Scott Triglia
Math 164 Final Report
4.18.08

Exploring evolution using genetic algorithms

Introduction

One of the trickiest things about explaining the concept of evolution to the average person is convincing them it actually works. Since, by its nature, evolution is not a particularly obvious or quick process, it is quite natural for the process to be hard to understand, or even unbelievable, upon first encounter. The concept of order rising from disorder, or quality emerging from mediocrity is hard to stomach when it is attributed to little more than time and a moderately effective selection process.

Given the current debates that are ongoing about evolution in the public sphere, I am interested in ways that scientists can work to make the process less mysterious to the average person. I'm of the firm belief that there is a whole sector of the American population that simply needs to see compelling evidence that evolution happens, before their eyes, to open their minds to the possibility.

For these reasons, my goal with this project has been to create a very simple model of evolution, both to show that it works, and to investigate some deeper questions about its behavior.

Construction of the model

References

Before discussing the actual specifics of the model, I'll briefly mention what information I used to decide on the details of the model. Feel free to skip to the next section if the specific sources I used are of no interest to you.

To construct the model, I relied on a few papers.

Modelling Crossover-Induced Linkage in Genetic Algorithms (2001) by Adam Prügel-Bennett caused me to consider the differences between various models of crossover, which will be discussed later.

For mutation, I chose to use a (fairly standard) equal probability mutation for each bit of information in the genes.

For the overall course of the algorithm, I used the basic outline given by comp.ai.genetic Usenet group at http://www.-faqs.org/faqs/ai-faq/genetic/part2/.

Genes

The first essential component of any evolutionary model is the genome of each Creature. For simplicity, I decided to make my creatures Math Creatures, whose genomes would consist of various arithmetic tables. In the end, I chose to have three genes, an addition, multiplication and modulus gene(or table) for positive integers 1-10. For the sake of the algorithm, these tables are each treated as a list of 100 integers.

The goal of our Math Creatures will be to attain the perfect addition, multiplication and modulus genes. This will be defined, naturally enough, as the set of genes where the tables in question are completely accurate.

For example, the following is a small 1x3 addition table, and a sample gene, which has an incorrect bottom row.

```
ActualTable = {{2, 3, 4}, {3, 4, 5}, {4, 5, 6}} // Grid
SampleGene = {{2, 3, 4}, {3, 4, 5}, {0, 0, 0}} // Grid
```

```
2 3 4
3 4 5
4 5 6
```

```
2 3 4
3 4 5
0 0 0
```

Now that we know the makeup of the genes in our Math Creatures, we must determine a metric for calculating how good a given creature's genes are.

Fitness

For simplicity's sake, the fitness of a creature is determined by the sum of squared errors from the addition/multiplication/modulus tables. As an example, consider the following (shortened) addition table, and it's accompanying fitness. It is important to remember that under this metric, larger values mean a gene is worse. In essence this is a "badness" metric.

```
SampleGene = {{2, 3, 4}, {3, 4, 5}, {0, 0, 0}};
ActualTable = {{2, 3, 4}, {3, 4, 5}, {4, 5, 6}};
GetFitness[S_, A_] := Module[{sum},
  sum = 0;
  For[i = 1, i ≤ 3, i = i + 1,
   For[j = 1, j ≤ 3, j = j + 1,
    sum = sum + (S[[i, j]] - A[[i, j]]) ^2 ;]];
  Return[sum]]
GetFitness[SampleGene, ActualTable]
```

```
77
```

The program also has a linear metric built in, simply summing the absolute error from ideal of each cell. For the same example, this metric would produce the following "badness" rating.

```
GetFitnessLinear[S_, A_] := Module[{sum},
  sum = 0;
  For[i = 1, i ≤ 3, i = i + 1,
   For[j = 1, j ≤ 3, j = j + 1,
    sum = sum + Abs[S[[i, j]] - A[[i, j]]];]];
  Return[sum]]
GetFitnessLinear[SampleGene, ActualTable]
```

```
15
```

Selection

The final details is deciding how to select the best organisms from a population. We have our fitness metric, so the surviving organisms are determined by simply picking the top 100 creatures, with each choice having some probability of picking randomly, instead of the best organism available. This probability is referred to elsewhere in this paper and the code as NSProb.

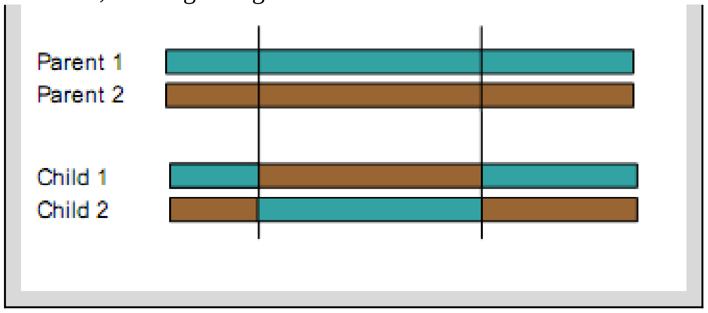
We now have a representation for a creature, a measure of fitness and a way to select the top organisms. What remains is to determine how to reproduce.

Reproduction Model

The model for reproduction is based in a large part off of the paper by Adam Prügel-Bennett mentioned in the References section. The two main parts of the model are deciding how to combine creatures and how to create a new child from two creatures.

The first decision, about combining creatures, was done naively for now. I simply take creatures in random pairs and create two children. Doing this 4 times per creature results in a population 4x larger. Then you can cull the top quarter of the population, restoring our original number of creatures, ready for the next generation.

For creating a child, there are two considerations. The first is how to combine the parents' genes into the child, and the second is how to mutate the child slightly. For crossover, I implemented two-point crossover, as discussed by Prügel-Bennett. This means each gene of the child will contain a third of one parent's gene, and two thirds of the other's. The following diagram illustrates this process(thanks Wikipedia!). You can see the parents' gene, along with two crossover points. The childrens' genes are then created by splitting the parent genes along these lines and swapping them back and forth, creating new genes.



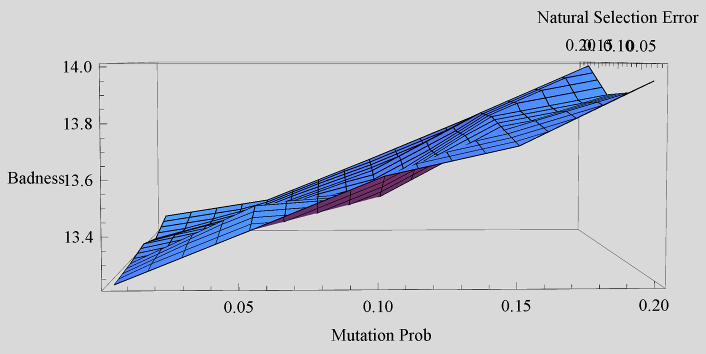
For mutation, I simply set a probability of each cell in the child's gene's increasing by a random value between -1 and 1. This probability is referred to in the code and this document as MutProb.

Exploring the model

For the program to work correctly, ensure that the MathCreatures folder this file came in is placed directly in you C:\ drive, so that the path to this file is C:\MathCreatures\triglia_final_report.nb

Here is a graph of the average final fitness, given varying values of natural selection probability and mutation probability, for 800 generations of Creatures. Notice that the rate at which natural selection makes errors doesn't much affect the final fitness (within the range shown), but the mutation rate significantly does. Because of this graph, the default NSProb is set to 0.1 and the default MutProb is set to 0.01.

```
L = Import["C:\\MathCreatures\\paramSearch.txt", "Table"];
ListPlot3D[L, AxesLabel → {"Mutation Prob", "Natural Selection Error", "Badness"},
 DataRange → {{0.005, 0.2}, {0.005, 0.2}}]
```



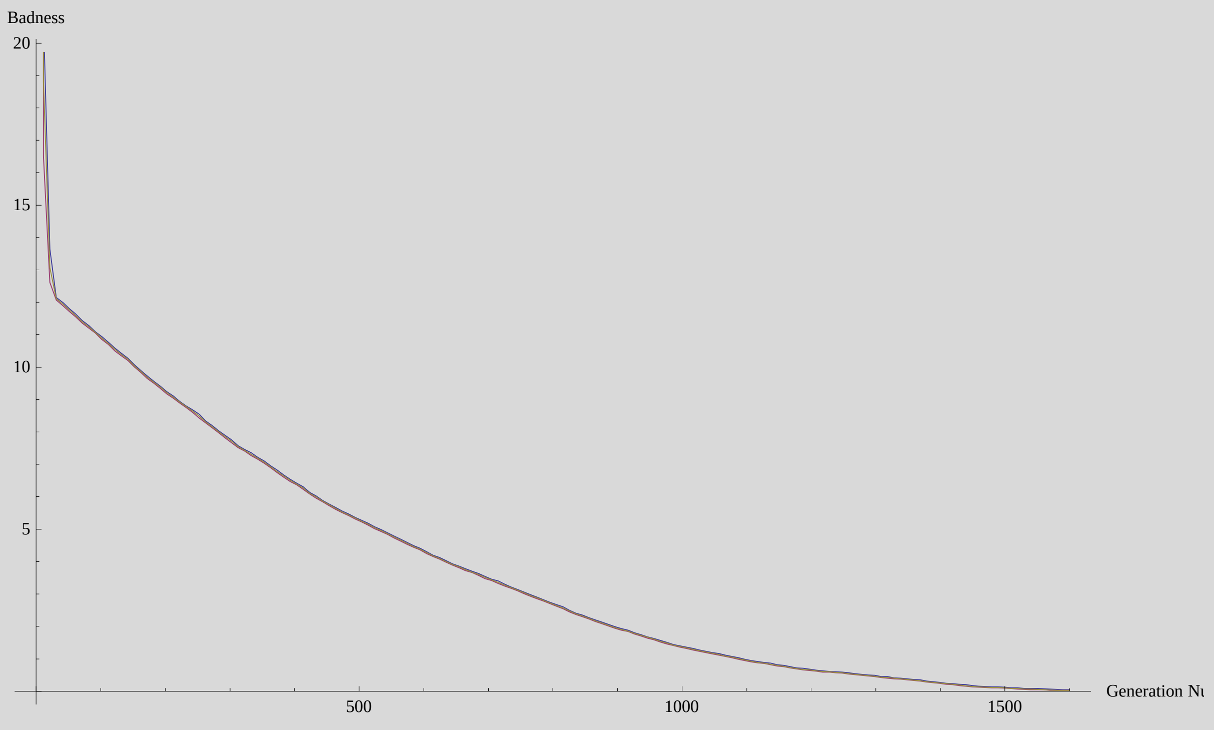
The following are the 8 parameters to the genetic algorithm. The first parameter determines which program is run. For more information on this parameter, refer to the appendix. The following 5 deal with the length and parameters of the simulation. The next three determine whether all three genes are considered, and what their relative importance is to fitness. The final parameter determines whether the fitness metric is the squared error as discussed, or the absolute error.

This is the fun part! Change the parameters around and watch the changes. Note that the graph is showing badness over time, so we expect it to decrease. A particularly interesting phenomenon is the huge bend in the graph after just 20 or so generations where improvement slows sharply. It seems that after the initial creatures are culled through a few generations, there is relatively little genetic diversity left, and so improvement must happen because of mutation rather than crossover during the reproductive process.

```
programName = "TrackAll";
numTrials = ToString[1];
(* The number of trials - averaged to produce the graph you see *)
numGens = ToString[1600]; (* The number of generations to run the world for *)
genRes = ToString[10];  (* How many generations between data points *)
NSProb = ToString[0.1];
(* The probability of natural selection not choosing rationally *)
MutProb = ToString[0.01];
(* The probability of a mutation in a single cell of a creature *)
(* The weights on the three genes *)
add = ToString[0];
mult = ToString[1];
mod = ToString[0];
fitStr = "Squared";
geneChoice = "Multiplication";
(* Which gene do you want to track over the simulation*)

(* Runs the .exe *)
Run[StringJoin["cd C:\MathCreatures && MathCreatures.exe ",
   programName, " ", numTrials, " ", numGens, " ", genRes, " ", NSProb, " ",
   MutProb, " ", add, " ", mult, " ", mod, " ", fitStr, " ", geneChoice]];

(* Reads the output table and graphs it *)
T = Import["C:\\MathCreatures\\runData.txt", "Table"];
ListLinePlot[{T[[1]], T[[2]], T[[3]]}, AxesLabel → {"Generation Number", "Badness"},
 PlotRange → Automatic, DataRange → {1, ToExpression[numGens]}]
```

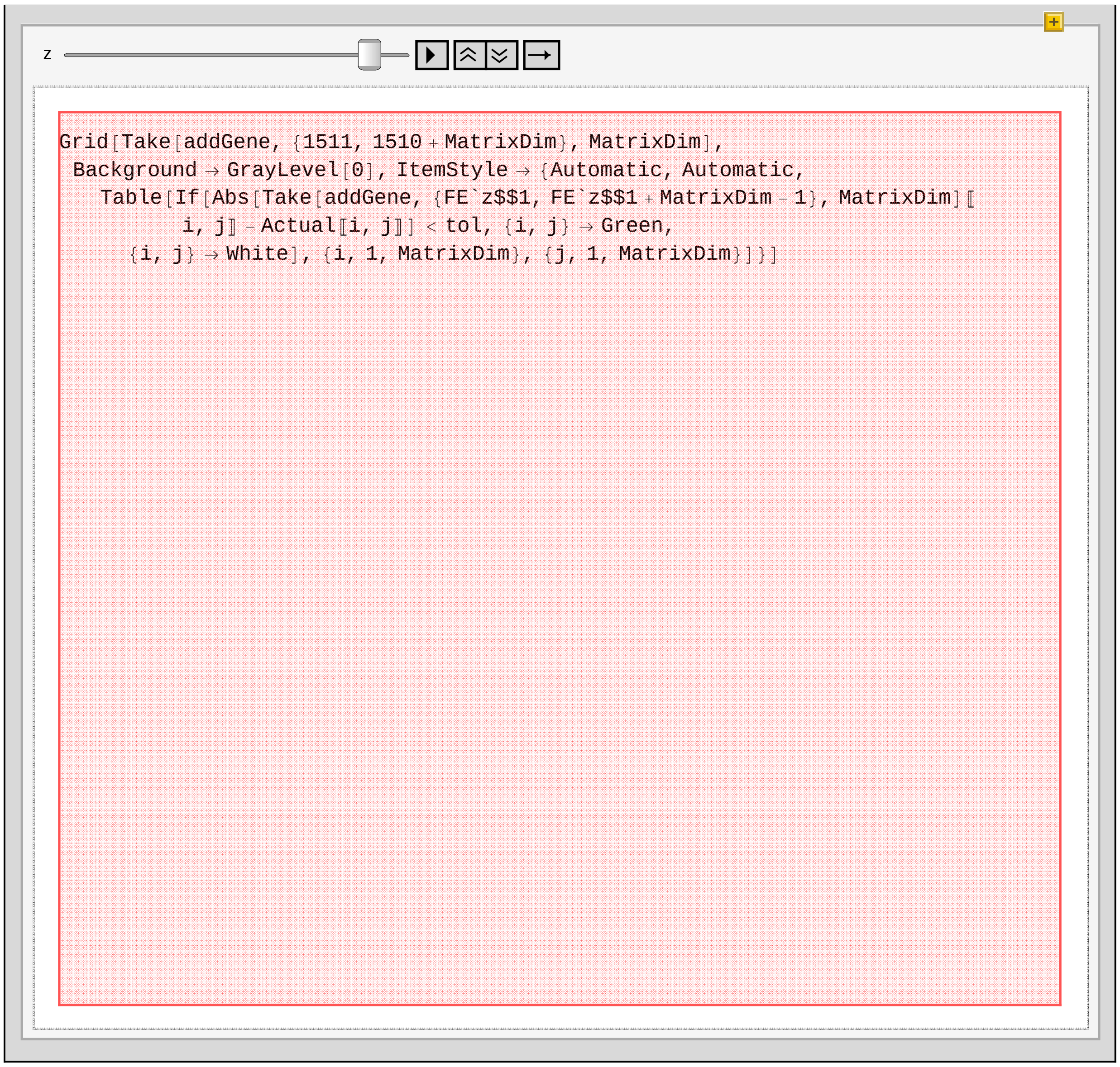


Another fun thing to view is the average multiplication gene as you go through the generations of Creatures. In this cell, you can view the average creature's multiplication cell from the same run whose graph is above.

```
(* How close to correct before we highlight it?*)
tol = .5;
MatrixDim = 10;
genIters = Floor[ToExpression[numGens] / ToExpression[genRes]];
numRows = (genIters - 1) * MatrixDim + 1;

addGene = Import["C:\\MathCreatures\\trackGenes.txt", "Table"];

Actual = Table[i * j, {i, 1, MatrixDim}, {j, 1, MatrixDim}];
Animate[ Grid[Take[addGene, {z, z + MatrixDim - 1}, MatrixDim],
  Background → Black, ItemStyle → {Automatic, Automatic,
    Flatten[Table[If[Abs[Take[addGene, {z, z + MatrixDim - 1}, MatrixDim][[i, j]] -
          Actual[[ i, j]]] < tol, {i, j} → Green, {i, j} →  White],
      {i, 1, MatrixDim}, {j, 1, MatrixDim}]]}], {z, 1, numRows,
  MatrixDim}, AnimationRunning → False, DefaultDuration →
  10]
```

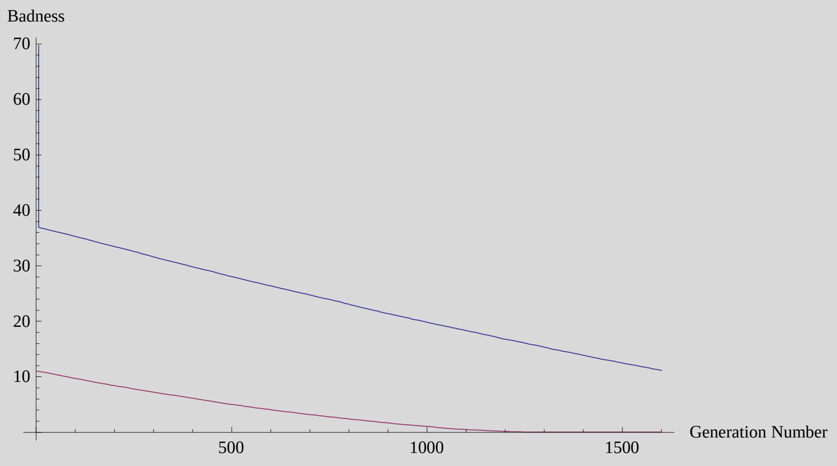


```
Grid[Take[addGene, {1511, 1510 + MatrixDim}, MatrixDim],
 Background → GrayLevel[0], ItemStyle → {Automatic, Automatic,
   Table[If[Abs[Take[addGene, {FE`z$$1, FE`z$$1 + MatrixDim - 1}, MatrixDim][[
         i, j]] - Actual[[i, j]]] < tol, {i, j} → Green,
     {i, j} → White], {i, 1, MatrixDim}, {j, 1, MatrixDim}]}]
```

```
programName = "TrackTwo";
numTrials = ToString[1];
(* The number of trials - averaged to produce the graph you see *)
numGens = ToString[1600]; (* The number of generations to run the world for *)
genRes1 = ToString[5];  (* How many generations between data points *)
genRes2 = ToString[10];  (* How many generations between data points *)
NSProb1 = ToString[0.1]; (* The probability of natural selection not choosing *)
NSProb2 = ToString[0.1];
(* The probability of natural selection not choosing rationally *)
MutProb1 = ToString[0.1];
MutProb2 = ToString[0.01];
(* The probability of a mutation in a single cell of a creature *)
(* The weights on the three genes *)
add = ToString[0];
mult = ToString[1];
mod = ToString[0];
fitStr1 = "Squared";
fitStr2 = "Squared";
geneChoice = "Multiplication";
(* Which gene do you want to track over the simulation*)

(* Runs the .exe *)
Run[StringJoin["cd C:\MathCreatures && MathCreatures.exe ",
   programName, " ", numTrials, " ", numGens, " ", genRes1, " ", genRes2,
   " ", NSProb1, " ", NSProb2, " ", MutProb1, " ", MutProb2, " ", add,
   " ", mult, " ", mod, " ", fitStr1, " ", fitStr2, " ", geneChoice]];

(* Reads the output table and graphs it *)
T = Import["C:\\MathCreatures\\runData.txt", "Table"];
ListLinePlot[{T[[1]], T[[2]]}, AxesLabel → {"Generation Number", "Badness"},
 PlotRange → Automatic, DataRange → {1, ToExpression[numGens]}]
```



Appendix - Code Use

The program can be used by calling MathCreatures.exe with the following options:

programName - This can be:
"TrackGenes" - creates a trackGenes.txt file which contains the average gene at each timestep for the gene given in geneChoice
"TrackFitness" - creates a runData.txt file which contains the best, worst and average fitness at each timestep
"TrackAll" - creates both files listed above
"PlotParams" - creates paramSearch.txt which shows the fitness after numGens generations for varying values of NSProb and MutProb. This can be viewed by using the code shown in the Parameters section of this document.

To run MathCreatures.exe from inside *Mathematica*, use the line

```
Run[StringJoin["cd C:\MathCreatures && MathCreatures.exe ", arguments]];
```

where "arguments" is a variable containing all command line arguments you wish to use. Naturally, you can always run the exe from the Windows command line if you'd prefer.

The other command line options vary based on which of these programs you want to run.

Tracking Programs

For all of the "Track" options, the remaining options are as follows

numTrials - How many populations should the results be averaged over
numGens - How many generations should each simulation run for
genRes - How many generations between data points
NSProb - What is the probability that natural selection chooses randomly
MutProb - What is the probability that each cell in each gene mutates
add - What is the fitness importance of the addition gene (a floating point number....add, mult and mod must sum to 1)
mult - What is the fitness importance of the multiplication gene
mod - What is the fitness importance of the modulus gene
fitStr - What type of fitness calculation should be used (must be "Linear" or "Squared")
geneChoice - What gene should be tracked for trackGene.txt (must be "Addition" "Multiplication" or "Modulus")

Plot Parameters

For PlotParams, the arguments are slightly different, as follows.

numTrials - How many populations should the results be averaged over
numGens - How many generations should each simulation run for
probRes - How thinly should the search space be divided (7 means there will be 7 values for NSProb and 7 for MutProb)
add - What is the fitness importance of the addition gene (a floating point number....add, mult and mod must sum to 1)
mult - What is the fitness importance of the multiplication gene
mod - What is the fitness importance of the modulus gene
fitStr - What type of fitness calculation should be used (must be "Linear" or "Squared")

Comparative Programs

For "TrackTwo", the program will take in two inputs for programs and output the two results, for comparison.

numTrials - How many populations should the results be averaged over

numGens - How many generations should each simulation run for
genRes1 - How many generations between data points
genRes2 - How many generations between data points
NSProb1 - What is the probability that natural selection chooses randomly
NSProb2 - What is the probability that natural selection chooses randomly
MutProb1 - What is the probability that each cell in each gene mutates
MutProb2 - What is the probability that each cell in each gene mutates
add - What is the fitness importance of the addition gene (a floating point number....add, mult and mod must sum to 1)
mult - What is the fitness importance of the multiplication gene
mod - What is the fitness importance of the modulus gene
fitStr1 - What type of fitness calculation should be used (must be "Linear" or "Squared")
fitStr2 - What type of fitness calculation should be used (must be "Linear" or "Squared")
geneChoice - What gene should be tracked for trackGene.txt (must be "Addition" "Multiplication" or "Modulus")