Online Recognition of Handwritten Mathematical Expressions with Support for Matrices

Chuanjun Li
Brown University
chuanjun@cs.brown.edu

Robert Zeleznik
Brown University
bcz@cs.brown.edu

Timothy Miller
Brown University
tsm@cs.brown.edu

Joseph J. LaViola Jr.
University of Central Florida
jjl@eecs.ucf.edu

Abstract

We present an online system for recognizing handwritten mathematical matrices in the context of an interactive computational tool called MathPaper. Automatic segmentation and recognition of multiple expressions are supported based on a spacing algorithm that leverages recognized symbol identities, sizes, and relative locations of individual symbols. Matrices with ellipses can be recognized and instantiated with non-ellipsis elements. Both well- and non-well-formed matrices can also be recognized. Matrix elements can be any general mathematical expressions including imbedded matrices. Our recognizer also addresses the poor column alignment problem of handwritten matrices, and allows for slight horizontal overlaps between elements in neighboring columns and different rows.

1 Introduction

As pen-based devices, especially Tablet PCs, become more prevalent in various fields [3], pen-based entry of 2D mathematical notations in conjunction with computational support is becoming more and more commonplace. Thus, accurate recognition of handwritten mathematical expressions [4] is critical for usable systems and applications.

Recognition of handwritten mathematical expressions have been widely investigated [1, 2, 13], and various structure analysis techniques based on baseline tree structures [13] or context free grammars for symbol groups have been explored for recognizing the structures of single mathematical expressions. MathPad^2 [4] handles multiple expressions by manual segmentation. Recently, research has been done on recognition of simple handwritten mathematical matrices [7, 8, 9, 11, 12] as well as scanned typeset matrices [10]. Due to variations and incremental updates of handwritten input, approaches that might work for scanned typeset matrices might not work for handwritten matrices, such as identification of matrix elements by projection of strokes onto the two primary axes [7, 8, 9]. Recognition of non-well-formed matrices where not all elements are available has not been addressed in these works or by commercial applications such as Microsoft Math [6]. In the latest work of Toyozumi et. al, insertion of new input between existing rows or columns is a limiting factor. In addition, some symbols such as summation ($\sum$), square root ($\sqrt{}$) and fraction lines etc. are not supported in matrices [11, 12].

To mitigate some of the issues raised above, we have developed a mathematical expression recognizer with support for matrices, that recognizes non-well-formed matrices, matrices with any general mathematical expression as elements, and matrices with horizontally overlapped elements in adjacent columns and different rows. Our recognizer also recognizes ellipses and can replace ellipses with non-ellipsis elements.

2 Mathematical Expression Recognition

Mathematical expressions are recognized in MathPaper by the steps shown in Figure 1. An ink stroke is first tested to see if it is a gesture, such as a scribble for stroke deletion, or a lasso for stroke selection, etc. If it is a gesture, the stroke is handled by the user interface (UI). Otherwise, it is passed to the Symbol Recognizer.

The Symbol Recognizer uses cusp-related features, such as straightness of stroke, average curvature between two cusps, arc length, etc, to recognize a stroke. The rule-based recognizer is adaptable to different writ-

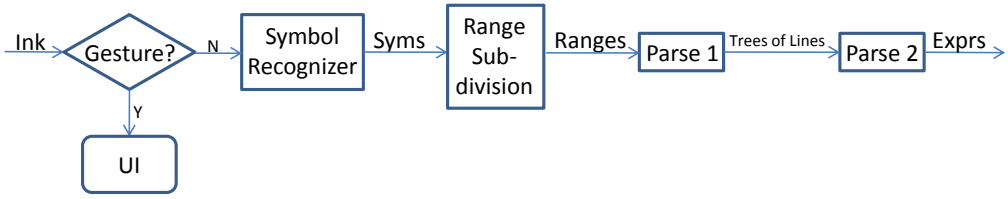

Figure 1. Online recognition of mathematical expressions (From [5])

ing styles by having allographs, or different ways of handwriting one symbol, for the input strokes. A user can choose a different recognition from a list of alternates for a stroke. The Symbol Recognizer also incorporates Microsoft's symbol recognizer for further recognition of strokes which cannot be recognized by the rule-based default recognizer.

Before passed to the structural analyzer called Parse 1, a Symbol object from the symbol recognizer is first examined for a range test. A range represents one grouping of symbols for a single expression. The symbol identity and size are used for inflating the bounding box of the symbol in order to determine which ranges are hit by the stroke, or if a new range is added by the stroke. The Symbol objects in all affected ranges are first sorted in Parse 1 by the left of the stroke bounding boxes for constructing baseline trees, one for each range. Each baseline tree is rooted by a main Line object, with its Symbol objects having lower lever Line objects for superscripts, subscripts, etc. The trees of Line objects are further analyzed by the semantics parser called Parse 2 and one Expr object is generated for each range of symbols. The Expr object is output as a typeset expression below the strokes as shown in Figure 2, and can be converted into other formats for different computing engines. One built-in engine and Mathematica are used for computation in MathPaper.

A matrix can be one component of an expression, and each of its elements can be a general expression. Hence, recognition of matrices is based on the recognition of general mathematical expressions as described above.

3 Matrix Recognition

For input simplicity, parentheses are used to specify matrices in MathPaper. Since parentheses are also used for non-matrix expressions, it is necessary to detect whether strokes between parentheses are matrix elements.

3.1 Matrix detection and element segmentation

Each open parenthesis Symbol object (*parenSym*) stores the Line objects for the matrix elements. Strokes between a pair of matched parentheses strokes, or to the right of an open parenthesis stroke, if the open parenthesis is not closed, are collected for each *parenSym* when the *parenSym*'s range is updated. A *parenSym*'s stroke collection is segmented into ranges, one range for each element, according to the recognized symbols' identities, sizes, and distances to other symbols just like range segmentation for non-matrix expressions as discussed above. Bounding boxes of symbols are inflated based on the symbols' identities and sizes and all intersected symbols fall into one range for an element. The bounding box inflating ratios for matrix element range segmentation are specific to individual symbols and are in general smaller than those for non-matrix expression range segmentation given that strokes for a matrix element tend to be closer than strokes for a non-matrix expression.

If there is only one row as identified in the following sections, it is likely that the one row is for a non-matrix expression, and not for a one-row matrix. Hence, if there is more than one column, a second round parse is applied to the *parenSym*'s strokes, with a larger bounding box inflating ratio based on non-matrix expression range segmentation. If there is still more than one column, the result of the first round parse is retrieved as matrix elements. If the second round parse generates only one column, then the strokes are for a non-matrix expression as shown in Figure 2. Without element 2 as shown on the right in the figure, the strokes are reparsed and only one column is generated rather than two columns as shown on the left.

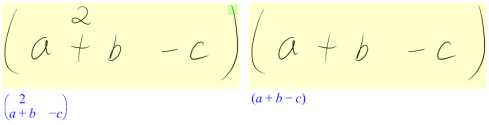

Figure 2. Recognition of matrix and single expression in parentheses.

3.2 Row identification

After range segmentation for elements, symbols in each element range are sorted and parsed by Parse 1 to

obtain one Line object for each element. The bounding box of an element Line $elementBox$ is compared with a bounding box of each row to identify to which row it belongs or if it is for a new row.

Since handwritten superscripts and subscripts in one row might intrude into neighboring rows as shown in Figure 3, we use the bounding box $rowBox$ which contains the bounding boxes of the main baselines for the elements in the row, rather than the bounding boxes of the Line trees of all the Line objects in the row, to compare with $elementBox$. Taking into account handwriting variations, we allow for certain overlaps between $elementBox$ and $rowBoxes$ of neighboring rows. An $elementBox$ falls into one row with $rowBox$ only when their overlap height is at least half of the height of either $elementBox$ or $rowBox$. Hence $c_{p_h}^{m^{b^2}}$ in Figure 3 is in a separate row.

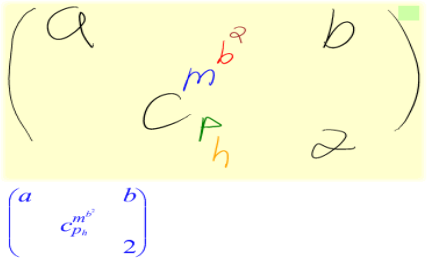

Figure 3. Recognition of non-well-formed matrix.

3.3 Column identification

There are two steps in identifying the column for a new element Line Object. The first step locates the relative position of the Line object in the list of Line objects for the row, and the second step identifies the column number after all element objects have been "inserted" into their rows in the first step. This allows for insertion of new rows or new columns.

Since adjacent elements in a row are separated by space, the first step needs only to check, from the first Line object, the right of a new $elementBox$ R_n and the left of an existing $elementBox$ L_e. If $R_n > L_e$, the new Line object is inserted after the current Line object in the Line object list. Otherwise the next Line object is checked until the end of the list.

The second step first identifies the total number of columns by traversing all rows and generates a bounding box $colBox_c$ for each column c. The list of $colBox_i$ is initialized by the first row, and updated while all the other rows are traversed. If there is a new column c, a new $colBox_c$ is inserted into the list $colBox_i$, allowing for insertion of new elements at any location.

The second step then checks the bounding box of each element b with the column bounding box list $colBox_i$. If b is contained by $colBox_i$, it is in column i. Since the column bounding box list $colBox_i$ contains one bounding box for each column, all elements' columns will be identified in the second step.

3.4 Ellipsis recognition and instantiation

Since dots for ellipsis can be far away from each other, they are recognized by being collected together and globally clustered to form horizontal, vertical or diagonal ellipses. Floating points are removed from the ellipsis dots, and the dots for i, j are not collected during ellipsis detection. If the ellipsis dots are not a multiple of 3 during incremental updates, the remaining one or two dots are recognized as a default horizontal ellipsis.

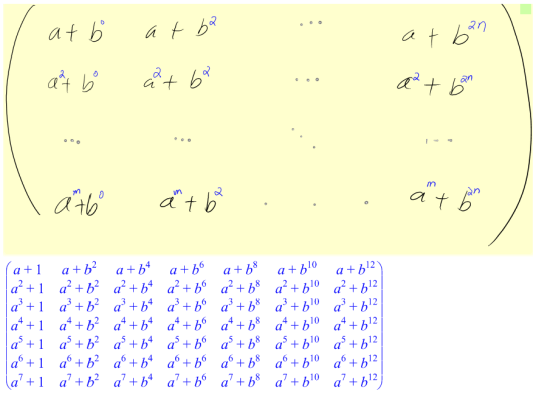

Figure 4. Pattern detection after recognition of ellipsis elements.

Since the bounding boxes of horizontal and vertical ellipses can be too narrow or flat, their rows and columns are identified by using the inflated bounding boxes which consider both lengths and heights of tight bounding boxes.

After ellipsis recognition, matrix elements are examined to detect certain patterns, such as repeated rows, columns, or diagonals, etc. Common patterns such as Toeplitz matrices or Hankel matrices are checked first. If a matrix is not any of these matrices, the element in the last row and the last column, which is normally a non-ellipsis element, is compared with another non-ellipsis element in the last row and another non-ellipsis element in the last column. If a pattern is found, the row and column variables are identified, and all the ellipsis elements can be instantiated with non-ellipsis elements with default matrix dimensions of 7, as shown in Figure 4 for a Vandermonde matrix.

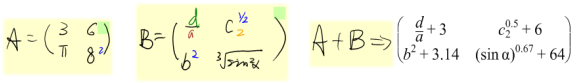

Figure 5. Matrix computation.

Matrices can be recognized individually or as components of an expression, as shown in Figure 5 for matrix computation after recognition. A double arrow ($\Rightarrow$) is used for expression evaluation. Linear algebra operations, such as computation of eigenvectors, singular value decompositions, etc, are also supported.

4 Experiments

We examined our recognition system's accuracy by examining both symbol recognition and matrix recognition. For symbol recognition, we had seven subjects each writing 891 symbols for a total of 6237 symbols in 43 different expressions. Note that each supported symbol (79 in total) was included at least twice in the 43 expressions. The average accuracy across all subjects was 91.6%, with the lowest accuracy 81.2% for one subject and the highest 99.5% for another subject.

Five different subjects performed experiments on recognition of 50 different matrices. Most of the matrices were taken from algebra textbooks and the literature. Each matrix was entered once by one subject. Among the 50 different matrices, 45 were correctly recognized with an overall recognition accuracy of 90%. A matrix is correctly recognized only if all symbols are correctly recognized and all elements are correctly parsed. Some of the recognized and failed matrices are listed in Figure 6 and Figure 7.

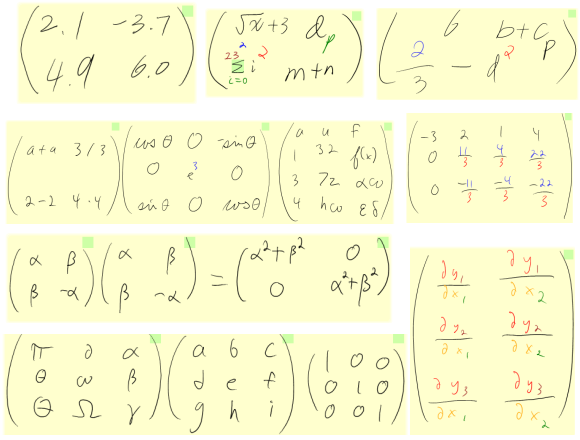

Figure 6. Examples of correctly recognized matrices.

Neighboring rows can overlap vertically, and neighboring columns can overlap horizontally as shown by the last two matrices on the top of Figure 6. Two typical cases where matrices cannot be recognized correctly are listed in Figure 7. The first one has slanted row elements, and the second one has part of one element outside the parenthesis range.

Figure 7. Failed matrix examples.

5 Conclusion

We have presented an approach for recognizing matrices in the context of our mathematical recognition and computation system, MathPaper. Matrices can be components of an expression, and matrix elements can be any general mathematical expression. Both well-formed and non-well-formed matrices can be recognized, and elements in adjacent columns can have horizontal overlaps if they are in different rows. Matrices with ellipses can also be recognized and instantiated with non-ellipsis elements.

References

[1] K.-F. Chan and D.-Y. Yeung. Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recogntion*, 3(1):3–15, 2000.

[2] U. Garain and B. B. Chaudhuri. Recognition of online handwritten mathematical expressions. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 34(6):2366–2376, Dec. 2004.

[3] S. Joy, F. Leslie, and K. Todd. Dataquest insight: Tablet pcs are slowly gaining momentum. 6 April 2007.

[4] J. LaViola and R. Zeleznik. Mathpad^2: A system for the creation and exploration of mathematical sketches. *ACM Transactions on Graphics*, 23(3):432–440, Aug. 2004. (Proc. of SIGGRAPH 2004).

[5] C. Li, T. Miller, R. Zeleznik, and J. LaViola. Algosketch: Algorithm sketching and interactive computation. In *Proc. of the 5th EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (SBIM 2008), pp. 175-182*, 2008.

[6] Microsoft. Microsoft math. Computer program. www.microsoft.com/math.

[7] E. Tapia and R. Rojas. Recognition of on-line handwritten mathematical expressions using a minimum spanning tree contruction and symbol dominance. In J. Llados and Y.-B. Kwon, editors, *Graphics Recognition*, volume 3088 of LNCS. Springer-Verlag, 2004.

[8] E. Tapia and R. Rojas. Recognition of on-line handwritten mathematical expressions in the e-chalk system - an extension. In *Proc. Int. Conf. on Document Analysis and Recognition (ICDAR'05)*, pages 1206–1210, Seoul, Korea, Aug./Sept. 2005.

[9] D. Tausky, G. Labahn, E. Lank, and M. Marzouk. Managing ambiguity in mathematical matrices. In *Proc. of the 4th EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (SBIM 2007)*, Aug. 2007.

[10] K. Toshihiro and S. Masakazu. Detection of matrices and segmentation of matrix elements in scanned images of scientific documents. In *Proc. Int. Conf. on Document Analysis and Recognition (ICDAR'03)*, pages 433–437, Edinburgh, Scotland, Aug. 2003.

[11] K. Toyozumi, T. Suzuki, K. Mori, and Y. Suenaga. A system for real-time recognition of handwritten mathematical formulas. In *Proc. Int. Conf. on Document Analysis and Recognition (ICDAR'01)*, pages 1059–1063, Seattle, WA, Sept. 2001.

[12] K. Toyozumi, T. Suzuki, K. Mori, and Y. Suenaga. An on-line handwritten mathematical equation recognition system that can process matrix expressions by referring to the relative positions of matrix elements. *Systems and Computers in Japan*, 37:87–96, 2006.

[13] R. Zanibbi, D. Blostein, and J. R. Cordy. Recognizing mathematical expressions using tree transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1455–1467, Nov. 2002.