantage that the SVD compression technique has over Huffman coding or Huffman shift coding lies in its potential use in a progressive transmission scheme, like the

[4]Such use of this code, one of the perfect Golay codes, is the reverse of its usual use: as a triple error-correcting 23-bit binary code for 12-bit data.

[5]The textbook [17] describes a poor method of incorporating the shift words; there, probabilities are assigned to these protocol words, and they are treated like everything else. An optimal method is given by Gallager [10], who shows how to construct a prefix code with an unused codeword of length two bits and whose redundancy r' satisfies $r' \leq 1$. Further, he shows that r' does not exceed the redundancy of the Huffman code by more than the quantity $2p_2$, and he states that one cannot do better. (Here, the probabilities are assumed to be ordered monotonically, so that p_2 denotes the second largest.) This is the method described in [12], where, instead of two shift words, a single 2-bit "cyclic shift" codeword is employed. Whether this method is best, or whether one ought to repeat Gallager's construction to obtain a second 2- or 3-bit shift word, I guess depends on the actual data.

one described by Rots. A practical implementation of SVD compression for data transmission would entail quantization of the singular vectors prior to their transmission. I haven't considered this matter in detail; however, in the case of 16-bit data, say, if one could get by with quantizing the singular vectors to 8 bits, then the compression factors implicit in my Examples would double. Quantization to about 12 bits would certainly be safer.[6] I'm starting to favor the ordinary encoding techniques over SVD compression for their simplicity and because, as in the case of SVD compression of *floating-point* data, their behavior is highly predictable and doesn't depend very greatly on statistical assumptions.

I apologize—especially to Arnold Rots—for not having checked the performance of Rots' pyramidal transmission scheme on my test examples. The appearances are that his scheme would outperform simple Huffman substitution encoding in the cases of coarsely quantized, high dynamic range data (such as the NGC 6251 jet image quantized to 8 bits). Huffman coding would probably perform better on images of high information content. Huffman shift coding, though, might not more than marginally outperform the pyramidal transmission scheme in these cases. Probably the chief attraction of Huffman coding is its simplicity.

Compression ratios somewhat higher than are mentioned here may be sorely needed with a slow communications link to the supercomputer. Then the question becomes what kind of distortion of the data to accept, and I don't think that a satisfactory answer has been given.

REFERENCES

1. H. C. Andrews, *Bibliography on rate distortion theory*, IEEE Trans. Inf. Theory **IT-17** (1971), 198–199.
2. H. C. Andrews and C. L. Patterson, *Outer product expansions and their uses in digital image processing*, Amer. Math. Monthly **82** (1975), 1–13.
3. ———————————, *Outer product expansions and their uses in digital image processing*, IEEE Trans. Computers **C-25** (1976), 140–147. Essentially identical to [1].
4. ———————————, *Singular value decompositions and digital image processing*, IEEE Trans. Acoust., Speech, Signal Processing **ASSP-24** (1976), 26–53.
5. H. C. Andrews and B. R. Hunt, *Digital Image Restoration*, Prentice–Hall, Englewood Cliffs, NJ, 1977.
6. E. F. Assmus and H. F. Mattson, *Coding and combinatorics*, SIAM Review **16** (1974), 349–388.
7. T. Berger, *Rate Distortion Theory*, Prentice–Hall, Englewood Cliffs, NJ, 1971.
8. L. D. Davisson and R. M. Gray, *Data Compression*, Dowden, Hutchinson & Ross, Stroudsburg, PA, 1976.
9. J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.
10. R. G. Gallager, *Variations on a theme by Huffman*, IEEE Trans. Inf. Theory **IT-24** (1978), 668–674.

[6]Don Wells has made an interesting suggestion: that some of the singular vectors, perhaps only the first pair or perhaps the first several pairs, should be quantized and transmitted, and then another SVD computed—the SVD of a residual image, corrected for the (easily calculable) quantization error already committed—and some of it transmitted; and the process repeated as long as need be. Indeed, some scheme of this ilk would probably be required for absolutely reliable quantized SVD transmission; and, given that the SVD computations are ideally suited to vectorization on the supercomputer, this might be entirely practical. Little or no extra effort would be required on the receiving end. An appropriate number of singular vector pairs to transmit might be that number such that the resulting r.m.s. quantization error is of about the same size as the singular value corresponding to the first omitted pair. Don also suggests using Huffman coding to compress the singular vectors prior to their transmission. The supercomputer then would have to make a "dry run" through the SVD compression procedure in order to make up a histogram of the singular vector quantization (it wouldn't be practical to separately encode each pair of vectors, since the image presumably has been partitioned into small blocks). It's difficult to judge the merits of a scheme of this complexity without actually trying it, and it's hard to know when one has done the job "right," because of the possibility for endless refinement.

11. G. H. Golub and C. Reinsch, *Singular value decomposition and least squares solutions*, Numerische Math. **14** (1970), 403–420.
12. R. C. Gonzalez and P. Wintz, *Digital Image Processing*, Addison–Wesley, Reading, MA, 1977.
13. T. C. Hu and A. C. Tucker, *Optimal computer search trees and variable-length alphabetical codes*, SIAM J. Appl. Math. **21** (1971), 514–532.
14. D. A. Huffman, *A method for the construction of minimum-redundancy codes*, Proc. IRE **40** (1952), 1098–1101.
15. D. E. Knuth, *Huffman's algorithm via algebra*, J. Combinatorial Theory, Series A **32** (1982), 216–224.
16. W. K. Pratt, *Digital Image Processing*, Wiley, New York, 1978.
17. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Volume 1, Second edition, Academic Press, New York, 1982.
18. A. Rots, *Transmission of digital images over slow communication lines*, VLA Computer Memo. No. 173, October 1984.
19. J. J. Sylvester, *On the reduction of a bilinear quantic of the* n*th order to the form of a sum of* n *products by a double orthogonal substitution*, Messenger of Math. **19** (1890), 42–46. Reprinted in *The Collected Mathematical Papers of James Joseph Sylvester*, Volume IV, Chelsea, New York, 1973, pp. 654–658.
20. L. C. Wilkins and P. A. Wintz, *Bibliography on data compression, picture properties, and picture coding*, IEEE Trans. Inf. Theory **IT-17** (1971), 180–197.
21. J. M. Yohe, *Algorithm 428: Hu–Tucker minimum redundancy alphabetic coding method*, Collected Algorithms from CACM, Association for Computing Machinery, New York, 1972.

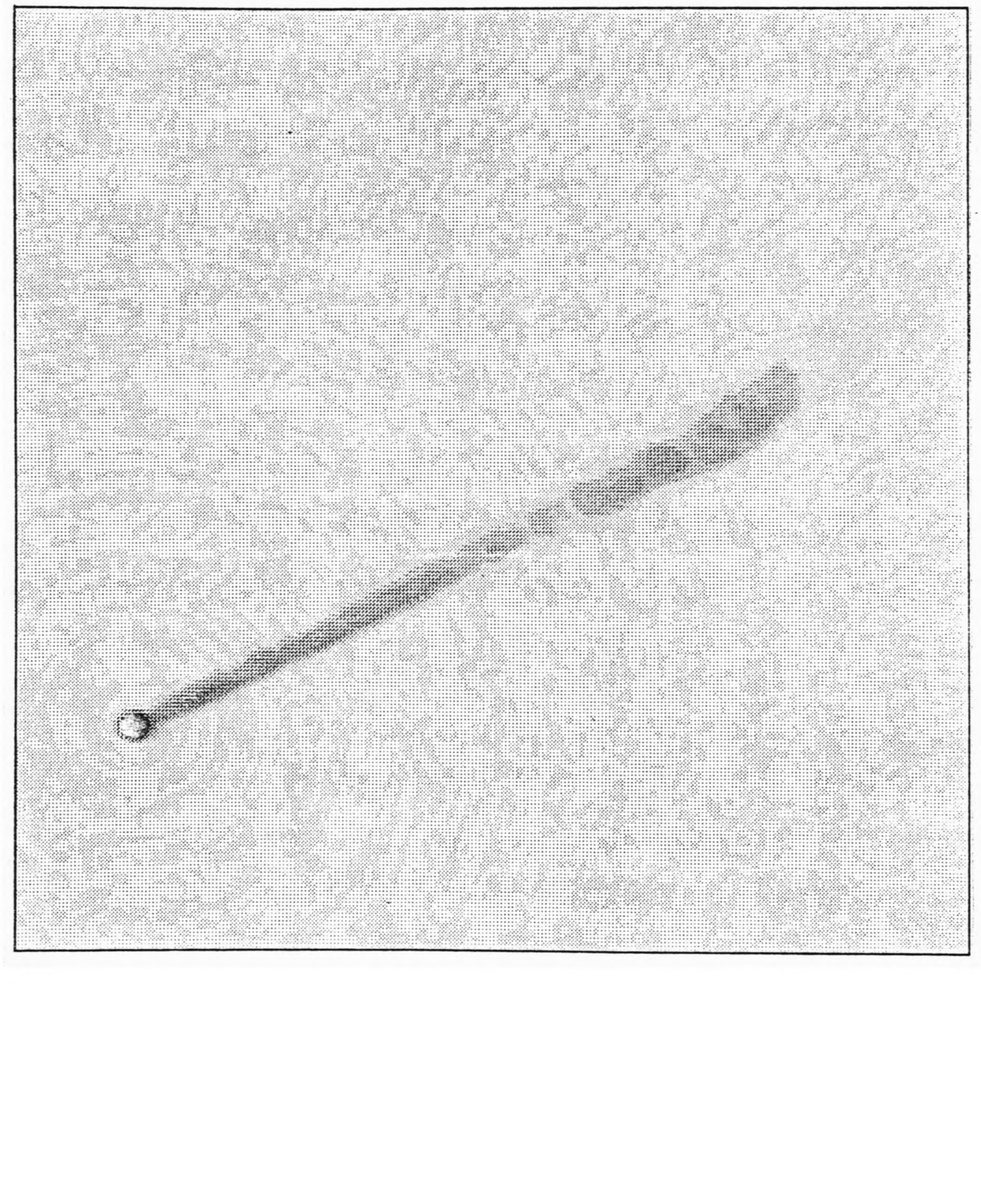


Figure 1. A photographic representation of a 256 × 256 "high dynamic range" digital image portraying radio emission (at 6 cm. wavelength) from the jet of NGC 6251 (the map courtesy of Rick Perley).

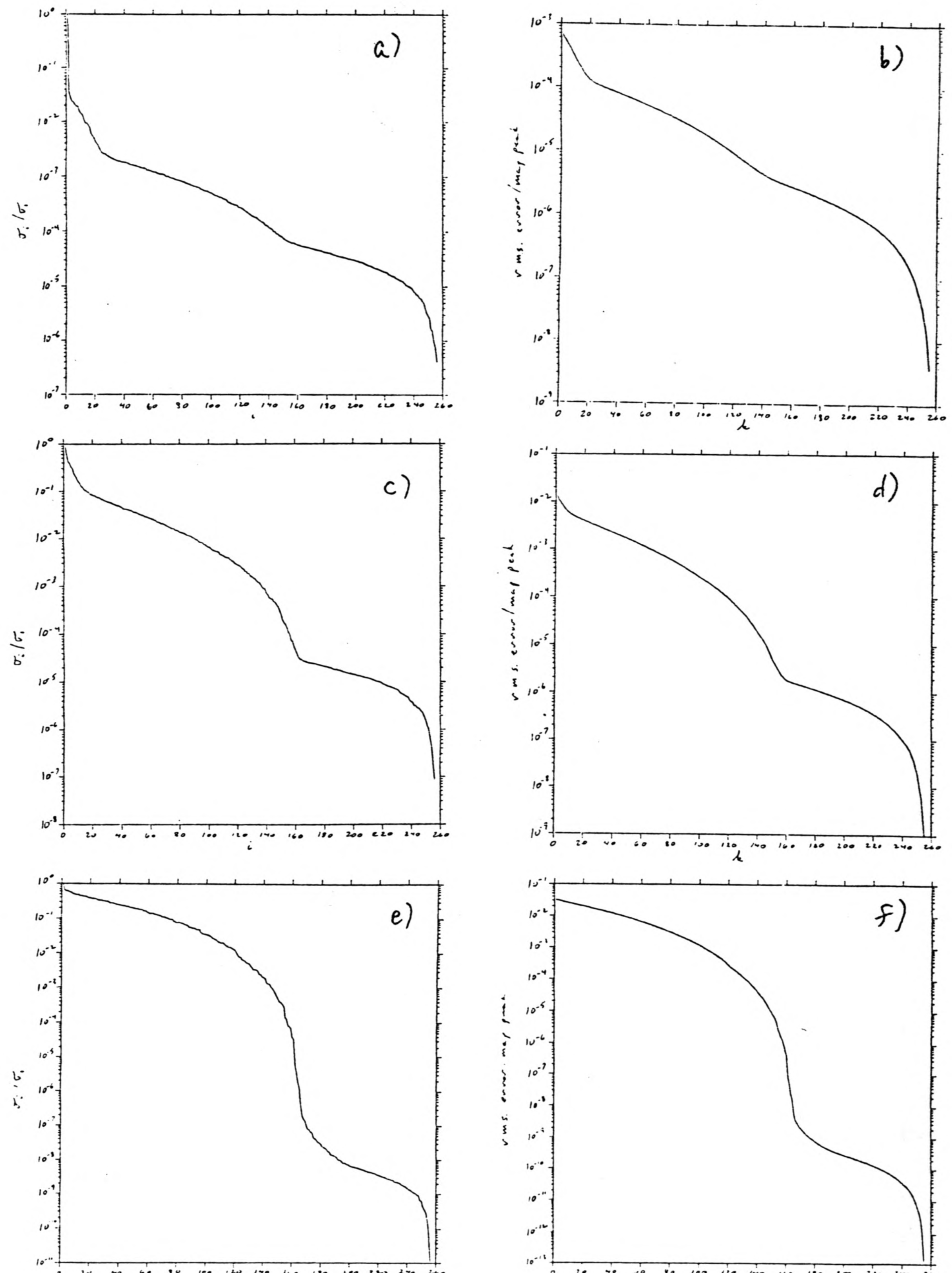


Figure 2. a) A plot of the singular values σ_i of a 256×256 image of the radio jet in NGC 6251 (see Fig. 1). b) A plot of the r.m.s. error of the k^{th} truncated SVD representation, A_k, of the same image. c) A plot of the singular values in Example 2. d) A plot of the r.m.s. error, for Example 2. e) A plot of the singular values in Example 3. f) A plot of the r.m.s. error, for Example 3.

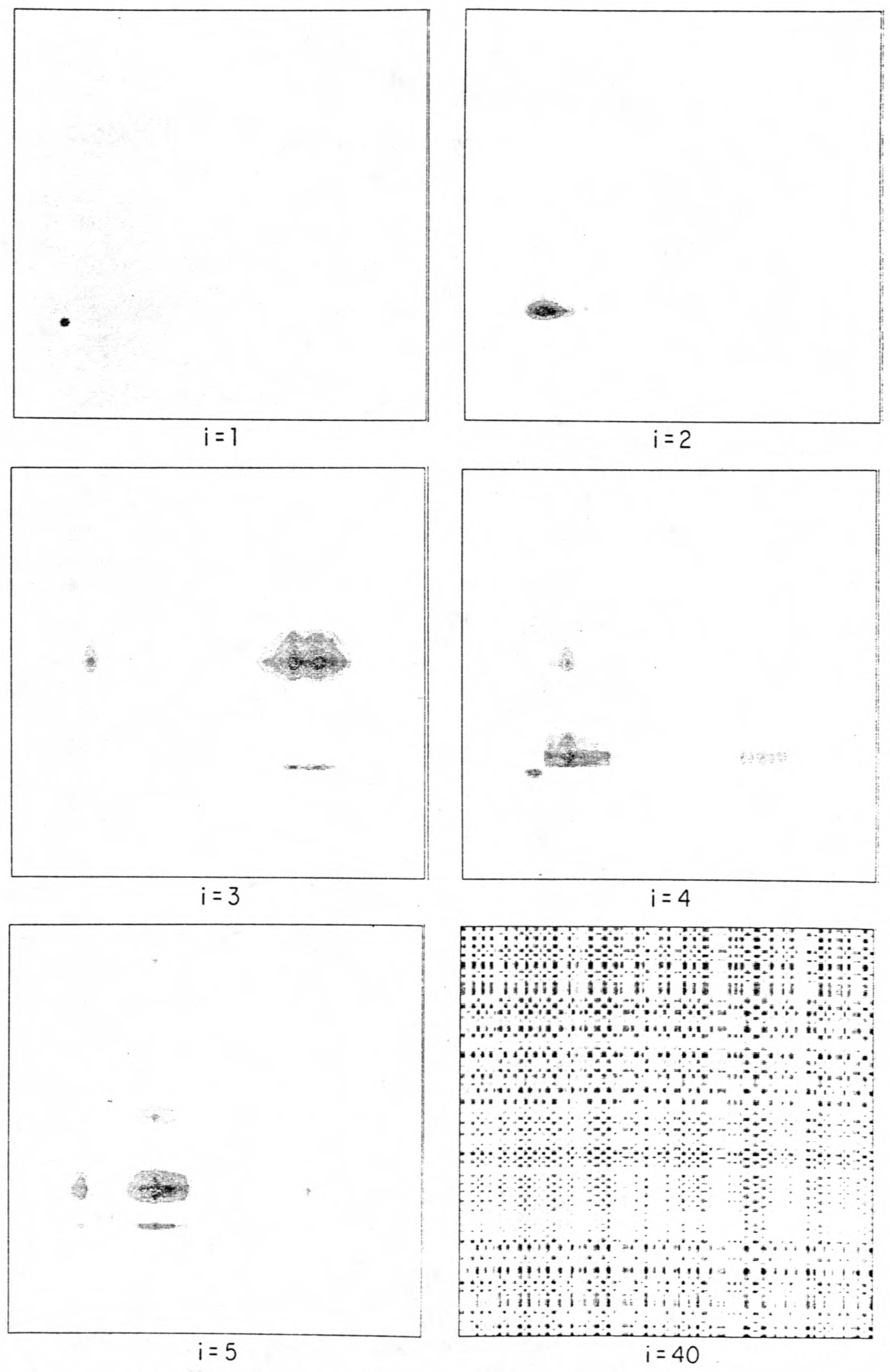


Figure 3. Selected "outer product" or rank 1 images, $\mathbf{u}_i \mathbf{v}_i^T$, comprising the SVD of the radio map of the jet in NGC 6251, for $i = 1, 2, 3, 4, 5$, and 40.

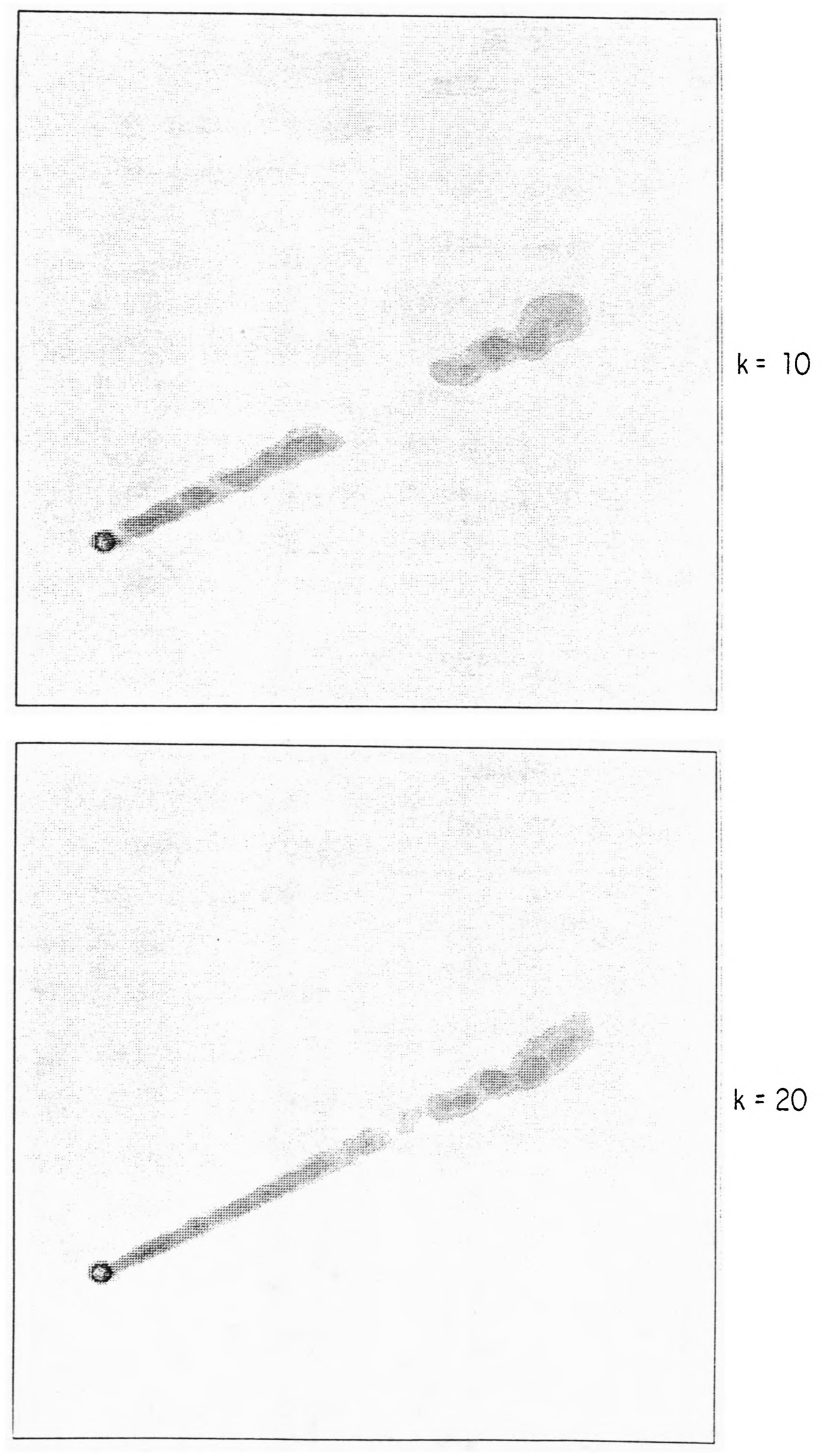
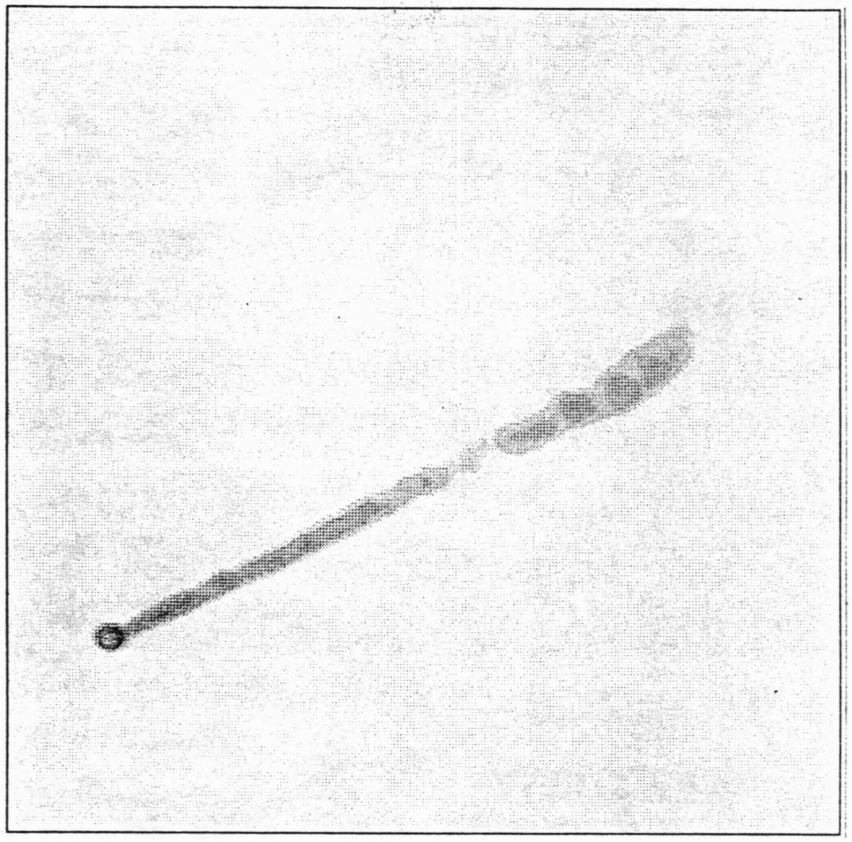


Figure 4. Photographs of the truncated SVD representations A_k of the image shown in Fig. 1, for $k = 10$, 20, 30, and 40. (Continued on next page.)



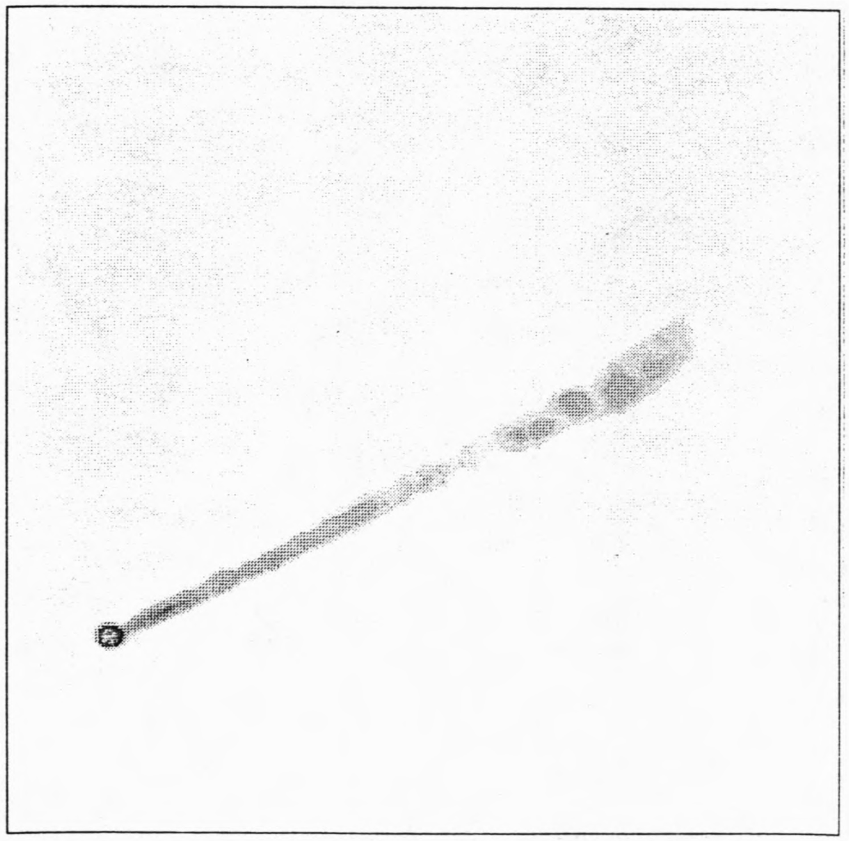
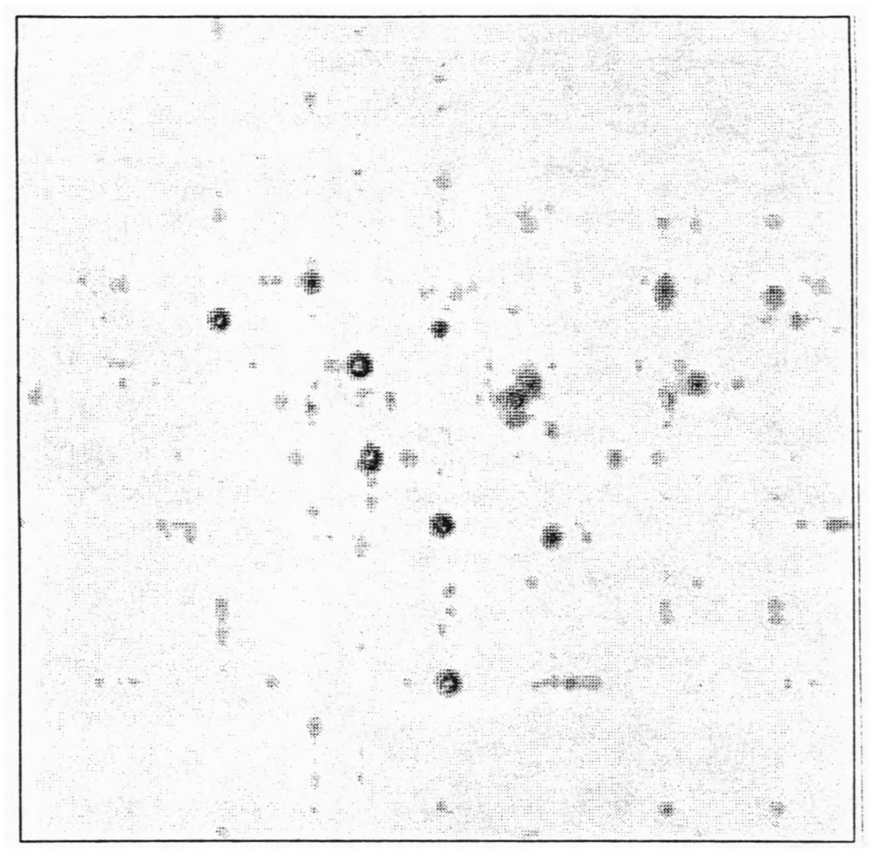


Figure 4 (cont'd).



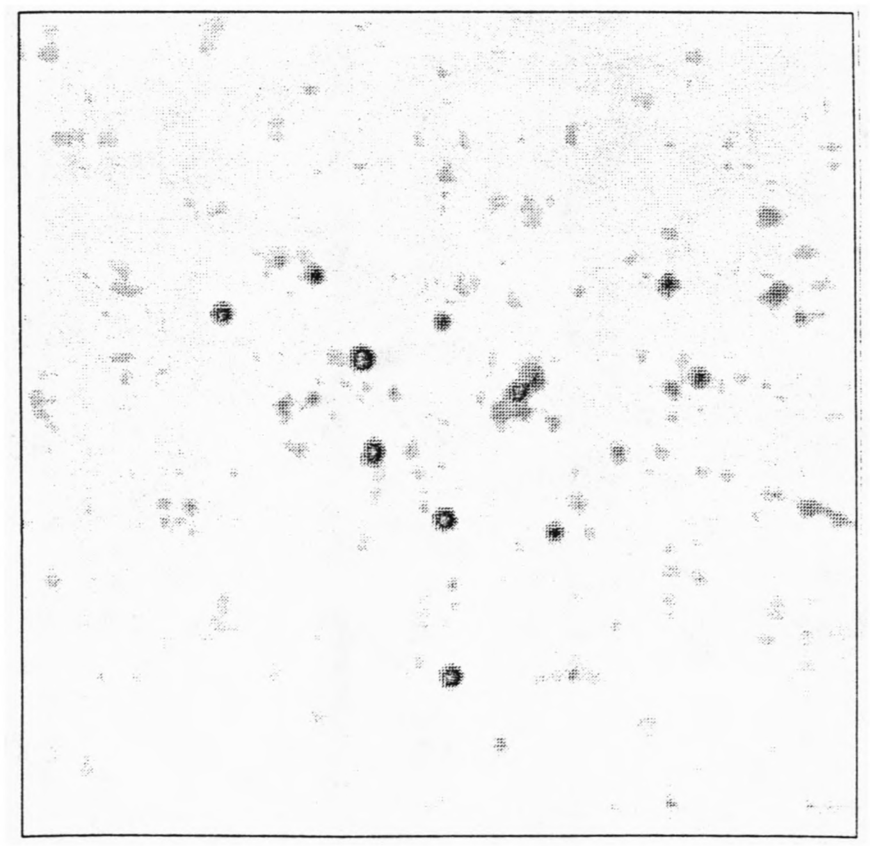
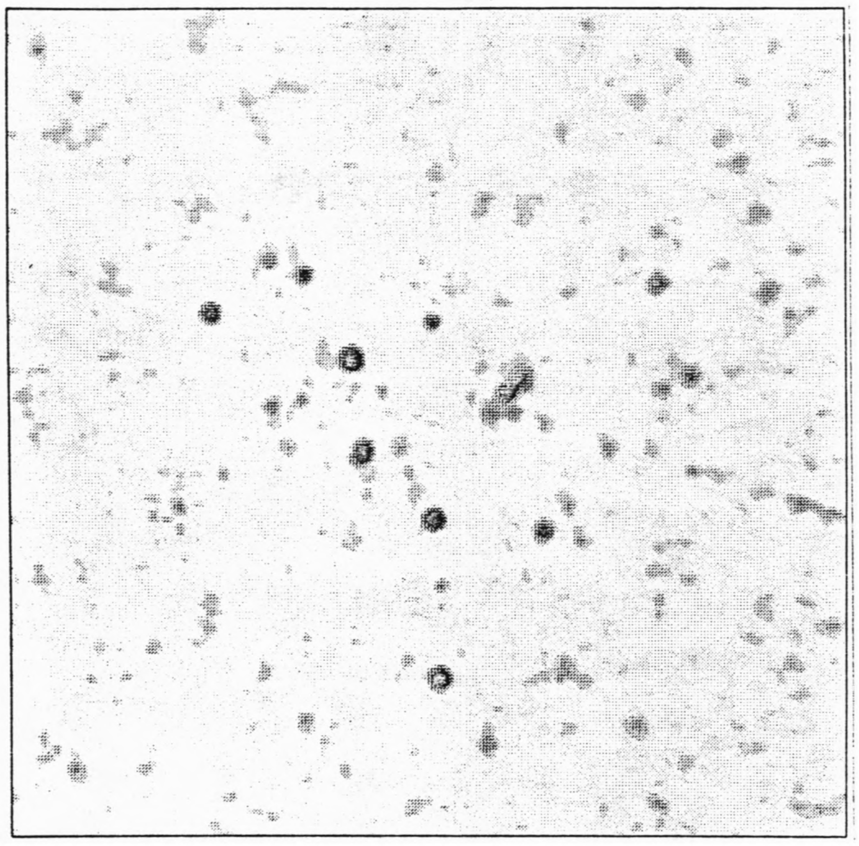


Figure 5. Truncated SVD representations A_k (k = 10, 20, 30, and 40) of a "low dynamic range" 256×256 (20 cm. wavelength) VLA map of a region—surrounding a Herbig–Haro object—which contains a large number of unresolved sources (this map the courtesy of Stephen Reynolds). (Continued on next page.)



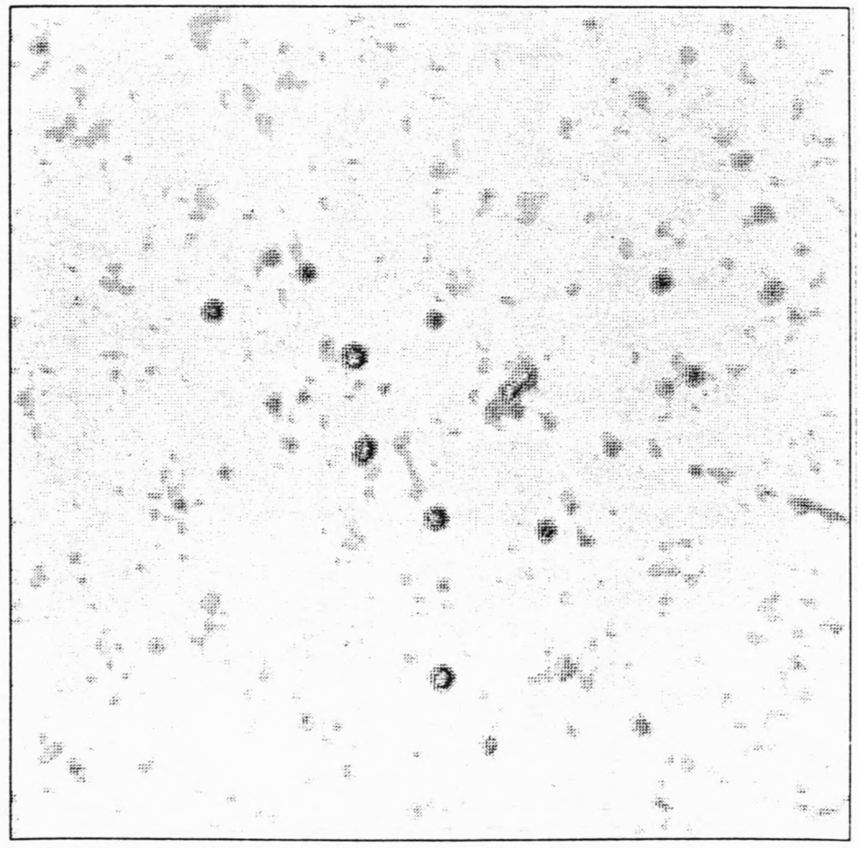
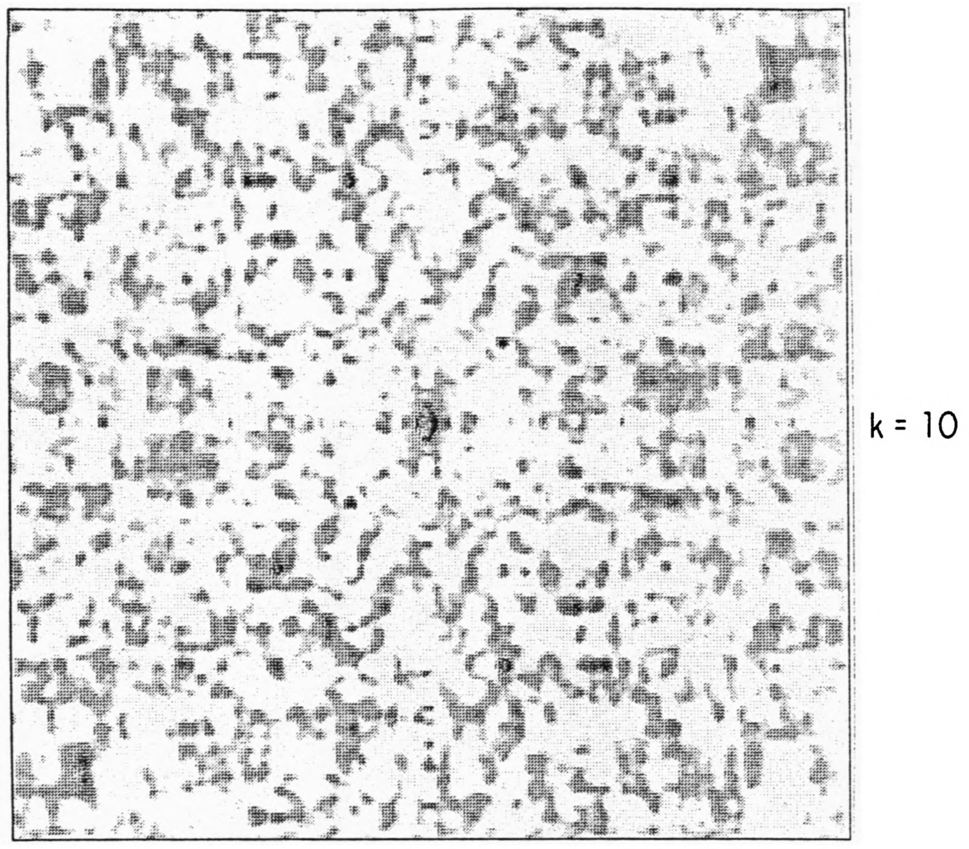


Figure 5 (cont'd).



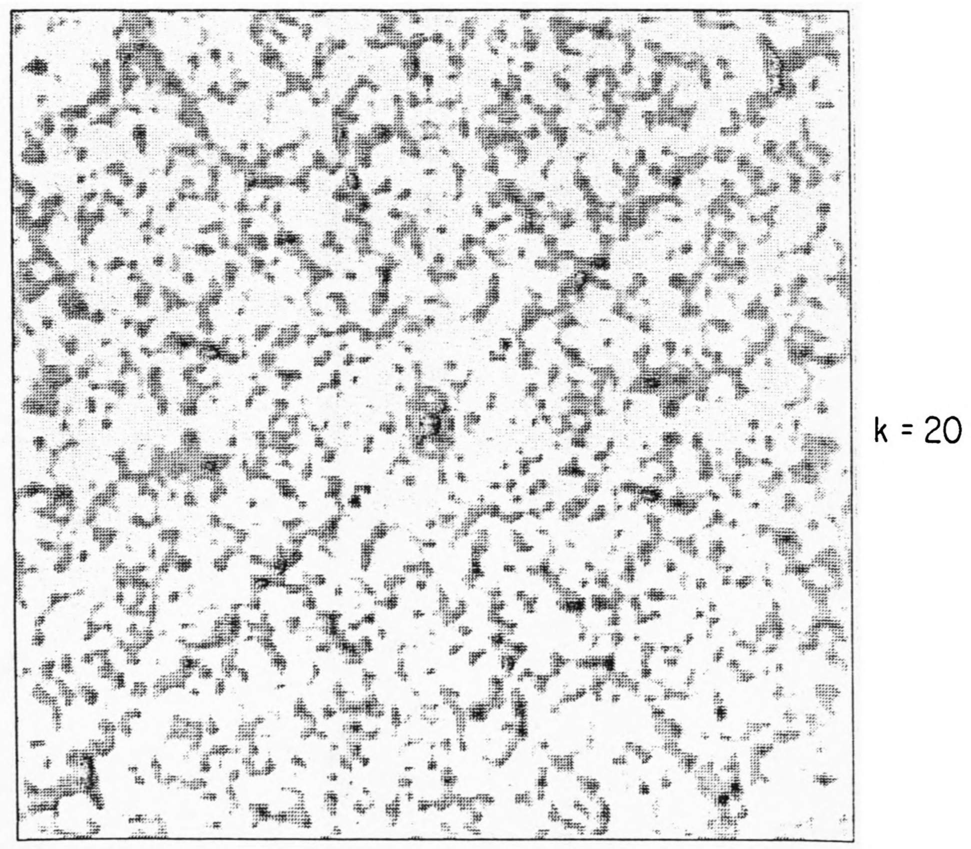
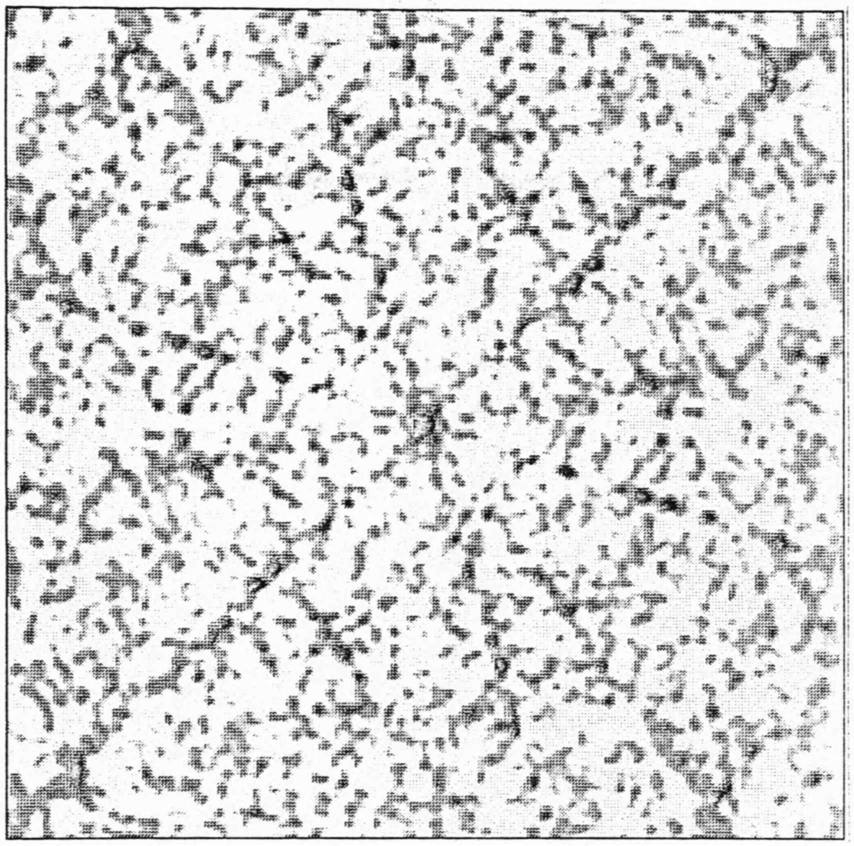


Figure 6. Select truncated SVD representations A_k ($k = 10$, 20, 40, and 80) of a "dirty beam." (Continued on next page.)



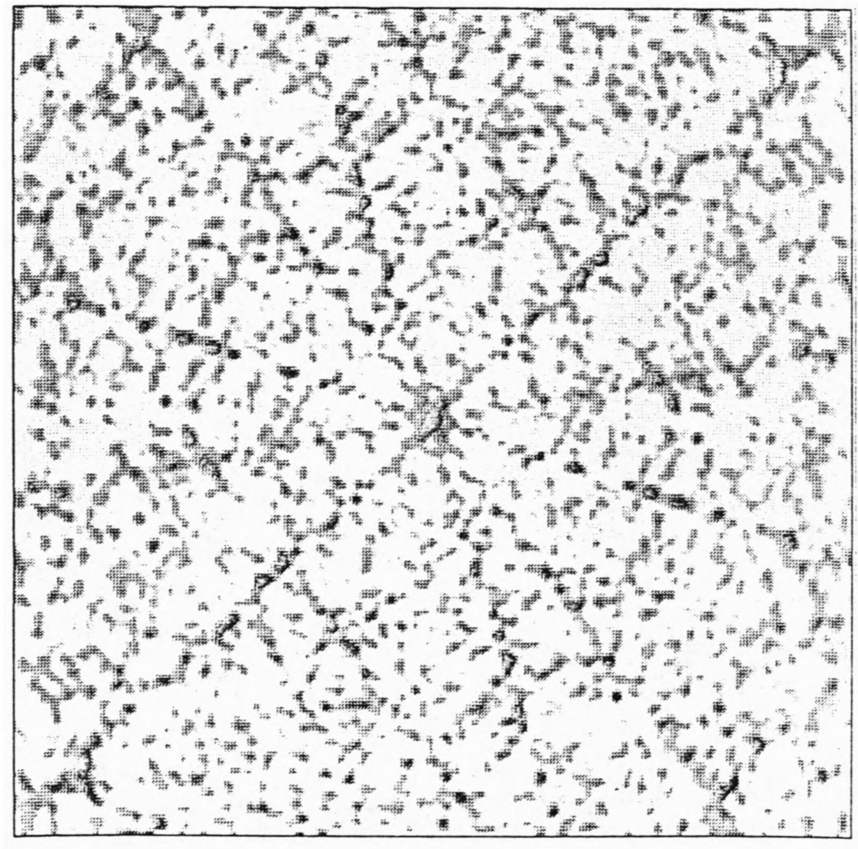


Figure 6 (cont'd).

Table 1. Huffman coding of the NGC6251 jet image.

Image	Number of bits	Entropy	# of occupied quant. levels	Ave. code-word length	Maximum length	max p_i	Compression ratio
Original	8	.2081	33	1.0381	10	.9732	7.706
	12	1.8278	111	1.8698	15	.5472	6.418
	16	5.6735	615	5.7069	16	.0416	2.804
1-D differenced	8	.1277	48	1.0248	9	.9862	7.806
	12	1.4964	104	1.6424	14	.6621	7.306
	16	5.0108	463	5.0316	16	.0630	3.180
2-D differenced	8	.1519	40	1.0280	9	.9824	7.782
	12	1.5556	89	1.6856	14	.6025	7.119
	16	4.5475	270	4.5824	16	.0822	3.492

Table 2. Huffman coding of the radioimage of the field containing the Herbig–Haro object.

Image	Number of bits	Entropy	# of occupied quant. levels	Ave. code-word length	Maximum length	max p_i	Compression ratio
Original	8	2.7077	112	2.7404	17	.2895	2.919
	12	6.6688	611	6.6929	16	.0195	1.793
	16	10.6013	3416	10.6276	16	.0016	1.506
1-D differenced	8	1.9830	90	2.0514	15	.4744	3.900
	12	5.7965	512	5.8159	16	.0357	2.063
	16	9.7402	2360	9.7607	16	.0026	1.639
2-D differenced	8	1.7309	55	1.8332	14	.5421	4.364
	12	5.0035	381	5.0289	16	.0606	2.386
	16	8.9527	1866	8.9784	16	.0043	1.782

Table 3. Huffman coding of the "dirty beam" test image.

Image	Number of bits	Entropy	# of occupied quant. levels	Ave. code-word length	Maximum length	max p_i	Compression ratio
Original	8	4.8146	93	4.8349	15	.0683	1.655
	12	8.7970	994	8.8201	16	.0049	1.361
	16	12.6414	6350	12.6690	16	.0005	1.263
1-D differenced	8	4.0112	83	4.0404	16	.1213	1.980
	12	7.9922	728	8.0167	16	.0084	1.497
	16	11.8904	6297	11.9163	16	.0008	1.343
2-D differenced	8	3.3301	65	3.3625	16	.1929	2.379
	12	7.2721	596	7.3116	16	.0136	1.641
	16	11.1834	4326	11.2142	16	.0012	1.427

APPENDIX

```
      SUBROUTINE HUTREE(N,Q,L)
C     This subroutine, a Fortran version of the Algol routine given
C  in CACM Algorithm #428, can be used to construct minimum-
C  redundancy variable-length binary codes, by an algorithm which
C  is due to Hu and Tucker.
C     On input, N is the number of letters in the 'alphabet' for
C  which a code is to be constructed, and Q(I) is the (integer)
C  frequency of occurrence, in the message, of the I'th letter.
C  On output, L(I) is the length, in bits, of the minimum-redundancy
C  encoding of the I'th letter of this alphabet.
C     (The actual encoding isn't computed by this subroutine, but
C  the routine can easily be modified to do the job.)  The code has
C  the property that both the sum of the L(I) and the maximal L(I)
C  are minimized, subject to the condition that the code be a
C  minimum-redundancy code (i.e., that sum Q(I)*L(I) be minimal).
C  As it is set up, the subroutine computes a so-called non-
C  alphabetic (Huffman) code.  If the L(I) are initialized to 0
C  rather than to one on entry, then an alphabetic code results.
C  (In the latter case, the numerical binary order of the code words
C  corresponds to the alphabetic order of the encoded letters ---
C  this constraint results in a minimum mean search time in decoding
C  (but greater redundancy, in general).)  See the CACM listing of
C  the Algol routine for more details.  (This routine doesn't seem
C  to work properly in the non-alphabetic case if any of the Q(I)
C  are equal to zero).
C     The approximate operation count is 4*N**2+2*N.  But it would
C  be possible to improve the algorithm to have run-time proportional
C  to  N log N  rather than N**2.
      IMPLICIT INTEGER (A-Z)
      DIMENSION Q(N),L(N)
      PARAMETER (NMAX=9350)
C  N mustn't exceed NMAX.
      DIMENSION P(NMAX),S(NMAX-1),D(NMAX-1)
      MAXN=1
      DO 10 I=1,N
         L(I)=1
         P(I)=Q(I)
10       MAXN=MAXN+Q(I)
      DO 70 M=1,N-1
         I=0
         PMIN=MAXN
20       I=I+1
30       IF (I.GE.N) GO TO 60
         IF (P(I).EQ.0) GO TO 20
         MIN2=MAXN
         J1=I
         MINL1=L(I)
         MIN1=P(I)
         DO 40 J=I+1,N
            IF (P(J).GT.0) THEN
               IF (P(J).LT.MIN1.OR.(P(J).EQ.MIN1.AND.L(J).LT.MINL1))
                  THEN
                  MIN2=MIN1
                  J2=J1
                  MINL2=MINL1
                  MIN1=P(J)
                  J1=J
                  MINL1=L(J)
               ELSE IF (P(J).LT.MIN2.OR.
                  (P(J).EQ.MIN2.AND.L(J).LT.MINL2)) THEN
                  MIN2=P(J)
                  J2=J
                  MINL2=L(J)
               END IF
               IF (L(J).EQ.0) GO TO 50
            END IF
40          CONTINUE
50       PT=P(J1)+P(J2)
         SUMLT=L(J1)+L(J2)
         IF (PT.LT.PMIN.OR.(PT.EQ.PMIN.AND.SUMLT.LT.SUML)) THEN
            PMIN=PT
            I1=J1
            I2=J2
            SUML=SUMLT
         END IF
         I=J
         GO TO 30
60       IF (I1.GT.I2) THEN
            J1=I1
            I1=I2
            I2=J1
         END IF
         S(M)=I1
         D(M)=I2
         P(I1)=PMIN
         P(I2)=0
70       L(I1)=SUML+1
      L(S(N-1))=0
      DO 80 M=N-1,1,-1
         L(D(M))=L(S(M))+1
80       L(S(M))=L(D(M))
      RETURN
      END
```