2010

Open Source Analysis of Biomedical Figures

David Shao
San Jose State University

Open Source Analysis of Biomedical Figures

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

David Shao

Spring 2010

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Sami Khuri — Department of Computer Science

Dr. Robert Fowler — Department of Biological Sciences

Dr. Mark Stamp — Department of Computer Science

APPROVED FOR THE UNIVERSITY

Abstract

With a selection of biomedical literature available for open access, a natural pairing seems to be the use of open source software to automatically analyze content, in particular, the content of figures. Considering the large number of possible tools and approaches, we choose to focus on the recognition of printed characters. As the problem of optical character recognition (OCR) under reasonable conditions is considered to be solved, and as open source software is fully capable of isolating the location of characters and identifying most of them accurately, we instead use OCR as an application area for the relatively recent development of compressive sampling, and in particular a fast implementation called compressive sensing matching pursuit (CoSaMP). Compressive sampling enables recovery of a signal from noisy measurements if certain rigorous mathematical conditions hold on previously measured samples, the mathematical conditions stating that measured samples must be essentially nearly perpendicular, orthogonal, to each other. For OCR, we investigate approximating such nearly orthogonal samples by selecting random curves, then using CoSaMP to determine a sparse number of samples approximating character shapes. We compare the accuracy of three different methods of applying CoSaMP to the problem of matching a blurred character to one of a set of previously sampled characters. We show numerically that selecting random curves does not satisfy the strict mathematical conditions for compressive sampling theory to guarantee optimal solutions. However, character matching strategies using CoSaMP transformed characters can be developed whose accuracy is roughly comparable to a baseline comparison of blurred characters with original characters, suggesting that OCR is an example where the performance of compressive sampling methods declines gracefully as conditions are weakened on the sampling matrix.

Acknowledgements

I would like to thank first my project advisor Dr. Sami Khuri of San José State University for the freedom to pursue a topic of interest, for the patience for guiding me so many years, for conducting the bioinformatics seminar that has exposed me to both a wide variety of machine learning algorithms and to their application, and for the ever practical advice on when and how the work must be finished.

I would like to thank Dr. Robert Fowler for his continual encouragement and education of computer science students in genetics. The random curve approach I use for sampling is my vague analogy for certain methods using many fragments for matching in biology. I would also like to thank Dr. Mark Stamp also of San José State University for his continued interest in my many potential topics and for giving me a perspective on networking that I applied for insight into compressive sampling applicability.

Apart from those on my committee, I would also like to give special thanks to Dr. Leslie Foster also of San José State University for giving me the fundamentals on numerical analysis and applied linear algebra to analyze my topic mathematically.

Most of all I would like to thank my wonderful wife Amy Shao who has been my constant companion for so long and who has continuously given me strength seeing how she has been able to survive and overcome life-threatening illness and disability.

Contents

List of Figures

List of Tables

1 Introduction

As discussed in [20], the sheer number of articles contributed each year to the biomedical literature makes it difficult for researchers to make links between separate sources to for example realize that there is the potential for a new drug or to make a connection why a genetic factor may explain why a drug works better or worse for certain classes of people. There is a need for automatic processing of biomedical literature to unearth such relationships across separate sources, especially with the accumulation of various biomedical literature databases. We now discuss how we continuously narrowed our focus until we developed a specific problem solvable using freely available open source software.

1.1 From open access to OCR

The approach taken in [20] is to use the Medline database [13], which has abstracts that have been curated to use the MeSH controlled biomedical vocabulary to search for such relationships, an effort that has continued as part of the Pharmacogenomics Knowledge Base [17]. Given that useful information can be extracted alone from abstracts, much more should be obtainable from full articles.

In addition to the Medline database, the United States National Institutes of Health (NIH) offers a full article subset of the biomedical literature through PubMed Central [18]. Such open access articles include text pre-parsed in an XML format and journal quality resolution figures. The existence of open access article databases has allowed for the development of systems that can use the quantity of figures for statistical pattern recognition training such as described in [2].

However, with statistical training there is the disadvantage that at some point someone quite knowledgable in biology must manually annotate a collection of sources to determine ground truth so that methods can be compared. The need for such knowledgable people is precisely the difficulty that automated tools are supposed to help overcome. For our research, we decided to focus on what can be deduced without having to consult a biologist; therefore, we reduce the problem of analyzing a figure to what we can see for ourselves. At the very foundation is the ability to recognize that a certain portion of a figure represents a character in some language and then to deduce what that character is, a field of inquiry known as *optical character recognition* (OCR).

The field of optical character recognition has been slowly and methodically been developed for decades with concrete problem solution, often combining multiple techniques, steadily advancing the field until problems such as OCR for English in a known font at reasonable resolution is essentially considered a solved problem, see for example books such as [14] or [4].

1.2 Quest for a simple algorithm

The references [14] or [4] show that machine learning algorithms such as neural nets, hidden Markov models (HMMs), and support vector machines (SVMs) have already been extensively investigated for OCR. But we have also observed that to code many machine learning algorithms, at some point we have to accept on faith some software package implementation whose theory would take months for us to fully understand. We had initially intended to use a statistical learning method known as a support vector machine (SVM), for which there is a rapidly maturing software package Shogun supported on many platforms and interfacing with many languages [23]. For support vector machines the geometry of the space for separating different classes is specified by a kernel that calculates the dot product between two vectors; however, we have no insight as to which kernel to use, nor could we see an easy way to experiment with extending application such as the ability to search for Greek letters.

Fortunately we have discovered that just in the past few years, post-2004, a new approach to processing signals, compressive sampling, has been developed that has a ready interpretation for our analysis of figures. For many implementations of compressive sampling there is also a need for something similar to solving complicated optimization problems, but again we are fortunate that just within the past three years, a much faster implementation of solving compressive sampling type problems has been developed that only uses concepts in applied linear algebra we are familiar with. And since this method called CoSaMP is by nature extremely fast, we are able to obtain results in just a few minutes, training included, so that we can rapidly develop intuition about how our OCR results can be explained through linear algebra.

From the start we have recognized that we are hardly likely to develop a new OCR method that can surpass open source solutions let alone commercial ones that evidently are quite suitable for their purposes. Neither are likely to discover that a new algorithm is better than any before. Instead we paradoxically seek what will in all likelihood be an analogous but suboptimal area of extension so that we can gain insight into a relatively new algorithm, CoSaMP. Furthermore, to bring the discussion back to our purported topic of open source, because CoSaMP is expressed in fundamental operations of linear algebra, there are many open source projects that provide suitable implementations. For our applied linear algebra needs, we easily found a package developed at CERN that works with the Java programming language.

In Chapter 2, we show that open source software, is capable of finding candidate connected components for characters, of basic OCR for printed English text of sufficiently high resolution, and of scripting character generation and conversion including blurring for test images. In Chapter 3, we discuss the basic theory of compressive sampling and how we plan to use random curves for our sampling matrix. In Chapter 4, we explain basic linear algebra concepts of near orthogonality and the singular value decomposition, in particular, showing how the singular value decomposition can be used to solve the least squares problem at the heart of each CoSaMP iteration

and how the singular value decomposition's diagonal entries enable us to find the condition number, the number we use to show our random curve sampling matrix does not satisfy the full theoretical conditions for compressive sampling solution optimality. In Chapter 5, we present a complete statement of the CoSaMP algorithm and explain our use of the singular value decomposition to solve least squares instead of using iterative methods. In Chapter 6, we provide initial examples of CoSaMP applied to approximate character images using random curves, show the condition numbers in CoSaMP iterations do not satisfy the strict conditions of compressive sampling theory, and discuss our strategies for using CoSaMP to match a blurred image of a character to previously sampled characters. In Chapter 7, we show the results of comparing these three strategies using CoSaMP to the baseline strategy not using CoSaMP, achieving our goal of character recognition rates roughly comparable to that of the baseline strategy. In Chapter 8, we discuss our conclusions, and in Chapter 9, we indicate further directions for research.

2 Open source OCR

Given a figure that has text composed of characters, we would like to know that open source software is capable of

- Detecting regions whose characters can occur and isolating such areas for character identification,
- Given an area known to have characters, identify what these characters are.

2.1 Existing OCR

There is commercially available optical character recognition software for English and other languages that is considered to be fairly comprehensive, although there are still some gaps depending on one's area of interest. A recent 2008 project [22] for automatic optical character recognition of mathematical articles combined the commercial product Fine Reader by Abbyy [1] to recognize English characters with an open source project InftyReader [9] to recognize mathematical objects such as square root signs and reported an accuracy rate of detection at around 99%. While an error rate of 1 out of 100 characters may seem high, one can possibly use previously acquired databases of likely occurring words or use surrounding context to improve word and phrase recognition. The open access articles from PubMed Central offer one such opportunity for using surrounding context as the articles come in a bundle both with text in an XML parsed format and with high quality figures. In addition, the Specialist Group offers a wide-ranging set of natural language processing open source software written in the Java programming language including databases of biomedical terms. We will not pursue this direction further; instead concentrating on our specific implementations.

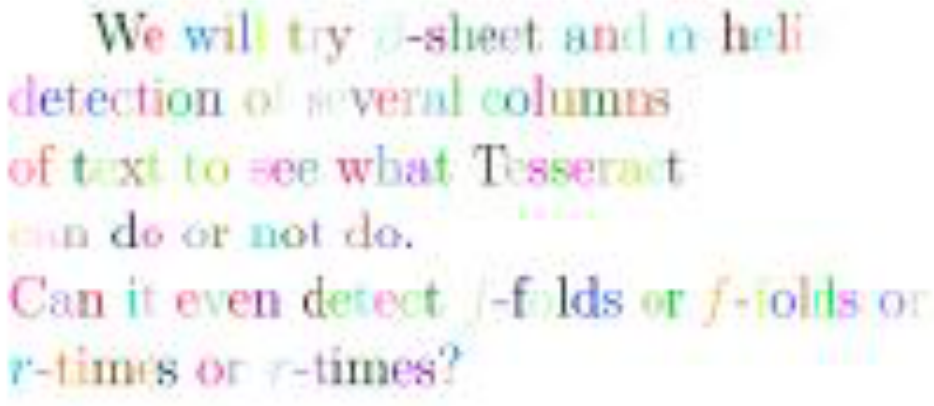

Figure 1: Leptonica can distinguish character bounding boxes

2.2 Leptonica for character region determination

As discussed in previously mentioned texts such as [14] or [4], to find parts of a figure that might be text, simply look for *connected components*, where by a connected component we mean a collection of points such that between any two points there exists a path contained in that component. By *path* for figures, we mean a sequence of points such that two adjacent points in the sequence really are neighbors. There are two reasonable concepts of neighbors if one considers points, or pixels, in a figure to be part of a rectangular gridlike array—either a point on one's diagonal is a neighbor, which leads to 8-connected components, or it is not necesarily a neighbor, which leads to 4-connected components. Regardless of concept, to find connected components one simply devises some sort of efficient scanning of a figure. In Figure 1, we show the output of modifying one line in the test program `conncomp_reg.c` from open source project Leptonica [12], version 1.65, to accept an arbitrary image as a command-line argument. Leptonica detects the individual characters as connected components, determines a box including a particular component, and then the `conncomp_reg.c` program randomly chooses colors for each component to prove that the characters are individually distinguished.

2.3 TEX and LATEX

Donald Knuth developed the TEX typesetting system, as described for example in a collected series of his papers in [10]. The LATEX extension to TEX as documented in

```
\documentclass[12pt]{article}
\thispagestyle{empty}
\begin{document}
We will try $\beta$-sheet and $\alpha$-helix \newline
detection of several columns\newline
of text to see what Tesseract\newline
can do or not do. \newline
Can it even detect $f$-folds or \textit{f}-folds or \newline
$r$-times or \textit{r}-times?
\end{document}
```

Figure 2: LaTeX source file

[11] provides the text specification in which we write the message eventually displayed in Figure 1. Figure 2 shows the source file that is used to generate the text Leptonica reads. Observe that in LaTeX one can specify a Greek letter such as `beta` β without having to leave the ordinary English letters. And because these are simple text files, we simply script the generation of individual files each containing one letter or character as training data.

2.4 Blurring used for testing data

One may observe in Figure 1 that the image is not very clear. A common cause of such blurring is incorrect use of software options. For example, if one were to take a PostScript file generated from a LaTeX file using a program such as `dvips` and then convert the PostScript file to a `.pdf` file without special care, then Figure 3 shows what can happen. Here the letter small `a` is badly converted, then afterwards it is difficult to recover the original sharpness. This is especially unwarranted because in TeX the font system is designed to allow fonts to remain sharp because fonts are specified using certain piecewise smooth cubic curves known as Beziér curves. As it happens these problems of conversion can be avoided by specifying certain resolution parameters, say 300 dpi, dots per inch, that retains a high enough resolution for comfortable OCR.

But we plan to use blurring to our advantage to generate test data. As a preview consider Figure 4 that shows a blurred letter `B` on the left that is processed to some intermediate form shown in the middle, then wrongly compared on the right to another intermediate for that represents the Greek letter `pi` π. Our idea will be to use TeX derived characters as training data, blur the same characters, then see if the trained detector can still find the character closest to the blurred image.

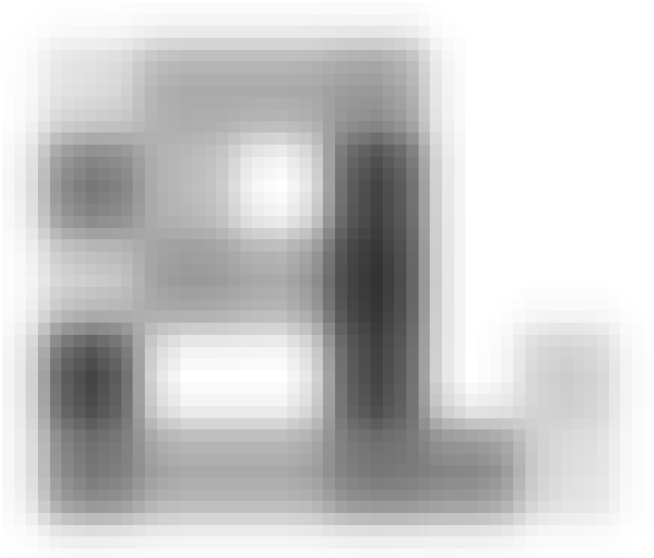

Figure 3: Blurred letter `a` from sloppy conversion

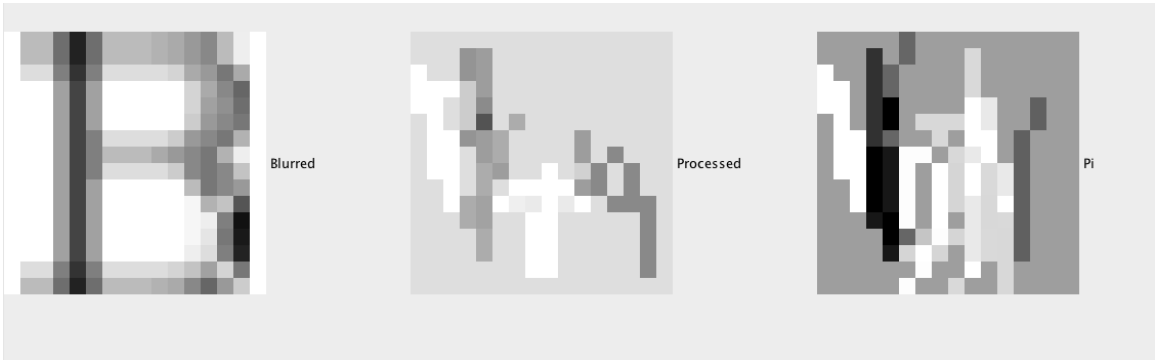

Figure 4: Blurred letter `B` used for tests

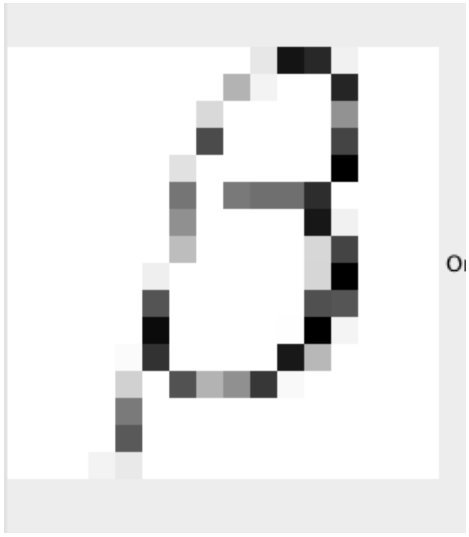

Figure 5: Greek letter `beta` β converted to 16×16

2.5 Image manipulation using ImageMagick

Suppose text in a figure starts out at 96 dpi, then if there are 6 lines per inch, there are approximately 16 pixels in the vertical direction per line. However, a character does not use all the vertical space. There is space between lines and differing heights for characters so that for a letter such as small "a," one may only have 5 pixels in either the vertical or horizontal direction.

Partly inspired by reading [15], we somewhat arbitrarily choose 16×16 pixels as a reasonable size for which to normalize character images. For normalizing images, converting between file formats, and for applying various transformations, we use the open source ImageMagick [8] program `convert`. The program `convert` has among other options one called `-trim` that can remove the white space doing nothing around a character isolating a character. Figure 5, a part of images we will see later, is a demonstration of our graphical user interface we programmed in Java to help us visualize images and transformations. Here careful use of `convert` has enabled us to scale the Greek letter `beta` β fairly reasonably to 16×16.

2.6 Tesseract-OCR

We have shown that open source software can find character connected components, can isolate them, and can convert them to various sizes and formats, and that we can choose either to try and preserve resolution or to deliberately blur to generate a testing set. The Tesseract-OCR [25] project, first developed at HP Labs in the 1990s then transferred to being maintained as open source by Google, has been used successfully by projects such as Sikuli [27]. In Figure 6, we see that the command-

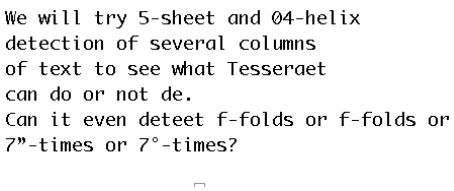

Figure 6: Tesseract-OCR reading most English text correctly

line version of Tesseract-OCR can read most of the English text we earlier fed to Leptonica. Such accuracy is not unexpected since even in the mid-1990s in contests at UNLV, Tesseract-OCR was performing at around 95% accuracy. The default English-trained Tesseract-OCR program we compiled from Subversion revision 319 appears to have some difficulty with reading the Greek letters `alpha` α and `beta` β; however, we recognize that Tesseract-OCR has an established if complicated procedure for retraining to recognize more characters, and that furthermore it has been ported to read several other European-type scripts. Tesseract-OCR's implementation appears to be somewhat akin to a Hidden Markov Model, and it is able to use context of surrounding characters and previously trained words to help decipher characters.

Open source therefore already has a software stack for OCR all the way from reading individual characters to deciphering passages. The problem of recognizing printed characters in a known font even using open source software alone is basically solved.

Instead of devising an optimal algorithm for OCR, we propose to use OCR to examine whether the techniques of compressive sampling can be extended to randomly generated sample matrices that do not quite satisfy conditions required to guarantee optimal solutions.

In the next chapter, we discuss the theory of compressive sampling and our proposal to use random curves as components of the sampling matrix.

3 Compressive Sampling using CoSaMP

Given that open source solutions such as Leptonica enable one to obtain individual printed characters and given that solutions such as Tesseract OCR enable one to obtain a significent fraction of characters unless the resolution of the characters is not sufficient, we change our focus from finding a better algorithm to do OCR to instead applying our intuition about OCR to understand better a recent algorithm. Our attention is drawn to a recent algorithm CoSaMP from [16] that uses matrix manipulations also easily found in open source. We discuss an implementation of CoSaMP that uses randomly chosen cubic Beziér curves in the plane and how the efficacy of this approach can be understood by using the singular value decomposition associated with the corresponding matrices.

3.1 Motivation for using compressive sampling

As an analogy, suppose one had the problem of trying to describe an acquaintance's face to a family member who had never met this acquaintance. Then one could conceive of describing the acquaintance's features using a combination of features from people with whom the family member was familiar with. As part of this description, it would probably be better for the family member to keep the number of template people as small as possible—using a couple of faces might be good, using fifty might become tiresome.

In a series of papers starting in 2004 such as [3], which extends results to the case of measurements with error, Candes, Tao, and others have developed the theory of *compressive sampling* to recover signals that express measurements as a sparse linear combination of known test samples. Let m measurements be made, where by measurement we mean for example taking the value of a pixel at a certain point location (x, y) of each photograph being considered. Let there be N known training objects where each training object j has m measurements, so that each training object can be identified with a column vector of measurements

$$\begin{pmatrix} \Phi_{1,j} \\ \vdots \\ \Phi_{m,j} \end{pmatrix}.$$

If any set of m measurements $\mathbf{y}$ we might later see is hypothesized to be a linear combination of the training objects, the *signal* is the set of linear coefficients $\mathbf{x}$ such that

$$\mathbf{y} = \Phi\mathbf{x}$$

where Φ is the matrix whose columns are those of the training object measurements. A vector $\mathbf{x}$ has *sparsity* s if $\mathbf{x}$ has no more than s nonzero components.

There is extensive theory for solving systems of a form similar to

$$\min \|\mathbf{x}\|, \text{ subject to } \mathbf{y} = \Phi\mathbf{x}.$$

Candes and Tao in their formulation of compressive sampling have developed a theory where if one assumes that almost all selections of s columns from test object matrix Φ are nearly orthogonal, that is, where columns are almost perpendicular to each other in multi-dimensional space, then solving certain systems can produce known recovery of signals up to some error level.

3.2 Motivation for using random curves

For analyzing figures we make an analogy between curves being nearly non-touching and columns being almost perpendicular to each other. We visualize the opposite, columns sharing a common direction, with curves touching many points.

Figure 7: Two random cubic Beziér curves in 16×16 space

We chose to use a class of randomized plane curves from the class of cubic Beziér curves because they were used by Donald Knuth for the design of fonts in his METAFONT program that generates the Computer Modern TEX fonts [10] and because in the Java API there are simple methods to generate such curves given the two endpoints (x_1, y_1) and (x_2, y_2) and two control points (c_1, d_1) and (c_2, d_2). We simply randomly generate all eight coordinates at once as real numbers between 0 and either the desired height or width of the rectangular box containing the image. Note that the resulting cubic curve must lie in the convex hull of its four specified points. For further details about the properties of Beziér and graphics see for example [6]. In Figure 7, we show the intersection of two randomly generated cubic Beziér curves in 16×16 pixel space. Two curves would represent completely orthogonal columns if they had no intersection at all. Actually the resulting image is blown up uniformly by a factor of 16 in each dimension so that each pixel becomes a 16×16 square for greater visibility. For this particular random choice the curves intersect only in one place in seemingly a small percentage of their total points. Of course if the random choice is unfortunate, the curves could intersect in a far greater number of points, and furthermore, there is no guarantee that several curves intersecting together might violate the assumption that as a collection they do not mutually intersect that often.

We know that choosing random curves cannot satisfy the hypothesis for compressive sampling. For a 16×16 grid there are only a small number of curves that can be chosen until the area is simply mostly covered, that is, lots of non-orthogonality. But that is exactly our contribution—we intend to show that even if the mathematical conditions no longer strictly hold, some part of the idea still applies, especially if we can use knowledge of linear algebra to repair problems.

In the next chapter, we discuss the applied linear algebra of orthogonality and the singular value decomposition that provide an understanding of the least squares computation iteratively used in CoSaMPand of the condition number that determines how close a matrix is to satisfying compressive sampling near-orthogonality requirements.

4 Mathematics of Compressive Sampling

We now explain what we mean by orthogonality and an efficient matrix decomposition used in the theory of CoSaMP and which will be used by us in our extension of CoSaMP to optical character recognition.

4.1 Orthogonality and SVD

We begin with a review of elementary linear algebra that will form the foundation of our mathematical analysis of CoSaMPś performance on classifying characters using random Beziér curves. The background for the following discussion can be found in [26].

4.1.1 Orthogonality and near-orthogonality

Let **a** and **b** be real-valued vectors of the same one-dimensionality, length m. Then **a** and **b** are said to be *orthogonal* if their *inner product* is 0, that is,

$$a \cdot b = \sum_{i=1}^{m} a_i \cdot b_i = 0.$$

Correspondingly in the theory of compressive sampling and for CoSaMP a concept of near orthogonality can be defined where the inner product of the two vectors is bounded by some small value δ.

A dot product induces a distance function where one simply takes the dot product of a vector by itself and then takes the square root. Such a distance is known as a *2-norm*. The general theory of compressive sampling can use other norms than the 2-norm, fortunately, we do not have to worry about such things in the context of CoSaMP.

The theory of compressive sampling uses two-dimensional matrices Φ of dimension $m \times T$ where $T < m$. The columns of Φ are said to be *orthogonal* if the inner products between columns of different indices is 0. But for the analogous concept of nearly orthogonal columns, it is not enough to merely consider pairs of columns and their dot products in pairs being relatively small. Collections of multiple columns must be considered together, such a procedure enabled by the following concept of the singular value decomposition.

4.1.2 Singular value decomposition

Let two-dimensional real-valued matrix A be $m \times T$ dimensioned where $m \leq T$. Then there exist an orthogonal $m \times T$ dimensioned matrix U, a $T \times T$ diagonal matrix Σ, and an orthogonal $T \times T$-dimensioned matrix V such that A has a *singular value decomposition* into the product of these three matrices

$$A = U\Sigma V^*,$$

where for real-valued matrices the conjugate adjoint V^* is the same as the transpose of V. (The transpose of a matrix is defined so that the element at row i and column j of the transposed matrix is the same as the element at row j and column i of the original matrix.) In addition the diagonal matrix Σ can be restricted to have its diagonal elements non-negative and sorted in descending order

$$\Sigma_1 \geq \Sigma_2 \geq \cdots \geq \Sigma_T \geq 0.$$

A more intuitive picture is that for multiplying a vector $\mathbf{x}$ by A, for each scaling factor Σ_i of the diagonal matrix, take the length of $\mathbf{x}$ in direction V_i, multiply that length by scaling factor Σ_i, and then assign the result along direction U_i. We can express A as a sum of so-called *outer products* as

$$A = \sum_{i=1}^{T} \Sigma_i U_i V_i^*.$$

4.1.3 SVD, stability, and least-squares

Because of the assumption of orthogonality of columns of U and of columns of V, algebraic calculation using an SVD decomposition is relatively easy. The calculation of the SVD is also relatively *stable* so that errors in calculations are minimized.

An example that will be of use later discussing CoSaMP is using the SVD to solve the *least squares* problem of given matrix A and a target vector $\mathbf{b}$, find a vector $\mathbf{x}$ that minimizes the norm of the difference

$$A\mathbf{x} - \mathbf{b}.$$

If the matrix A^*A is invertible, then the solution of this problem is often written as

$$A^+\mathbf{b} = \mathbf{x},$$

where A^+ is called the *pseudo-inverse* of A with algebraic solution

$$A^+ = (A^*A)^{-1}A^*.$$

Using a matrix inverse $(A^*A)^{-1}$ after explicitly multiplying out A and its transpose A^* is less stable because in effect such a multiplication is squaring the effect of A. On the other hand it is well-known that if one has an SVD one can simply invert the order and use the inverse of the diagonal matrix Σ to obtain an expression for the pseudo-inverse that does not involve a square term

$$A^+ = V\Sigma^{-1}U^* = \sum_{i=1}^{m} \Sigma_i^{-1} V_i U_i^*,$$

assuming none of the $\Sigma_i = 0$.

4.1.4 SVD and condition number

Suppose that all diagonal elements $\Sigma_i > 0$, recall they are assumed to all be non-negative. Then a measure of how differently A distributes values in different directions is given by the ratio, the *condition number*,

$$\text{cond}\,(A) = \frac{\Sigma_1}{\Sigma_T}$$

is the ratio of the largest scaling in some direction to the smallest scaling in some direction. Observe that if in some direction the corresponding scaling is 0, that is, $\Sigma_i = 0$, then the condition number above is not well-defined.

Having discussed the SVD, in the next chapter we present the full CoSaMP algorithm and our strategies for using the CoSaMP algorithm for matching characters.

5 Methods: CoSaMP

We explain CoSaMP and show how we apply it to perform OCR.

5.1 Motivation for CoSaMP

Suppose the columns of the training matrix Φ were exactly orthogonal to each other, suppose we were given some vector $\mathbf{u}$ to be approximated in terms of the columns of Φ, and suppose we wanted to use s or fewer columns to explain as much of $\mathbf{u}$ as possible. Then the obvious approach would be to simply take the dot product of $\mathbf{u}$ with each column Φ_i of Φ and take the columns corresponding to the largest s values of

$$\{\mathbf{u} \cdot \Phi_i\}\,.$$

But the columns are not exactly orthogonal to each other, only approximately so. If we consider the problem of finding the optimal s columns of Φ to be a search problem, we are hoping that finding something plausibly close will lead us to the answer close by, that is, using many of the columns we previously found.

We also recall that given some T columns of Φ where $T \geq s$, the most optimal way of approximating $\mathbf{u}$ is precisely to solve the least squares problem using the T columns.

5.2 CoSaMP formal definition

The algorithm described below is Algorithm 2.1 from [16]. We make the following observations and definitions:

- Let function $\mathsf{findLargest}(\mathbf{x}; j)$ for vector $\mathbf{x}$ and integer j represent the *indices* of the j largest in absolute value components of $\mathbf{x}$.

- Let function $\mathsf{values}(\mathbf{x}; j)$ for vector $\mathbf{x}$ and integer j represent the *values* of the j largest in absolute value components of $\mathbf{x}$.
- Let function $\mathsf{LS}(A; \mathbf{b})$ for matrix A and vector $\mathbf{b}$ represent the least squares solution $\mathbf{x}$ that minimizes the norm of $A\mathbf{x} - \mathbf{b}$. Recall finding $\mathsf{LS}()$ can be done using the pseudo-inverse of A.
- By definition of matrix transpose, the i-th row of Φ^* is the transpose of the i-th column of Φ.
- For any matrix-vector product Ay, the resulting i-th row of Ay represents the dot product of the i-th row of A with y. To find all of the dot products of some m vector $\mathbf{v}$ with the columns of Φ, first transpose Φ to Φ^* and just take the matrix-vector product $\Phi^*\mathbf{y}$.
- From now on let S represent the best guess for the s sparsity indices and let T represent the next set of indices in which to search for the next S. T must contain the previous S. Since we have no information at the start, S is initialized to be the empty set .
- Let $\mathbf{z}$ represent the current guess for the s sparse signal values. Initially as we know nothing, $\mathbf{z}$ is initialized to be the zero vector.
- Let Φ_T represent the matrix formed from the T indexed columns of Φ.
- Let $\mathbf{r}$ represent the residual $\mathbf{u} - \Phi\mathbf{z}$, that is, what is left to be estimated.
- Let function $\mathsf{toContinue}()$ be somewhat arbitrarily defined to be true if the number of iterations has not exceeded some limit $\mathsf{maxIter}$ or if the signal guess $\mathbf{z}$ is still changing between iterations.

5.3 CoSaMP implementation

For finding a least-squares estimate, in the context of CoSaMP's theoretical framework where the columns of the sample matrix Φ are approximately orthogonal, [16] suggests that iterative methods, either Richardson's iteration method or conjugate gradient, can be used to considerably speed up the computation relative to the more stable but slower use of the SVD.

However these iterative methods for their rapid convergence depend on the approximately orthogonal assumption that in retrospect simply does not hold. Our initial attempts to program CoSaMP with iterative methods failed badly. Even had our code been bug-free, the lack of orthogonality would magnify errors so that after a few iterations massive divergences would occur on the order of 10^7 or more.

Input: $m \times N$ matrix Φ, sparsity s, $m \times 1$ vector $\mathbf{u}$ to be approximated
Result: s-sparse signal $N \times 1$ vector $\mathbf{z}$ such that $\Phi\mathbf{z}$ approximates $\mathbf{u}$
begin
  $S \leftarrow \emptyset$;
  $\mathbf{z} \leftarrow 0$;
  $\mathbf{r} \leftarrow \mathbf{u}$;
  while toContinue() **do**
    $T \leftarrow S \cup \mathsf{findLargest}(\Phi^*\mathbf{r}; 2s)$;
    $S \leftarrow \mathsf{findLargest}(\mathsf{LS}(\Phi_T; \mathbf{u}); s)$;
    $\mathbf{z} \leftarrow \mathsf{values}(\mathsf{LS}(\Phi_T; \mathbf{u}); s)$;
    $\mathbf{r} \leftarrow \mathbf{u} - \Phi\mathbf{z}$;
  end
end

Algorithm 1: CoSaMP

5.4 Time bounds

Even if there is no other known structure, a matrix-vector multiply of a $m \times T$ dimensioned matrix with a $T \times 1$ dimensioned vector will in a brute force implementation cost

$$O\left(m \cdot T^2\right)$$

operations. According to [26], this is also the theoretical cost of computing the SVD. However, according to [16], there are situations where at least one of the following holds that advantage iterative methods:

- matrix-vector multiplications are faster than the upper bound,
- operations are not strictly matrix-vector,
- additional storage of the $m \times T$ matrix U is inconvenient.

The breakthrough of [16] is not just an acceleration of compressive sampling, but proven bounds on number of iterations, perhaps only five, if the near-orthogonality assumptions hold.

5.5 Reversion to SVD and Colt

Given that the near-orthogonality bounds do not hold strongly enough for the theorems from [16] to apply, the safest and easiest way to implement CoSaMP is to simply use the SVD to calculate the least squares solution despite that method being disparaged. We use the Colt [5] linear algebra package developed at CERN for the Java programming language that conveniently has its own SVD method.

In the next chapter, we show images from our graphical user interface illustrating CoSaMP approximations to characters using random curves, we describe our baseline

Iteration	$\|\mathbf{r}\|$	$\|\Delta\mathbf{z}\|$	rank	—T—	cond(Φ_T)
0	628.55	508.94	20	20	5.10
1	599.00	340.83	30	30	4.91
2	602.83	190.47	30	30	5.26
3	601.94	20.48	30	30	4.82
4	601.32	5.89	30	30	4.72
5	601.32	0.00	30	30	4.72

Table 1: CoSaMP applied to Greek letter β

strategy for matching a blurred character to one of previously sampled characters, and we describe our three strategies for comparison using CoSaMP for such blurred character matching.

6 Methods

6.1 GUI illustrating CoSaMP visually

In Table 1, we show the first results we have for using CoSaMP on the image pixels of the character `beta` β. Somewhat arbitrarily we choose the sparsity s to be 10 and the number of random curves to be 1000. The first column is the number of iterations of CoSaMP the second column is the norm of the residual, the third is the change in the signal estimate, the fourth is the rank of the matrix Φ_T, the fifth is the order of T, and the sixth column is the condition number of the same matrix Φ_T. If one retrains using new random sample shapes, one can obtain different rates of running CoSaMP on one character. In Table 1 we see a very short run where it takes only 6 iterations for the signal estimate to completely convert. Looking at the sixth column, the condition numbers, we see that the matrices Φ cannot possibly fulfill the near-orthogonality constraints, because then the condition numbers, the diagonal elements Σ_i, would have to be all nearly 1, while these in column 6 have a ratio at least 4.

We also observe that the presence of both the rank of the matrix Φ_T and the expected dimension the number of indices in T is not redundant. We have observed calculations where the matrix Φ_T has deficient rank.

In Figure 8, we show from the "Comparison" panel of our Java platform graphical user interface the letter `beta` β and its transformation using CoSaMP. The leftmost image in Figure 8 is the original letter β converted to 16×16, the middle image is the CoSaMP transform of β shown as an image, and the rightmost image is the area spanned by the $s = 10$ CoSaMP-selected random curves. The middle image represents an approximation by CoSaMP selecting a linear combination of s random curves, and the rightmost image shows that our choice of $s = 10$ does not simply induce a covering of all pixels of the image by the s curves.

Figure 8: Visualization of CoSaMP applied to β

Figure 9: Visualization stroke width 2.5 applied to β

6.2 Stroke width

Examining the Java API for cubic Beziér curves, we see that in addition to specifying coordinates for points, we need to specify a stroke width. If we use a larger stroke width more of the area is covered.

In Figure 8, we use a relatively modest stroke width of 1. Changing only the stroke width to 2.5, we obtain in Figure 9 a seemingly better fit in appearance if we are interested in including the interior of the shape as well. In Table 2 we show the numerical results during the CoSaMP transform of β, which in this case involved more iterations.

6.3 Choices for testing

We now explore more choices that can be made with testing sets.

We identify with CoSaMP the following choices for how to compare an unknown character, blurred, with the previously trained set of characters including Greek let-

Iteration	$\lvert\mathbf{r}\rvert$	$\lvert\Delta\mathbf{z}\rvert$	rank	—T—	cond(Φ_T)
0	798.56	580.41	20	20	10.38
1	722.09	708.23	30	30	10.56
2	932.01	632.96	30	30	10.33
3	946.74	352.96	30	30	14.71
4	989.45	126.65	28	28	13.37
5	963.29	63.55	28	28	14.53
6	961.38	47.88	28	28	13.41
7	963.29	47.88	28	28	14.53
8	961.38	47.88	28	28	13.41
9	963.29	47.88	28	28	14.53
10	961.38	47.88	28	28	13.41
11	963.29	47.88	28	28	14.53
12	961.38	47.88	28	28	13.41
13	963.29	47.88	28	28	14.53
14	961.38	47.88	28	28	13.41
15	963.29	47.88	28	28	14.53
16	961.38	47.88	28	28	13.41
17	963.29	47.88	28	28	14.53
18	961.38	47.88	28	28	13.41
19	963.29	47.88	28	28	14.53

Table 2: CoSaMP applied stroke width 2.5 β

ters and some arrows. Observe the scaled dot product of two vectors a and b

$$(a, b) = \frac{a \cdot b}{|a| \cdot |b|}$$

is a measure of how much the two vectors point in the same direction. We take the absolute value of the scaled dot product. Consider CoSaMP when applied to a vector x, given sparsity s, to be in effect a transform $C(x)$ to its sparsity s approximation.

1. The obvious and best approach is to simply compare the new blurred character with the original of each other character using the scaled dot product. For each fixed new vector x, with every previously encountered character y, we find the maximum over the y of

   $$(x, y).$$

   This choice corresponds to simply identifying each character with the part of the two-dimensional rectangle it occupies, then the dot product essentially measures well these areas overlap, fully in accordance with our intuition about how to match characters.

2. Suppose we consider the CoSaMP approximation for a vector given the sparsity s to be a transform, then perhaps taking the dot products of two transforms, both transforming the new character and using the previous transform of the trained characters, will preserve something similar to have Fourier transforms into the frequency domain preserve some notion of inner product. This approach will be the slowest since a brand new CoSaMP approximation will be applied to each new vector. The dot product of the two transforms then maximizes over y

   $$(Cx, Cy)\,.$$

3. Suppose we take the dot product of the new vector x with the CoSaMP transform Cy. If we write Cy as

   $$Cy = \sum_{i=1}^{s} \alpha_i \Phi_i$$

   for some columns Φ_i of the sample matrix, then the dot product

   $$x \cdot Cy = \sum_{i=1}^{s} \alpha_i \left(x \cdot \Phi_i\right)$$

   is almost pretending that the Φ_i are orthogonal and that we are taking a dot product relative to those coordinates.

4. Suppose we try a hybrid approach of using our intuition that the selected sparse s curves should somehow represent an important region on which two

characters should match, but that we should consider this sparse s region on its own terms. That is, CoSaMP is used to determine the indices of the sampling matrix columns $\Phi(s) = \{\Phi_1, \ldots, \Phi_s\}$, but for both the new and the currently considered character y, instead of using CoSaMP again, we try to find the best projection on the space of $\Phi(s)$. We propose taking the scaled dot product of the least squares solution for both x and y on $\Phi(s)$.

6.4 Characters tested

In TEX or LATEX it is relatively easy to generate characters including Greek and mathematical symbols and then to script these characters into image files. We used the English version of the Latin alphabet, both upper and lower case, the numbers from 0 through 9, the unique Greek lower and upper case symbols from TEX but not the variant versions such as ϖ for π, and three arrow symbols $\leftarrow$, $\rightarrow$, and $\mapsto$.

In the next chapter, we discuss the results of comparing the three strategies for matching characters using CoSaMP with the baseline strategy.

7 Results

We show the results for an experiment comparing various strategies for using CoSaMP for OCR of English printed characters and Greek / mathematical symbols.

7.1 Comparison strategy results

For each character x, we blur the character using a sloppy conversion from a PostScript to a `.pdf` file, then try to find the best fit from the previously encountered characters y using four different strategies. We might as well keep track for each x what place the true character finished in the comparison, and the scaled dot products give us some sort of measure of how closely pairs match. For the test run we use 16×16 images, sparsity $s = 10$, stroke width 2.5, number of random curves 2000, maximum number of iterations for CoSaMP of 20, and dpi 300.

7.1.1 Comparison for vector vs. vector, no CoSaMP

We begin with the matching for the strategy of just using the (blurred) image and taking its scaled dot product with other images, no CoSaMP. To explain the columns of Table 3, the first column represents the actual character x, blurred, the second column is the character y with the largest scaled dot product relative to x, the third column is that largest scaled dot product, the fourth column is the place in which the actual character x finished in the comparisons, the fifth column is if x finished first the character that finished second, and the sixth column is either the score of the second placed character if x finished first or otherwise x's score.

Actual	Best fitted	Best score	Place	Alternative	Actual score
C	c	0.812	6	C	0.620
F	P	0.817	6	F	0.655
G	o	0.752	2	G	0.636
I	t	0.816	13	I	0.412
L	U	0.721	23	L	0.443
O	o	0.854	3	O	0.576
Q	Omega	0.786	5	Q	0.625
S	s	0.868	2	S	0.770
T	t	0.824	16	T	0.380
U	H	0.728	18	U	0.491
V	v	0.836	3	V	0.622
X	x	0.736	3	X	0.626
Z	z	0.875	2	Z	0.865
a	sigma	0.605	4	a	0.530
f	l	0.817	3	f	0.803
n	u	0.822	2	n	0.794
p	r	0.817	6	p	0.644
r	T	0.774	8	r	0.606
t	I	0.806	10	t	0.544

Table 3: Mismatched, no CoSaMP, alphabetical characters

Actual	Best fitted	Best score	Place	Alternative	Actual score
0	n	0.742	4	0	0.700
1	l	0.895	2	1	0.883
3	s	0.701	3	3	0.681
4	d	0.736	8	4	0.526
5	s	0.676	2	5	0.656
6	epsilon	0.700	6	6	0.660
9	q	0.720	3	9	0.681

Table 4: Mismatched, no CoSaMP, digits

Actual	Best fitted	Best score	Place	Alternative	Actual score
beta	eta	0.549	18	beta	0.400
theta	epsilon	0.683	2	theta	0.666
xi	zeta	0.672	2	xi	0.640
pi	w	0.635	3	pi	0.534
tau	w	0.549	12	tau	0.368
upsilon	b	0.530	2	upsilon	0.517
chi	Y	0.658	3	chi	0.650
Gamma	E	0.772	24	Gamma	0.463
Theta	o	0.797	2	Theta	0.667
Omega	o	0.623	2	Omega	0.614
leftarrow	C	0.460	3	leftarrow	0.457
mapsto	alpha	0.501	2	mapsto	0.456
rightarrow	mapsto	0.408	4	rightarrow	0.394

Table 5: Mismatched, no CoSaMP, Greek and arrows

Table 3 shows the mismatched Latin alphabetic characters for the strategy of simply comparing the blurred x with the previously stored images y, no CoSaMP transformation to either and no restriction to a subset of pixels in the rectangle. We see that blurring has what is to us a surprisingly big effect—we are after all comparing characters using the same font, size, etc. to blurred versions of themselves and still finding that for some many other characters appear to match better using this strategy. It also appears to us from Table 3 that lower case letters match better than upper case letters. In Table 4, we see the results for comparing characters directly for digits. The digits are often mismatched; however, the number of places of mismatch is relatively small so that at least the true digit is a close candidate. So far the rate at which even in ideal conditions blurred characters are recognized is rather low. But in Table 5, which show differences in matching for Greek letters and for a few arrows, we see that the accurate recognition of Greek letters brings the

recognition rate up to an overall

$$60/99 \approx 0.6$$

with average place around 3.05. For our tests the Greek symbols are typeset in mathematics; furthermore, some Greek symbols are not specially made, particularly for capital letters, so presumably the letters that have special versions are perhaps more different from other Latin alphabet letters than usual.

7.1.2 Comparison least squares on CoSaMP determined subset

In Table 6, we see the results for small Greek letters of finding the least squares approximation on the CoSaMP derived s-sparse subspace, presumably a set of curves that somehow approximate the shape intended. We observe that the scaled dot product scores are higher for the best matches, often above 0.9, presumably due to the approximating shape tightening up where matches in pixels should occur. We observe that scores even for correct matches can be somewhat close to the second place score. And we note that for character `tau` τ matching is considerably worse than for the case of no transform, the place of the correct match down to 62, worse than chance. In Table 7, we show a comparison on all numeral characters from 0 through 9 of the case of no transforms versus the case of least squares projection on the s-sparse subspace found by CoSaMP. We see that least squares never has the numeral detected at a worse place than the case of no transform and that least squares detects 5 out of 10 correctly versus 3 out of 10 for no transform. For the least squares on all characters we obtain

$$49/99 \approx 0.49$$

correctly identified with average place approximately 6.1. For one reason why the average place rose, consider Table 8 showing the results for least squares projection on capital letters. While the no transform capital letters analysis, refer to Table 3, has four letters with double digit places, we see that the least squares approach has seven with double digit places, and the case of `tau` τ has deteriorated to a place down to 62, worse than just taking half the number of characters considered.

7.1.3 Comparison for CoSaMP vs. CoSaMP

Next, we consider comparing a new blurred character transformed by CoSaMP with previously sampled unblurred characters also transformed by CoSaMP. In Table 9 we show the results for small Greek letters. We observe that 15 out of 23 are correctly recognized, as compared to the case of no transform where 16 out of 23 are correctly recognized.

Over all characters not just small Greek ones the accuracy rate of matching is

$$52/99 \approx 0.525$$

and average place of matching is approximately 4.4.

Actual	Best fitted	Best score	Place	Alternative	Actual score
alpha	alpha	0.955	2	A	0.866
beta	Z	0.908	2	beta	0.882
gamma	gamma	0.903	2	Xi	0.872
delta	delta	0.955	2	8	0.914
epsilon	epsilon	0.967	2	C	0.951
zeta	zeta	0.941	2	6	0.884
eta	eta	0.970	2	pi	0.891
theta	6	0.948	2	theta	0.936
iota	epsilon	0.933	3	iota	0.898
kappa	kappa	0.948	2	nu	0.933
lambda	lambda	0.949	2	Lambda	0.948
mu	mu	0.916	2	0	0.914
nu	nu	0.956	2	0	0.952
xi	zeta	0.911	2	xi	0.906
pi	pi	0.873	2	p	0.870
rho	rho	0.941	2	0	0.939
sigma	sigma	0.959	2	alpha	0.930
tau	chi	0.902	62	tau	0.539
upsilon	psi	0.877	14	upsilon	0.791
phi	phi	0.913	2	0	0.896
chi	Upsilon	0.936	4	chi	0.907
psi	r	0.872	2	psi	0.860
omega	O	0.949	2	omega	0.939

Table 6: Comparison on least squares subspace, small Greek letters

Actual	Best fitted	Best score	Place	Alternative	Actual score
No CoSaMP blurred vs no CoSaMP					
0	n	0.742	4	0	0.700
1	l	0.895	2	1	0.883
2	2	0.851	2	R	0.600
3	s	0.701	3	3	0.681
4	d	0.736	8	4	0.526
5	s	0.676	2	5	0.656
6	epsilon	0.700	6	6	0.660
7	7	0.736	2	Z	0.614
8	8	0.819	2	s	0.779
9	q	0.720	3	9	0.681
Least squares blurred vs Least squares					
0	0	0.981	2	d	0.958
1	l	0.953	2	1	0.952
2	2	0.984	2	S	0.893
3	3	0.965	2	x	0.958
4	J	0.898	6	4	0.841
5	Xi	0.909	2	5	0.908
6	3	0.943	3	6	0.934
7	7	0.884	2	f	0.869
8	8	0.974	2	s	0.936
9	q	0.966	2	9	0.935

Table 7: Comparison on numerals of no transform vs. least squares

Actual	Best fitted	Best score	Place	Alternative	Actual score
A	A	0.982	2	Lambda	0.945
B	B	0.976	2	P	0.958
C	G	0.938	14	C	0.850
D	D	0.965	2	Omega	0.938
E	v	0.961	6	E	0.933
F	Z	0.961	23	F	0.821
G	G	0.959	2	o	0.911
H	n	0.986	4	H	0.968
I	t	0.963	33	I	0.589
J	J	0.950	2	4	0.943
K	K	0.990	2	X	0.928
L	U	0.923	44	L	0.644
M	M	0.969	2	n	0.945
N	N	0.961	2	X	0.951
O	n	0.938	8	O	0.878
P	P	0.980	2	e	0.960
Q	n	0.967	7	Q	0.917
R	e	0.979	3	R	0.974
S	s	0.984	2	S	0.968
T	t	0.980	31	T	0.663
U	c	0.989	64	U	0.599
V	Y	0.963	4	V	0.904
W	w	0.975	2	W	0.969
X	N	0.921	12	X	0.846
Y	v	0.922	2	Y	0.882
Z	Z	0.983	2	z	0.971

Table 8: Comparison for least squares, capital letters

Actual	Best fitted	Best score	Place	Alternative	Actual score
alpha	alpha	0.810	2	sigma	0.721
beta	beta	0.696	2	eta	0.687
gamma	gamma	0.855	2	7	0.716
delta	delta	0.836	2	k	0.687
epsilon	C	0.868	2	epsilon	0.840
zeta	zeta	0.813	2	6	0.662
eta	eta	0.865	2	0	0.763
theta	6	0.746	5	theta	0.661
iota	iota	0.734	2	r	0.721
kappa	kappa	0.768	2	sigma	0.675
lambda	lambda	0.906	2	Lambda	0.816
mu	mu	0.829	2	p	0.765
nu	nu	0.730	2	M	0.715
xi	zeta	0.808	2	xi	0.794
pi	w	0.758	13	pi	0.561
rho	rho	0.777	2	theta	0.765
sigma	sigma	0.905	2	alpha	0.702
tau	w	0.733	23	tau	0.446
upsilon	psi	0.645	11	upsilon	0.552
phi	6	0.702	2	phi	0.679
chi	Y	0.745	5	chi	0.671
psi	psi	0.729	2	k	0.621
omega	omega	0.750	2	Psi	0.619

Table 9: Comparison CoSaMP vs. CoSaMP, small Greek letters

Actual	Best fitted	Best score	Place	Alternative	Actual score
alpha	alpha	0.835	2	sigma	0.707
beta	beta	0.626	2	eta	0.619
gamma	gamma	0.806	2	Upsilon	0.635
delta	delta	0.814	2	theta	0.660
epsilon	C	0.843	2	epsilon	0.816
zeta	zeta	0.810	2	xi	0.621
eta	eta	0.851	2	0	0.672
theta	0	0.736	2	theta	0.734
iota	r	0.733	2	iota	0.728
kappa	kappa	0.774	2	x	0.683
lambda	lambda	0.893	2	Lambda	0.795
mu	mu	0.827	2	p	0.756
nu	nu	0.769	2	upsilon	0.671
xi	zeta	0.768	2	xi	0.730
pi	w	0.734	5	pi	0.590
rho	rho	0.782	2	theta	0.710
sigma	sigma	0.859	2	alpha	0.755
tau	w	0.690	20	tau	0.419
upsilon	psi	0.614	3	upsilon	0.584
phi	phi	0.670	2	6	0.642
chi	Y	0.754	3	chi	0.699
psi	psi	0.696	2	k	0.609
omega	omega	0.709	2	Phi	0.470

Table 10: Comparison actual blurred vs. CoSaMP, small Greek letters

7.1.4 Comparison for actual vs. CoSaMP

For the strategy of comparing the actual blurred vs. CoSaMP previously trained images, Table 10 shows the results for the lower case Greek letters. The recognition rate is 15 out of 23, and the place of `tau` τ has dropped to 20. We also see that the character `beta` β is correctly recognized with its best score 0.626 now better than the `eta` η score of 0.619.

The overall recognition rate for all characters is

$$61/99 \approx 0.616,$$

and the average place of the actual character's score matching is approximately 3.9.

Our intuition is that CoSaMP by choosing a random shape is narrowing what areas are used for analyzing a character, thus resulting in scores being bunched closer together at an upper range.

7.2 Time

We observe that for training and testing CoSaMP using the SVD on 99 characters, the elapsed times on an Apple Intel Macbook 2.1, Core 2 Duo, 2 GB memory, OS X 10.6.3, Apple bundled version of Java 1.6.0_17, is listed in various parts of our output as:

```
21523 millis to obtain 2000 random curves
152638 millis to train 99 characters
...
144235 millis to test 99 characters
```

that is, 21.5 seconds for generating 2000 random curves, 152 seconds for training 99 characters, and 144.2 seconds for running CoSaMP on the blurred images then comparing that transformed measurement to the previous CoSaMP transformed measurements.

8 Conclusions

From a software engineering perspective, this report shows the value of having a field of investigation that allows for visualization such as OCR and of using algorithms that combine speed, simplicity, and stability. Using our graphical user interface we are able to detect and correct bugs in our implementation that we would have missed otherwise. Because of the speed of CoSaMP, we are able to shift to using the more stable SVD method of solving least squares and still have our tests all finish within a few minutes, both training and testing times. We never have to wait overnight for a run to finish.

The simplicity of CoSaMP and its ready interpretation enables us to suggest several plausible strategies for using CoSaMP for OCR. Numerical experiments testing various strategies for matching blurred characters suggest that using CoSaMP can be made competitive with an optimal strategy of matching a blurred character against itself.

We find that compressive sampling for application to OCR does gracefully decline in performance as the mathematical conditions for optimal signal recovery weaken, at least for an implementation using CoSaMP.

9 Further Research

The recognition rate we have observed for CoSaMP compressive sampling in OCR surprises us with how competitive it is with what we thought would have been an optimal stategy of comparing a blurred image with the original. We can now extend our investigations to see if compressive sampling can somehow generalize the portion

of a character that is important by using the sparsity to paradoxically focus on specific areas identified with well-matching random curves. In particular we can compare how well the CoSaMP strategies fare when trying to match italicized versions of characters.

We can also extend using compressive sampling to a wider range of mathematical symbols. And if we can detect mathematical symbols, we anticipate we can generalize to detecting more general shapes in biomedical literature figures.

If compressive sampling ideas can apply to more intuitive methods of generating samples such as our generating random curves, we anticipate far easier and faster means of training pattern recognizers.

A Source Code

We leave some comments including copyright boilerplate and also leave out trivial functions and imports.

A.1 CoSaMP implementation

Our general approach is to simply implement as many concepts in CoSaMP as methods so that it is easier to check for errors.

```
import java.io.*;
import java.util.*;

/**
Class to implement the cosamp algorithm
*/
public class DSCosamp implements Serializable
{

/**
Finds s-sparse approximation, use getApprox() for value
@param u sample vector
@param s sparsity
*/
   public void cosamp(double[] u, int s)
   {
      cosampInit(u, s);
      boolean notDone = true;
      while (notDone)
      {
         cosampIteration();
         notDone = cosampStop();
         kIter++;
      }
   }

/**
SVD least squares estimation
@param Tindex column indices needed from sample matrix Phi
@param b to be approximated
@param x preallocated T by 1 for pseudoInverse A applied to b
@param bApprox preallocated m by 1 for A applied to x
```

```
@param condNum first position will have condition number of Phi T
@return rank of Phi T to check for being nonsingular
*/
   public int svdLS
   (
      int[] Tindex, double[] b, double[] x, double[] bApprox,
      double[]condNum
   )
   {
// A <- Phi T
      double[][] A = new double[m][Tindex.length];
      DSLib.submatrix(Tindex, Phi, A);
      double[][] U = new double[m][Tindex.length];
      double[] sigma = new double[Tindex.length];
      double[][] V = new double[Tindex.length][Tindex.length];
      int rank = DSColt.svd(A, U, sigma, V);
      double cond = sigma[0] / sigma[rank - 1];
      DSLib.pseudoInverse(U, sigma, V, rank, b, x);
      if (bApprox != null)
      {
         multPhiT(Tindex, x, bApprox);
      }
      condNum[0] = cond;
      return rank;
   }

/**
Choose which method to solve least squares
*/
   public void setMethodLS(int methodLS)
   {
      this.methodLS = methodLS;
   }

/**
Set the maximum number of iterations
@param maxIter maximum number
*/
   public void setMaxIter(int maxIter)
   {
      this.maxIter = maxIter;
   }
```

```
/**
Gets best approximation to noisy signal
@return copy of m by 1 approximation
*/
   public double[] getApprox()
   {
      double[] approxCopy = new double[m];
      DSLib.vecCopy(approxCache[indexMinDiffSample], approxCopy);
      return approxCopy;
   }

/**
Gets the indices of Phi for the previous best estimate
@return copy of s by 1 indices vector
*/
   public int[] getASupp()
   {
      int[] aSuppCopy = new int[s];
      for (int i = 0; i < s; i++)
      {
         aSuppCopy[i] = aSupp[i];
      }
      return aSuppCopy;
   }

/**
Gets sparsity s of previous call
@return sparsity
*/
   public int getS()
   {
      return s;
   }

/**
Gets union of previous call to cosamp sets
@param vecUnion to store the union
*/
   public double[] getUnion(double threshold)
   {
      double[] vecUnion = new double[m];
```

```
      DSLib.unionSet(threshold, s, aSupp, m, Phi, scale, vecUnion);
      return vecUnion;
   }

/**
Get sampling matrix, actual not a copy
@return actual sampling matrix, can alter state
*/
   public double[][] getPhi()
   {
      return Phi;
   }

/**
Sets the sampling matrix
@param Phi sampling matrix, m by N
*/
   public void setPhi(double[][] Phi)
   {
      this.Phi = Phi;
// m = number of rows, N = number of columns
      this.m = Phi.length;
      this.N = Phi[0].length;
      this.scale = new double[N];
      for (int j = 0; j < N; j++)
      {
         double norm2 = 0;
         for (int i = 0; i < m; i++)
         {
            System.out.printf("Phi(%d, %d) = %6.2f\n", i, j, Phi[i][j]);
            norm2 += (Phi[i][j] * Phi[i][j]);
         }
         norm2 = Math.sqrt(norm2);
         scale[j] = norm2;
         for (int i = 0; i < m; i++)
         {
            Phi[i][j] = Phi[i][j] / norm2;
         }
      }
   }

/**
```

```
Sets the scaling factor to recover the original
@param scale
*/
   public void setScale(double[] scale)
   {
      this.scale = scale;
   }

/**
Sets whether to issue debugging output to the terminal
@param debug to true if want debugging output
*/
   public void setDebug(boolean debug)
   {
      this.debug = debug;
   }

/**
Sets whether to issue debugging output to the terminal
@param debug to true if want debugging output
*/
   public void setPrintMatch(boolean printMatch)
   {
      this.printMatch = printMatch;
   }

/**
Sets whether or not to keep computing after diff in signal min
@param cutShort true if can cut short computation
*/
   public void setCutShort(boolean cutShort)
   {
      this.debug = debug;
   }

/**
Clears log
*/
   public void clearLog()
   {
      log.clear();
   }
```

```
/**
Gets log
@return log
*/
   public ArrayList<String> getLog()
   {
      return log;
   }

   public static void main(String[] args)
   {

   }

/**
Initialize CoSaMP run
@param u sample vector
@param s sparsity level
*/
   private void cosampInit(double[] u, int s)
   {
      clearLog();
      this.u = u;
      this.s = s;
      this.approxCache = new double[maxIter + 1][m];
      this.aCache = new double[maxIter + 1][N];
      this.diffSignal = new double[maxIter + 1];
      this.diffSample = new double[maxIter + 1];
      this.tolReached = false;
      this.otherLine = "";
      this.otherTitle = "";

// Pre-allocations
      this.y = new double[N];
      this.Omega = new int[2 * s];
      this.yAbs = new double[N];
      this.b = new double[0];
      this.preSupp = new int[s];
      this.approx = new double[m];

// a <- 0 and support(a) is empty
```

```
      this.aPrev = new double[N];
      DSLib.vecInit(aPrev, 0);
      this.aCurr = new double[N];
      DSLib.vecInit(aCurr, 0);
      this.aLoc = new int[s];
      this.aSupp = new int[0];
      this.aSparse = new double[s];

// v <- u, v is the sample residual
      v = new double[u.length];
      DSLib.vecCopy(u, v);

// k <- 0
      kIter = 0;

      String message = "kIter = " + kIter;
      if (debug)
      {
         log.add(message);
         System.out.println("kIter = " + kIter);
         if (kIter == 0)  // initial v = u
         {
            for (int i = 0; i < m; i++)
            {
               System.out.println("v[" + i + "] = " + v[i]);
            }
         }
      }
   }

/**
Run one iteration of the CoSaMP Recovery Algorithm
*/
   private void cosampIteration()
   {
      String message = "";

      DSLib.vecCopy(aCurr, aPrev);

// Calculate signal proxy y <- Phi* v
      multPhiStar(v, y);
```

```
      if (debug)
      {
         if (kIter == 0)
         {
            for (int i = 0; i < N; i++)
            {
               System.out.println("y[" + i + "] = " + y[i]);
            }
         }
      }

// Find 2 * s largest components of |y|
      DSLib.vecAbs(y, yAbs);
      DSLib.findLargest(yAbs, 2 * s, Omega);

// Merge the supports Omega and support of a.
// Note: resulting T might have less than 3s indices.
      T = DSLib.mergeSupport(Omega, aSupp, aLoc);

      if (debug)
      {
         if ((kIter == 0) || (kIter == 1))
         {
            for (int i = 0; i < T.length; i++)
            {
               System.out.println("T[" + i + "] = " + T[i]);
            }
         }
      }

// zCurr <- pseudoInverse Phi T u
      zCurr = new double[T.length];

// Solve least-squares to obtain zCurr = bT = Phi_T^+ u
      if (methodLS == USE_SVD)
      {
         svdLS();
      }
      else if (methodLS == USE_CG)
      {
         conjugateGradientLS();
      }
```

```
// Prune to the largest s elements of b_T.
// But b_T contains all non-zero elements of b.
// aSupp <- the s largest indices of b relative to columns of Phi.
// aSparse <- the s largest values of b relative to columns of Phi.
// aCurr <- N vector that has as nonzero elements values of aSparse
      if (kIter == 0)  // aSupp was empty originally
      {
         aSupp = new int[s];
      }
      prune();
      multPhiT(aSupp, aSparse, approx);

// Update residual v
      DSLib.vecDiff(u, approx, v);
   }

/**
Stop either when maxIter iterations exceeded or
when no further difference in signal estimate
*/
   private boolean cosampStop()
   {
      String message = "";
      DSLib.vecCopy(approx, approxCache[kIter]);
      DSLib.vecCopy(aCurr, aCache[kIter]);
      diffSample[kIter] = DSLib.norm2(v);
      diffSignal[kIter] = DSLib.norm2(aPrev, aCurr);
      if (kIter == 0)
      {
         indexMinDiffSignal = 0;
         minDiffSignal = diffSignal[0];
         indexMinDiffSample = 0;
         minDiffSample = diffSample[0];
         tolReached = (minDiffSignal < SIGNAL_TOL);
      }
      else if (!tolReached)
      {
         if (diffSignal[kIter] < minDiffSignal)
         {
            indexMinDiffSignal = kIter;
            minDiffSignal = diffSignal[kIter];
```

```
            tolReached = (minDiffSignal < SIGNAL_TOL);
         }
         if (diffSample[kIter] < minDiffSample)
         {
            indexMinDiffSample = kIter;
            minDiffSample = diffSample[kIter];
         }
      }

      StringBuilder sb = new StringBuilder();
      Formatter formatter = new Formatter(sb, Locale.US);
      formatter.format("%4d %10.2f %10.2f %s",
         kIter, diffSample[kIter], diffSignal[kIter], otherLine);
      iterLine = sb.toString();
      log.add(iterLine);
      if (printMatch)
      {
         System.out.printf("%s\n", iterLine);
      }
      return ((kIter < (maxIter - 1)) && (!tolReached));
   }

/**
Multiply m by N matrix Phi with N by 1 vector
@param vec vector
@param mult result, assumed N by 1
*/
   private void multPhi(double[] vec, double[] mult)
   {
      for (int i = 0; i < m; i++)
      {
         mult[i] = 0;
         for (int j = 0; j < N; j++)
         {
            mult[i] += Phi[i][j] * vec[j];
         }
      }
   }

/**
Multiply N by m matrix PhiStar with m by 1 vector
@param vec vector
```

```
@param mult result, assumed N by 1
*/
   private void multPhiStar(double[] vec, double[] mult)
   {
      for (int i = 0; i < N; i++)
      {
         mult[i] = 0;
         for (int j = 0; j < m; j++)
         {
// Phi^* [i][j] = Phi[j][i]
            mult[i] += (Phi[j][i] * vec[j]);
         }
      }
   }

/**
Multiply T.length by m matrix PhiTStar with m by 1 vector
@param T column indices in Phi
@param vec vector
@param mult result, assumed T.length by 1
*/
   private void multPhiTStar
   (
      int[] T, double[] vec, double[] mult
   )
   {
      int nT = T.length;
      for (int i = 0; i < nT; i++)
      {
         mult[i] = 0;
         int coli = T[i];
         for (int j = 0; j < m; j++)
         {
// Phi[j][coli] = Phi^*[coli][j]
            mult[i] += Phi[j][coli] * vec[j];
         }
      }
   }

/**
Multiply m by T.length m matrix PhiT with T.length by 1 vector
@param T indices
```

```
@param vec vector, assumed T.length by 1
@param mult result, assumed preallocated m by 1
*/
   private void multPhiT
   (
      int[] T, double[] vec, double[] mult
   )
   {
      int nT = T.length;
      for (int i = 0; i < m; i++)
      {
         mult[i] = 0;
         for (int j = 0; j < nT; j++)
         {
            mult[i] += Phi[i][T[j]] * vec[j];
         }
      }
   }

/**
Richardson iteration least squares estimation
*/
   private void richardsonLS()
   {
      richardsonInit();
      while (!stopLS())
      {
         richardsonIteration();
      }
   }

/**
Make Phi_T* u the initial guess
*/
   private void richardsonInit()
   {
      errorLSNormal.clear();
      errorLSIter.clear();
      iterLS = 0;
      zLHS = new double[T.length];
      vInter = new double[m];
// z0 = A^* u
```

```
      z0 = new double[T.length];
      DSLib.vecInit(z0, 0);
      AStaru = new double[T.length];
      multPhiTStar(T, u, AStaru);
      if (kIter != 0)
      {
         for (int i = 0; i < s; i++)
         {
            z0[aLoc[i]] = aSparse[i];
         }
      }
// zPrev is z^l
      zPrev = new double[T.length];
      zCurr = new double[T.length];
      DSLib.vecCopy(z0, zCurr);
   }

/**
One Richardson's iteration for least-squares
*/
   private void richardsonIteration()
   {
      DSLib.vecCopy(aCurr, aPrev);
      DSLib.vecCopy(zCurr, zPrev);

// vInter = Phi_T  zPrev
      multPhiT(T, zPrev, vInter);

// zLHS = Phi_T^* vInter = Phi_T^* Phi_T zPrev
      multPhiTStar(T, vInter, zLHS);
// Normal equation solution requires
//    | zLHS - z0 | to decrease to near 0.
      double errorNormal = DSLib.norm2(zLHS, z0);
      errorLSNormal.add(errorNormal);
      log.add("norm2(A*A zPrev - A* u) = " + errorNormal);
// zLHS = zLHS - zPrev
//       = ( Phi_T^* Phi_T - I ) zPrev
//       = ( M - I ) zPrev
      DSLib.vecDiff(zLHS, zPrev, zLHS);
// z = z0 - zLHS
      DSLib.vecDiff(AStaru, zLHS, zCurr);
      errorIter = DSLib.norm2(zCurr, zPrev);
```

```
      errorLSIter.add(errorIter);
      log.add("norm2(zCurr - zPrev) = " + errorIter);
      iterLS++;
   }

/**
Conjugate gradient least squares estimation
*/
   private void conjugateGradientLS()
   {
      conjugateGradientInit();
      while (!stopLS())
      {
         conjugateGradientIteration();
      }
   }

/**
*/
   private void conjugateGradientInit()
   {
      errorLSNormal.clear();
      errorLSIter.clear();
      iterLS = 0;
      zLHS = new double[T.length];
      vInter = new double[m];
// z0 = A^* u
      z0 = new double[T.length];
      DSLib.vecInit(z0, 0);
      AStaru = new double[T.length];
      multPhiTStar(T, u, AStaru);
      if (kIter != 0)
      {
         for (int i = 0; i < s; i++)
         {
            z0[aLoc[i]] = aSparse[i];
         }
      }
// zPrev is z^l
      zPrev = new double[T.length];
      zCurr = new double[T.length];
      DSLib.vecCopy(z0, zCurr);
```

```
      pvec = new double[T.length];
      DSLib.vecCopy(AStaru, pvec);
      qvec = new double[m];
      svec = new double[T.length];
      DSLib.vecCopy(AStaru, svec);

// gamma[0] = s[0]
      gammaPrev = DSLib.norm2(svec);
      gammaPrev = gammaPrev * gammaPrev;
      gamma = gammaPrev;
   }

   private void conjugateGradientIteration()
   {
      DSLib.vecCopy(zCurr, zPrev);
      DSLib.vecCopy(aCurr, aPrev);
      multPhiT(T, pvec, qvec);
      alpha = DSLib.norm2(qvec);
      alpha = gammaPrev / (alpha * alpha);
      double[] zCurrcorr = new double[pvec.length];
      DSLib.vecCopy(zCurr, zPrev);
      DSLib.vecScale(alpha, pvec, zCurrcorr);
      DSLib.vecAdd(zCurr, zCurrcorr, zCurr);
      double[] sveccorr = new double[svec.length];
      multPhiTStar(T, qvec, sveccorr);
      DSLib.vecScale(-alpha, sveccorr, sveccorr);
      DSLib.vecAdd(svec, sveccorr, svec);
      gammaPrev = gamma;
      gamma = DSLib.norm2(svec);
      gamma = gamma * gamma;
      System.out.println("iterLS = " + iterLS + ", gamma = " + gamma);
      beta = gamma / gammaPrev;
      double[] pveccorr = new double[pvec.length];
      DSLib.vecScale(beta, pvec, pveccorr);
      DSLib.vecAdd(svec, pveccorr, pvec);
      iterLS++;
   }

// svd least squares

/**
SVD least squares estimation
```

```
*/
   private void svdLS()
   {
/*
      double[][] A = new double[m][T.length];
      DSLib.submatrix(T, Phi, A);
      double[][] U = new double[m][T.length];
      double[] sigma = new double[T.length];
      double[][] V = new double[T.length][T.length];
      int rank = DSColt.svd(A, U, sigma, V);
      double cond = sigma[0] / sigma[rank - 1];
      zCurr = new double[T.length];
      DSLib.pseudoInverse(U, sigma, V, rank, u, zCurr);
*/
      int rank = svdLS(T, u, zCurr, (double[])null, condStore);
      double cond = condStore[0];

      StringBuilder sb = new StringBuilder();
      Formatter formatter = new Formatter(sb, Locale.US);
      formatter.format("%5d %5d %10.2f", rank, T.length, cond);
      otherLine = sb.toString();
   }

/**
Stopping criterion for conjugate gradient least squares
@return true if should stop
*/
   private boolean stopLS()
   {
      if ((iterLS > s) || (gamma < 0.01))
      {
         return true;
      }
      return false;
   }

/**
Prunes for the s largest components of least-squares b_T = zCurr
*/
   private void prune()
   {
// Least-squares estimate in zCurr for column indices in T
```

```
// Find s largest values in zCurr
      double[] zCurrAbs = new double[zCurr.length];
      DSLib.vecAbs(zCurr, zCurrAbs);
      DSLib.findLargest(zCurrAbs, s, preSupp);
      DSLib.vecInit(aCurr, 0);
      for (int i = 0; i < s; i++)
      {
         aSupp[i] = T[preSupp[i]];
         aSparse[i] = zCurr[preSupp[i]];
         aCurr[aSupp[i]] = aSparse[i];
      }
   }

   public static final int USE_SVD = 0;
   public static final int USE_CG = 1;
   public static final int USE_RICHARDSON = 2;
   public static final int MAX_ITERATION = 15;
   public static final double SIGNAL_TOL = 0.01;

   public static final long serialVersionUID = 1225393870755107024L;

// Private state for all cosamp variations

   private int methodLS = USE_SVD;
   private boolean debug = false;
   private boolean printMatch = true;
   private boolean cutShort = false;

// For monitoring how errors evolve
   private double[][] approxCache;
   private double[][] aCache;
   private double[] diffSignal;
   private double[] diffSample;
   private int indexMinDiffSignal = 0;
   private double minDiffSignal = 0;
   private int indexMinDiffSample = 0;
   private double minDiffSample = 0;
   private boolean tolReached = false;

// For output
   private String iterLine = "";
   private String otherLine = "";
```

```
private String otherTitle = "";

private int m;
private int N;
private double[][] Phi;
private double[] scale;
private double[] y;
private double[] yAbs;
private int[] Omega;
private int[] T;
private int[] aSupp;
private int[] preSupp;
private double[] aSparse;
private int kIter;
private int s;
private double[] u;
private double[] v;
private double[] vInter;
private double[] aPrev;
private double[] aCurr;
private int[] aLoc;
private double[] approx;
private double[] AStaru;
private double[] z;
private double[] z0;
private double[] zPrev;
private double[] zCurr;
private double[] zLHS;
private double[] b;
private double errorZ;
private double errorIter;
private ArrayList<Double> errorLSNormal =
   new ArrayList<Double>();
private ArrayList<Double> errorLSIter =
   new ArrayList<Double>();
private ArrayList<Double> errorA = new ArrayList<Double>();
private ArrayList<Double> errorResidual = new ArrayList<Double>();
private int iterLS;
private ArrayList<String> log = new ArrayList<String>();

private int maxIter = MAX_ITERATION;
```

```
// Conjugate gradient
   private double[] pvec;
   private double[] qvec;
   private double[] rvec;
   private double[] svec;
   private double[] xvec;
   private double gammaPrev;
   private double gamma;
   private double alpha;
   private double beta;

// Private state for SVD
   private double[] condStore = new double[1];
}
```

A.2 DSLib.java

We implement most of our functionality as concepts as static functions. We even add functions for simple vector operations such as copying to simplify the linguistic task of translation.

```
import java.io.*;
import java.util.*;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.filechooser.*;
import javax.imageio.*;
import javax.imageio.stream.*;

public class DSLib
{

/**
Converts a double to the nearest integer from 0 to 255
for 8-bit grayscale
@param intensity number to be converted
@return nearest integer from 0 to 255
*/
```

```
   public static int toGray(double intensity)
   {
      int gray = (int)Math.round(intensity);
      if (gray < 0)
      {
         gray = 0;
      }
      else if (gray > 255)
      {
         gray = 255;
      }
      return gray;
   }

/**
Converts an xBound by yBound array of doubles to
a grayscale image
@param xBound width
@param yBound height
@param pixels grayscale values as pixels
*/
   public static void doubleToGrayImage
   (
      double[] values,
      BufferedImage aBufferedImage
   )
   {
      int width = aBufferedImage.getWidth();
      int height = aBufferedImage.getHeight();
      WritableRaster aWritableRaster = aBufferedImage.getRaster();
// getPixels() for TYPE_BYTE_INT seemed to indicate int array length 1
      int[] toLoad = new int[1];
      for (int i = 0; i < height; i++)
      {
         for (int j = 0; j < width; j++)
         {
            toLoad[0] = toGray(values[i * width + j]);
            aWritableRaster.setPixel(i, j, toLoad);
         }
      }
   }
```

```
/**
Converts a grayscale image to an array of doubles
@param aBufferedImage image
@param values grayscale values
*/
   public static void grayImageToDouble
   (
      BufferedImage aBufferedImage, double[] values
   )
   {
      int width = aBufferedImage.getWidth();
      int height = aBufferedImage.getHeight();
      WritableRaster aWritableRaster = aBufferedImage.getRaster();
      int[] toLoad;
      for (int i = 0; i < height; i++)
      {
         for (int j = 0; j < width; j++)
         {
            toLoad =
               aWritableRaster.getPixel(i, j, (int[])null);
            values[i * width + j] = (double)toLoad[0];
         }
      }
   }

/**
Takes a subset of sampling matrix and
creates a black 255, white 0 vector
@param threshold for conversion
@param Tlength length of indices subset
@param T vector of indices subset
@param m number of measurements
@param Phi sample matrix
@param vecUnion m by 1 vector union
*/
   public static void unionSet
   (
      double threshold,
      int Tlength, int[] T, int m, double[][] Phi, double[] scale,
      double[] vecUnion
   )
   {
```

```
      for (int i = 0; i < m; i++)
      {
         double sum = 0;
         for (int j = 0; j < Tlength; j++)
         {
            sum += (Phi[i][T[j]] * scale[T[j]]);
         }
         if (sum >= threshold)
         {
            vecUnion[i] = 255;
         }
         else
         {
            vecUnion[i] = 0;
         }
      }
   }

/**
Creates an image resized by an integer factor
@param factor factor
@param aBufferedImage original image
@return new resized image
*/
   public static BufferedImage resizeImage
   (
      int factor, BufferedImage aBufferedImage
   )
   {
      int width = aBufferedImage.getWidth();
      int height = aBufferedImage.getHeight();
      int resizedWidth = factor * width;
      int resizedHeight = factor * height;
      WritableRaster aWritableRaster = aBufferedImage.getRaster();
      BufferedImage resized = new BufferedImage(
         resizedWidth, resizedHeight, BufferedImage.TYPE_BYTE_GRAY);
      WritableRaster bWritableRaster = resized.getRaster();
      for (int i = 0; i < resizedHeight; i++)
      {
         for (int j = 0; j < resizedWidth; j++)
         {
            int[] toLoad = aWritableRaster.getPixel(
```

```
               i / factor, j / factor, (int[])null);
            bWritableRaster.setPixel(i, j, toLoad);
         }
      }
      return resized;
   }

// findLargest and merge operations on indices

/**
Finds the toGet largest values of vec, and
records the corresponding indices.
@param vec vector
@param toGet how many of the largest values to find
@param indices indices of largest values in ascending order,
   assumed to be preallocated of length toGet
*/
   public static void findLargest
   (
      double[] vec, int toGet, int[] indices
   )
   {

// DSDoubleSort carries indices with values when sorted
      int n = vec.length;
      DSDoubleSort[] vecSorted = new DSDoubleSort[n];
      for (int i = 0; i < n; i++)
      {
         vecSorted[i] = new DSDoubleSort(i, vec[i]);
      }

// Takes time n log n, replaces vecSorted
      java.util.Arrays.sort(vecSorted);

// toGet largest value of sorted
//    (n - 1) - (n - toGet) + 1 == toGet
      for (int i = 0; i < toGet; i++)
      {
         indices[i] = vecSorted[i + (n - toGet)].getIndex();
      }

// Sort indices again so that they are in order
```

```
      java.util.Arrays.sort(indices);
   }

/**
Merge two lists of ints already in ascending order
@param vecone first list
@param vectwo second list
@param Tint already allocated result
@param twoloc already allocated, indices of vectwo in Tint
*/
   public static int merge
   (
      int[] vecone, int[] vectwo, int[] Tint, int[] twoloc
   )
   {
      int headone = 0;
      int headtwo = 0;
      int leftone = vecone.length - headone;
      int lefttwo = vectwo.length - headtwo;
      int filled = 0;
      while ((leftone > 0) || (lefttwo > 0))
      {
         if (leftone == 0) // only from list two
         {
            Tint[filled] = vectwo[headtwo];
            twoloc[headtwo] = filled;
            filled++;
            headtwo++;
            lefttwo--;
         }
         else if (lefttwo == 0) // only from list one
         {
            Tint[filled] = vecone[headone];
            filled++;
            headone++;
            leftone--;
         }
         else // both lists, compare values at heads
         {
            if (vecone[headone] > vectwo[headtwo])
            {
               Tint[filled] = vectwo[headtwo];
```

```
               twoloc[headtwo] = filled;
               filled++;
               headtwo++;
               lefttwo--;
            }
            else if (vecone[headone] < vectwo[headtwo])
            {
               Tint[filled] = vecone[headone];
               filled++;
               headone++;
               leftone--;
            }
            else // special case values equal both removed
            {
               Tint[filled] = vectwo[headtwo];
               twoloc[headtwo] = filled;
               filled++;
               headone++;
               leftone--;
               headtwo++;
               lefttwo--;
            }
         }
      }
      return filled;
   }

/**
Merge two sorted vectors of indices in ascending order
@param vecone first vector
@param vectwo second vector
@param twoloc preallocated where vectwo is in merged
@return sorted vector of unique indices in ascending order
*/
   public static int[] mergeSupport
   (
      int[] vecone, int[] vectwo, int[] twoloc
   )
   {
      int combined = vecone.length + vectwo.length;
      int[] Tint = new int[combined];
```

```
      int filled = DSLib.merge(vecone, vectwo, Tint, twoloc);
      if (filled == combined)
      {
         return Tint;
      }
      int[] T = new int[filled];
      for (int i = 0; i < filled; i++)
      {
         T[i] = Tint[i];
      }
      return T;
   }

// Vector operations

/**
Find the absolute value of each entry of a vector
@param vec original vector
@param absVec result
*/
   public static void vecAbs(double[] vec, double[] absVec)
   {
      int n = vec.length;
      for (int i = 0; i < n; i++)
      {
         absVec[i] = Math.abs(vec[i]);
      }
   }


/**
Initialize a vector to some value
@param vec vector
@param value value
*/
   public static void vecInit(double[] vec, double value)
   {
      int n = vec.length;
      for (int i = 0; i < n; i++)
      {
         vec[i] = value;
      }
```

```
   }

/**
Finds the 2-norm of a vector
@param vec vector
@returns 2-norm
*/
   public static double norm2(double[] vec)
   {
      double norm2 = 0;
      int nVec = vec.length;
      for (int i = 0; i < nVec; i++)
      {
         double diff = vec[i];
         norm2 += (diff * diff);
      }
      return Math.sqrt(norm2);
   }

/**
Finds the 2-norm of the difference between two vectors
@param a first vector
@param b second vector
@returns 2-norm
*/
   public static double norm2(double[] a, double[] b)
   {
      double norm2 = 0;
      for (int i = 0; i < a.length; i++)
      {
         double diff = a[i] - b[i];
         norm2 += (diff * diff);
      }
      return Math.sqrt(norm2);
   }

/**
Finds the scalar product between two vectors
@param a first vector
@param b second vector
@return scalar product
*/
```

```
public static double dotProduct(double[] a, double[] b)
{
   double dot = 0;
   for (int i = 0; i < a.length; i++)
   {
      dot += (a[i] * b[i]);
   }
   return dot;
}

public static void vecDiff(double[] a, double[] b, double[] c)
{
   int n = a.length;
   for (int i = 0; i < n; i++)
   {
      c[i] = a[i] - b[i];
   }
}

public static void vecAdd(double[] a, double[] b, double[] c)
{
   int n = a.length;
   for (int i = 0; i < n; i++)
   {
      c[i] = a[i] + b[i];
   }
}

public static void vecScale(double a, double[] b, double[] c)
{
   int n = b.length;
   for (int i = 0; i < n; i++)
   {
      c[i] = a * b[i];
   }
}

public static void vecCopy(double[] from, double[] to)
{
   int n = from.length;
   for (int i = 0; i < n; i++)
   {
```

```
      to[i] = from[i];
   }
}

public static void matrixCopy(double[][] A, double[][] B)
{
   int m = A.length;
   int n = A[0].length;
   for (int i = 0; i < m; i++)
   {
      for (int j = 0; j < n; j++)
      {
         B[i][j] = A[i][j];
      }
   }
}

public static void multMatrixVec
(
   double[][] A, double[] x, double[] b
)
{
   int m = A.length;
   int n = A[0].length;
   for (int i = 0; i < m; i++)
   {
      b[i] = 0;
      for (int j = 0; j < n; j++)
      {
         b[i] += (A[i][j] * x[j]);
      }
   }
}

public static void multMatrixTVec
(
   double[][] A, double[] x, double[] b
)
{
   int m = A.length;
   int n = A[0].length;
   for (int i = 0; i < n; i++)
```

```
      {
         b[i] = 0;
         for (int j = 0; j < m; j++)
         {
            b[i] += (A[j][i] * x[j]);
         }
      }
   }

/**
*/
   public static void pseudoInverse
   (
      double[][] U, double[] sigma, double[][] V, int rank,
      double[] b, double[] x
   )
   {
      int m = U.length;
      int n = V.length;
      double[] xInter = new double[x.length];
// x = V * sigma-1 * UT b
      multMatrixTVec(U, b, xInter);
      for (int i = 0; i < rank; i++)
      {
         xInter[i] *= (1.0 / sigma[i]);
      }
      multMatrixVec(V, xInter, x);
   }

   public static void submatrix
   (
      int[] T, double[][] Phi, double[][] A
   )
   {
      int m = Phi.length;
      for (int i = 0; i < m; i++)
      {
         for (int j = 0; j < T.length; j++)
         {
            A[i][j] = Phi[i][T[j]];
         }
      }
```

```
   }

// Utility code for manipulating images

   public static void writeLatex
   (
      String filename, File parentDir, String letterString
   )
      throws IOException
   {
      PrintWriter toLatex = new PrintWriter(
         new FileWriter(
            new File(parentDir, filename + ".tex"), true));
      toLatex.println("\\documentclass[12pt]{article}");
      toLatex.println("\\thispagestyle{empty}");
      toLatex.println("\\begin{document}");
      toLatex.println(letterString);
      toLatex.println("\\end{document}");
      toLatex.close();
   }

   public static ArrayList<String> texAlphabetical
   (
      String pt,
      String series, String shape, String family,
      String charLine
   )
   {
      ArrayList<String> texText = new ArrayList<String>();
      texText.add("\\documentclass[" + pt + "]{article}");
      texText.add("\\thispagestyle{empty}");
      texText.add("\\begin{document}");
      texText.add("\\" + series + "{" +
         "\\" + shape + "{" +
         "\\" + family + "{" +
         charLine + "}}}");
      texText.add("\\end{document}");
      return texText;
   }

   public static void writeTextFile
   (
```

```
      String fullFilename, ArrayList<String> text
   ) throws IOException
   {
      PrintWriter aWriter = new PrintWriter(
         new FileWriter(fullFilename, true)
      );
      for (String str: text)
      {
         aWriter.println(str);
      }
      aWriter.close();
   }

/**
Create a new process and run a command
@param command command to run
@param directory working directory
*/
   public static int execCommand
   (
      ArrayList<String> command,
      File directory,
      ArrayList<String> output
   ) throws IOException, InterruptedException
   {
//      for (String str : command)
//      {
//         System.out.print(str + " ");
//      }
//      System.out.println();

      ProcessBuilder pb = new ProcessBuilder(command);
      pb.directory(directory);
      Process p = pb.start();
      Scanner execOutput = new Scanner(p.getInputStream());
      int n = p.waitFor();
      if (output != null)
      {
         while (execOutput.hasNextLine())
         {
            output.add(execOutput.nextLine());
         }
```

```
      }
      return n;
   }

/**
LaTeX a file
@param filename file name without extension
*/
   public static void latexFile(String filename, File parentDir)
   {
      ArrayList<String> command = new ArrayList<String>();
      String program = "latex";
      command.add(program);
      command.add(filename + ".tex");
      try
      {
         int n = DSLib.execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
   }

/**
Create PostScript file from .dvi file
@param filename file name without extension
@param parentDir parent directory of .dvi file
@param dpi resolution
*/
   public static void dvipsFile
   (
      String filename, File parentDir, int dpi
   )
   {
      ArrayList<String> command = new ArrayList<String>();
      String program = "dvips";
      command.add("dvips");
      command.add("-D" + dpi);
      command.add(filename + ".dvi");
      try
```

```
      {
         int n = DSLib.execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
   }

/**
Command and args for converting PostScript to another image format
@param filename file name without extension
@param parentDir parent directory of file
@param dpi resolution
@param device device for GhostScript, image format
@param ext image format extension
*/
   public static void psToImage
   (
      String filename, File parentDir,
      int dpi, String device, String ext
   )
   {
      ArrayList<String> command = new ArrayList<String>();
      String program = "gs";
      command.add(program);
      command.add("-r" + dpi);
      command.add("-DEPSCrop");
      command.add("-DTextAlphaBits=4");
      command.add("-sDEVICE=" + device);
      command.add("-sOutputFile=" + filename + "." + ext);
      command.add("-dBATCH");
      command.add("-dNOPAUSE");
      command.add(filename + ".ps");
      try
      {
         int n = DSLib.execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
```

```
         ioe.printStackTrace();
      }
   }

/**
Commands and args for converting PostScript to blurred image format
@param filename file name without extension
@param parentDir parent directory
@param ext image format extension
*/
   public static void psToBlurredImage
   (
      String filename, File parentDir,
      String ext
   )
   {
      ArrayList<String> command = new ArrayList<String>();

// First convert from .ps to .pdf, causes blurring
      String program = "convert";
      command.add(program);
      command.add(filename + ".ps");
      command.add(filename + ".pdf");

      try
      {
         int n = DSLib.execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);

// Now convert from .pdf to image format
         command.clear();
         command.add(program);
         command.add(filename + ".pdf");
         command.add(filename + "." + ext);
         n = DSLib.execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
   }
```

```
/**
Command and args for trimming image to character outline
@param filename file name without extension
@param parentDir parent directory of file
@param width width of resulting image
@param height height of resulting image
@param extFrom image format extension of original
@param extTo image format for result
*/
   public static void trimImage
   (
      String filename, File parentDir,
      int width, int height,
      String extFrom, String extTo
   )
   {
      ArrayList<String> list = new ArrayList<String>();
      String program = "convert";
      list.add(program);
      list.add(filename + "." + extFrom);
      list.add("-trim");
      list.add("-depth");
      list.add("8");
      list.add("-type");
      list.add("Grayscale");
      list.add("-adaptive-resize");
      list.add(width + "x" + height);
      list.add("-gravity");
      list.add("center");
      list.add("-extent");
      list.add(width + "x" + height);
//      list.add("-morphology");
//      list.add("Erode");
//      list.add("Diamond");
      list.add(filename + "." + extTo);
      try
      {
         int n = DSLib.execCommand(list, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
```

```
      {
         ioe.printStackTrace();
      }
   }

/**
Command and args for thickening lines of image
@param filename file name without extension
@param parentDir parent directory of file
@param width width of resulting image
@param height height of resulting image
@param extFrom image format extension of original
@param extTo image format for result
*/
   public static void thickenImage
   (
      String filename, File parentDir,
      int width, int height,
      String extFrom, String extTo
   )
   {
      ArrayList<String> list = new ArrayList<String>();
      String program = "convert";
      list.add(program);
      list.add(filename + "." + extFrom);
      list.add("-morphology");
      list.add("Erode");
      list.add("Diamond");
      list.add(filename + "." + extTo);
      try
      {
         int n = execCommand(list, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
   }

/**
Loads an image file given its filename and extension
```

```
@param filename file name
@param ext extension
@return buffered image
*/
   public static BufferedImage readImageFile
   (
      File parentDir, String filename, String ext
   ) throws IOException
   {
      File imageFile = new File(
         parentDir.getPath() + File.separator + filename + "." + ext);
//      System.out.printf("Reading %s\n", imageFile.getName());
      BufferedImage aBufferedImage = ImageIO.read(imageFile);
      return aBufferedImage;
   }

/**
Remove file if it exists
@param filename file name with extension
@param parentDir parent directory
*/
   public static void rmFile
   (
      String filename, File parentDir
   )
   {
      ArrayList<String> command = new ArrayList<String>();
      String program = "rm";
      command.add(program);
      command.add(filename);
      try
      {
         int n = execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
   }

/**
```

```
Creates gray image from character string in LaTeX.
@param letterString character string to produce character in LaTeX
@param dpi dpi
*/
   public static BufferedImage charToImage
   (
      String letterString, int dpi, int xBound, int yBound
   ) throws IOException, InterruptedException
   {
      String filename = TLATEX;
      File parentDir = new File(".");
      String device = "pnggray";
      String extFrom = "png";
      String extTo = "png";
      DSLib.rmFile(filename + ".tex", parentDir);
      DSLib.writeLatex(filename, parentDir, letterString);
      DSLib.latexFile(filename, parentDir);
      DSLib.dvipsFile(filename, parentDir, dpi);
      DSLib.psToImage(filename, parentDir, dpi, device, extFrom);
      DSLib.trimImage(filename, parentDir, xBound, yBound, extFrom, extTo);
      BufferedImage image = DSLib.readImageFile(
         parentDir, filename, extTo);
      return image;
   }

/**
Creates thickened gray image from character string in LaTeX.
@param letterString character string to produce character in LaTeX
@param dpi dpi
*/
   public static BufferedImage charToThickenedImage
   (
      String letterString, int dpi, int xBound, int yBound
   ) throws IOException, InterruptedException
   {
      String filename = TLATEX;
      File parentDir = new File(".");
      String device = "pnggray";
      String extFrom = "png";
      String extTo = "png";
      DSLib.rmFile(filename + ".tex", parentDir);
      DSLib.writeLatex(filename, parentDir, letterString);
```

```
      DSLib.latexFile(filename, parentDir);
      DSLib.dvipsFile(filename, parentDir, dpi);
      DSLib.psToImage(filename, parentDir, dpi, device, extFrom);
      DSLib.trimImage(filename, parentDir, xBound, yBound,
         extFrom, extTo);
      DSLib.thickenImage(filename, parentDir, xBound, yBound,
         extTo, extTo);
      BufferedImage image = DSLib.readImageFile(
         parentDir, filename, extTo);
      return image;
   }

/**
Creates blurred gray image from character string in LaTeX.
@param letterString character string to produce character in LaTeX
@param dpi dpi
@param xBound width
@param yBound height
*/
   public static BufferedImage charToBlurredImage
   (
      String letterString, int dpi, int xBound, int yBound
   ) throws IOException, InterruptedException
   {
      String filename = TLATEX;
      File parentDir = new File(".");
      String device = "pnggray";
      String extFrom = "png";
      String extTo = "png";
      DSLib.rmFile(filename + ".tex", parentDir);
      DSLib.writeLatex(filename, parentDir, letterString);
      DSLib.latexFile(filename, parentDir);
      DSLib.dvipsFile(filename, parentDir, dpi);
      DSLib.psToBlurredImage(filename, parentDir, extFrom);
      DSLib.trimImage(filename, parentDir, xBound, yBound, extFrom, extTo);
      BufferedImage image = DSLib.readImageFile(
         parentDir, filename, extTo);
      return image;
   }

// Generating random shapes in a rectangle
```

```
/**
Makes a random quadratic Bezier curve restricted to half the area
@param xBound x pixels numbered from 0 to xBound - 1
@param yBound y pixels numbered from 0 to yBound - 1
@param fracDistance maximum fraction of distance allowed
@param rand random number generator
@return random QuadCurve2D in bounds
*/
   public static QuadCurve2D randQuad
   (
      int xBound, int yBound, double fracDistance, Random rand
   )
   {
      double x1 = Math.floor(xBound * rand.nextDouble());
      double y1 = Math.floor(yBound * rand.nextDouble());
      double ctrlx = Math.floor(xBound * rand.nextDouble());
      double ctrly = Math.floor(yBound * rand.nextDouble());
      double x2 = Math.floor(yBound * rand.nextDouble());
      double y2 = Math.floor(yBound * rand.nextDouble());
      double hX = (double)xBound * fracDistance;
      double hY = (double)yBound * fracDistance;
      while
      (
         (Math.abs(x1 - x2) > hX) || (Math.abs(y1 - y2) > hY) ||
         (Math.abs(x1 - ctrlx) > hX) || (Math.abs(y1 - ctrly) > hY) ||
         (Math.abs(ctrlx - x2) > hX) || (Math.abs(ctrly - y2) > hY)
      )
      {
         x1 = Math.floor(xBound * rand.nextDouble());
         y1 = Math.floor(yBound * rand.nextDouble());
         ctrlx = Math.floor(xBound * rand.nextDouble());
         ctrly = Math.floor(yBound * rand.nextDouble());
         x2 = Math.floor(yBound * rand.nextDouble());
         y2 = Math.floor(yBound * rand.nextDouble());
      }
      return new QuadCurve2D.Double(x1, y1, ctrlx, ctrly, x2, y2);
   }

/**
Makes a random cubic Bezier curve with two control points
@param xBound x pixels numbered from 0 to xBound - 1
@param yBound y pixels numbered from 0 to yBound - 1
```

```
@return random QuadCurve2D in bounds
*/
   public static CubicCurve2D randCubic
   (
      int xBound, int yBound, double fracDistance, Random rand
   )
   {
      double x1 = Math.floor(xBound * rand.nextDouble());
      double y1 = Math.floor(yBound * rand.nextDouble());
      double ctrlx1 = Math.floor(xBound * rand.nextDouble());
      double ctrly1 = Math.floor(yBound * rand.nextDouble());
      double ctrlx2 = Math.floor(xBound * rand.nextDouble());
      double ctrly2 = Math.floor(yBound * rand.nextDouble());
      double x2 = Math.floor(yBound * rand.nextDouble());
      double y2 = Math.floor(yBound * rand.nextDouble());
      double hX = (double)xBound * fracDistance;
      double hY = (double)yBound * fracDistance;
      while
      (
         (Math.abs(x1 - x2) > hX) || (Math.abs(y1 - y2) > hY) ||
         (Math.abs(x1 - ctrlx1) > hX) || (Math.abs(y1 - ctrly1) > hY) ||
         (Math.abs(x1 - ctrlx2) > hX) || (Math.abs(y1 - ctrly2) > hY) ||
         (Math.abs(ctrlx1 - x2) > hX) || (Math.abs(ctrly1 - y2) > hY) ||
         (Math.abs(ctrlx2 - x2) > hX) || (Math.abs(ctrly2 - y2) > hY)
      )
      {
         x1 = Math.floor(xBound * rand.nextDouble());
         y1 = Math.floor(yBound * rand.nextDouble());
         ctrlx1 = Math.floor(xBound * rand.nextDouble());
         ctrly1 = Math.floor(yBound * rand.nextDouble());
         ctrlx2 = Math.floor(xBound * rand.nextDouble());
         ctrly2 = Math.floor(yBound * rand.nextDouble());
         x2 = Math.floor(yBound * rand.nextDouble());
         y2 = Math.floor(yBound * rand.nextDouble());
      }
      return new CubicCurve2D.Double(
         x1, y1, ctrlx1, ctrly1, ctrlx2, ctrly2, x2, y2);
   }

/**
Makes one random cubic Bezier stroke
@param xBound x bound of image rectangle
```

```
@param yBound y bound of image rectangle
@param strokeWidth stroke width
@param fracDistance maximum distance control pt from end pt
@param rand random number generator
@return pixels enumerated top down and left to right from 0 to 255
*/
   public static double[] randomCubicStroke
   (
      int xBound, int yBound, float strokeWidth,
      double fracDistance, Random rand
   )
   {
      BufferedImage image = new BufferedImage(xBound, yBound,
         BufferedImage.TYPE_BYTE_GRAY);
      Graphics2D graphics2D = image.createGraphics();
      graphics2D.setStroke(new BasicStroke(strokeWidth));
      graphics2D.draw(
         DSLib.randCubic(xBound, yBound, fracDistance, rand));
      double[] pixels = new double[xBound * yBound];
      DSLib.grayImageToDouble(image, pixels);
      return pixels;
   }

/**
Makes random samples
@param xBound width
@param yBound height
@param strokeWidth stroke width
@param N number of samples to make
@param fracDistance fractional distance allowed
@param rand random number generator
@return Phi, sample matrix
*/
   public static double[][] makeRandomCubicSample
   (
      int width, int height, float strokeWidth, int N,
      double fracDistance, Random rand
   )
   {
      int m = width * height;
      double[][] Phi = new double[m][N];
      for (int j = 0; j < N; j++)
```

```
      {
         double[] pixels = DSLib.randomCubicStroke(
            width, height, strokeWidth, fracDistance, rand);
//         DSLib.flipColor(pixels, pixels);
         for (int i = 0; i < m; i++)
         {
            Phi[i][j] = pixels[i];
         }
      }
      return Phi;
   }

/**
For integer valued colors from 0 to 255, flip
@param orig original source
@param toFlip new picture with colors flipped
*/
   public static void flipColor(double[] orig, double[] toFlip)
   {
      for (int i = 0; i < orig.length; i++)
      {
         toFlip[i] = 255 - orig[i];
      }
   }

// Public constants
/* Filename without extension for temporary latex file */
   public static final String TLATEX = "templ";

} /* Class DSLib */
```

A.3 DSColt.java

This is basically a wrapper around calling Colt's SVD methods.

```
import cern.colt.*;
import cern.colt.matrix.*;
import cern.colt.matrix.impl.*;
import cern.colt.matrix.linalg.*;

public class DSColt
{
```

```
/**
Least squares using SVD
@param A matrix with number of columns n less than number of rows m
@param U left orthogonal m by n matrix
@param sigma singular values of A
@param V right orthogonal n by n matrix such that A = U sigma VT
*/
   public static int svd
   (
      double[][] A, double[][] U, double [] sigma, double[][] V
   )
   {
      int m = A.length;
      int n = A[0].length;
      DoubleMatrix2D coltA = new DenseDoubleMatrix2D(A);
      SingularValueDecomposition coltSVD =
         new SingularValueDecomposition(coltA);
      double[] coltSigma = coltSVD.getSingularValues();
      DSLib.vecCopy(coltSigma, sigma);
      double[][] coltU = coltSVD.getU().toArray();
      DSLib.matrixCopy(coltU, U);
      double[][] coltV = coltSVD.getV().toArray();
      DSLib.matrixCopy(coltV, V);
      return coltSVD.rank();
   }
}
```

A.4 DSCharacter.java

Having defined our terms in `DSLib.java`, we can mass process as many LaTeX characters as we want.

```
import java.io.*;
import java.util.*;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.filechooser.*;
import javax.imageio.*;
```

```
import javax.imageio.stream.*;

/**
Class to classify patterns as characters
*/
public class DSCharacter implements Serializable
{

/**
Sets the implementation of CoSaMP, in particular, the
sampling matrix Phi
@param model used for analyzing patterns
*/
   public void setCosamp(DSCosamp model)
   {
      this.model = model;
   }

/**
Assumed cosamp implementation with Phi already set,
then simply use this on each character's image
@param dpi dpi
@param xBound width in pixels
@param yBound height in pixels
*/
   public void trainSimple
   (
      int dpi, int xBound, int yBound, int s
   ) throws IOException, InterruptedException
   {
      int nChar = baseChar.length;
      int m = xBound * yBound;
      V = new double[nChar][1][m];
      origChar = new double[nChar][1][m];
      double[] u = new double[m];
      double[] uFlipped = new double[u.length];
      double[] approx = new double[u.length];
      for (int k = 0; k < nChar; k++)
      {
         String letterString = "";
         if (baseChar[k][1].equals(GREEK))
         {
```

```
            letterString = "$\\" + baseChar[k][0] + "$";
         }
         else // default, simple alphabetical / numeric
         {
            letterString = baseChar[k][0];
         }
         System.out.println("\nTraining character " +
            baseChar[k][0]);
         BufferedImage image = DSLib.charToImage(
            letterString, dpi, xBound, yBound);
         DSLib.grayImageToDouble(image, u);
         DSLib.flipColor(u, uFlipped);
         DSLib.vecCopy(uFlipped, origChar[k][0]);
         model.cosamp(uFlipped, s);
         DSLib.vecCopy(model.getApprox(), V[k][0]);
         aSuppChar[k][0] = model.getASupp();
      }
   }

/**
Gets LaTeX string to produce character
@param index index into character array
@return LaTeX string
*/
   public String getLetterString(int k)
   {
      if (baseChar[k][1].equals(GREEK))
      {
         return "$\\" + baseChar[k][0] + "$";
      }
      else // default, simple alphabetical / numeric
      {
         return baseChar[k][0];
      }
   }

/**
Assuming cosamp implementation with Phi already set,
trains on thickened character's image
@param dpi dpi
@param xBound width in pixels
@param yBound height in pixels
```

```
*/
   public void trainThicken
   (
      int dpi, int xBound, int yBound, int s
   ) throws IOException, InterruptedException
   {
      int nChar = baseChar.length;
      int m = xBound * yBound;
      V = new double[nChar][1][m];
      origChar = new double[nChar][1][m];
      double[] u = new double[m];
      double[] uFlipped = new double[u.length];
      double[] approx = new double[u.length];
      for (int k = 0; k < nChar; k++)
      {
         String letterString = "";
         if (baseChar[k][1].equals(GREEK))
         {
            letterString = "$\\" + baseChar[k][0] + "$";
         }
         else // default, simple alphabetical / numeric
         {
            letterString = baseChar[k][0];
         }
         System.out.println("\nTraining character " +
            baseChar[k][0]);
         BufferedImage image = DSLib.charToThickenedImage(
            letterString, dpi, xBound, yBound);
         DSLib.grayImageToDouble(image, u);
         DSLib.flipColor(u, uFlipped);
         DSLib.vecCopy(uFlipped, origChar[k][0]);
         model.cosamp(uFlipped, s);
         DSLib.vecCopy(model.getApprox(), V[k][0]);
         aSuppChar[k][0] = model.getASupp();
      }
   }

/**
Classify using simple dot product of test with original char
@param toTest m by 1 vector to test
@param forResult m by 1 character vector closest
@param score scores for all characters
```

```
@return character name of best match
*/
   public String classifyOriginal
   (
      double[] toTest, double[] forResult, DSDoubleSort[] score
   )
   {
      int nChar = baseChar.length;
      int indexMax = 0;
      double valueMax = 0;
      int m = toTest.length;
      double[] charOrig = new double[m];
      double[] condNum = new double[1];

      for (int k = 0; k < nChar; k++)
      {
         DSLib.vecCopy(origChar[k][0], charOrig);
         double dotProd = DSLib.dotProduct(charOrig, toTest);
         double scale1 = DSLib.norm2(charOrig);
         double scale2 = DSLib.norm2(toTest);
// Find best projections on indices aSupp
         double tested = Math.abs(dotProd / (scale1 * scale2));
         if (tested > valueMax)
         {
            indexMax = k;
            valueMax = tested;
         }
         if (score != null)
         {
            score[k] = new DSDoubleSort(k, tested);
         }
         if (debug)
         {
            System.out.printf("%8s %8.5f\n",
               baseChar[k][0], tested);
         }
      }
      DSLib.vecCopy(V[indexMax][0], forResult);
      return baseChar[indexMax][0];
   }

/**
```

```
Classify by taking test dot product with trained CoSaMP vector
@param toTest m by 1 vector to test
@param forResult m by 1 character vector closest
@param score scores for all characters
@return character name of best match
*/
   public String classifySimple
   (
      double[] toTest, double[] forResult, DSDoubleSort[] score
   )
   {
      int nChar = baseChar.length;
      int indexMax = 0;
      double valueMax = 0;
      int m = toTest.length;
      double[] charApprox = new double[m];
      for (int k = 0; k < nChar; k++)
      {
         DSLib.vecCopy(V[k][0], charApprox);
         double scale1 = DSLib.norm2(charApprox);
         double scale2 = DSLib.norm2(toTest);
         double dotProd = DSLib.dotProduct(toTest, charApprox);
         double tested = Math.abs(dotProd / (scale1 * scale2));
         if (tested > valueMax)
         {
            indexMax = k;
            valueMax = tested;
         }
         if (score != null)
         {
            score[k] = new DSDoubleSort(k, tested);
         }
         if (debug)
         {
            System.out.printf("%8s %8.5f\n",
               baseChar[k][0], tested);
         }
      }
      DSLib.vecCopy(V[indexMax][0], forResult);
      return baseChar[indexMax][0];
   }
```

```
/**
Classify using least squares projection on s-sparse support
@param toTest m by 1 vector to test
@param forResult m by 1 character vector closest
@param score scores for all characters
@return character name of best match
*/
   public String classifySupport
   (
      double[] toTest, double[] forResult, DSDoubleSort[] score
   )
   {
      int s = aSuppChar[0][0].length;
      int nChar = baseChar.length;
      int indexMax = 0;
      double valueMax = 0;
      int m = toTest.length;
      int[] aSupp = new int[s];
      double[] charOrig = new double[m];
      double[] testSignal = new double[s];
      double[] charSignal = new double[s];
      double[] testApprox = new double[m];
      double[] charApprox = new double[m];
      double[] condNum = new double[1];

      for (int k = 0; k < nChar; k++)
      {
         DSLib.vecCopy(origChar[k][0], charOrig);
         model.svdLS(aSuppChar[k][0], charOrig, charSignal,
            charApprox, condNum);
         model.svdLS(aSuppChar[k][0], toTest, testSignal,
            testApprox, condNum);
         double dotProd = DSLib.dotProduct(
            charApprox, testApprox);
         double scale1 = DSLib.norm2(charApprox);
         double scale2 = DSLib.norm2(testApprox);
// Find best projections on indices aSupp
         double tested = Math.abs(dotProd / (scale1 * scale2));
         if (tested > valueMax)
         {
            indexMax = k;
            valueMax = tested;
```

```
         }
         if (score != null)
         {
            score[k] = new DSDoubleSort(k, tested);
         }
         if (debug)
         {
            System.out.printf("%8s %8.5f\n",
               baseChar[k][0], tested);
         }
      }
      DSLib.vecCopy(V[indexMax][0], forResult);
      return baseChar[indexMax][0];
   }

/**
Classify by taking test CoSaMP dot trained CoSaMP vector
@param toTest m by 1 vector to test, assumed already CoSaMPed!
@param forResult m by 1 character vector closest
@param score scores for all characters
@return character name of best match
*/
   public String classifyCosamp
   (
      double[] toTest, double[] forResult, DSDoubleSort[] score
   )
   {
      model.setPrintMatch(false);
      int nChar = baseChar.length;
      int indexMax = 0;
      double valueMax = 0;
      int m = toTest.length;
      double[] charApprox = new double[m];
      for (int k = 0; k < nChar; k++)
      {
         DSLib.vecCopy(V[k][0], charApprox);
         int s = model.getS();
         double scale1 = DSLib.norm2(charApprox);
         double scale2 = DSLib.norm2(toTest);
         double dotProd = DSLib.dotProduct(toTest, charApprox);
         double tested = Math.abs(dotProd / (scale1 * scale2));
         if (tested > valueMax)
```

```
         {
            indexMax = k;
            valueMax = tested;
         }
         if (score != null)
         {
            score[k] = new DSDoubleSort(k, tested);
         }
         if (debug)
         {
            System.out.printf("%8s %8.5f\n",
               baseChar[k][0], tested);
         }
      }
      DSLib.vecCopy(V[indexMax][0], forResult);
      return baseChar[indexMax][0];
   }

/**
Sets whether to have debugging output
@param debug true for output
*/
   public void setDebug(boolean debug)
   {
      this.debug = debug;
   }

/**
Gets number of characters
@return number of characters
*/
   public int getNChar()
   {
      return baseChar.length;
   }

/**
Gets character name given index
@param index index into character array
@return character name
*/
   public String getChar(int index)
```

```
   {
      return baseChar[index][0];
   }

/**
Gets character LaTeX string
@param index index into character array
@return LaTeX string
*/
   public String getCharLatex(int index)
   {
      return baseChar[index][1];
   }

/**
Prints out how well for index k character its score was
@param k index
@param score score from one of the classification algorithms
@return place
*/
   public int printCompare(int k, DSDoubleSort[] score)
   {
      Arrays.sort(score);
      int place = 1;
      boolean notDone = true;
      int toSearch = score.length - 1;
      while (notDone)
      {
         if (score[toSearch].getIndex() == k)
         {
            notDone = false;
         }
         else
         {
            place++;
            toSearch--;
         }
      }
      System.out.printf("%12s & %12s & %5.3f & ", getChar(k),
         getChar(score[score.length - 1].getIndex()),
         score[score.length - 1].getValue());
      if (place == 1)
```

```
      {
         System.out.printf("   2 & %12s & % 5.3f",
            getChar(score[score.length - 2].getIndex()),
            score[score.length - 2].getValue());
      }
      else
      {
         System.out.printf("%4d  & %12s & %5.3f",
            place,
            getChar(score[score.length - place].getIndex()),
            score[score.length - place].getValue());
      }
      System.out.printf("\\\\\n");
      return place;
   }

/**
Run comparisons for blurred images of all characters
*/

// Public constants
   public static final String ALPHABETICAL = "alphabetical";
   public static final String NUMERIC = "numeric";
   public static final String GREEK = "greek";
   public static final String SYMBOL = "symbol";
   public static final String[][] baseChar =
   {
      {"A", ALPHABETICAL},
      {"B", ALPHABETICAL},
      {"C", ALPHABETICAL},
      {"D", ALPHABETICAL},
      {"E", ALPHABETICAL},
      {"F", ALPHABETICAL},
      {"G", ALPHABETICAL},
      {"H", ALPHABETICAL},
      {"I", ALPHABETICAL},
      {"J", ALPHABETICAL},
      {"K", ALPHABETICAL},
      {"L", ALPHABETICAL},
      {"M", ALPHABETICAL},
      {"N", ALPHABETICAL},
      {"O", ALPHABETICAL},
```

```
{"P", ALPHABETICAL},
{"Q", ALPHABETICAL},
{"R", ALPHABETICAL},
{"S", ALPHABETICAL},
{"T", ALPHABETICAL},
{"U", ALPHABETICAL},
{"V", ALPHABETICAL},
{"W", ALPHABETICAL},
{"X", ALPHABETICAL},
{"Y", ALPHABETICAL},
{"Z", ALPHABETICAL},
{"a", ALPHABETICAL},
{"b", ALPHABETICAL},
{"c", ALPHABETICAL},
{"d", ALPHABETICAL},
{"e", ALPHABETICAL},
{"f", ALPHABETICAL},
{"g", ALPHABETICAL},
{"h", ALPHABETICAL},
{"i", ALPHABETICAL},
{"j", ALPHABETICAL},
{"k", ALPHABETICAL},
{"l", ALPHABETICAL},
{"m", ALPHABETICAL},
{"n", ALPHABETICAL},
{"o", ALPHABETICAL},
{"p", ALPHABETICAL},
{"q", ALPHABETICAL},
{"r", ALPHABETICAL},
{"s", ALPHABETICAL},
{"t", ALPHABETICAL},
{"u", ALPHABETICAL},
{"v", ALPHABETICAL},
{"w", ALPHABETICAL},
{"x", ALPHABETICAL},
{"y", ALPHABETICAL},
{"z", ALPHABETICAL},
{"0", NUMERIC},
{"1", NUMERIC},
{"2", NUMERIC},
{"3", NUMERIC},
{"4", NUMERIC},
```

```
{"5", NUMERIC},
{"6", NUMERIC},
{"7", NUMERIC},
{"8", NUMERIC},
{"9", NUMERIC},
{"alpha", GREEK},
{"beta", GREEK},
{"gamma", GREEK},
{"delta", GREEK},
{"epsilon", GREEK},
{"zeta", GREEK},
{"eta", GREEK},
{"theta", GREEK},
{"iota", GREEK},
{"kappa", GREEK},
{"lambda", GREEK},
{"mu", GREEK},
{"nu", GREEK},
{"xi", GREEK},
{"pi", GREEK},
{"rho", GREEK},
{"sigma", GREEK},
{"tau", GREEK},
{"upsilon", GREEK},
{"phi", GREEK},
{"chi", GREEK},
{"psi", GREEK},
{"omega", GREEK},
{"Gamma", GREEK},
{"Delta", GREEK},
{"Theta", GREEK},
{"Lambda", GREEK},
{"Xi", GREEK},
{"Pi", GREEK},
{"Sigma", GREEK},
{"Upsilon", GREEK},
{"Phi", GREEK},
{"Psi", GREEK},
{"Omega", GREEK},
{"leftarrow", GREEK},
{"mapsto", GREEK},
{"rightarrow", GREEK}
```

```
   };

   public static final long serialVersionUID = 4095369392171259835L;

/* Private state for characters */
   private double[][][] origChar;
   private int[][][] aSuppChar = new int[baseChar.length][1][];
   private double[][][] V;

// Private state, trained
   private DSCosamp model;

   private boolean debug = false;
}
```

A.5 DSGui.java

Note: the code for the graphical user interface is adapted from Horstmann and Cornell's *Core Java* books as noted in the comments [7].

```
/**

   COPYRIGHT (C) 2010 David Shao. All Rights Reserved.

   Library functions.

   Masters project

   Note: Much of this code is adapted from
   Cay S. Horstmann and Gary Cornell
   Core Java, Volumes I and II, 8th Edition
   Prentice Hall, Upper Saddle River, NJ: 2008.
   We refer to this book later as Horstmann and Cornell.

   @author David Shao

   @version 2.02 2010/04/28

*/

import java.io.*;
import java.util.*;
```

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.filechooser.*;
import javax.imageio.*;
import javax.imageio.stream.*;

public class DSLib
{

/**
Converts a double to the nearest integer from 0 to 255
for 8-bit grayscale
@param intensity number to be converted
@return nearest integer from 0 to 255
*/
   public static int toGray(double intensity)
   {
      int gray = (int)Math.round(intensity);
      if (gray < 0)
      {
         gray = 0;
      }
      else if (gray > 255)
      {
         gray = 255;
      }
      return gray;
   }

/**
Converts an xBound by yBound array of doubles to
a grayscale image
@param xBound width
@param yBound height
@param pixels grayscale values as pixels
*/
   public static void doubleToGrayImage
   (
```

```
      double[] values,
      BufferedImage aBufferedImage
   )
   {
      int width = aBufferedImage.getWidth();
      int height = aBufferedImage.getHeight();
      WritableRaster aWritableRaster = aBufferedImage.getRaster();
// getPixels() for TYPE_BYTE_INT seemed to indicate int array length 1
      int[] toLoad = new int[1];
      for (int i = 0; i < height; i++)
      {
         for (int j = 0; j < width; j++)
         {
            toLoad[0] = toGray(values[i * width + j]);
            aWritableRaster.setPixel(i, j, toLoad);
         }
      }
   }

/**
Converts a grayscale image to an array of doubles
@param aBufferedImage image
@param values grayscale values
*/
   public static void grayImageToDouble
   (
      BufferedImage aBufferedImage, double[] values
   )
   {
      int width = aBufferedImage.getWidth();
      int height = aBufferedImage.getHeight();
      WritableRaster aWritableRaster = aBufferedImage.getRaster();
      int[] toLoad;
      for (int i = 0; i < height; i++)
      {
         for (int j = 0; j < width; j++)
         {
            toLoad =
               aWritableRaster.getPixel(i, j, (int[])null);
            values[i * width + j] = (double)toLoad[0];
         }
      }
```

```
   }

/**
Takes a subset of sampling matrix and
creates a black 255, white 0 vector
@param threshold for conversion
@param Tlength length of indices subset
@param T vector of indices subset
@param m number of measurements
@param Phi sample matrix
@param vecUnion m by 1 vector union
*/
   public static void unionSet
   (
      double threshold,
      int Tlength, int[] T, int m, double[][] Phi, double[] scale,
      double[] vecUnion
   )
   {
      for (int i = 0; i < m; i++)
      {
         double sum = 0;
         for (int j = 0; j < Tlength; j++)
         {
            sum += (Phi[i][T[j]] * scale[T[j]]);
         }
         if (sum >= threshold)
         {
            vecUnion[i] = 255;
         }
         else
         {
            vecUnion[i] = 0;
         }
      }
   }

/**
Creates an image resized by an integer factor
@param factor factor
@param aBufferedImage original image
@return new resized image
```

```
*/
   public static BufferedImage resizeImage
   (
      int factor, BufferedImage aBufferedImage
   )
   {
      int width = aBufferedImage.getWidth();
      int height = aBufferedImage.getHeight();
      int resizedWidth = factor * width;
      int resizedHeight = factor * height;
      WritableRaster aWritableRaster = aBufferedImage.getRaster();
      BufferedImage resized = new BufferedImage(
         resizedWidth, resizedHeight, BufferedImage.TYPE_BYTE_GRAY);
      WritableRaster bWritableRaster = resized.getRaster();
      for (int i = 0; i < resizedHeight; i++)
      {
         for (int j = 0; j < resizedWidth; j++)
         {
            int[] toLoad = aWritableRaster.getPixel(
               i / factor, j / factor, (int[])null);
            bWritableRaster.setPixel(i, j, toLoad);
         }
      }
      return resized;
   }

// findLargest and merge operations on indices

/**
Finds the toGet largest values of vec, and
records the corresponding indices.
@param vec vector
@param toGet how many of the largest values to find
@param indices indices of largest values in ascending order,
   assumed to be preallocated of length toGet
*/
   public static void findLargest
   (
      double[] vec, int toGet, int[] indices
   )
   {
```

```
// DSDoubleSort carries indices with values when sorted
      int n = vec.length;
      DSDoubleSort[] vecSorted = new DSDoubleSort[n];
      for (int i = 0; i < n; i++)
      {
         vecSorted[i] = new DSDoubleSort(i, vec[i]);
      }

// Takes time n log n, replaces vecSorted
      java.util.Arrays.sort(vecSorted);

// toGet largest value of sorted
//    (n - 1) - (n - toGet) + 1 == toGet
      for (int i = 0; i < toGet; i++)
      {
         indices[i] = vecSorted[i + (n - toGet)].getIndex();
      }

// Sort indices again so that they are in order
      java.util.Arrays.sort(indices);
   }

/**
Merge two lists of ints already in ascending order
@param vecone first list
@param vectwo second list
@param Tint already allocated result
@param twoloc already allocated, indices of vectwo in Tint
*/
   public static int merge
   (
      int[] vecone, int[] vectwo, int[] Tint, int[] twoloc
   )
   {
      int headone = 0;
      int headtwo = 0;
      int leftone = vecone.length - headone;
      int lefttwo = vectwo.length - headtwo;
      int filled = 0;
      while ((leftone > 0) || (lefttwo > 0))
      {
         if (leftone == 0) // only from list two
```

```
{
   Tint[filled] = vectwo[headtwo];
   twoloc[headtwo] = filled;
   filled++;
   headtwo++;
   lefttwo--;
}
else if (lefttwo == 0) // only from list one
{
   Tint[filled] = vecone[headone];
   filled++;
   headone++;
   leftone--;
}
else // both lists, compare values at heads
{
   if (vecone[headone] > vectwo[headtwo])
   {
      Tint[filled] = vectwo[headtwo];
      twoloc[headtwo] = filled;
      filled++;
      headtwo++;
      lefttwo--;
   }
   else if (vecone[headone] < vectwo[headtwo])
   {
      Tint[filled] = vecone[headone];
      filled++;
      headone++;
      leftone--;
   }
   else // special case values equal both removed
   {
      Tint[filled] = vectwo[headtwo];
      twoloc[headtwo] = filled;
      filled++;
      headone++;
      leftone--;
      headtwo++;
      lefttwo--;
   }
}
```

```
      }
      return filled;
   }

/**
Merge two sorted vectors of indices in ascending order
@param vecone first vector
@param vectwo second vector
@param twoloc preallocated where vectwo is in merged
@return sorted vector of unique indices in ascending order
*/
   public static int[] mergeSupport
   (
      int[] vecone, int[] vectwo, int[] twoloc
   )
   {
      int combined = vecone.length + vectwo.length;
      int[] Tint = new int[combined];

      int filled = DSLib.merge(vecone, vectwo, Tint, twoloc);
      if (filled == combined)
      {
         return Tint;
      }
      int[] T = new int[filled];
      for (int i = 0; i < filled; i++)
      {
         T[i] = Tint[i];
      }
      return T;
   }

// Vector operations

/**
Find the absolute value of each entry of a vector
@param vec original vector
@param absVec result
*/
   public static void vecAbs(double[] vec, double[] absVec)
   {
      int n = vec.length;
```

```
      for (int i = 0; i < n; i++)
      {
         absVec[i] = Math.abs(vec[i]);
      }
   }


/**
Initialize a vector to some value
@param vec vector
@param value value
*/
   public static void vecInit(double[] vec, double value)
   {
      int n = vec.length;
      for (int i = 0; i < n; i++)
      {
         vec[i] = value;
      }
   }

/**
Finds the 2-norm of a vector
@param vec vector
@returns 2-norm
*/
   public static double norm2(double[] vec)
   {
      double norm2 = 0;
      int nVec = vec.length;
      for (int i = 0; i < nVec; i++)
      {
         double diff = vec[i];
         norm2 += (diff * diff);
      }
      return Math.sqrt(norm2);
   }

/**
Finds the 2-norm of the difference between two vectors
@param a first vector
@param b second vector
```

```
@returns 2-norm
*/
   public static double norm2(double[] a, double[] b)
   {
      double norm2 = 0;
      for (int i = 0; i < a.length; i++)
      {
         double diff = a[i] - b[i];
         norm2 += (diff * diff);
      }
      return Math.sqrt(norm2);
   }

/**
Finds the scalar product between two vectors
@param a first vector
@param b second vector
@return scalar product
*/
   public static double dotProduct(double[] a, double[] b)
   {
      double dot = 0;
      for (int i = 0; i < a.length; i++)
      {
         dot += (a[i] * b[i]);
      }
      return dot;
   }

   public static void vecDiff(double[] a, double[] b, double[] c)
   {
      int n = a.length;
      for (int i = 0; i < n; i++)
      {
         c[i] = a[i] - b[i];
      }
   }

   public static void vecAdd(double[] a, double[] b, double[] c)
   {
      int n = a.length;
      for (int i = 0; i < n; i++)
```

```
   {
      c[i] = a[i] + b[i];
   }
}

public static void vecScale(double a, double[] b, double[] c)
{
   int n = b.length;
   for (int i = 0; i < n; i++)
   {
      c[i] = a * b[i];
   }
}

public static void vecCopy(double[] from, double[] to)
{
   int n = from.length;
   for (int i = 0; i < n; i++)
   {
      to[i] = from[i];
   }
}

public static void matrixCopy(double[][] A, double[][] B)
{
   int m = A.length;
   int n = A[0].length;
   for (int i = 0; i < m; i++)
   {
      for (int j = 0; j < n; j++)
      {
         B[i][j] = A[i][j];
      }
   }
}

public static void multMatrixVec
(
   double[][] A, double[] x, double[] b
)
{
   int m = A.length;
```

```
      int n = A[0].length;
      for (int i = 0; i < m; i++)
      {
         b[i] = 0;
         for (int j = 0; j < n; j++)
         {
            b[i] += (A[i][j] * x[j]);
         }
      }
   }

   public static void multMatrixTVec
   (
      double[][] A, double[] x, double[] b
   )
   {
      int m = A.length;
      int n = A[0].length;
      for (int i = 0; i < n; i++)
      {
         b[i] = 0;
         for (int j = 0; j < m; j++)
         {
            b[i] += (A[j][i] * x[j]);
         }
      }
   }

/**
*/
   public static void pseudoInverse
   (
      double[][] U, double[] sigma, double[][] V, int rank,
      double[] b, double[] x
   )
   {
      int m = U.length;
      int n = V.length;
      double[] xInter = new double[x.length];
// x = V * sigma-1 * UT b
      multMatrixTVec(U, b, xInter);
      for (int i = 0; i < rank; i++)
```

```
      {
         xInter[i] *= (1.0 / sigma[i]);
      }
      multMatrixVec(V, xInter, x);
   }

   public static void submatrix
   (
      int[] T, double[][] Phi, double[][] A
   )
   {
      int m = Phi.length;
      for (int i = 0; i < m; i++)
      {
         for (int j = 0; j < T.length; j++)
         {
            A[i][j] = Phi[i][T[j]];
         }
      }
   }

// Utility code for manipulating images

   public static void writeLatex
   (
      String filename, File parentDir, String letterString
   )
      throws IOException
   {
      PrintWriter toLatex = new PrintWriter(
         new FileWriter(
            new File(parentDir, filename + ".tex"), true));
      toLatex.println("\\documentclass[12pt]{article}");
      toLatex.println("\\thispagestyle{empty}");
      toLatex.println("\\begin{document}");
      toLatex.println(letterString);
      toLatex.println("\\end{document}");
      toLatex.close();
   }

   public static ArrayList<String> texAlphabetical
   (
```

```
      String pt,
      String series, String shape, String family,
      String charLine
   )
   {
      ArrayList<String> texText = new ArrayList<String>();
      texText.add("\\documentclass[" + pt + "]{article}");
      texText.add("\\thispagestyle{empty}");
      texText.add("\\begin{document}");
      texText.add("\\" + series + "{" +
         "\\" + shape + "{" +
         "\\" + family + "{" +
         charLine + "}}}");
      texText.add("\\end{document}");
      return texText;
   }

   public static void writeTextFile
   (
      String fullFilename, ArrayList<String> text
   ) throws IOException
   {
      PrintWriter aWriter = new PrintWriter(
         new FileWriter(fullFilename, true)
      );
      for (String str: text)
      {
         aWriter.println(str);
      }
      aWriter.close();
   }

/**
Create a new process and run a command
@param command command to run
@param directory working directory
*/
   public static int execCommand
   (
      ArrayList<String> command,
      File directory,
      ArrayList<String> output
```

```
   ) throws IOException, InterruptedException
   {
//      for (String str : command)
//      {
//         System.out.print(str + " ");
//      }
//      System.out.println();

      ProcessBuilder pb = new ProcessBuilder(command);
      pb.directory(directory);
      Process p = pb.start();
      Scanner execOutput = new Scanner(p.getInputStream());
      int n = p.waitFor();
      if (output != null)
      {
         while (execOutput.hasNextLine())
         {
            output.add(execOutput.nextLine());
         }
      }
      return n;
   }

/**
LaTeX a file
@param filename file name without extension
*/
   public static void latexFile(String filename, File parentDir)
   {
      ArrayList<String> command = new ArrayList<String>();
      String program = "latex";
      command.add(program);
      command.add(filename + ".tex");
      try
      {
         int n = DSLib.execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
```

```
   }

/**
Create PostScript file from .dvi file
@param filename file name without extension
@param parentDir parent directory of .dvi file
@param dpi resolution
*/
   public static void dvipsFile
   (
      String filename, File parentDir, int dpi
   )
   {
      ArrayList<String> command = new ArrayList<String>();
      String program = "dvips";
      command.add("dvips");
      command.add("-D" + dpi);
      command.add(filename + ".dvi");
      try
      {
         int n = DSLib.execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
   }

/**
Command and args for converting PostScript to another image format
@param filename file name without extension
@param parentDir parent directory of file
@param dpi resolution
@param device device for GhostScript, image format
@param ext image format extension
*/
   public static void psToImage
   (
      String filename, File parentDir,
      int dpi, String device, String ext
   )
```

```
   {
      ArrayList<String> command = new ArrayList<String>();
      String program = "gs";
      command.add(program);
      command.add("-r" + dpi);
      command.add("-DEPSCrop");
      command.add("-DTextAlphaBits=4");
      command.add("-sDEVICE=" + device);
      command.add("-sOutputFile=" + filename + "." + ext);
      command.add("-dBATCH");
      command.add("-dNOPAUSE");
      command.add(filename + ".ps");
      try
      {
         int n = DSLib.execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
   }

/**
Commands and args for converting PostScript to blurred image format
@param filename file name without extension
@param parentDir parent directory
@param ext image format extension
*/
   public static void psToBlurredImage
   (
      String filename, File parentDir,
      String ext
   )
   {
      ArrayList<String> command = new ArrayList<String>();

// First convert from .ps to .pdf, causes blurring
      String program = "convert";
      command.add(program);
      command.add(filename + ".ps");
      command.add(filename + ".pdf");
```

```
      try
      {
         int n = DSLib.execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);

// Now convert from .pdf to image format
         command.clear();
         command.add(program);
         command.add(filename + ".pdf");
         command.add(filename + "." + ext);
         n = DSLib.execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
   }

/**
Command and args for trimming image to character outline
@param filename file name without extension
@param parentDir parent directory of file
@param width width of resulting image
@param height height of resulting image
@param extFrom image format extension of original
@param extTo image format for result
*/
   public static void trimImage
   (
      String filename, File parentDir,
      int width, int height,
      String extFrom, String extTo
   )
   {
      ArrayList<String> list = new ArrayList<String>();
      String program = "convert";
      list.add(program);
      list.add(filename + "." + extFrom);
      list.add("-trim");
      list.add("-depth");
```

```
      list.add("8");
      list.add("-type");
      list.add("Grayscale");
      list.add("-adaptive-resize");
      list.add(width + "x" + height);
      list.add("-gravity");
      list.add("center");
      list.add("-extent");
      list.add(width + "x" + height);
//      list.add("-morphology");
//      list.add("Erode");
//      list.add("Diamond");
      list.add(filename + "." + extTo);
      try
      {
         int n = DSLib.execCommand(list, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
   }

/**
Command and args for thickening lines of image
@param filename file name without extension
@param parentDir parent directory of file
@param width width of resulting image
@param height height of resulting image
@param extFrom image format extension of original
@param extTo image format for result
*/
   public static void thickenImage
   (
      String filename, File parentDir,
      int width, int height,
      String extFrom, String extTo
   )
   {
      ArrayList<String> list = new ArrayList<String>();
      String program = "convert";
```

```
      list.add(program);
      list.add(filename + "." + extFrom);
      list.add("-morphology");
      list.add("Erode");
      list.add("Diamond");
      list.add(filename + "." + extTo);
      try
      {
         int n = execCommand(list, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
   }

/**
Loads an image file given its filename and extension
@param filename file name
@param ext extension
@return buffered image
*/
   public static BufferedImage readImageFile
   (
      File parentDir, String filename, String ext
   ) throws IOException
   {
      File imageFile = new File(
         parentDir.getPath() + File.separator + filename + "." + ext);
//      System.out.printf("Reading %s\n", imageFile.getName());
      BufferedImage aBufferedImage = ImageIO.read(imageFile);
      return aBufferedImage;
   }

/**
Remove file if it exists
@param filename file name with extension
@param parentDir parent directory
*/
   public static void rmFile
   (
```

```
      String filename, File parentDir
   )
   {
      ArrayList<String> command = new ArrayList<String>();
      String program = "rm";
      command.add(program);
      command.add(filename);
      try
      {
         int n = execCommand(command, parentDir, null);
//         System.out.printf("Return value %d for %s\n", n, program);
      }
      catch (Exception ioe)
      {
         ioe.printStackTrace();
      }
   }

/**
Creates gray image from character string in LaTeX.
@param letterString character string to produce character in LaTeX
@param dpi dpi
*/
   public static BufferedImage charToImage
   (
      String letterString, int dpi, int xBound, int yBound
   ) throws IOException, InterruptedException
   {
      String filename = TLATEX;
      File parentDir = new File(".");
      String device = "pnggray";
      String extFrom = "png";
      String extTo = "png";
      DSLib.rmFile(filename + ".tex", parentDir);
      DSLib.writeLatex(filename, parentDir, letterString);
      DSLib.latexFile(filename, parentDir);
      DSLib.dvipsFile(filename, parentDir, dpi);
      DSLib.psToImage(filename, parentDir, dpi, device, extFrom);
      DSLib.trimImage(filename, parentDir, xBound, yBound, extFrom, extTo);
      BufferedImage image = DSLib.readImageFile(
         parentDir, filename, extTo);
      return image;
```

```
   }

/**
Creates thickened gray image from character string in LaTeX.
@param letterString character string to produce character in LaTeX
@param dpi dpi
*/
   public static BufferedImage charToThickenedImage
   (
      String letterString, int dpi, int xBound, int yBound
   ) throws IOException, InterruptedException
   {
      String filename = TLATEX;
      File parentDir = new File(".");
      String device = "pnggray";
      String extFrom = "png";
      String extTo = "png";
      DSLib.rmFile(filename + ".tex", parentDir);
      DSLib.writeLatex(filename, parentDir, letterString);
      DSLib.latexFile(filename, parentDir);
      DSLib.dvipsFile(filename, parentDir, dpi);
      DSLib.psToImage(filename, parentDir, dpi, device, extFrom);
      DSLib.trimImage(filename, parentDir, xBound, yBound,
         extFrom, extTo);
      DSLib.thickenImage(filename, parentDir, xBound, yBound,
         extTo, extTo);
      BufferedImage image = DSLib.readImageFile(
         parentDir, filename, extTo);
      return image;
   }

/**
Creates blurred gray image from character string in LaTeX.
@param letterString character string to produce character in LaTeX
@param dpi dpi
@param xBound width
@param yBound height
*/
   public static BufferedImage charToBlurredImage
   (
      String letterString, int dpi, int xBound, int yBound
   ) throws IOException, InterruptedException
```

```
   {
      String filename = TLATEX;
      File parentDir = new File(".");
      String device = "pnggray";
      String extFrom = "png";
      String extTo = "png";
      DSLib.rmFile(filename + ".tex", parentDir);
      DSLib.writeLatex(filename, parentDir, letterString);
      DSLib.latexFile(filename, parentDir);
      DSLib.dvipsFile(filename, parentDir, dpi);
      DSLib.psToBlurredImage(filename, parentDir, extFrom);
      DSLib.trimImage(filename, parentDir, xBound, yBound, extFrom, extTo);
      BufferedImage image = DSLib.readImageFile(
         parentDir, filename, extTo);
      return image;
   }

// Generating random shapes in a rectangle

/**
Makes a random quadratic Bezier curve restricted to half the area
@param xBound x pixels numbered from 0 to xBound - 1
@param yBound y pixels numbered from 0 to yBound - 1
@param fracDistance maximum fraction of distance allowed
@param rand random number generator
@return random QuadCurve2D in bounds
*/
   public static QuadCurve2D randQuad
   (
      int xBound, int yBound, double fracDistance, Random rand
   )
   {
      double x1 = Math.floor(xBound * rand.nextDouble());
      double y1 = Math.floor(yBound * rand.nextDouble());
      double ctrlx = Math.floor(xBound * rand.nextDouble());
      double ctrly = Math.floor(yBound * rand.nextDouble());
      double x2 = Math.floor(yBound * rand.nextDouble());
      double y2 = Math.floor(yBound * rand.nextDouble());
      double hX = (double)xBound * fracDistance;
      double hY = (double)yBound * fracDistance;
      while
      (
```

```
         (Math.abs(x1 - x2) > hX) || (Math.abs(y1 - y2) > hY) ||
         (Math.abs(x1 - ctrlx) > hX) || (Math.abs(y1 - ctrly) > hY) ||
         (Math.abs(ctrlx - x2) > hX) || (Math.abs(ctrly - y2) > hY)
      )
      {
         x1 = Math.floor(xBound * rand.nextDouble());
         y1 = Math.floor(yBound * rand.nextDouble());
         ctrlx = Math.floor(xBound * rand.nextDouble());
         ctrly = Math.floor(yBound * rand.nextDouble());
         x2 = Math.floor(yBound * rand.nextDouble());
         y2 = Math.floor(yBound * rand.nextDouble());
      }
      return new QuadCurve2D.Double(x1, y1, ctrlx, ctrly, x2, y2);
   }

/**
Makes a random cubic Bezier curve with two control points
@param xBound x pixels numbered from 0 to xBound - 1
@param yBound y pixels numbered from 0 to yBound - 1
@return random QuadCurve2D in bounds
*/
   public static CubicCurve2D randCubic
   (
      int xBound, int yBound, double fracDistance, Random rand
   )
   {
      double x1 = Math.floor(xBound * rand.nextDouble());
      double y1 = Math.floor(yBound * rand.nextDouble());
      double ctrlx1 = Math.floor(xBound * rand.nextDouble());
      double ctrly1 = Math.floor(yBound * rand.nextDouble());
      double ctrlx2 = Math.floor(xBound * rand.nextDouble());
      double ctrly2 = Math.floor(yBound * rand.nextDouble());
      double x2 = Math.floor(yBound * rand.nextDouble());
      double y2 = Math.floor(yBound * rand.nextDouble());
      double hX = (double)xBound * fracDistance;
      double hY = (double)yBound * fracDistance;
      while
      (
         (Math.abs(x1 - x2) > hX) || (Math.abs(y1 - y2) > hY) ||
         (Math.abs(x1 - ctrlx1) > hX) || (Math.abs(y1 - ctrly1) > hY) ||
         (Math.abs(x1 - ctrlx2) > hX) || (Math.abs(y1 - ctrly2) > hY) ||
         (Math.abs(ctrlx1 - x2) > hX) || (Math.abs(ctrly1 - y2) > hY) ||
```

```
         (Math.abs(ctrlx2 - x2) > hX) || (Math.abs(ctrly2 - y2) > hY)
      )
      {
         x1 = Math.floor(xBound * rand.nextDouble());
         y1 = Math.floor(yBound * rand.nextDouble());
         ctrlx1 = Math.floor(xBound * rand.nextDouble());
         ctrly1 = Math.floor(yBound * rand.nextDouble());
         ctrlx2 = Math.floor(xBound * rand.nextDouble());
         ctrly2 = Math.floor(yBound * rand.nextDouble());
         x2 = Math.floor(yBound * rand.nextDouble());
         y2 = Math.floor(yBound * rand.nextDouble());
      }
      return new CubicCurve2D.Double(
         x1, y1, ctrlx1, ctrly1, ctrlx2, ctrly2, x2, y2);
   }

/**
Makes one random cubic Bezier stroke
@param xBound x bound of image rectangle
@param yBound y bound of image rectangle
@param strokeWidth stroke width
@param fracDistance maximum distance control pt from end pt
@param rand random number generator
@return pixels enumerated top down and left to right from 0 to 255
*/
   public static double[] randomCubicStroke
   (
      int xBound, int yBound, float strokeWidth,
      double fracDistance, Random rand
   )
   {
      BufferedImage image = new BufferedImage(xBound, yBound,
         BufferedImage.TYPE_BYTE_GRAY);
      Graphics2D graphics2D = image.createGraphics();
      graphics2D.setStroke(new BasicStroke(strokeWidth));
      graphics2D.draw(
         DSLib.randCubic(xBound, yBound, fracDistance, rand));
      double[] pixels = new double[xBound * yBound];
      DSLib.grayImageToDouble(image, pixels);
      return pixels;
   }
```

```
/**
Makes random samples
@param xBound width
@param yBound height
@param strokeWidth stroke width
@param N number of samples to make
@param fracDistance fractional distance allowed
@param rand random number generator
@return Phi, sample matrix
*/
   public static double[][] makeRandomCubicSample
   (
      int width, int height, float strokeWidth, int N,
      double fracDistance, Random rand
   )
   {
      int m = width * height;
      double[][] Phi = new double[m][N];
      for (int j = 0; j < N; j++)
      {
         double[] pixels = DSLib.randomCubicStroke(
            width, height, strokeWidth, fracDistance, rand);
//         DSLib.flipColor(pixels, pixels);
         for (int i = 0; i < m; i++)
         {
            Phi[i][j] = pixels[i];
         }
      }
      return Phi;
   }

/**
For integer valued colors from 0 to 255, flip
@param orig original source
@param toFlip new picture with colors flipped
*/
   public static void flipColor(double[] orig, double[] toFlip)
   {
      for (int i = 0; i < orig.length; i++)
      {
         toFlip[i] = 255 - orig[i];
      }
```

```
   }

// Public constants
/* Filename without extension for temporary latex file */
   public static final String TLATEX = "templ";

} /* Class DSLib */
```

A.6 DSTest.java

The following code is what is used for the main test cases. Note: We do not use the `main()` code for our tests.

```
import java.io.*;
import java.util.*;

import java.awt.image.*;

/**
Class for tests for project
*/
public class DSTest
{

/**
Tests projecting both onto s-space subspace

*/
   public void makeRandom
   (
      int xBound, int yBound, float strokeWidth,
      int N, double fracDistance, Random rand, int maxIter
   )
   {
      model.setMaxIter(maxIter);
      model.setPhi(
         DSLib.makeRandomCubicSample(
            xBound, yBound, strokeWidth, N, fracDistance, rand
      ));
   }

/**
Sets the random number generator
```

```
@param rand random number generator
*/
   public void setRandom(Random rand)
   {
      this.rand = rand;
   }

/**
Trains using characters as given, maybe thin
@param dpi for gs
@param xBound width
@param yBound height
@param s sparsity
*/
   public void trainSimple
   (
      int dpi, int xBound, int yBound, int s
   ) throws IOException, InterruptedException
   {
      classifier.setCosamp(model);
      classifier.trainSimple(dpi, xBound, yBound, s);
   }

/**
Trains using thickened characters
@param dpi for gs
@param xBound width
@param yBound height
@param s sparsity
*/
   public void trainThicken
   (
      int dpi, int xBound, int yBound, int s
   ) throws IOException, InterruptedException
   {
      classifier.setCosamp(model);
      classifier.trainThicken(dpi, xBound, yBound, s);
   }

/**
Run tests both least squares mapped to cosamp s sparse subspace
@param dpi dpi for gs
```

```
@param xBound width
@param yBound height
*/
   public void testSubspace
   (
      int dpi, int xBound, int yBound
   ) throws IOException, InterruptedException
   {
      int nChar = classifier.getNChar();
      int m = xBound * yBound;
      double[] u = new double[m];
      double[] best = new double[m];
      DSDoubleSort[] score = new DSDoubleSort[nChar];
      int numRight = 0;
      int totalPlace = 0;
      for (int k = 0; k < nChar; k++)
      {
         String letterString = classifier.getLetterString(k);
         BufferedImage blurredImage = DSLib.charToBlurredImage(
            letterString, dpi, xBound, yBound);
         DSLib.grayImageToDouble(blurredImage, u);
         double[] uFlipped = new double[u.length];
         DSLib.flipColor(u, uFlipped);
         String closest = classifier.classifySupport(
            uFlipped, best, score);
         totalPlace += classifier.printCompare(k, score);
         if (closest.equals(classifier.getChar(k)))
         {
            numRight++;
         }
      }
      System.out.println(numRight + " correct out of " + nChar +
         ", percent " + (double)numRight / (double)nChar +
         ", avg place " + (double)totalPlace / (double)nChar);
   }

/**
Run tests both simple original vectors
@param dpi dpi for gs
@param xBound width
@param yBound height
*/
```

```
   public void testOriginal
   (
      int dpi, int xBound, int yBound
   ) throws IOException, InterruptedException
   {
      int nChar = classifier.getNChar();
      int m = xBound * yBound;
      double[] u = new double[m];
      double[] best = new double[m];
      DSDoubleSort[] score = new DSDoubleSort[nChar];
      int numRight = 0;
      int totalPlace = 0;
      for (int k = 0; k < nChar; k++)
      {
         String letterString = classifier.getLetterString(k);
         BufferedImage blurredImage = DSLib.charToBlurredImage(
            letterString, dpi, xBound, yBound);
         DSLib.grayImageToDouble(blurredImage, u);
         double[] uFlipped = new double[u.length];
         DSLib.flipColor(u, uFlipped);
         String closest = classifier.classifyOriginal(
            uFlipped, best, score);
         totalPlace += classifier.printCompare(k, score);
         if (closest.equals(classifier.getChar(k)))
         {
            numRight++;
         }
      }
      System.out.println(numRight + " correct out of " + nChar +
         ", percent " + (double)numRight / (double)nChar +
         ", avg place " + (double)totalPlace / (double)nChar);
   }

/**
Run tests where new is CoSaMPed maybe different support
@param dpi dpi for gs
@param xBound width
@param yBound height
*/
   public void testCosamp
   (
      int dpi, int xBound, int yBound
```

```
   ) throws IOException, InterruptedException
   {
      int nChar = classifier.getNChar();
      int m = xBound * yBound;
      double[] u = new double[m];
      double[] best = new double[m];
      DSDoubleSort[] score = new DSDoubleSort[nChar];
      int numRight = 0;
      int totalPlace = 0;
      double[] toTest = new double[m];
      for (int k = 0; k < nChar; k++)
      {
         String letterString = classifier.getLetterString(k);
         BufferedImage blurredImage = DSLib.charToBlurredImage(
            letterString, dpi, xBound, yBound);
         DSLib.grayImageToDouble(blurredImage, u);
         double[] uFlipped = new double[u.length];
         DSLib.flipColor(u, uFlipped);
// Special case compute cosamp of uFlipped now
         int s = model.getS();
         model.cosamp(uFlipped, s);
         toTest = model.getApprox();
         String closest = classifier.classifyCosamp(
            toTest, best, score);
         totalPlace += classifier.printCompare(k, score);
         if (closest.equals(classifier.getChar(k)))
         {
            numRight++;
         }
      }
      System.out.println(numRight + " correct out of " + nChar +
         ", percent " + (double)numRight / (double)nChar +
         ", avg place " + (double)totalPlace / (double)nChar);
   }

/**
Run tests where new is CoSaMPed maybe different support
@param dpi dpi for gs
@param xBound width
@param yBound height
*/
   public void testSimple
```

```
   (
      int dpi, int xBound, int yBound
   ) throws IOException, InterruptedException
   {
      int nChar = classifier.getNChar();
      int m = xBound * yBound;
      double[] u = new double[m];
      double[] best = new double[m];
      DSDoubleSort[] score = new DSDoubleSort[nChar];
      int numRight = 0;
      int totalPlace = 0;
      for (int k = 0; k < nChar; k++)
      {
         String letterString = classifier.getLetterString(k);
         BufferedImage blurredImage = DSLib.charToBlurredImage(
            letterString, dpi, xBound, yBound);
         DSLib.grayImageToDouble(blurredImage, u);
         double[] uFlipped = new double[u.length];
         DSLib.flipColor(u, uFlipped);
         String closest = classifier.classifySimple(
            uFlipped, best, score);
         totalPlace += classifier.printCompare(k, score);
         if (closest.equals(classifier.getChar(k)))
         {
            numRight++;
         }
      }
      System.out.println(numRight + " correct out of " + nChar +
         ", percent " + (double)numRight / (double)nChar +
         ", avg place " + (double)totalPlace / (double)nChar);
   }

/**
Run tests
*/
   public static void main(String[] args)
      throws IOException, InterruptedException
   {
      Random rand = new Random(139);
      DSTest runit = new DSTest();
      int xBound = 16;
      int yBound = 16;
```

```
      float strokeWidth = 2.5f;
      int N = 2000;
      double fracDistance = 1;
      int s = 20;
      int maxIter = 20;
      int dpi = 300;
      long before = System.currentTimeMillis();
      runit.makeRandom(xBound, yBound, strokeWidth, N, fracDistance,
         rand, maxIter);
      long after = System.currentTimeMillis();
      System.out.println((after - before) + " millis to obtain " +
         N + " random curves");
      before = System.currentTimeMillis();
      runit.trainThicken(dpi, xBound, yBound, s);
      after = System.currentTimeMillis();
      System.out.println((after - before) + " millis to train " +
         runit.classifier.getNChar() + " characters");
      before = System.currentTimeMillis();
      runit.testSubspace(dpi, xBound, yBound);
      after = System.currentTimeMillis();
      System.out.println((after - before) + " millis to test " +
         runit.classifier.getNChar() + " characters");
   }

// Private state

   DSCosamp model = new DSCosamp();
   DSCharacter classifier = new DSCharacter();
   Random rand = new Random(139);
}
```

A.7 DSRunThin.java

The following program runs the main test cases we use for comparing the performance of CoSaMP on blurred images after having initialized CoSaMP. The test cases are simply run from `main()` so that compiling and executing

```
java DSRunThin
```

will run the tests and print the output onto the terminal. Note that the `CLASSPATH` must contain the Colt `.jar` file.

```
import java.io.*;
```

```
import java.util.*;

import java.awt.image.*;

public class DSRunThin
{
/**
Run tests
*/
   public static void main(String[] args)
      throws IOException, InterruptedException
   {
      Random rand = new Random(237);
      DSTest runit = new DSTest();
      int xBound = 16;
      int yBound = 16;
      float strokeWidth = 2.5f;
      int N = 2000;
      double fracDistance = 1;
      int s = 10;
      int maxIter = 20;
      int dpi = 300;
      long before = System.currentTimeMillis();
      runit.makeRandom(xBound, yBound, strokeWidth, N, fracDistance,
         rand, maxIter);
      long after = System.currentTimeMillis();
      System.out.println((after - before) + " millis to obtain " +
         N + " random curves");
      before = System.currentTimeMillis();
      runit.trainSimple(dpi, xBound, yBound, s);
      after = System.currentTimeMillis();
      System.out.println((after - before) + " millis to train " +
         runit.classifier.getNChar() + " characters");

      System.out.println("\nTest original vs training original\n");
      before = System.currentTimeMillis();
      runit.testOriginal(dpi, xBound, yBound);
      after = System.currentTimeMillis();
      System.out.println((after - before) + " millis to test " +
         runit.classifier.getNChar() + " characters");

      System.out.println("\nTest original vs training CoSaMPed\n");
```

```
      before = System.currentTimeMillis();
      runit.testSimple(dpi, xBound, yBound);
      after = System.currentTimeMillis();
      System.out.println((after - before) + " millis to test " +
         runit.classifier.getNChar() + " characters");

      System.out.println("\nTest LS s vs training LS s\n");
      before = System.currentTimeMillis();
      runit.testSubspace(dpi, xBound, yBound);
      after = System.currentTimeMillis();
      System.out.println((after - before) + " millis to test " +
         runit.classifier.getNChar() + " characters");

      System.out.println("\nTest CoSaMPed vs training CoSaMPed\n");
      before = System.currentTimeMillis();
      runit.testCosamp(dpi, xBound, yBound);
      after = System.currentTimeMillis();
      System.out.println((after - before) + " millis to test " +
         runit.classifier.getNChar() + " characters");
   }

}
```

References

[1] Abbyy. [Online.] http://www.abbyy.com/

[2] A. Ahmed et al., "Structured Correspondence Topic Models for Mining Captioned Figures in Biological Literature," *KDD-2009*, June 28–July 1, 2009, Paris, France. ACM, pp. 39–47.

[3] E. Candes, J. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Comm. Pure Appl. Math*, vol. 59, 2006, pp. 1206–1223.

[4] M. Cheriet, N. Kharma, C.-L. Liu, and C. Y. Suen, *Character Recognition Systems*, Hoboken, NJ: John Wiley, 2007.

[5] Colt Project. [Online.] http://acs.lbl.gov/software/colt/

[6] J. F. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice, 2nd ed*, Reading, MA: Addison Wesley, 1996.

[7] C. S. Horstmann and G. Cornell, "Core Java, Volumes I and II, 8th Ed.", Upper Saddle River, NJ: Prentice Hall, 2008.

[8] ImageMagick Studio LLC, "ImageMagick." [Online]. Available: `http://www.imagemagick.org/script/index.php`

[9] Infty Project. [Online.] Available: `http://www.inftyproject.org/en/index.html`.

[10] D. Knuth, *The METAFONTbook*, Reading, MA: Addison Wesley, 1986.

[11] L. Lamport, *LaTeX: A Document Preparation System, 2nd Ed.*, Boston, MA: Addison Wesley, 1994.

[12] D. Bloomberg, "Leptonica," April 5, 2010, [Software]. Available: `http://www.leptonica.com/source/leptonlib-1.65.tar.gz`.

[13] Medline. http://www.ncbi.nlm.nih.gov/pubmed

[14] S. Mori, H. Nishida, and H. Yamada, *Optical Character Recognition*, New York: John Wiley, 1999.

[15] A. Majumdar and R. K. Ward, "Nearest subspace classifier: application to character recognition," [Online.] http://ubc.academia.edu/documents/0009/0488/NSC.pdf

[16] D. Needell and J. A. Tropp, "CoSaMP: Iterative signal recovery from incomplete and inaccurate samples," *Appl. Comput. Harmon. Anal.*, vol. 26, 2009, pp. 301–321.

[17] Pharmacogenomics Knowledge Base [Online]. http://www.pharmgkb.org/

[18] PubMed Central [Online]. Available: `http://www.pubmedcentral.nih.gov`.

[19] R. Rodriguez-Esteban and I. Iossifov, "Figure Mining for Biomedical Research," *Bioinformatics*, vol. 25, no. 16, 2009, pp. 2082–2084.

[20] D. L. Rubin, C. F. Thorn, T. E. Klein, and R. B. Altman, "A statistical approach to scanning the biomedical literature for pharmacogenetics knowledge," *Journal of the American Medical Informatics Association*, vol. 12, no. 2, 2005, pp. 121–129.

[21] H. Shatkay, N. Chen, and D. Blostein, "Integrating Image Data into Biomedical Text Categorization," *Bioinformatics*, vol. 22, no. 14, 2006, pp. e446–e453.

[22] P.Sojka, R. Panák, and T. Mudrák, "Optical character recognition of mathematical texts in the DML-CZ project," Presentation dated October 14, 2008, [Online]. Available: `http://www.fi.muni.cz/usr/sojka/presentations/abbyy-dml-cz-pres.pdf`

[23] S. Sonnenburg et al., "Large Scale Multiple Kernel Learning," *Journal of Machine Learning Research*, vol. 7, July 2006, pp. 1531–1565.

[24] The Lexical Systems Group of The Lister Hill National Center for Biomedical Communications, *SPECIALIST NLP Tools* http://lexsrv3.nlm.nih.gov/SPECIALIST/

[25] tesseract-ocr: An OCR Engine that was developed at HP Labs between 1985 and 1995 ... and now at Google, "Project home." [Online]. Available: `http://code.google.com/p/tesseract-ocr/`

[26] L. N. Trefethen and D. Bau, III, *Numerical Linear Algebra*, Philadelphia: SIAM, 1997.

[27] T. Yeh, T.-H. Chang, and R.C. Miller, "Sikuli: Using GUI Screenshots for Search and Automation," *UIST 2009*, ACM, 2009, pp. 183–192.

[28] M. Yousef, S. Jung, L. C. Showe, and M. K. Showe "Recursive Cluster Elimination (RCE) for classification and feature selection from gene expression data", *BMC Bioinformatics*, **Vol. 8**: 144, 2007. [Online]. Available: `ftp://ftp.ncbi.nlm.nih.gov/pub/pmc/bd/33/BMC_Bioinformatics-8-_-1877816.tar.gz`.