COMP6026: Evolution of Complexity
Assignment II: Coevolution

Ashley J. Robinson
ajr2g10@ecs.soton.ac.uk

School of Electronics and Computer Science
University of Southampton

January 6, 2014

1 Introduction

This assignment is based on the paper *Coevolutionary Dynamics in a Minimal Substrate* (Watson and Pollack, 2001). It describes a very simple greater than number game space allowing for coevolving populations to be examined; especially intransitive superiority (Cliff and Miller, 1995). In this case two populations consisting of 25 individuals coevolve on the minimal substrate. Equation (1) is used to to produce a subjective measure of fitness defined by scoring against a sample of members, S, from the other population. These scoring functions are defined in Equations (2), (3) and (4).

Equation (2) is a basic comparison with only two possible return values. Sending the first parameter (a) as the individual to be evaluated and second (b) to be compared against. The sum of these scores, in Equation (1), gives the subjective fitness where it is possible to vary the sample size from one to all individuals in the coevolving population therefore setting the fitness range as 0 to $|S|$. The remaining scoring functions are used for multi dimensional applications of the game. These both call Equation (2) and only determine which dimension the one dimensional game is played upon. Equations (3) and (4) are two dimensional examples where the chosen dimension to play on is either the furtherest or closest respectively. The same principle can be easily extended to cover higher dimensional games.

$$f(a, S) = \sum_{i=1}^{|S|} score(a, S_i) \tag{1}$$

$$score1(a, b) = \begin{cases} 1, \, if \; a > b \\ 0, \, otherwise \end{cases} \tag{2}$$

$$score2((a_x, a_y), (b_x, b_y)) = \begin{cases} score(a_x, b_x), \, If \; (|a_x - b_x| > |a_y - b_y|) \\ score(a_y, b_y), \, Otherwise \end{cases} \tag{3}$$

$$score3((a_x, a_y), (b_x, b_y)) = \begin{cases} score(a_x, b_x), \, If \; (|a_x - b_x| < |a_y - b_y|) \\ score(a_y, b_y), \, Otherwise \end{cases} \tag{4}$$

The individuals in each population are bit strings which are 100 bits long. Mutation occurs during reproduction but there is no crossover. This is manifested as flipping bits with a probability set by the mutation rate. This rate is fixed at 0.005 throughout all simulations in section 2 therefore making the probability of performing a logical inversion of an individual 0.005^{100}. Objective fitness of an individual is the number of 1's in the bit string creating a cap on the discrete fitness measure at 100. Mutation bias is an important factor in this setup because it is expected that a randomly mutating bit string will drift towards a fitness of 50. This is because any non-balanced bit string will have a greater chance of mutating the higher frequency value. Any algorithm outputting individuals with an objective fitness of 50 or less can be considered poor.

Pseudo code describing the basic outline of the generation update kernel of the evolutionary algorithm used is held in Listing 1. A sample taken from the other population is used to determine the fitness of the current population. These fitness vectors then influence the proportionate selection routine which will create the next generation. Finally mutation is applied to all new individuals to introduce variation but no crossover.

```
Sample1 = GetRandomSample(Pop1,S);
Sample2 = GetRandomSample(Pop2,S);
For i = 1 to PopSize
   Fitness1[i] = Quality(Pop1[i],Sample2);
   Fitness2[i] = Quality(Pop2[i],Sample1);
End
For i = 1 to PopSize
   Pop1[i] = ProportionateSelection(Pop1,Fitness1);
   Pop1[i] = Mutate(Pop1[i]);
   Pop2[i] = ProportionateSelection(Pop2,Fitness2);
   Pop2[i] = Mutate(Pop2[i]);
End
```

Listing 1: Evolutionary algorithm pseudo code

2 Reimplemented Results

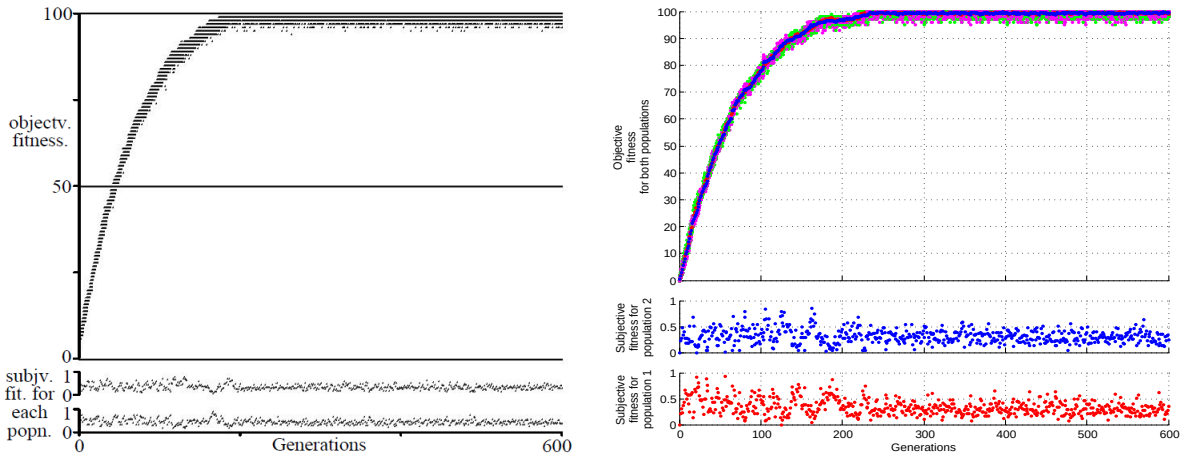
Throughout this section records of simulations are included; these are graphs holding objective and subjective fitness changing over generations. Colours are used in the reimplementation figures for clarity. See Table 1.

Population	**Individuals Colour**	**Mean Colour**
1	Pink	Blue
2	Green	Red

Table 1: Colour coding.

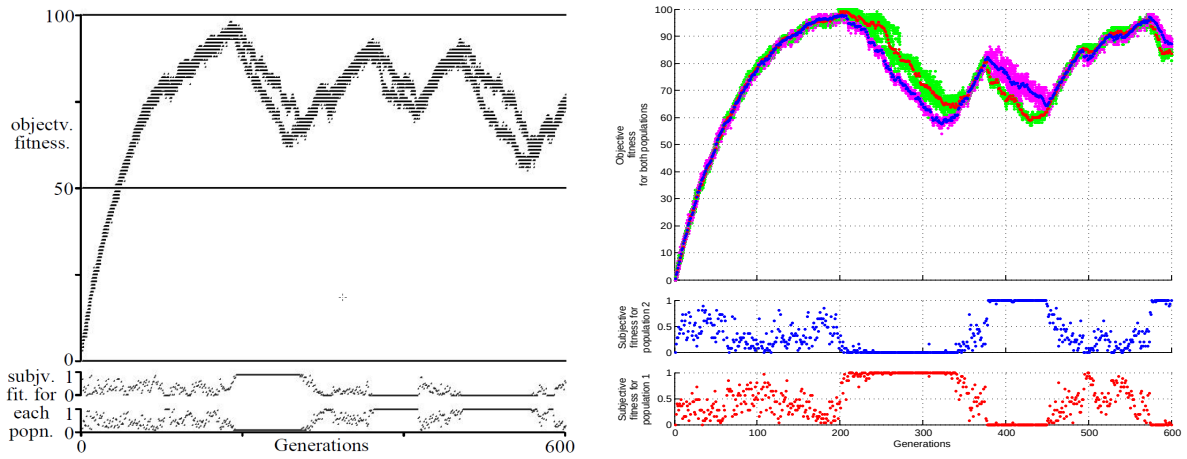
2.1 Experiment I: Loss of Gradient

Using Equation (2) the two populations are evolved using a sample size of 25 in Figure 1 and just a single sample in Figure 2. Using the entire population as a sample yields good results and all individuals are driven and stay above 90% objective fitness before 200 generations. When the sample size is set to a single individual the initial fitness rises similarly to the previous simulation but disconnects at which point both populations drift towards the mutation bias. Viewing further simulations reveals it rarely reaches mutation bias and tends to increase in fitness before when the coevolving populations are reconnected.



(a) Original (b) Reimplementation

FIGURE 1: Coevolution using Equations 1 and 2 where $S = 25$. Reimplemented from Watson and Pollack (2001).

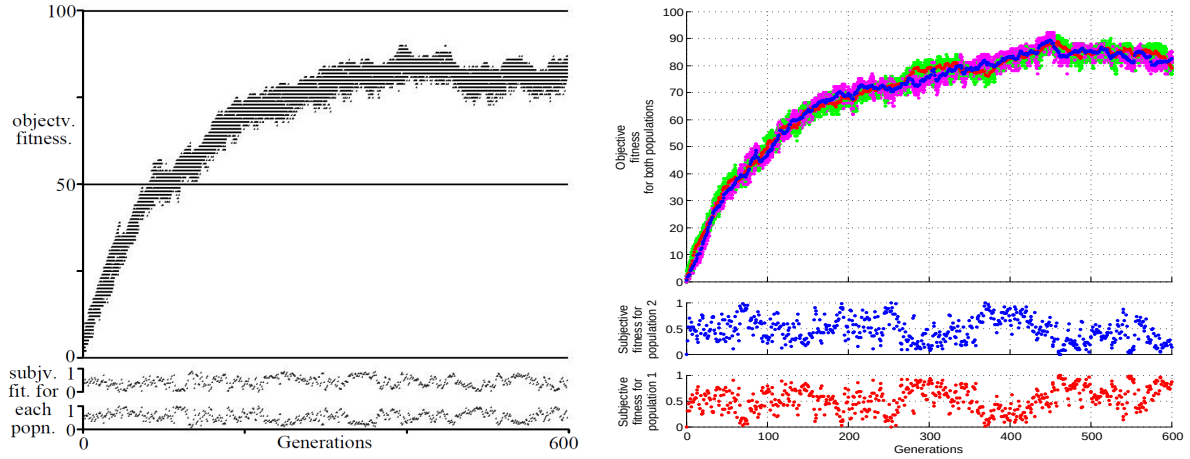


(a) Original (b) Reimplementation

FIGURE 2: Coevolution using Equations 1 and 2 where $S = 1$. Taken and reimplemented from Watson and Pollack (2001).

2.2 Experiment II: Focussing

Extending the scoring function held in Equation (3) allows a simulation of individuals with 10 dimensions. It can be seen in figure 3 that objective fitness is restricted typically to less than 90%. This is due to the game only being played in a single dimension while the other dimensions drift towards the mutation bias. One dimension is focussed on and therefore over-specialised when compared to the remaining dimensions. Sample size here is 25 and performance is significantly reduced when compared to Figure 1 which is of the same size.

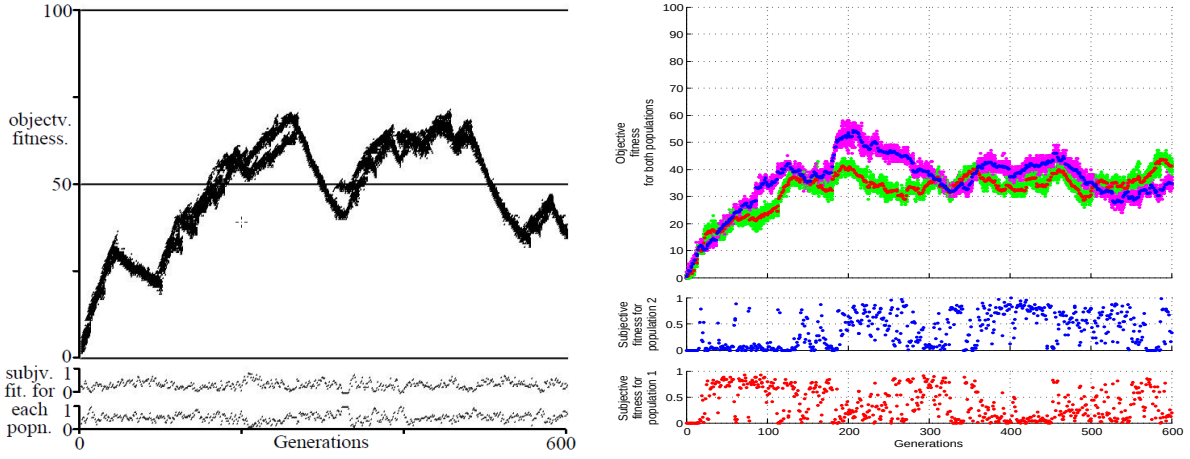


(a) Original (b) Reimplementation

FIGURE 3: Coevolution using Equations 1 and 3, 10 dimensions. Taken and reimplemented from Watson and Pollack (2001).

2.3 Experiment III: Relativism

Extending Equation (4) as in section 2.2 Figure 4 holds very poor simulation results. The objective fitness is typically below the mutation bias which means it is actively driven to a lesser fitness rather than drifting. This low fitness can be explained because it is possible to switch the dimension focus by lowering the value of the current dimension. When the focus is shifted the individual may beat it's opponent on the new dimension even though the objective fitness has decreased. Importantly this means increasing subjective fitness does not always mean increasing objective fitness.



(a) Original (b) Reimplementation

FIGURE 4: Coevolution using Equations 1 and 3, 10 dimensions. Taken and reimplemented from Watson and Pollack (2001).

3 Extension Hypothesis: An adaptive mutation rate can reduce the number of generations required to resolve a population disconnection.

3.1 Description and Expectations

Adaptive mutation rates can be considered as an improvement on using a static parameter provided by the algorithm designer (De Jong, 2006). When a simulation suffers from a population disconnect and they both drift towards the mutation bias the gradient is defined by the mutation rate (Figure 2). It is possible for the gradients of the drifting populations to remain parallel for several generations before reconnecting but if they had different mutation rates then a resolution may come quicker. The hypothesis suggests an adaptive mutation rate may cause these differing drift gradients in such a manor to prevent a disconnect or cause a reconnect.

This hypothesis arose from noticing a guaranteed way to suppress the beginnings of population disengagement is to replace both populations, when subjective fitness is polarised, with the most fit. Any further investigation was stopped because of the cooperative and competitive setup fitness sharing (Uchibe and Asada, 2006) would introduce. One method of enabling similar behaviour would be a dynamic mutation rate where the parameter is setup as an accumulator. High or low subjective fitness will add to the rate while anything else would reduce the rate. Seeking to tangle the two populations when one gains an upper hand (Thierens, 2002). However this method requires a lot of tuning which is one of the features of design trying to be eradicated.

When the subjective fitness approaches zero there is a greater chance of a disconnect and the opposing population gaining the upper hand. Decreasing the mutation rate would reduce the drift gradient and allow the populations to remain connected. Inversely increasing the mutation rate of the higher subjective fitness population will allow a broader stochastic search increasing the chance of a reconnect if necessary. A mutation rate coefficient (γ) is defined in Equation (5) which is a function of the normalised subjective fitness (β) and a tuning parameter (λ).

$$\gamma = \lambda + (1 - \lambda)\beta \tag{5}$$

The introduction of this coefficient is expected to create the desired reaction in the case of a disconnect. It does however introduce unwanted behaviour. The mutation rate will be operating at diminished value when the populations are actively competing. This may be rectified by increasing the raw mutation value but will not be consider in this investigation.

3.2 Parameter Tuning and Results

Setting λ to 0 would mean mutation is controlled entirely by the subjective fitness so possibly if both subjective fitnesses were 0 the system would lock-up. Setting λ to 1 completely suppresses the subjective fitness influence. The tuning parameter is experimented with in Figure 5 by varying from 0.1 to 0.9. As λ=0.5 yields acceptable results, with the raw mutation rate at 0.005, this will be used for further experimentation (Equation (6) to be explicit).

$$0.005\gamma = 0.005(0.5 + 0.5\beta) = 0.0025 + 0.0025\beta \tag{6}$$

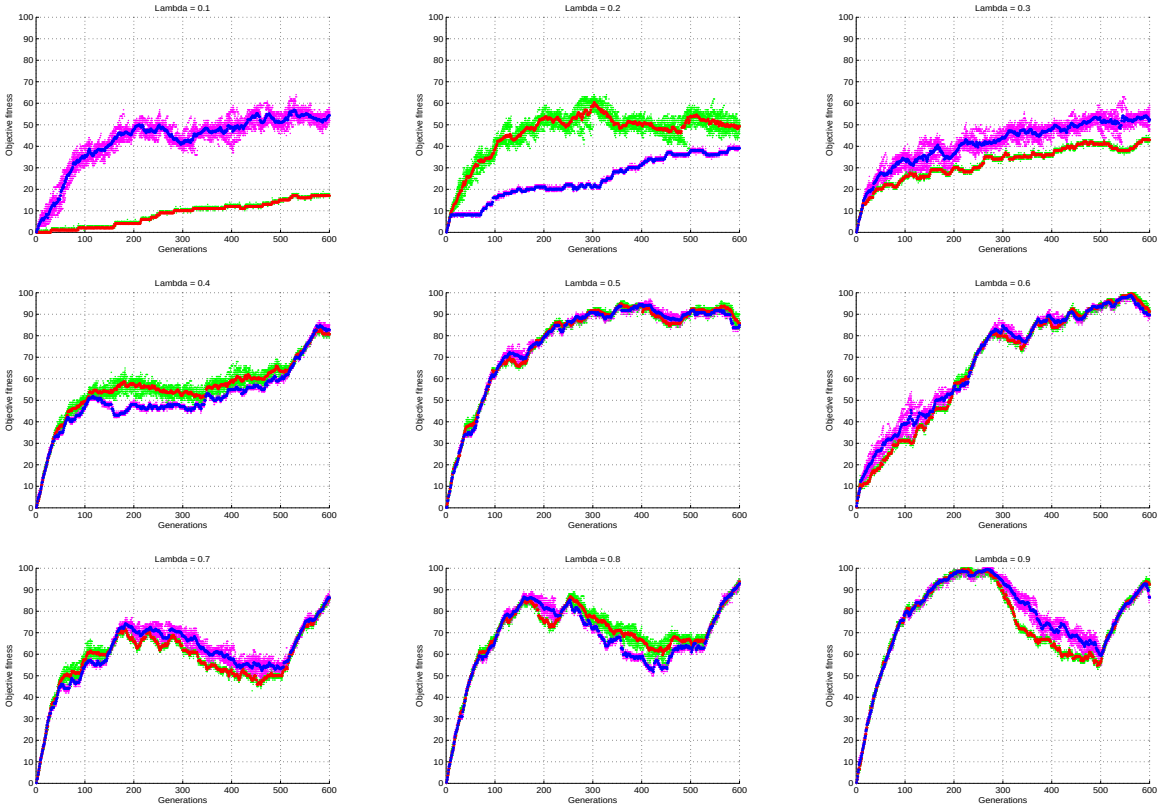


FIGURE 5: Tuning λ where the raw mutation rate=0.005 and S=1.

Two longer simulations are held in Figures 6 and 7. Over 2000 generations it is obvious the introduction of the mutation rate coefficient has improved the performance of the algorithm. Figure 6 has a higher mean objective fitness and less obvious disconnects than Figure 7 which is just a longer simulation of the setup in Figure 2. The subjective fitness while exhibiting polarisation in both simulations doesn't persist as much or appear as concentrated in Figure 6. Tests were also performed on the higher dimensional application (code held in Listing 22) using Equation (3). This produces no obvious performance increase and the output is the same as in Figure 3.

3.3 Conclusion

The hypothesis is correct. An adaptive mutation rate does reduce the number of generations to resolve a population disconnection. It does not produce an ideal simulation, as in Figure 1, but it is certainly an improvement for single sample coevolution on this minimal substrate.

One issue occurs when the populations are below the mutation bias at which point the population with the lowest subjective fitness is pinned. This can be seen in Figure 5 when λ is set to a lower value but is can also happen for higher values. The mutation rate is decreased and therefore the drift gradient is much lower for one population taking far longer to drift to the mutation bias. The population with the upper hand goes straight to the mutation bias waiting for the less fit population to catch up. Eventually this will happen but takes many generations and will even reach a 90% fitness similar to other setups.

The higher dimensional equations did not see an increase in performance from the extension. This can be explained by the same reasons as the reduced performance in the first place for higher dimensions

(see section 2.3). The performance increase was also quite small and if only assisting one dimension at a time any increase would be drowned out overall.

The reason for mutation bias comes from the experiment setup in Watson and Pollack (2001). Rather than seeking to improve the impact of the symptoms it may be possible to remove the issue completely by redefining the individuals. That is using pure numbers instead of binary strings and counting 1's.

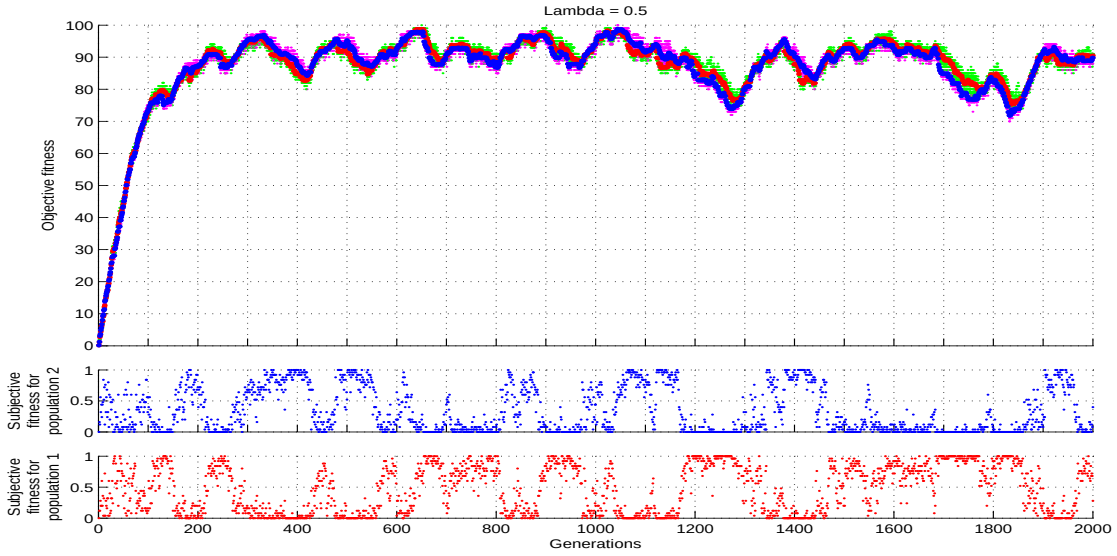


FIGURE 6: Raw mutation rate=0.005, S=1 and λ=0.5.

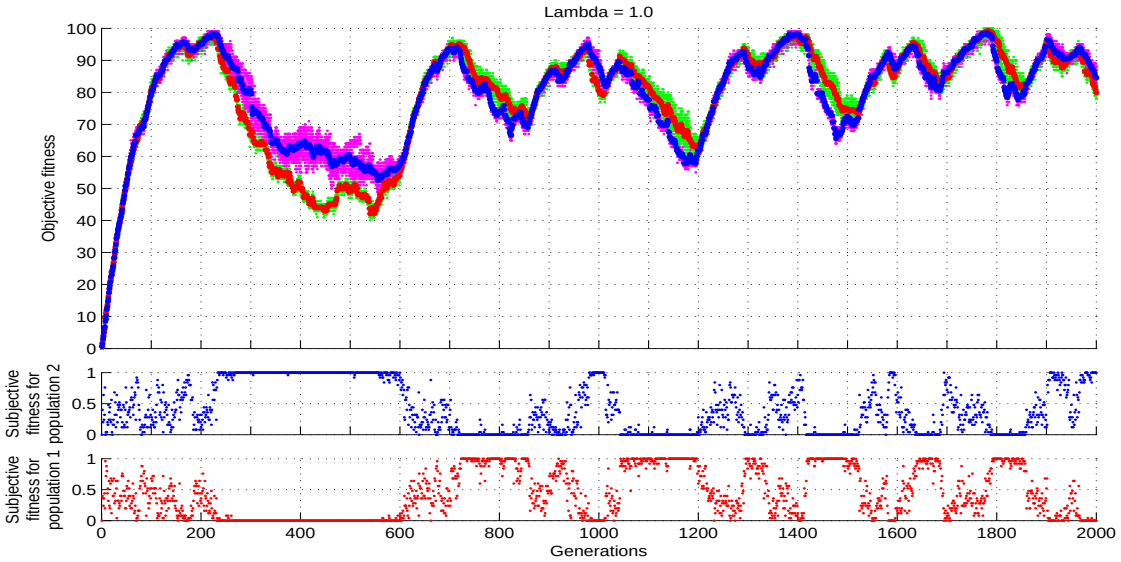


FIGURE 7: Raw mutation rate=0.005, S=1, λ=1 (No adaptive mutation rate coefficient).

References

Dave Cliff and Geoffrey F. Miller. Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In *In*, pages 200–218. Springer Verlag, 1995.

K.A. De Jong. *Evolutionary Computation: A Unified Approach*. Bradford Book. Mit Press, 2006. ISBN 9780262041942.

D. Thierens. Adaptive mutation rate control schemes in genetic algorithms. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 980–985, 2002.

E. Uchibe and M. Asada. Incremental coevolution with competitive and cooperative tasks in a multirobot environment. *Proceedings of the IEEE*, 94(7):1412–1424, 2006. ISSN 0018-9219.

Richard A. Watson and Jordan B. Pollack. Coevolutionary dynamics in a minimal substrate. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 702–709, 2001.

A Source code description

All code submitted for this assignment, in the zip archive and section B, is written for MATALB and tested on version R2012a. A description of the folders containing code submitted...

- **Control** Mutation bias example when popultions converge of 50.
- **LossOfGradient** Run *Fig2LossofGradient* or *Fig3LossofGradient* to reproduce those figures.
- **Focussing** Run *Fig4Focussing*.
- **Relativism** Run *Fig5Relativism*.
- **Extension** Extension code to produce all figures and content expressed in section 3. Run *Extension* files in folders *OneDim* and *TenDim*.

B Source code

B.1 Control

```
clear
% Evolution
lenString = 100;          % Bit string length
popSize = 25;             % Population size
mutateRate = 0.005;       % Mutation rate
genLimit = 600;           % Run fro this many generations
%  Init Pop  %
pop1 = zeros(lenString,popSize);
pop2 = ones(lenString,popSize);
for gen=1:genLimit
    for member=1:popSize
        pop1(:,member) = Mutate(pop1(:,member),mutateRate);
        pop2(:,member) = Mutate(pop2(:,member),mutateRate);
    end
    X = sprintf('Generation = %d', gen);    % Text output
    disp(X)
    for pop=1:popSize
        pop1_x(pop,gen) = gen;
        pop1_y(pop,gen) = ObjFit(pop1(:,pop));
        pop2_x(pop,gen) = gen;
        pop2_y(pop,gen) = ObjFit(pop2(:,pop));
    end
end
for pop=1:popSize
    scatter(pop1_x(pop,:),pop1_y(pop,:),100,'r','.')
    hold on
    scatter(pop2_x(pop,:),pop2_y(pop,:),100,'b','.')
    hold on
end
```

Listing 2: Reproduce mutation bias figure.

```
function[mutant] = Mutate(normal,rate) % Any length bit string
    mutant = normal;                   % Init output as input
    for i=1:size(normal)               % Access each bit
        if(rand(1) < rate)             % Supplies mutation rate
            if(mutant(i) == 0)         % Flip bits
                mutant(i) = 1;
            else
                mutant(i) = 0;
            end
        end
    end
end
```

Listing 3: Mutation function.

```
function[fitness] = ObjFit(bitString)
    fitness = sum(bitString);          % Just sim the vectors
end
```

Listing 4: Objective fitness function.

B.2 Loss of Gradient

```
clear
% Evolution data
lenString = 100;          % Bit string length
popSize = 25;             % Population size
mutateRate = 0.005;       % Mutation rate
genLimit = 600;           % Run fro this many generations
samSize = 15;             % Number of individuals in sample to test eagainst
%  Init Pop
pop1      = zeros(lenString,popSize);          % Some inits and some places holder
pop2      = zeros(lenString,popSize);
popBuf1   = zeros(lenString,popSize);
popBuf2   = zeros(lenString,popSize);
samBuf    = zeros(lenString,samSize);
for gen=1:genLimit
   % Pop 1
    total = 0;
    for member=1:samSize                                          % Get sample out
     of first individuals in population
       samBuf(:,member) = pop2(:,member);
    end
    for member=1:popSize                                          % Acculumate
     fitness in wheel
       fitness(member) = f(pop1(:,member),samBuf);                % Every member
     against sample TODO: random(er) sample?
       total = total + fitness(member);
       wheel(member) = total;
    end
    for member=1:popSize
       pick = rand(1)*total;
       i = 1;
       while(wheel(i) < pick)                                     % Spin wheel
          i = i + 1;
       end
       popBuf1(:,member) = pop1(:,i);                             % Update and
     mutate
       popBuf1(:,member) = Mutate(popBuf1(:,member),mutateRate);
    end
    sub1_y(gen) = total./(popSize*samSize);
   % Pop 2
    total = 0;                                                    % Repeat above
     but for second population
    for member=1:samSize
       samBuf(:,member) = pop1(:,member);
    end
    for member=1:popSize
       fitness(member) = f(pop2(:,member),samBuf);
       total = total + fitness(member);
       wheel(member) = total;
    end
    for member=1:popSize
       pick = rand(1)*total;
       i = 1;
       while(wheel(i) < pick)
          i = i + 1;
       end
       popBuf2(:,member) = pop2(:,i);
       popBuf2(:,member) = Mutate(popBuf2(:,member),mutateRate);
    end
    sub2_y(gen) = total./(popSize*samSize);
   % Population update
```

```
    pop1 = popBuf1;
    pop2 = popBuf2;
    % Record for graph
    X = sprintf('Generation = %d', gen);   % Text output
    disp(X)
    avg1_y(gen) = 0;
    avg2_y(gen) = 0;
    for pop=1:popSize
        pop1_x(pop,gen) = gen;
        pop1_y(pop,gen) = ObjFit(pop1(:,pop));
        avg1_y(gen) = avg1_y(gen) + pop1_y(pop,gen);
        pop2_x(pop,gen) = gen;
        pop2_y(pop,gen) = ObjFit(pop2(:,pop));
        avg2_y(gen) = avg2_y(gen) + pop2_y(pop,gen);
        sub1_x(gen) = gen;
        sub2_x(gen) = gen;
    end

end
% Graph
figure(1)
grid on;
subplot(6,1,6)
scatter(sub1_x,sub1_y,25,'r','.')
ylim([0 1])
xlabel('Generations')
ylabel(sprintf('Subjective\nfitness for\npopulation 1'))
grid on;
subplot(6,1,5)
scatter(sub2_x,sub2_y,25,'b','.')
ylim([0 1])
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
ylabel(sprintf('Subjective\nfitness for\npopulation 2'))
subplot(6,1,[1 4])
for pop=1:popSize
    scatter(pop1_x(pop,:),pop1_y(pop,:),25,'g','.')
    hold on
    scatter(pop2_x(pop,:),pop2_y(pop,:),25,'m','.')
    hold on
end
avg1_y = avg1_y./popSize
scatter(pop1_x(pop,:),avg1_y,100,'r','.')
avg2_y = avg2_y./popSize
scatter(pop2_x(pop,:),avg2_y,100,'b','.')
ylabel(sprintf('Objective\nfitness\nfor both populations'))
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
```

LISTING 5: Reproduce Figure 2 from Watson and Pollack (2001).

```
clear
% Evolution
lenString = 100;          % Bit string length
popSize = 25;             % Population size
mutateRate = 0.005;       % Mutation rate
genLimit = 600;           % Run fro this many generations
%  Init Pop  %
pop1     = zeros(lenString,popSize);          % Some inits and some places holder
pop2     = zeros(lenString,popSize);
popBuf1  = zeros(lenString,popSize);
popBuf2  = zeros(lenString,popSize);
for gen=1:genLimit
   % Pop 1
    total = 0;
    for member=1:popSize                                                % Acculumate
     fitness in wheel
       fitness(member) = f(pop1(:,member),pop2(:,randi(popSize)));                    %
      Every member against sample TODO: random(er) sample?
       total = total + fitness(member);
       wheel(member) = total;
    end
    for member=1:popSize
       pick = rand(1)*total;
       i = 1;
       while(wheel(i) < pick)                                           % Spin wheel
          i = i + 1;
       end
       popBuf1(:,member) = pop1(:,i);                                   % Update and
     mutate
       popBuf1(:,member) = Mutate(popBuf1(:,member),mutateRate);
    end
    sub1_y(gen) = total./(popSize);
    % Pop 2
    total = 0;                                                          % Repeat above
     but for second population
    for member=1:popSize
       fitness(member) = f(pop2(:,member),pop1(:,randi(popSize)));
       total = total + fitness(member);
       wheel(member) = total;
    end
    for member=1:popSize
       pick = rand(1)*total;
       i = 1;
       while(wheel(i) < pick)
          i = i + 1;
       end
       popBuf2(:,member) = pop2(:,i);
       popBuf2(:,member) = Mutate(popBuf2(:,member),mutateRate);
    end
    sub2_y(gen) = total./(popSize);
    % Population update
    pop1 = popBuf1;
    pop2 = popBuf2;
    % Record for graph
    X = sprintf('Generation = %d', gen);   % Text output
    disp(X)
    avg1_y(gen) = 0;
    avg2_y(gen) = 0;
    for pop=1:popSize
       pop1_x(pop,gen) = gen;
       pop1_y(pop,gen) = ObjFit(pop1(:,pop));
```

```
        avg1_y(gen) = avg1_y(gen) + pop1_y(pop,gen);
        pop2_x(pop,gen) = gen;
        pop2_y(pop,gen) = ObjFit(pop2(:,pop));
        avg2_y(gen) = avg2_y(gen) + pop2_y(pop,gen);
        sub1_x(gen) = gen;
        sub2_x(gen) = gen;
    end

end
% Graph
figure(1)
grid on;
subplot(6,1,6)
scatter(sub1_x,sub1_y,25,'r','.')
ylim([0 1])
xlabel('Generations')
ylabel(sprintf('Subjective\nfitness for\npopulation 1'))
grid on;
subplot(6,1,5)
scatter(sub2_x,sub2_y,25,'b','.')
ylim([0 1])
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
ylabel(sprintf('Subjective\nfitness for\npopulation 2'))
subplot(6,1,[1 4])
for pop=1:popSize
    scatter(pop1_x(pop,:),pop1_y(pop,:),25,'g','.')
    hold on
    scatter(pop2_x(pop,:),pop2_y(pop,:),25,'m','.')
    hold on
end
avg1_y = avg1_y./popSize
scatter(pop1_x(pop,:),avg1_y,100,'r','.')
avg2_y = avg2_y./popSize
scatter(pop2_x(pop,:),avg2_y,100,'b','.')
ylim([0 100])
ylabel(sprintf('Objective\nfitness\nfor both populations'))
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
```

Listing 6: Reproduce Figure 3 from Watson and Pollack (2001).

```
function[score] = Score(a,b)
   if(ObjFit(a) > ObjFit(b))                    % Scoring from paper
      score = 1;
   else
      score = 0;
   end
end
```

Listing 7: Scoring function.

```
function[fitness] = f(a,S)
   fitness = 0;
   tests = size(S);
   for test=1:tests(2)
      fitness = fitness + Score(a,S(:,test));
   end
end
```

Listing 8: Subjective fitness function.

```
function[mutant] = Mutate(normal,rate) % Any length bit string
   mutant = normal;                    % Init output as input
   for i=1:size(normal)                % Access each bit
      if(rand(1) <= rate)              % Supplies mutation rate
         if(mutant(i) == 0)            % Flip bits
            mutant(i) = 1;
         else
            mutant(i) = 0;
         end
      end
   end
end
```

Listing 9: Mutation function.

```
function[fitness] = ObjFit(bitString)
   fitness = sum(bitString);           % Just sum the vectors
end
```

Listing 10: Objective fitness function.

B.3 Focussing

```
clear
% Evolution
lenX = 10;                  % Bit string length
lenY = 10;                  % Bit string length - together X x Y matrix
popSize = 25;               % Population size
mutateRate = 0.005;         % Mutation rate
genLimit = 600;             % Run fro this many generations
%  Init Pop  %
pop1     = zeros(lenX,lenY,popSize);          % Some inits and some places holder
pop2     = zeros(lenX,lenY,popSize);
popBuf1  = zeros(lenX,lenY,popSize);
popBuf2  = zeros(lenX,lenY,popSize);
objFit1  = zeros(popSize);
objFit2  = zeros(popSize);
for gen=1:genLimit
   X = sprintf('Generation = %d', gen);   % Text output
   disp(X);
   % Pop 1
   total = 0;
   for member=1:popSize                                                % Acculumate
    fitness in wheel
      fitness(member) = f2(pop1(:,:,member),pop2);
      total = total + fitness(member);
      wheel(member) = total;
   end
   for member=1:popSize
      pick = rand(1)*total;
      i = 1;
      while(wheel(i) < pick)                                           % Spin wheel
         i = i + 1;
      end
      popBuf1(:,:,member) = pop1(:,:,i);
      for Y=1:lenY
         popBuf1(:,Y,member) = Mutate(popBuf1(:,Y,member),mutateRate);
      end
   end
   sub1_y(gen) = total./(popSize^2);
   % Pop 2
   total = 0;
   for member=1:popSize                                                % Acculumate
    fitness in wheel
      fitness(member) = f2(pop2(:,:,member),pop1);
      total = total + fitness(member);
      wheel(member) = total;
   end
   for member=1:popSize
      pick = rand(1)*total;
      i = 1;
      while(wheel(i) < pick)                                           % Spin wheel
         i = i + 1;
      end
      popBuf2(:,:,member) = pop2(:,:,i);
      for Y=1:lenY
         popBuf2(:,Y,member) = Mutate(popBuf2(:,Y,member),mutateRate);
      end
   end
   sub2_y(gen) = total./(popSize^2);
   % Update
   pop1 = popBuf1;
   pop2 = popBuf2;
```

```
    % Objective Fitness
    for member=1:popSize
        objFit1(member) = 0;
        objFit2(member) = 0;
        for Y=1:lenY              % Go through each dimension and find fitness of each
            objFit1(member) = objFit1(member) + ObjFit(pop1(:,Y,member));
            objFit2(member) = objFit2(member) + ObjFit(pop2(:,Y,member));
        end
    end
    % Record for graph
    avg1_y(gen) = 0;
    avg2_y(gen) = 0;
    for member=1:popSize
        pop1_x(member,gen) = gen;
        pop1_y(member,gen) = objFit1(member);
        avg1_y(gen) = avg1_y(gen) + pop1_y(member,gen);
        pop2_x(member,gen) = gen;
        pop2_y(member,gen) = objFit2(member);
        avg2_y(gen) = avg2_y(gen) + pop2_y(member,gen);
        sub1_x(gen) = gen;
        sub2_x(gen) = gen;
    end

end
% Graph
figure(1)
grid on;
subplot(6,1,6)
scatter(sub1_x,sub1_y,25,'r','.')
xlabel('Generations')
ylabel(sprintf('Subjective\nfitness for\npopulation 1'))
grid on;
subplot(6,1,5)
scatter(sub2_x,sub2_y,25,'b','.')
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
ylabel(sprintf('Subjective\nfitness for\npopulation 2'))
subplot(6,1,[1 4])
for pop=1:popSize
    scatter(pop1_x(pop,:),pop1_y(pop,:),25,'g','.')
    hold on
    scatter(pop2_x(pop,:),pop2_y(pop,:),25,'m','.')
    hold on
end
avg1_y = avg1_y./popSize
scatter(pop1_x(pop,:),avg1_y,100,'r','.')
avg2_y = avg2_y./popSize
scatter(pop2_x(pop,:),avg2_y,100,'b','.')
ylim([0 100])
ylabel(sprintf('Objective\nfitness\nfor both populations'))
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
```

LISTING 11: Reproduce Figure 4 from Watson and Pollack (2001).

```
function[score] = Score2(a,b)
    dim = size(a);
    x = dim(1);
    y = dim(2);
    playDim = 1;        % Dimension to play on
    diffDimMax = 0;
    for i=1:y
        diffDim = abs(sum(a(:,i)) - sum(b(:,i)));
        if(diffDim > diffDimMax)
            diffDimMax = diffDim;
            playDim = i;
        end
    end
    score = Score(a(:,playDim),b(:,playDim));
end
```

LISTING 12: Scoring function.

```
function[fitness] = f2(a,S)
    fitness = 0;
    tests = size(S);
    for test=1:tests(3)
        fitness = fitness + Score2(a,S(:,:,test));
    end
end
```

LISTING 13: Subjective fitness function.

```
function[mutant] = Mutate(normal,rate) % Any length bit string
    mutant = normal;                   % Init output as input
    for i=1:size(normal)               % Access each bit
        if(rand(1) <= rate)            % Supplies mutation rate
            if(mutant(i) == 0)         % Flip bits
                mutant(i) = 1;
            else
                mutant(i) = 0;
            end
        end
    end
end
```

LISTING 14: Mutation function.

```
function[fitness] = ObjFit(bitString)
    fitness = sum(bitString);          % Just sim the vectors
end
```

LISTING 15: Objective fitness function.

B.4 Relativism

```
clear
% Evolution
lenX = 10;                  % Bit string length
lenY = 10;                  % Bit string length - together X x Y matrix
popSize = 25;               % Population size
mutateRate = 0.005;         % Mutation rate
genLimit = 600;             % Run fro this many generations
%  Init Pop
pop1      = zeros(lenX,lenY,popSize);          % Some inits and some places holder
pop2      = zeros(lenX,lenY,popSize);
popBuf1   = zeros(lenX,lenY,popSize);
popBuf2   = zeros(lenX,lenY,popSize);
objFit1   = zeros(popSize);
objFit2   = zeros(popSize);
for gen=1:genLimit
   X = sprintf('Generation = %d', gen);    % Text output
   disp(X);
   % Pop 1
   total = 0;
   for member=1:popSize                                                  % Acculumate
    fitness in wheel
      fitness(member) = f3(pop1(:,:,member),pop2);
      total = total + fitness(member);
      wheel(member) = total;
   end
   for member=1:popSize
      pick = rand(1)*total;
      i = 1;
      while(wheel(i) < pick)                                             % Spin wheel
         i = i + 1;
      end
      popBuf1(:,:,member) = pop1(:,:,i);
      for Y=1:lenY
         popBuf1(:,Y,member) = Mutate(popBuf1(:,Y,member),mutateRate);
      end
   end
   sub1_y(gen) = total./(popSize^2);
   % Pop 2
   total = 0;
   for member=1:popSize                                                  % Acculumate
    fitness in wheel
      fitness(member) = f3(pop2(:,:,member),pop1);
      total = total + fitness(member);
      wheel(member) = total;
   end
   for member=1:popSize
      pick = rand(1)*total;
      i = 1;
      while(wheel(i) < pick)                                             % Spin wheel
         i = i + 1;
      end
      popBuf2(:,:,member) = pop2(:,:,i);
      for Y=1:lenY
         popBuf2(:,Y,member) = Mutate(popBuf2(:,Y,member),mutateRate);
      end
   end
   sub2_y(gen) = total./(popSize^2);
   % Update
   pop1 = popBuf1;
   pop2 = popBuf2;
```

```
    % Objective Fitness
    for member=1:popSize
        objFit1(member) = 0;
        objFit2(member) = 0;
        for Y=1:lenY              % Go through each dimension and find fitness of each
            objFit1(member) = objFit1(member) + ObjFit(pop1(:,Y,member));
            objFit2(member) = objFit2(member) + ObjFit(pop2(:,Y,member));
        end
    end
    % Record for graph
    avg1_y(gen) = 0;
    avg2_y(gen) = 0;
    for member=1:popSize
        pop1_x(member,gen) = gen;
        pop1_y(member,gen) = objFit1(member);
        avg1_y(gen) = avg1_y(gen) + pop1_y(member,gen);
        pop2_x(member,gen) = gen;
        pop2_y(member,gen) = objFit2(member);
        avg2_y(gen) = avg2_y(gen) + pop2_y(member,gen);
        sub1_x(gen) = gen;
        sub2_x(gen) = gen;
    end

end
% Graph %
figure(1)
grid on;
subplot(6,1,6)
scatter(sub1_x,sub1_y,25,'r','.')
xlabel('Generations')
ylabel(sprintf('Subjective\nfitness for\npopulation 1'))
grid on;
subplot(6,1,5)
scatter(sub2_x,sub2_y,25,'b','.')
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
ylabel(sprintf('Subjective\nfitness for\npopulation 2'))
subplot(6,1,[1 4])
for pop=1:popSize
    scatter(pop1_x(pop,:),pop1_y(pop,:),25,'g','.')
    hold on
    scatter(pop2_x(pop,:),pop2_y(pop,:),25,'m','.')
    hold on
end
avg1_y = avg1_y./popSize
scatter(pop1_x(pop,:),avg1_y,100,'r','.')
avg2_y = avg2_y./popSize
scatter(pop2_x(pop,:),avg2_y,100,'b','.')
ylim([0 100])
ylabel(sprintf('Objective\nfitness\nfor both populations'))
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
```

LISTING 16: Reproduce Figure 4 from Watson and Pollack (2001).

```
function[score] = Score2(a,b)
    dim = size(a);
    x = dim(1);
    y = dim(2);
    playDim = 1;        % Dimension to play on
    diffDimMin = x;
    for i=1:y
        diffDim = abs(sum(a(:,i)) - sum(b(:,i)));
        if(diffDim < diffDimMin)
            diffDimMin = diffDim;
            playDim = i;
        end
    end
    score = Score(a(:,playDim),b(:,playDim));
end
```

LISTING 17: Scoring function.

```
function[fitness] = f3(a,S)
    fitness = 0;
    tests = size(S);
    for test=1:tests(3)
        fitness = fitness + Score3(a,S(:,:,test));
    end
end
```

LISTING 18: Subjective fitness function.

```
function[mutant] = Mutate(normal,rate) % Any length bit string
    mutant = normal;                   % Init output as input
    for i=1:size(normal)               % Access each bit
        if(rand(1) <= rate)            % Supplies mutation rate
            if(mutant(i) == 0)         % Flip bits
                mutant(i) = 1;
            else
                mutant(i) = 0;
            end
        end
    end
end
```

LISTING 19: Mutation function.

```
function[fitness] = ObjFit(bitString)
    fitness = sum(bitString);          % Just sim the vectors
end
```

LISTING 20: Objective fitness function.

B.5 Extension

This section uses additional funtions but these are all included throughout the appendix.

```
clear
% Evolution
lenString = 100;          % Bit string length
popSize = 25;             % Population size
mutateRate = 0.005;       % Mutation rate
genLimit = 6000;           % Run fro this many generations
lambda=0.1
%  Init Pop  %
    pop1      = zeros(lenString,popSize);          % Some inits and some places holder
    pop2      = zeros(lenString,popSize);
    popBuf1   = zeros(lenString,popSize);
    popBuf2   = zeros(lenString,popSize);
    for gen=1:genLimit
       %  EA Stuff  %
       total = 0;
       for member=1:popSize                                                    % Acculumate
    fitness in wheel
           fitness(member) = f(pop1(:,member),pop2(:,randi(popSize)));
      % Every member against sample TODO: random(er) sample?
           total = total + fitness(member);
           wheel(member) = total;
       end
       sub1_y(gen) = total./(popSize);
       for member=1:popSize
           pick = rand(1)*total;
           i = 1;
           while(wheel(i) < pick)                                              % Spin wheel
               i = i + 1;
           end
           popBuf1(:,member) = pop1(:,i);                                      % Update and
    mutate
           popBuf1(:,member) = Mutate(popBuf1(:,member),(mutateRate*(lambda+((1-lambda)*
    sub1_y(gen)))));
       end
       total = 0;                                                              % Repeat above
     but for second population
       for member=1:popSize
           fitness(member) = f(pop2(:,member),pop1(:,randi(popSize)));
           total = total + fitness(member);
           wheel(member) = total;
       end
       sub2_y(gen) = total./(popSize);
       for member=1:popSize
           pick = rand(1)*total;
           i = 1;
           while(wheel(i) < pick)
               i = i + 1;
           end
           popBuf2(:,member) = pop2(:,i);
           popBuf2(:,member) = Mutate(popBuf2(:,member),(mutateRate*(lambda+((1-lambda)*
    sub2_y(gen)))));
       end
       pop1 = popBuf1;    % Population update
       pop2 = popBuf2;    % Population update
       % Record for graph %
       X = sprintf('Generation = %d', gen);    % Text output
       disp(X)
       avg1_y(gen) = 0;
```

```
        avg2_y(gen) = 0;
        for pop=1:popSize
            pop1_x(pop,gen) = gen;
            pop1_y(pop,gen) = ObjFit(pop1(:,pop));
            avg1_y(gen) = avg1_y(gen) + pop1_y(pop,gen);
            pop2_x(pop,gen) = gen;
            pop2_y(pop,gen) = ObjFit(pop2(:,pop));
            avg2_y(gen) = avg2_y(gen) + pop2_y(pop,gen);
            sub1_x(gen) = gen;
            sub2_x(gen) = gen;
        end

    end
    % Graph
figure(1)
ylabel(sprintf('Objective fitness'))
xlabel(sprintf('Generations'))
title(sprintf('Lambda = %1.1f',lambda))

grid on;
subplot(6,1,6)
scatter(sub1_x,sub1_y,25,'r','.')
ylim([0 1])
ylabel(sprintf('Subjective\nfitness for\npopulation 1'))
xlabel(sprintf('Generations'))
grid on;
subplot(6,1,5)
scatter(sub2_x,sub2_y,25,'b','.')
ylim([0 1])
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
ylabel(sprintf('Subjective\nfitness for\npopulation 2'))
subplot(6,1,[1 4])
for pop=1:popSize
    scatter(pop1_x(pop,:),pop1_y(pop,:),25,'g','.')
    hold on
    scatter(pop2_x(pop,:),pop2_y(pop,:),25,'m','.')
    hold on
end
avg1_y = avg1_y./popSize
scatter(pop1_x(pop,:),avg1_y,100,'r','.')
avg2_y = avg2_y./popSize
scatter(pop2_x(pop,:),avg2_y,100,'b','.')
ylim([0 100])
ylabel(sprintf('Objective\nfitness\nfor both populations'))
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
ylabel(sprintf('Objective fitness'))
title(sprintf('Lambda = %1.1f',lambda))
```

LISTING 21: 2000 generations testing adaptive mutation rate.

```
clear
% Evolution
lenX = 10;                  % Bit string length
lenY = 10;                  % Bit string length - together X x Y matrix
popSize = 25;               % Population size
mutateRate = 0.005;         % Mutation rate
genLimit = 2000;             % Run fro this many generations
lambda=0.5
%  Init Pop  %
pop1      = zeros(lenX,lenY,popSize);          % Some inits and some places holder
pop2      = zeros(lenX,lenY,popSize);
popBuf1   = zeros(lenX,lenY,popSize);
popBuf2   = zeros(lenX,lenY,popSize);
objFit1   = zeros(popSize);
objFit2   = zeros(popSize);
for gen=1:genLimit
   X = sprintf('Generation = %d', gen);    % Text output
   disp(X);
   % Pop 1
   total = 0;
   for member=1:popSize                                                  % Acculumate
    fitness in wheel
      fitness(member) = f2(pop1(:,:,member),pop2);
      total = total + fitness(member);
      wheel(member) = total;
   end
   sub1_y(gen) = total./(popSize^2);
   for member=1:popSize
      pick = rand(1)*total;
      i = 1;
      while(wheel(i) < pick)                                             % Spin wheel
         i = i + 1;
      end
      popBuf1(:,:,member) = pop1(:,:,i);
      for Y=1:lenY
         popBuf1(:,Y,member) = Mutate(popBuf1(:,Y,member),(mutateRate*(lambda+((1-
    lambda)*sub1_y(gen)))));
      end
   end
   % Pop 2
   total = 0;
   for member=1:popSize                                                  % Acculumate
    fitness in wheel
      fitness(member) = f2(pop2(:,:,member),pop1);
      total = total + fitness(member);
      wheel(member) = total;
   end
   sub2_y(gen) = total./(popSize^2);
   for member=1:popSize
      pick = rand(1)*total;
      i = 1;
      while(wheel(i) < pick)                                             % Spin wheel
         i = i + 1;
      end
      popBuf2(:,:,member) = pop2(:,:,i);
      for Y=1:lenY
         popBuf2(:,Y,member) = Mutate(popBuf2(:,Y,member),(mutateRate*(lambda+((1-
    lambda)*sub1_y(gen)))));
      end
   end
   % Update
```

```
    pop1 = popBuf1;
    pop2 = popBuf2;
    % Objective Fitness
    for member=1:popSize
        objFit1(member) = 0;
        objFit2(member) = 0;
        for Y=1:lenY            % Go through each dimension and find fitness of each
            objFit1(member) = objFit1(member) + ObjFit(pop1(:,Y,member));
            objFit2(member) = objFit2(member) + ObjFit(pop2(:,Y,member));
        end
    end
    % Record for graph
    avg1_y(gen) = 0;
    avg2_y(gen) = 0;
    for member=1:popSize
        pop1_x(member,gen) = gen;
        pop1_y(member,gen) = objFit1(member);
        avg1_y(gen) = avg1_y(gen) + pop1_y(member,gen);
        pop2_x(member,gen) = gen;
        pop2_y(member,gen) = objFit2(member);
        avg2_y(gen) = avg2_y(gen) + pop2_y(member,gen);
        sub1_x(gen) = gen;
        sub2_x(gen) = gen;
    end

end
% Graph
figure(2)
grid on;
subplot(6,1,6)
scatter(sub1_x,sub1_y,25,'r','.')
xlabel('Generations')
ylabel(sprintf('Subjective\nfitness for\npopulation 1'))
grid on;
subplot(6,1,5)
scatter(sub2_x,sub2_y,25,'b','.')
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
ylabel(sprintf('Subjective\nfitness for\npopulation 2'))
subplot(6,1,[1 4])
for pop=1:popSize
    scatter(pop1_x(pop,:),pop1_y(pop,:),25,'g','.')
    hold on
    scatter(pop2_x(pop,:),pop2_y(pop,:),25,'m','.')
    hold on
end
avg1_y = avg1_y./popSize
scatter(pop1_x(pop,:),avg1_y,100,'r','.')
avg2_y = avg2_y./popSize
scatter(pop2_x(pop,:),avg2_y,100,'b','.')
ylim([0 100])
ylabel(sprintf('Objective\nfitness\nfor both populations'))
set(gca,'xtick',[0:100:genLimit], 'xticklabel',{})
grid on;
```

LISTING 22: 2000 generations testing adaptive mutation rate with ten dimensional individuals.