Discovering Clusters using the Fuzzy C-Means Algorithm

Joseph Anthony C. Hermocilla

Institute of Computer Science
University of the Philippines Los Banos
jach@ics.uplb.edu.ph

October 2003

I. Introduction

Analysis of data to generate relevant information is an important activity not only in computing systems but in other fields as well. The analysis can either be confirmatory of exploratory based on the availability of a model. In confirmatory analysis, a postulated model exists wherein the data being analyzed is tested against the model. On the otherhand, exploratory analysis lacks a model and the association among the data is discovered naturally. Both types use some form of similarity measure to determine the degree of association among the data.

Cluster analysis is a form of exploratory analysis which makes use of a variety of mathematical methods to assign similar data into groups called clusters. It has been a subject of interest by researchers in different fields including biology, statistics, economics, manufacturing, aeronautics, image processing, among others, where some form of grouping or classification is desired. The results of a clustering activity can be used to formulate new hypotheses or validate certain assumptions regarding the data

being analyzed. Although cluster analysis is an old discipline on its own, much of the theories behind it have not been applied in practice due to the expensive computations required to discover the clusters, especially when computers are not available.

Data clustering or cluster analysis organizes a collection of patterns into clusters and is often refererred to as unsupervised classification because the result is solely based on the data itself. In order to accomplish, a similarity measure is employed to determine similar objects. In applications like image processing and information retrieval, little prior information is available about the data being analyzed and thus are viable candidates for data clustering.

Several approaches to data clustering have been proposed. There are two basic categories, *partitional* and *heirarchical* clustering. Partitional clustering assigns each data to belong to single particular class or cluster. Heirarchical, on the otherhand, generates a *tree* or *dendrogram* which produces nested levels of partitions. Partitional clustering algorithms are more advantageous when using large datasets because the construction of a dendrogram is expensive. The most popular partional clustering algorithm is *K-means*, which attempts to minimize the squared error in a clustering. Partitional algorithms work by minimizing a particular objective function, a combinatorial problem and therefore is expensive to solve.

II. K-Means Algorithm

Let **X**=(x_1,x_2,...,x_N) be a *feature vector* where the scalars, x_1,x_2,...,x_n represent the individual *features* of of each object. N is the dimensionality of the vector, the number of features. Let **V**=(v_1,v_2,...,v_k) be the set of k clusters where the v_1,v_2,...,v_k represent the *centroid* of each cluster. In a clustering activity, there are usually M objects under study, each described by a feature vector.

The *centroid* of a cluster is defined as:

$$v_k = \frac{\sum_{i=1}^{N} x_i \in v_k}{|v_k|}$$

The above formula simply computes the mean or the average of the values of features belonging to the particular cluster.

The objective function to be minimized is defined below:

$$J(X,V) = min \sum_{k=1}^{K} \sum_{i=1}^{N} d(v_k, x_i)$$

where $d(v_k, x_i)$ is a *distance function.* There are several distance functions used in practice some of them are described below:

Euclidean: $$d(v_k, x_i) = \|x_i - v_k\| = \sqrt{\sum_{j=1}^{N} (x_{ij} - v_{kj})^2}$$

Squared Euclidean: $$d(v_k, x_i) = \|x_i - v_k\|^2 = \sum_{j=1}^{N} (x_{ij} - v_{kj})^2$$

Manhattan Distance: $$d(v_k, x_i) = \sum_{j=1}^{N} |(x_{ij} - v_{kj})|$$

General Minkowsky Metric: $d(v_k, x_i) = \left(\sum_{k=1}^{d} (x_{ij} - v_{kj})^p\right)^{1/p}$

The K-means algorithm has been a popular choice because of its simplicity and ease of implmentation. It is basically a *variance minimization* technique, attempting to minimize the error sum of squares. The objective function can be minimized using calculus.

The algorithm for K-Means can be stated as follows:

1. Choose *k* cluster centers to coincide with *k* randomly chosen objects
2. Assign each object to the closest centers
3. Recompute cluster centers using current membership
4. Goto step 2 until a convergence criterion is met. Usually a minimal reassignment or minimal decrease in squared error

A common problem encountered with variance minimization algorithms is the effect of *outliers* on the function being minimized, which tends to affect the resulting clusters.

III.Fuzzy C-Means

The K-means algorithm described above performs a *hard clustering* on the dataset . Hard clustering assign each object to belong to a single cluster only. However, there are instances when objects can belong to several clusters, that is, each object has a

certain degree of membership in each cluster. Clustering algorithms that performs this task are said to be *fuzzy*, attributed to Zadeh, the proponent of fuzzy set theory. A variant of K-Means that performs fuzzy clustering is called *Fuzzy C-Means (FCM).* The output of a fuzzy clustering is not a partition but still a clustering with each object having a certain degree of membership to a particular cluster.

The objective function being minimized in FCM is given below:

$$J(X,V)=min\sum_{k=1}^{K}\sum_{i=1}^{N}(u_{ki})^m d(v_k,x_i)$$

where $d(v_k,x_i)$ is the *distance function* similar to the one discussed above and u_{ki} is the degree of membership of object i to cluster k subject to the following constraints:

$u_{ki}\in[0,1]$ and $\sum_{k=1}^{K}u_{ki}=1,\forall i$, which means that the sum of the membership values of the objects to all of the fuzzy clusters must be one.

m is the fuzzifier which determines the degree of fuzziness of the resulting clusters.

The objective function can be minimized by using Lagrange multipliers. By taking the first derivatives of J with respect to u_{ki} and v_k and setting them to zero results in two necessary but not sufficient conditions for J to be at the local extrema. The result of the derivation is given below:

$$v_k=\frac{\sum_{i=1}^{N}u_{ki}^m x_i}{\sum_{i=1}^{N}u_{ki}^m}$$ which computes the centroid of cluster k .

$$u_{ki} = \frac{1}{\sum_{j=1}^{K} \left(d(v_k, x_i) / d(v_k, x_j) \right)^{\frac{1}{m-1}}}$$ computes the membership of object i to cluster k. The d function refers to the Squared Euclidean norm above.

The algorithm for FCM can be stated as follows:

1. Construct an initial membership matrix **U** of the N objects and K clusters. This represents the initial fuzzy partition
2. Compute the cluster centroids using v_k
3. Update the membership values using u_{ki}
4. Compute the value of the objective function , J using the values computed from steps 2 and 3
5. Goto step 2 until some convergence criterion is met($|\Delta J| < \varepsilon$)

IV. Discussion

The Fuzzy C-Means algorithm was implemented using the Java programming language in a program called Fuzzy Clusterer (source code included). The program was tested using microarray data from the web. The data included yeast and serum gene expressions. The program created is capable of reading PCL files from Stanford's Microarray Data repository. Although the main objective of the study is to study the gene expressions of rice, the unavailability of the data forced the author to test other data sets only. With regards to performance, the program was slow because of the expensive computation required especially in computing the objective function and the updating of

the membership values which takes O(MxK). On successive runs of the program on the data sets, the author observed that the membership values of the objects to a cluster are almost evenly distributed and there is little deviation.

Below are screenshots of the program running in linux.

Joseph's Terminal - [1-jach@booch:/home/jach/projects/CMSC244/FuzzyPaper]
1-Shell
------>Updating membership values...
------>Computing next...13.332170057303795
------>Delta: 0.006866187767572407
0.24703729077379272 0.7562311498070275 0.002488408821103072 4
0.3341110727725897 0.04089259662025481 0.0010090511458818565
0.9442413633911829 0.012798396508887944 3.4962504586526 18E-4
0.6912579488092704 0.1234482894262151 1.1147838451827823E-4
0.4715982822696325 0.465980402646992 7 0.1012779677438847 1
0.35171250389707287 0.6769581066565312 0.009124986705416784
0.19827427038525064 0.7618624821205119 0.006139010285269861
0.43972288750569666 0.5848495201264151 0.009280269062369 21
0.43054308087955685 0.5608007514059838 0.018535761911962233
0.3371397145061124 0.5704438926347429 0.0013401929315741703
0.7717407084201239 0.9942738544735423 0.006906135536638 8044
0.6789515698038072 0.39789809446535357 0.0245083964152241
0.527944284491574 0.4482638985913658 0.0408428363280680 7
0.5504105576968018 0.4940202113189818 3 0.005331995926089676
0.5037963352091455 0.44554223103846846 0.024028747047561264
0.5455736387049472 0.41468727569829356 0.021726824862683274
0.46500862803064336 0.39536773733740904 0.13826023604572532
0.6033839812248841 0.3847384481550379 0.010567089385620272
0.5768491239599178 0.4388658583974772 0.015480368360684935
0.5519659029036931 0.5586705609590054 0.0017256469011039943
[jach@booch:~/projects/CMSC244/FuzzyPaper]$

Joseph's Terminal - [1-jach@booch:/home/jach/projects/CMSC244/FuzzyPaper]
1-Shell
Loaded rows: 3396
Loaded rows: 3397
Loaded rows: 3398
Loaded rows: 3399
Loaded rows: 3400
Loaded rows: 3401
Loaded rows: 3402
Loaded rows: 3403
Loaded rows: 3404
Loaded rows: 3405
Loaded rows: 3406
Loaded rows: 3407
Loaded rows: 3408
Loaded rows: 3409
Loaded rows: 3410
Loaded rows: 3195
Loaded rows: 3196
Loaded rows: 3197
Loaded rows: 3198
Loaded rows: 3199
Loaded rows: 3200
Loaded rows: 3201
Loaded rows: 3202
Loaded row

References

Kauffman, L. and Rousseeu P. *Finding Groups in Data: An introduction to Cluster Analysis*. John Wiley & Sons, Inc: Canada. 1990.

Kolman, B. *Elementary Linear Algebra*. Prentice-Hall: USA. 1998.

Yan, J. etc. *Using Fuzzy Logic: Towards Intelligent Systems*. Prentice-Hall:UK. 1994.

Source Code:

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

/**
 * Class to read a preclustering file (pcl) and retrieve the
 * information from the file.
 *
 * @author Joseph Anthony C. Hermocilla
 * @version 1.0
 *
 */

public class PCLReader{
        //Abstract path of the pcl file
        File source;

        //A hashtable of the column titles (column #, fieldname)
        Hashtable experiments;

        //A hashtable of the row titles (row #, fieldname)
        Hashtable genes;

        //The data matrix
        DataMatrix data;


        //Maximum allowable genes
        public static final int MAXGENES=10000;


        public PCLReader(String fileName){
                source = new File(fileName);
                BufferedReader in=null;

                try{
                        in = new BufferedReader(new FileReader(source));
                }catch(FileNotFoundException fnfe){
                        System.err.println("File not found: " + fileName);
                        System.exit(1);
                }

                //The first pass extracts stuctural information regarding the
                //data set

                String rawInput1=null,rawInput2=null;
                try{
                        rawInput1 = in.readLine();
                        rawInput2 = in.readLine();
                }catch(IOException ioe){}

                Vector line1 = processLine(rawInput1);
                Vector line2 = processLine(rawInput2);

                int a=-1,b=-1,c=-1,d=-1,e=-1,f=-1;
                for (int i = 0; i < line1.size(); i++){
                        String temp = (String)line1.elementAt(i);
                        temp.toUpperCase().trim();

                        //determine the column numbers
                        if (temp.equals("GID"))
                                a = i;
                        else if (temp.equals("UID"))
                                b = i;
                        else if (temp.equals("NAME"))
                                c = i;
                        else if (temp.equals("YORF"))
                                d = i;
                        else if (temp.equals("GWEIGHT"))
                                e = i;
                        else if (f == -1)
                                f = i;          //column # of data
                }
```

```
                //store experiment descriptions
                experiments = new Hashtable();
                for (int i = f; i < line1.size(); i++){
                        ExperimentInfo info = new ExperimentInfo(line1.elementAt(i).toString(),

        Double.parseDouble(line2.elementAt(i).toString()));
                        experiments.put(new Integer(i-f),info);
                }



                //store gene descriptions
                int rowCount=0;
                int colCount = line1.size() - f;
                genes = new Hashtable();
                data = new DataMatrix(MAXGENES,colCount);


                try{
                while ((rawInput1=in.readLine()) != null){
                        line1 = processLine(rawInput1);

                        GeneInfo info = new GeneInfo(line1.elementAt(a).toString(),
                                                                        line1.elementAt(a).toString(),
                                                                        line1.elementAt(a).toString(),
                                                                        line1.elementAt(a).toString(),
                                                                        /*Double.parseDouble
(line1.elementAt(e).toString())*/ 0);


                        genes.put(new Integer(rowCount),info);


                        //quick hack!!!
                        int temp = colCount;
                        if (temp > line1.size())
                                temp = line1.size();
                        temp=1; //Try temp features first

                        for (int i = f; i < temp+f+1; i++){
                                //System.out.println(line1.elementAt(i).toString());
                                data.setData(rowCount,i-f,Double.parseDouble(line1.elementAt(i).
toString()));
                        }
                        rowCount++;
                        System.out.println("Loaded rows: " + rowCount);
                        data.setColCount(temp);

                }
                }catch(IOException ioe){
                }
                data.setRowCount(genes.size());
        }

        /**
         * Process the input line and return a vector containing the columns
         */
        private Vector processLine(String line){
                StringTokenizer tokenizer = new StringTokenizer(line,"\t");
                Vector colList = new Vector();
                while(tokenizer.hasMoreTokens()){
                        String token = tokenizer.nextToken();
                        colList.addElement(token);
                }
                return colList;
        }


        /** Returns the number of data columns */
        public int getColCount(){
                return data.getColCount();
        }


        /** Returns the number of data rows */
```

```
        public int getRowCount(){
                return data.getRowCount();
        }


        /** Returns the Experiment info*/
        public ExperimentInfo getExperimentInfo(int i){
                return (ExperimentInfo)experiments.get(new Integer(i));
        }


        /** Returns the GeneInfo */
        public GeneInfo getGeneInfo(int i){
                return (GeneInfo)genes.get(new Integer(i));
        }


        /** Returns a clone of the data matrix contained
    *  in the file.
    */
        public DataMatrix getDataMatrix(){
                return data;
        }


        /**
    * Returns a string representation
    */
        public String toString(){
                String temp="";

                temp += "Filename: " + source.getName() + "\n";
                temp += "MicroArray Experiment Count: " + data.getColCount() + "\n";
                temp += "Gene Count: " + data.getRowCount() + "\n";
                return temp;

        }


        /** Main method */
        public static void main(String args[]){
                PCLReader pcl;

                if (args.length < 1){
                        System.err.println("Please specify a filename.");
                        System.exit(0);
                }
                pcl = new PCLReader(args[0]);

                /*
                System.out.println(pcl.getExperimentInfo(2));
                System.out.println(pcl.getGeneInfo(2));
                System.out.println(pcl);
                MathUtil.dumpData(pcl.getDataMatrix().getCol(0));
                */

                //MembershipMatrix mem = MathUtil.randomMembership(pcl.getDataMatrix(),10);
                //MathUtil.dumpData(mem.getRow(0));
                //System.out.println(MathUtil.sum(mem.getRow(0)));

                //double mvf = MathUtil.computeClusterCenter(0,0,mem,pcl.getDataMatrix());
                //double norm = MathUtil.computeEuclideanNorm(0,0,mem,pcl.getDataMatrix());
                //double objective = MathUtil.computeObjective(2,mem,pcl.getDataMatrix());

                //System.out.println("Cluster Center: " + mvf);
                //System.out.println("Norm: " + norm);
                //System.out.println("Objective: " + objective);
                //CentroidMatrix cent = MathUtil.computeClusterCentroids(mem,pcl.getDataMatrix
());
                //MathUtil.dumpData(cent.getCol(0));

                FuzzyClusterer fuzzy = new FuzzyClusterer(pcl,Integer.parseInt(args[1]));
                fuzzy.cluster();


        }
```

```
}


/*
 * Encapsulates Gene Information
 */
class GeneInfo{
        //gene id
        String gid;

        //user id
        String uid;

        //gene name
        String name;

        //yorf
        String yorf;

        //gene weight
        double gweight;


        public GeneInfo(String gid,String uid, String name,String yorf,double gweight){
                this.gid = gid;
                this.uid = uid;
                this.name = name;
                this.gweight = gweight;
                this.yorf = yorf;
        }

        public String getGID(){return gid;}
        public String getUID(){return uid;}
        public String getName(){return name;}
        public double getGweight(){return gweight;}

        public String toString(){
                String s="";

                s += "Group ID: " + gid + "\n";
                s += "User ID : " + uid + "\n";
                s += "Name    : " + name + "\n";
                s += "YORF    : " + yorf + "\n";
                s += "GWEIGHT : " + gweight + "\n";
                return s;
        }
}


/**
 * Encapsulates the Experiment Info
 */

class ExperimentInfo{
        //description of the experiment
        String desc;

        //the weight
        double weight;

        public ExperimentInfo(String desc,double weight){
                this.desc = desc;
                this.weight = weight;
        }

        public String getDesc(){return desc;}
        public double getWeight(){return weight;}

        public String toString(){
                String s="";
                s += "Description: " + desc + "\n";
                s += "Weight      : " + weight + "\n";
                return s;
        }
}
```

```
/**
 * A generalized data matrix of doubles
 */
class DataMatrix{
        //The two dimensional array representing the
        //data sets, data[gene][exp]
        protected double data[][];
        protected int colCount;
        protected int rowCount;

        public DataMatrix(){
                rowCount = 10;
                colCount = 10;
                data = new double[rowCount][colCount];
        }

        public DataMatrix(int rowCount, int colCount){
                this.rowCount = rowCount;
                this.colCount = colCount;

                data = new double[rowCount][colCount];
        }

        public double getData(int row,int col){
                return data[row][col];
        }

        public void setData(int row,int col,double val){
                data[row][col]= val;
        }

        public double getGeneExperiment(int gene, int expr){
                return getData(gene,expr);
        }

        public void setRow(double data[], int row){
                for (int i=0;i < colCount; i++)
                        this.data[row][i] = data[i];
        }

        public double[] getRow(int row){
                double[] temp = new double[colCount];
                for (int i=0;i < colCount; i++){
                        temp[i] = data[row][i];
                }
                return temp;
        }

        public void setCol(double data[], int col){
                for (int i=0; i < rowCount; i++)
                        this.data[i][col] = data[i];
        }

        public double[] getCol(int col){
                double[] temp= new double[rowCount];
                for (int i = 0; i < rowCount; i++)
                        temp[i] = data[i][col];
                return temp;
        }

        public int getGeneCount(){
                return getRowCount();
        }


        //A quick hack
        public void setRowCount(int count){
                rowCount = count;
        }

        public void setColCount(int count){
                colCount = count;
        }

        public int getColCount(){
```

```
                return colCount;
        }

        public int getRowCount(){
                return rowCount;
        }

        public int getExperimentCount(){
                return getColCount();
        }

}

/**
 * The partition matrix containing the membership
 * values of each variable for a particular object *
 */

class MembershipMatrix extends DataMatrix{
        public MembershipMatrix(int row,int col){
                super(row,col);
        }

        public double getMembership(int gene,int cluster){
                return getData(gene,cluster);
        }

        public void setMembership(int gene,int cluster, double val){
                setData(gene,cluster,val);
        }

        public int getClusterCount(){
                return getColCount();
        }

}

class CentroidMatrix extends DataMatrix{
        public CentroidMatrix(int row,int col){
                super(row,col);
        }

        public double getCentroid(int experiment,int cluster){
                return getData(experiment,cluster);
        }

        public void setCentroid(int experiment,int cluster,double center){
                setData(experiment,cluster,center);
        }
}


/**
 * A utility class for math operations
 *
 */
class MathUtil{

        /** Returns the sum of the elements of an array */
        public static double sum(double data[]){
                double sum = 0;
                for (int i = 0; i < data.length; i++)
                        sum += data[i];
                return sum;
        }

        /** Raise the elements of the array to b*/
        public static void pow(double data[],double b){
                for (int i = 0; i < data.length; i++)
                        data[i] = Math.pow(data[i],b);
        }

        /** Multiply the contents of the array by a given scalar c*/
        public static void scale(double data[],double c){
                for (int i =0 ; i < data.length; i++)
                        data[i] = c * data[i];
```

```
        }

        public static void dumpData(double data[]){
                for (int i = 0 ; i < data.length; i++)
                        System.out.print(" " + data[i]);
                System.out.println();
        }


        public static double computeObjective(int clusters,MembershipMatrix mem,

DataMatrix data){
                double memsquare;
                double sum=0;
                for (int i = 0; i < data.getGeneCount(); i++){
                        for (int j = 0; j < clusters;j++){
                                memsquare=mem.getMembership(i,j);
                                memsquare*=memsquare;
                                sum += (memsquare * computeEuclideanNorm(i,j,mem,data));
                                //System.out.println(sum);
                        }
                }
                return sum;
        }


        public static double computeEuclideanNorm(int gene,int cluster,

                 MembershipMatrix mem, DataMatrix data){
                double sum=0;
                double square;
                for (int i = 0; i < data.getExperimentCount(); i++){
                        square = (data.getData(gene,i) - computeClusterCenter
(cluster,i,mem,data))*
                                                (data.getData(gene,i) - computeClusterCenter
(cluster,i,mem,data));
                        sum+=square;
                }
                return sum;
        }

        public static double computeClusterCenter(int cluster, int experiment,
                                                                        MembershipMatrix mem,
DataMatrix data){
                double nume = 0;
                double deno = 0;
                double membership = 0;
                double memsquare;

                for (int i=0; i < data.getGeneCount(); i++){
                        memsquare=mem.getMembership(i,cluster);
                        memsquare*=memsquare;
                        nume += memsquare + data.getGeneExperiment(i,experiment);
                        deno += memsquare;
                }
                return (nume /* /deno */);
        }

        public static CentroidMatrix computeClusterCentroids(MembershipMatrix mem,

                DataMatrix data){
                CentroidMatrix retval = new CentroidMatrix(data.getExperimentCount(),

                        mem.getClusterCount());
                for (int i = 0; i < data.getExperimentCount(); i++){
                        for (int j = 0; j < mem.getClusterCount(); j++){
                                retval.setCentroid(i,j,computeClusterCenter(j,i,mem,data));
                        }
                }
                return retval;
        }


        public static MembershipMatrix randomMembership(DataMatrix data,

                                        int clusters){
```

```
                double val = 1.0 / clusters;
                double t=0;
                double remaining=1;
                int j;
                MembershipMatrix mem = new MembershipMatrix(data.getGeneCount(),

                                        clusters);
                for (int i = 0; i < data.getGeneCount(); i++){
                        remaining = 1;
                        for (j = 0; j < clusters-1; j++){
                                val = Math.random() * remaining;
                                remaining = remaining - val;
                                mem.setMembership(i,j,val);
                        }
                        mem.setMembership(i,j,remaining);
                }
                return mem;
        }


        public static void computeMembership(MembershipMatrix mem,DataMatrix data,
                                                                        CentroidMatrix centroids)
{
                double newval=0;

                for (int i = 0; i < data.getGeneCount(); i++){
                        for (int j = 0; j < mem.getClusterCount(); j++){
                                double sum = 0;
                                for (int k = 0; k < mem.getClusterCount(); k++){
                                        double nume = computeEuclideanNorm(i,j,mem,data);
                                        double deno = computeEuclideanNorm(i,k,mem,data);
                                        sum += (nume/deno);
                                }
                                newval=1/sum;
                                mem.setMembership(i,j,newval);
                                //System.out.println("Gene " + i + " has " + newval + " to
cluster " + j);
                        }
                }
        }
}

class FuzzyClusterer{
        MembershipMatrix mem;
        DataMatrix data;
        CentroidMatrix centroids;
        PCLReader source;
        int k;

        public FuzzyClusterer(){}

        public FuzzyClusterer(PCLReader source,int k){
                this.k = k;
                this.source = source;
                data = source.getDataMatrix();
        }


        public void cluster(){

                mem = MathUtil.randomMembership(data,k);
                long pass=1;
                double prev=0,current=0,delta=0,temp;
                while (true){
                        System.out.println("Pass #" + pass);
                        System.out.print("------>Computing current...");
                        prev = MathUtil.computeObjective(k,mem,data);
                        System.out.println(prev);
                        System.out.println("------>Recomputing centroids...");
                        centroids = MathUtil.computeClusterCentroids(mem,data);
                        System.out.println("------>Updating membership values...");
                        MathUtil.computeMembership(mem,data,centroids);
                        System.out.print("------>Computing next...");
                        current = MathUtil.computeObjective(k,mem,data);
                        System.out.println(current);
                        delta = Math.abs(current - prev);
```

```
                System.out.println("------>Delta: " + delta);
                pass++;
                if (delta < 0.01 || Double.isNaN(delta))
                        break;
        }
        if (Double.isNaN(delta))
                System.out.println("Failed to cluster!");
        else
                for (int i=0; i < data.getGeneCount(); i++)
                        MathUtil.dumpData(mem.getRow(i));


    }
}
```