DataLab-J: A Signal and Image Processing Laboratory for Teaching & Research

Jonathan Campbell, Member, IEEE, Fionn Murtagh and Münevver Köküer, Member, IEEE

School of Computer Science
The Queen’s University of Belfast, Belfast, BT7 1NN.
email: jg.campbell@qub.ac.uk
Tel. +44 28 9027 4623.

November 14, 2004
DataLab-J is a software signal and image processing laboratory which has proved effective both as an educational “workbench” and in practical operational use. It requires: a pedagogical tool, a research environment, and a fully operational data analysis system, i.e. it is used not only in undergraduate engineering courses, but in graduate study and general research. The system must be easily extendable, e.g. to allow undergraduates to perform practical programming of standard digital filters and image processing algorithms, or to provide a realistic platform upon which novel algorithms can be implemented. On a further dimension, the system must handle seamlessly and efficiently three broad data types: digital signals (sequences), images (possibly multiband), and multivariate data sets. The system is implemented in the programming language Java™.

DataLab-J has been operational for four years, has been used in an undergraduate image processing course, and as a platform for a great many dissertation projects. In addition it is in everyday use within a university signal and image processing research group.

Key Words: DSP, image processing, pattern recognition, software engineering, education.
1 Introduction

Signal processing, image processing, and related topics, such as pattern recognition (the core science of data mining), need to be taught on an increasingly broad range of courses. However, students studying these topics now are more likely to be familiar with computer algorithms and be skilled in computer programming than be very comfortable with applied mathematics which traditionally has been the sole basis for such courses. Mathematics cannot be dispensed with entirely, but our objective is to make the subjects referred to above more accessible to students from a wider range of abilities and inclinations. In addition, it is noted that even the most modest personal computer is now capable of performing as a powerful scientific workstation.

In spite of the existence of various widely used commercially available software packages, of which MATLAB [1] is probably the most popular, and also software available as public domain packages, such as Octave [2], DataLab's chief attractions derive from the availability of the complete source code and from the fact that the system was designed for ease of programming particularly from the point of view of the wide range of carefully chosen functions/abstractions available. Insight may be gained by probing of algorithms and the inspection of intermediate data, something which is not possible in a third-party system. Moreover software engineering is now a major skill for electronic engineers, so that, in both undergraduate and graduate courses, it is valuable to provide cross-linkages between signal processing related courses and software engineering. Also, there are distinct advantages in having a teaching aid that can double as a research tool, thus providing an easy bridge between learning and doing and between undergraduate and graduate use. This provides an environment in which undergraduate students, graduate students and professors speak a common language.

The requirements for DataLab are based on the following objectives: (a) utility as a pedagogical tool, (b) suitability as an experimental algorithm development laboratory, and (c) performance efficiency and effectiveness as an operational data analysis system. Further underlying design goals are simplicity and ease of programming.

DataLab includes a lot more than image processing: everything from signal processing to monochrome and multivariate image processing, multivariate statistics and data mining. In [3] the long history of the DataLab project is outlined including mention of the significant advantages of the combination, in one package, of this range of functions.

Section 2 elaborates on the scope and requirements for DataLab, in particular, the data objects that are involved, plus the sorts of processing functions that are to be applied to the objects. Section 3 describes the design and implementation. Section 4 discusses the major issues and critical decisions in such designs.

2 Data Objects and Operations

DataLab is concerned with signal processing, image processing, and pattern classification and regression/estimation. From these three general data types can immediately be identified: (a) one-dimensional digital signals [4]; (b) two-dimensional images, perhaps multiband [5]; and (c) multivariate data sets [6].

The following gives an abstract mathematical outline.

2.1 Digital Signal

The primary data object is a data sequence [4]

\[ x[n], \quad n = 0, 1, 2 \ldots, N - 1 \]

where \( N \) is the length of the sequence.

At this stage no commitment is made as to representation of the sequence; it must be possible to read (get) the numerical value at index \( n \) and to write (put) a new value; in other words, there is no commitment to representation as a Java array. An abstract sequence \( x \) is defined as a function taking natural numbers (indices) to reals (values), \( x : \mathbb{N} \rightarrow \mathbb{R} \).

A typical digital signal process is one-dimensional convolution:

\[ y[n] = \sum_{m=0}^{N-1} x[n-m] h[m] \]

2.2 Digital Image

Here the primary data object is a two-dimensional image [5]

\[ f[r,c], \quad r = 0, 1, 2 \ldots, N_r - 1; c = 0, 1, 2 \ldots, N_c - 1 \]

where \( N_r \) is the number of rows, \( N_c \) number of columns.

Again, it is tempting to commit to a representation of an image as a two-dimensional array; however, at this stage an image, \( f \), is defined as a function from a discrete lattice to a real number, \( f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R} \).

A typical image process is two-dimensional convolution
\[ g[r,c] = \sum_{k=r-M_r+1}^{r} \sum_{l=c-M_c+1}^{c} f[k,l] h[r-k,c-l] \]

where \( M_r \) is the number of rows, \( M_c \) number of columns in the convolution kernel.

### 2.3 Multivariate Data Set

A multivariate data set used in a pattern recognition context [6] may be summarized as follows:

\[ X_T = \{ x_i, \omega_i \}, i = 0, 1, 2, \ldots, N_s - 1 \]

where \( N_s \) is the size of the sample, \( x_i \) is a typical datum, and \( \omega_i \) is its class label.

Given a “training” data set, \( X_T \), and a new vector \( x_j \), the pattern classification problem may be stated as the inference of its class label:

\[ \omega_j = f(x_j; X_T) \]

### 3 Analysis and Design

#### 3.1 Data Representation

**Image and Signal** A simple solution is to represent a monochrome image as a two-dimensional array. Multiband images may be represented as a Java Vector [7] of images. Vector is used, instead of a third dimension of an array, for its generality and ease of resizing when additional bands are to be inserted.

A digital signal may be represented as an image with just one row.

**Multivariate Data Set** For unordered data collections (sets) one row of an image may again be used, taking care to avoid processes which make use of spatial or sequential proximity of data (i.e. image or signal processes).

**Class Labels** Labels, i.e. typically class labels in pattern classification experiments, are stored in a data structure that corresponds to the associated data except, by their nature, labels may be represented by integers.

**Ancillary Data** From the point of view of data structures, regression may be viewed as similar to pattern classification except that the output, \( y \), the ancillary data, is continuous rather than discrete. Thus, ancillary data are stored in a data structure that corresponds to the associated primary data.

**Fuzzy Labels** Fuzzy labels are appropriate in a variety of cases, for example fuzzy c-means clustering, [8] and fuzzy supervised classification [9]. These are not the only cases; in problems involving conventional classifiers [10], it may be appropriate to retain intermediate (posterior) probabilities.

In contrast to crisp (discrete) labels, for fuzzy labels one-of-N coding must be used; thus fuzzy labels are represented by real valued vectors; therefore their structure is similar to vector data or vector ancillary data.

### 3.2 Object Design

A summary UML (Unified Modeling Language) diagram is given in Figure 1.

**Class Im** Figure 2 shows part of a class Im, which is the basic DataLab class. Its data field is a simple float two-dimensional data array; as has already been indicated, a digital signal (data sequence) and likewise an unsequenced data collection may be implemented by a single row Im object.

In the spirit of data abstraction [11] data objects, e.g. of class Im, are viewed as black boxes containing the private representation data (information hiding), but these data may be accessed only through defined operations – the methods or interface functions.

Class Matvec provides no data abstraction, but is a collection of vector and matrix operations, and incorporates some of the Basic Linear Algebra Subroutines (BLAS) Java conversions provided by [12].

With the current approach, external software can be used unmodified, and DataLab software tools are usable on other projects. Moreover, since Java (native) arrays are in fact objects, Java provides their memory management, i.e. they are allocated on the heap, and, eventually, garbage collected [7].
Class **Imd**  Class **Imd** represent a multiband image. A Java **Vector of Im objects** is used; consequently, the number of bands/images can be altered dynamically; given the high-level of this class, the overhead of the reduced access efficiency of **Vector** (versus plain array) is insignificant. The methods of **Imd** follow the same pattern as **Im**.

Class **Dld**  Class **Dld** represents the overall DataLab data object. As has been described, it consists of four data members, three of which are optional: primary data – **dat**, optional labels – **lab**, optional ancillary data – **anc**, and optional fuzzy label data – **fla**.

Class **Cmd**  This class handles the command-line user interface by reading commands from the keyboard (or from a file, which provides a rudimentary script facility), and by decoding the commands and their parameters ready for the dispatch of the appropriate process.

Class **Dlj**  This class provides the main program:

1. Calls the command fetcher-and-decoder (class **Cmd**);
2. Dispatches the commands to DataLab functions;
3. Manages the computational environment of **Dld** objects as they are created and/or destroyed during an interactive session. Currently a **Dld** object list implemented as a Java **Vector** is used, with objects referenced simply by their position in the list. In the next version plans are to incorporate an environment class which will be implemented using an appropriate dictionary/symbol-table class. Very basic Help information is included in the menu data.

**DataLab functions**  A typical DataLab image processing function – two-dimensional convolution – is shown in Figure 3. Another example, in this case multivariate data analysis, is the Karhunen-Loève transformation (**kl**) implementation shown in Figure 4.

A complete list of DataLab functions is given in Appendix A.

### 4 Discussion

#### 4.1 Data Model

As can be seen in the previous section, the data model is extremely simple. In previous systems, the limitations of the raster/lattice data model was questioned. Would a pyramidal representation be worth considering? More importantly, would it be easily understood by client programmers and users. Moreover, what about a symbolic or relational representation for higher level processing?

In fact, earlier DataLab [13, 14] representations have been simplified. No longer may sequences and images have non-zero-based indices; this was possible to fake in C [15]. However, this feature is one that seemed nice to have; in practice, it was never used; but it did require significant programming.

#### 4.2 Data Implementation

Although the image data model is clearly that of a function on a restricted lattice, so that a grid based raster storage is natural, our data abstraction properly denies any statement about the actual implementation of the storage. Our data could be stored in some sparse structure, using for example run-length encoding.

Yet another possibility is the inclusion of functional representation, e.g. \( f[r,c] = \text{constant}(1.0) \), or \( x[n] = \sin(2\pi an/N) \).

#### 4.3 Descriptor Data

By descriptor data we mean data such as image size, flags signifying the existence of optional data blocks etc. What is remarkable about the classes described in the previous section is the complete absence of descriptor data fields. This is because Java provides built-in methods for interrogating the size of array and **Vector** objects.

The absence of descriptor data may seem a minor point but it has some significance. Related is the decision between store or compute data such as as statistics, e.g. covariance matrices. Since they seem computationally expensive to compute, there is a temptation to store them. However storing brings its own expense [3]: stored statistics must be kept up to date so that the data object at all times retains internal consistency.

Measures such as lazy evaluation and memoising [16] are partial solutions, but have been rejected from the current system owing to pressure from our design goal – simplicity and ease of programming.
4.4 Multiple Numeric Types

In previous systems it was necessary to store, at least initially, large images in byte and integer format. However fixed-point computations are at best inconvenient, so that there is always a temptation, where feasible, to work in floating-point. In the current system, all numerical data are stored in float – though of course this is hidden by the data abstraction.

4.5 Software Architecture

The software architecture based on function-table/shared memory/objects has some drawbacks compared to other architectures, [11]. The chief drawback is that of configuration management. It is difficult to extend the system across distributed sites without serious divergence occurring (multiple configurations come into being). A UNIX filter architecture is one notable and attractive alternative.

To solve these problems, it may be possible to draw on lessons learned from so-called open-source projects, [17].

4.6 Ease of Programming

The chief motivation for the design goals of simplicity and ease of programming is the objective of development efficiency. Implementation of new algorithms and ongoing projects must be easy.

DataLab-J provides a number of methods of use/programming: (a) command line interaction, for example, create a test image, add noise, apply an edge-detector, (b) script-level programming via sequences of commands contained in “script” files which are invoked like the commands in (a), (c) programming using the native Java language, using appropriate high-level library calls; see, for example Figure 4, which accomplishes a significant task (Karhunen-Loève transform) in remarkably few statements; it is noted that it is possible to call one DataLab operation from within another; (d) obviously, programmers may develop at any level, though it will rarely ever be prudent to tamper with basic data class (e.g. Dld) internals.

DataLab-J has been used for the practical part of in a course on Image Processing and Pattern Recognition [18].

4.7 Efficiency

Although, so far, run-time efficiency has been ignored, this is clearly an issue owing to the interpreted execution of Java. Even in exploratory algorithm development, rapidity of feedback may be crucial; processes that take a long time to run discourage exploration.

Regarding time efficiency, comparisons between DataLab-C (programmed in C) [13] and DataLab-J, using ordinary, and Just-in-Time (JIT) Java interpreters, have been carried out [19], yielding the following indicative figures: C programming language, under the Linux operating system, DataLab-C, 1 time unit; Java 1.1.5 under Windows-95, 4.5 units; Java JIT (just in time) compiler under Windows-95, 2.3 units. Because these tests are time consuming to carry out, the figures have not been updated for recent compilers and interpreters. However, the indicative nature of the comparisons remain valid; the major conclusion is that the reduced performance is quite tolerable.

5 Conclusions

DataLab-J is a portable software package for research and development of algorithms in signal processing, image processing, pattern recognition, estimation, and general multivariate data processing. Within often competing constraints, it achieves three significant goals. First, it provides comfortable environment for developing and experimenting with algorithms and thereby facilitates the rapid implementation of new algorithms with no systems programming; second, it offers reasonably efficient use of machine resources so that at the early stages of DataLab-J in 1998, significant work could be done on a typical student laboratory machine, a Pentium 200 MHz with 32 MB RAM. Finally, as a pedagogical tool, it has proved its educational worth by providing the basis for a range of practical activities within a course on image processing. Moreover, in the context of undergraduate (and postgraduate) laboratories its cost-effectiveness (free) cannot be disregarded.

A Functions currently available

A.1 File

- Read (various formats)
- Save
- Print (various formats)
A.2 General

- Show current environment
- Provide menu
- Help

A.3 Data generation

- Image with rectangle
- Image with disc
- Image with random lines
- Uniform noise
- Gaussian noise
- Sine wave

A.4 Arithmetic

- Add, subtract, multiply
- Integrate – signal/sequence
- Extract sub-image, subsignal
- Zoom
- Transformations: log, dBs, threshold, scale, normalize, negate, contrast stretch, histogram equalization, remove d.c. component

A.5 Image Morphology

- Binary: erode, dilate, open, close, and, or, xor
- Gray-level: erode, dilate, open, close

A.6 Pattern Classification and related

- Bayes-Gauss classifier
- Nearest mean, nearest neighbour, k-nearest neighbour classifiers
- Fuzzy rule-base classifier
- Probabilistic neural network classifier (kernel density estimation)
- Fisher discriminant analysis, multiple discriminant analysis
- Karhunen-Loève transform
- Whitening transform
- C-means clustering and fuzzy c-means clustering
- Kohonen self-organizing map
A.7 Regression, Estimation

- Generalized regression neural network
- Fuzzy rule-base regression
- Kalman filter

A.8 Wavelet transforms

- One-dimensional, forward, inverse, Daubechies four-coefficient basis
- Two-dimensional, forward, inverse, Daubechies four-coefficient basis
- Two-dimensional, forward, inverse, Haar basis
- À-trous [20] redundant transform, one-, two-dimensional

A.9 Discrete Fourier Transforms

- Forward: real, imaginary; amplitude, phase
- Inverse
- Walsh-Hadamard transform

Acknowledgement

The authors wish to thank Andy Fugard for producing the UML diagram.

References


Figure 1: DataLab-J — Summary UML diagram
package dlj;
import dlj.*;
import java.io.*;

public class Im{

    private float[][] dat; // basic data

    // constructors
    public Im(int nr,int nc){dat= Matvec.make(nr,nc); } // etc...

    // accessors
    public int nrows(){ return dat.length; }
    public int ncols(){ return dat[0].length; }
    public float get(int r, int c){ return dat[r][c];}
    public float max(){ return Matvec.max(dat); } // etc...

    // mutators
    public void put(float val, int r, int c){dat[r][c]= val; }

    // etc...
}

Figure 2: Class Im (part of)
/**
 * This method performs a 2-d convolution, f convolved with h: f(x)h.
 *
 * @param f the image
 * @param h the kernel
 * @param return g = f(*)h.
 *
 */

public static Im conv2d(Im f, Im h) {
    int nr = f.nrows(); int nc = f.ncols();
    int v = h.ncols(); int w = h.nrows();
    Im g = new Im(nr, nc);
    float sum;
    int sr, sc, r, c, k, l, m;

    float[] hh = h.to1d(); // for efficiency: kernel -> 1-d.

    for (r = 0; r < nr; r++) {
        for (c = 0; c < nc; c++) {
            sum = 0.0f;
            /*
             * g[r,c] = sum sum f[r-k,c-l] * h[k,l]
             * k=0 l=0
             */
            m = 0;
            for (k = 0; k < w; k++) {
                for (l = 0; l < v; l++) {
                    sr = r - k; sc = c - l;
                    if (sr < 0) sr = -sr - 1; // reflection: -1->0, -2->1
                    if (sc < 0) sc = -sc - 1;
                    sum += f.get(sr, sc) * hh[m]; // h.get(k,l);
                    m++;
                }
                g.put(sum, r, c);
            }
        }
    }
    return g;
}

Figure 3: Image Processing Example: Two-dimensional Convolution
package dlj; import java.io.*;
import linear_algebra.*;
public class DldMa{
    public static Dld kl(Dld source, Dld train){
        PrintStream out=System.out; Dld dest= null;
        if(!source.ndimsdEqual(train))return dest;
        int nd= train.ndimsd(); int nr= source.nrows();int nc= source.ncols();

        dest= new Dld(source, "la"); //copies ‘lab’ and ‘anc’
        //compute mean and covariance of ‘training data’
        double[][] s= train.covdDouble(); double[] m= train.meansdDouble();

        double eval[]=new double[nd]; double work[]=new double[nd];
        double e[]=new double[nd]; double u[][]=new double[nd][nd];
        double v[][]=new double[nd][nd]; double x[]=new double[nd];
        double y[]=new double[nd]; int job=10;
        try {
            SVDC_j.dsvdc_j(s,nd,nd,eval,e,u,v,work,job);
        } catch (SVDCException svdce) {
            System.out.print("\nThere was an SVDC_j exception\n" +
            "The info value from SVDC_j was " + svdce.info +".\n");
        }
        double ut[][]= Matvec.mattran(u); //transpose eigenvector matrix
        for(int r=0; r<nr; r++){ //source vector to be transformed
            for(int c=0; c< nc; c++){
                x= source.getpvecdDouble(r,c); //source vector to be transformed
                Matvec.axpy(-1.0, m, x); // remove mean
                Matvec.matvec(ut, x, y); // transform: y = U’x
                for(int d=0; d< nd; d++){
                    dest.putd((float)y[d], d, r, c); //store transformed vector
                }
            }
        }
        return dest;
    }
}

Figure 4: Multivariate Data Analysis Example – K-L Transform