Self-organizing Map – 2-d arrows example

This is a partial listing of the Java code for the Self-organizing map program working on two-dimensional data.

It includes all the code from the **Initialize** and **Evolution** sections, but does not include the variables' descriptions or the display section. The specification of the variables, for example the type of each variable and reasonable dimensions for the arrays, should be clear from the context.

Initialization section

```
cycle=0;
scaleit=10.0;

for (int i=0; i<n; i++)
{
   for (int j=0; j<m; j++)
   {
      arrows2d[i][j][0]=15*i/m+2;
      arrows2d[i][j][1]=15*j/n+3;
      arrows2d[i][j][2]=scaleit*(0.5-Math.random());
      arrows2d[i][j][3]=scaleit*(0.5-Math.random());
      arrows2d[i][j][4]=25;
   }
}
updatemult=0.05;
```

Evolution section

```
cycle++;

/* Generate a random angle by generating random values to be compared with the
elements [2] and [3] in the arrows array */

double two, three;

two=scaleit*(0.5-Math.random());
three=scaleit*(0.5-Math.random());

/* Find the winning node */

int ibest, jbest;
ibest=0;
jbest=0;
```

```
double Euclid;
Euclid=1.0e10;

for (int i=0; i<m; i++)
{
   for (int j=0; j<n; j++)
   {
      distance=(arrows2d[i][j][2]-two)*(arrows2d[i][j][2]-two)+(arrows2d[i][j][3]-
three)*(arrows2d[i][j][3]-three);
      if (distance<Euclid)
      {
         Euclid=distance;
         ibest=i;
         jbest=j;
      }
   }
}

/* Now we have found the winning node, update the weights to that node and to the
nodes in the neighbourhood. Assume a linear fall-off with distance from the nodes
and that the updating diminishes with cycle number */

arrowtotal=0.0;  // Will be used later for scaling
for (int i=0; i<m; i++)
{
   for (int j=0; j<n; j++)
   {
      double distance_from_target;
      distance_from_target=1.0+Math.sqrt((double)((i-ibest)*(i-ibest)+(j-jbest)*(j-
jbest)));
      double update=updatemult/(distance_from_target+0.00001*cycle*cycle);
      arrows2d[i][j][2]=(arrows2d[i][j][2]-update*(arrows2d[i][j][2]-
two))/(1.0+0.5*update);
      arrows2d[i][j][3]=(arrows2d[i][j][3]-update*(arrows2d[i][j][3]-
three))/(1.0+0.5*update);
      arrowtotal=arrowtotal+Math.abs(arrows2d[i][j][2])+Math.abs(arrows2d[i][j][3]);
   }
}

/* Scale the arrows to ensure their length is reasonable. */

for (int i=0; i<m; i++)
{
   for (int j=0; j<n; j++)
   {
      arrows2d[i][j][2]=arrows2d[i][j][2]*200.0/arrowtotal;
      arrows2d[i][j][3]=arrows2d[i][j][3]*200.0/arrowtotal;
   }
}
```