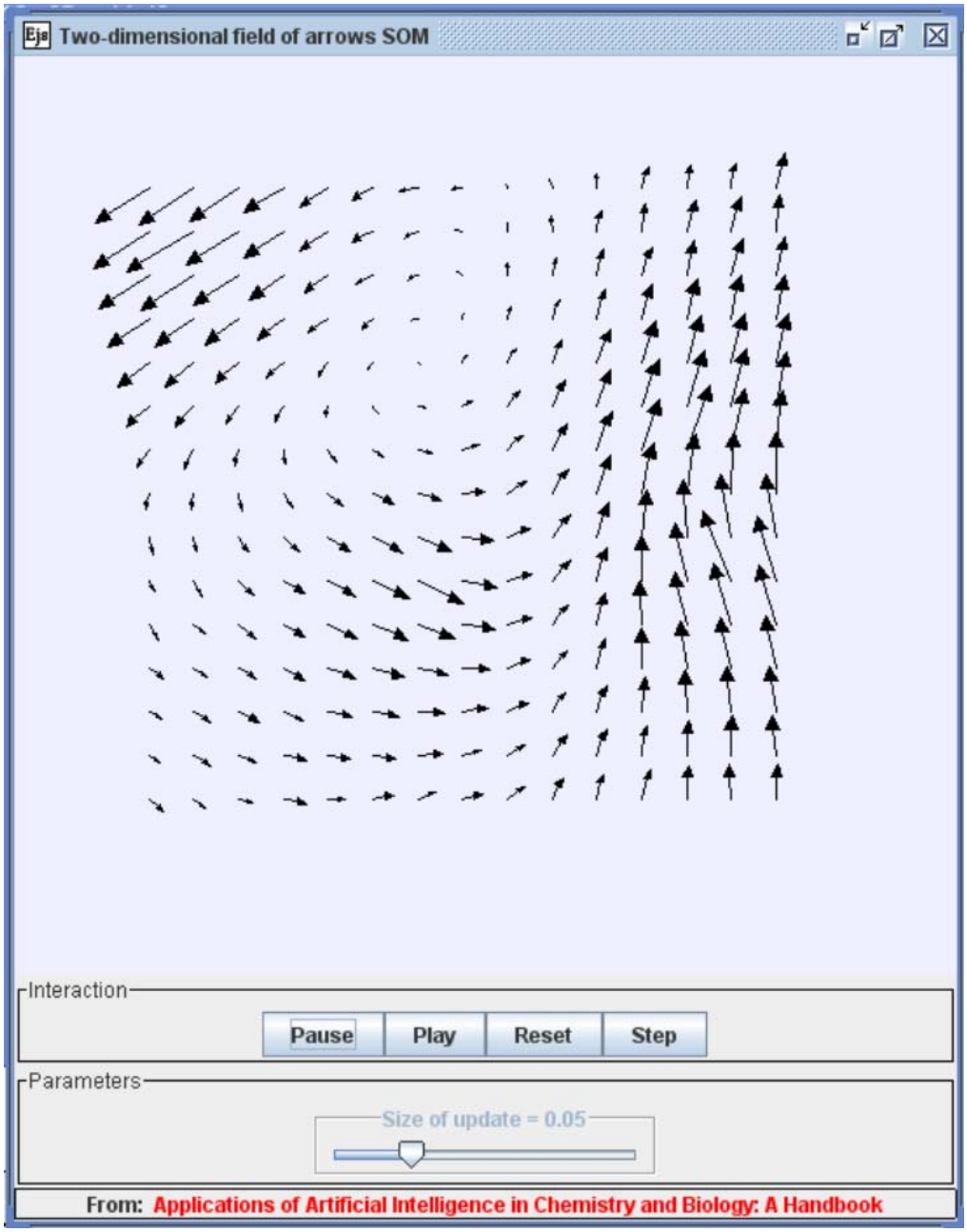
Self-organizing Map – Two-dimensional Arrow Set

This self-organizing map program follows on from the SOM that displayed the 1d-arrows set.



In that example the input data were single values in the range 0 to 360. Here the same data set is used.

```
/* 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());
```

(Actually, strictly speaking it is not the same set, because in both programs we merely pick data points at random from within the range 0-360, rather than using some predefined stored data.)

In this instance, the SOM nodes are spread across two dimensions, so the appearance of the map is rather different. However, just as when the nodes were arranged along a line, the set of node weights quickly becomes correlated, as shown by the arrows in the image above.

BUTTONS

Buttons in the window created by the program have the following functions:

Button	Function
Pause	Temporarily halt execution.
Play	Restart the program if it has previously been paused.
Reset	Restart the calculation from scratch; the update size will be returned to its default value.
Step	Execute a single cycle of the program

SLIDERS

The parameter that can be adjusted by the user is:

Slider	Default	Comment
Size of update	0.05	Determines the magnitude of the change to the node weights made per cycle.

Investigations and Exercises

1. Default parameters

Run the simulation several times. Each run generates its own fresh set of random numbers in the range 0-360 as input data, but qualitatively each data set is the same. Satisfy yourself that the appearance of the map that is produced each time is unpredictable – sometimes the pattern is a swirl, at other times the map divides into segments – but that on each occasion the weights of nodes that are close together are strongly correlated.

2 Size of update

Restart the simulation and check the effect of changing the size of the update. If the update is large and the dataset complicated, a large update size can lead to the map forgetting what has been learnt, but the data set in this example is so simple that even large updates do not usually destroy the correlation among the nodes.

However, very large updates are counterproductive. Try going to the **View** page in Ejs and change the amount by which the update can be altered. Right-click on the **slider** in the left hand panel and choose **Properties**. You can type a new value into the **Maximum** box, then click on the X in the top right corner of the **Properties for slider** box to store the new value. Rerun the simulation. Move the update to a high value and use the **Step** function to follow how the network weights change each cycle. You will understand why, though the update value must be greater than zero, a very large value does not lead to rapid convergence to a meaningful map.

3 Colorful arrows

If you are familiar with Java, try modifying the program, or writing one of your own, in which the input data consist of a random value between 0 and 360, together with three values to be interpreted as RGB (red/green/blue) values between 0 and 255. In displaying the arrows, draw them in a color defined by the rgb values. You will find that a map of at least 30 x 30 is required to give good separation.