Genetic Algorithm Curve Fitter

This is a partial listing of the Java code for the Genetic Algorithm curve fitter.

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

```
/* This is a simple genetic algorithm whose role is to determine the combination of
sin waves that best fits a chosen input pattern, itself constructed by summing a set of
sin waves. */

nsincurves=5;            // # of sin curves to be combined to make the target pattern
oldnsincurves=nsincurves;
nstrings=80;             // # of genetic algorithm strings
oldnstrings=nstrings;
npoints=100+30*nsincurves;  // # of points to be displayed
mutationRate=0.2;        // Initial GA mutation rate
crossoverRate=1.0;       // Initial GA crossover rate
showbeststring=true;     // true if the best string is to be displayed
showtrace=true;          // true if the curve to which a fit is attempted is shown
showrandomstring=false;  // when true, will display the output of a random GA string

/* Define the values of x and y at which points are drawn. */

for (ipoints=0; ipoints<npoints; ipoints++)
{
   x[ipoints]=2.0+ipoints*95.0/npoints;
   y[ipoints]=50.0;
}

/* Choose random coefficients for the sin curves which will be summed to prepare the
trace. */

for (icoeff=0; icoeff<nsincurves; icoeff++)
{
   xcoeff[icoeff]=15.0*Math.random();
   offset[icoeff]=15.0*Math.random();
}

/* Calculate the z-coordinate for each of the points */

for (ipoints=0; ipoints<npoints; ipoints++)
```

```
{
   z[ipoints]=50.0;
   {
      for (j=0; j<nsincurves; j++)
      z[ipoints]=z[ipoints]+xcoeff[j]*Math.sin(offset[j]+j*x[ipoints]/10.0);
   }
}

/* Fill the GA strings with random values; the "15" appears because the values chosen
   for the target coefficients lie in the same range. */

for (istring=0; istring<nstrings; istring++)
{
   for (j=0; j<nsincurves; j++)
   {
      GAstring[istring][j][0]=15.0*Math.random();
      GAstring[istring][j][1]=15.0*Math.random();
   }
}

bestfitness=0.0;      // The best fitness found so far by the GA.
```

Evolution section

```
/* The first part of this section is executed if the user presses the "Use updated
params" button, or if the program detects that  the number of sin curves or the number
of GA strings has changed. It  reinitializes the program, using any updated
parameters, such as number of GA strings or number of sin curves, that the user  has
selected.

If, instead, the "Start from scratch" button is pressed, the program restarts from the
beginning and reinitializes. */

if (oldnsincurves != nsincurves)
{
   oldnsincurves=nsincurves;
   revisedParameters=true;
}

if (oldnstrings != nstrings)
{
   oldnstrings=nstrings;
   revisedParameters=true;
}

if (revisedParameters) // Start of section to be executed if parameters have changed
{
```

```
   revisedParameters=false;  // Reset so that this section is not executed every cycle
   cycle=0;
   npoints=100+30*nsincurves;
   showbeststring=true;
   showtrace=true;
   showrandomstring=false;

/* npoints may have changed, so redefine the values of x and y at which points are
drawn. */

   for (ipoints=0; ipoints<npoints; ipoints++)
   {
      x[ipoints]=2.0+ipoints*95.0/npoints;
      y[ipoints]=50.0;
   }

/* Choose random coefficients for the sin curves used to prepare the trace. */

   for (icoeff=0; icoeff<nsincurves; icoeff++)
   {
      xcoeff[icoeff]=15.0*Math.random();
      offset[icoeff]=15.0*Math.random();
   }

/* Calculate the z-coordinate for each of the points */

   for (ipoints=0; ipoints<npoints; ipoints++)
   {
      z[ipoints]=50.0;
      {
         for (j=0; j<nsincurves; j++)
         z[ipoints]=z[ipoints]+xcoeff[j]*Math.sin(offset[j]+j*x[ipoints]/10.0);
      }
   }

/* Fill the GA strings with random values */

   for (istring=0; istring<nstrings; istring++)
   {
      for (j=0; j<nsincurves; j++)
      {
         GAstring[istring][j][0]=15.0*Math.random();
         GAstring[istring][j][1]=15.0*Math.random();
      }
   }
   bestfitness=0.0;
}    // End of the section to be executed if parameters have changed.

cycle++;
xcycle=(double)cycle;
```

```
/* The last line is needed because the slider that allows the user to adjust the rate at
which the program runs requires a double, not an integer. */

// Select a random string in case the user wants to display it later

stringtoshow=(int)((float)nstrings*Math.random());

/*                  +++ Start of the main GA loop +++

   First calculate the fitness of every GA string, using a least squares measure. Start by
calculating the z-coordinates predicted by each GA string, then compare with the
actual curve. */

for (istring=0; istring<nstrings; istring++)
{
   for (ipoints=0; ipoints<npoints; ipoints++)
   {
      zpredicted[ipoints]=50.0;
      for (j=0; j<nsincurves; j++)
      {
zpredicted[ipoints]=zpredicted[ipoints]+GAstring[istring][j][0]*Math.sin(GAstring[is
tring][j][1]+j*x[ipoints]/10.0);
      }
   }  // trace has been calculated; now find the least squares sum

   lssum=0.0;
   for (ipoints=0; ipoints<npoints; ipoints++)
   {
      lssum=lssum+Math.pow(z[ipoints]-zpredicted[ipoints],2);
   }
   fitness[istring]=10.0/(1.0+lssum);

/* Check whether the current string is fitter than any previous string this cycle. If it is,
copy the trace that this string defines into zdisplaybest  so that it can be displayed. */

   if(showbeststring && fitness[istring]>bestfitness)
   {
      bestfitness=fitness[istring];
      for(ipoints=0; ipoints<npoints; ipoints++)
      {
         zdisplaybest[ipoints]=zpredicted[ipoints];
      }
   }

/* Also check to see whether a random string is to be displayed; if it is, check whether
the current string is the one we chose at random earlier. If it is, copy the z-coordinates
into an array for later display. */

   if(showrandomstring && stringtoshow==istring)
```

```
    {
      for(ipoints=0; ipoints<npoints; ipoints++)
      {
        zdisplayrandom[ipoints]=zpredicted[ipoints];
      }
    }
}

/* Here come the Genetic Algorithm bits...
   Now that the fitness has been calculated, select the better strings. In this example we
use a binary tournament to do this. */

newnstrings=0;   // newnstrings will be used to count through the newly created
strings

// Pick two strings at random, making sure they are different

do
{
   do
   {
      string1=(int)(nstrings*Math.random());
      string2=(int)(nstrings*Math.random());
   }
   while (string1 == string2);  // Now got two different strings

   beststring=string1;
   if(fitness[string2]>fitness[string1]) beststring=string2;  // pick the fitter string
   for (icoeff=0; icoeff<nsincurves; icoeff++)
   {
      newGAstring[newnstrings][icoeff][0]=GAstring[beststring][icoeff][0];
      newGAstring[newnstrings][icoeff][1]=GAstring[beststring][icoeff][1];
   }
newnstrings++;
}
while (newnstrings<nstrings-1);

/* Once newnstrings=nstrings-1, the parent pool is full, so now we do crossover.  We
do this a number of times equal to half the crossover rate multiplied by the number of
strings */

ncross=(int)(0.5*nstrings*crossoverRate);
for (icross=0; icross<ncross; icross++)
{

/* Pick two different strings at random */

   do
   {
      string1=(int)(nstrings*Math.random());
```

```
      string2=(int)(nstrings*Math.random());
   }
   while (string1 == string2);

/* Now we've picked the strings, choose a random crossing point... */

   cutpoint=(int)(nsincurves*Math.random());

/* ... and swap the segments of the two strings from the cutpoint onwards */

   double hold;
   for (icoeff=cutpoint; icoeff<nsincurves; icoeff++)
   {
      hold=newGAstring[string1][icoeff][0];
      newGAstring[string1][icoeff][0]=newGAstring[string2][icoeff][0];
      newGAstring[string2][icoeff][0]=hold;

      hold=newGAstring[string1][icoeff][1];
      newGAstring[string1][icoeff][1]=newGAstring[string2][icoeff][1];
      newGAstring[string2][icoeff][1]=hold;
   }
}

/* The final GA step is mutation. Go through every string, selecting it with probability
pmutation. If the string is selected, choose a random position at which to make the
change. */

for (istring=0; istring<nstrings; istring++)
{
   if (Math.random()<mutationRate)
   {
      icoeff=(int)(nsincurves*Math.random());
      int oneorzero=0;
      if (Math.random()>0.5) oneorzero=1;
      newGAstring[istring][icoeff][oneorzero]=15.0*Math.random();
   }
}

/* Child population is complete, so copy it into the working GA array, then go back to
the top to start again.... */

for (istring=0; istring<nstrings; istring++)
{
   for (icoeff=0; icoeff<nsincurves; icoeff++)
   {
      GAstring[istring][icoeff][0]=newGAstring[istring][icoeff][0];
      GAstring[istring][icoeff][1]=newGAstring[istring][icoeff][1];
   }
}
```