```
%Matlab file. Insert values at ###
%Note that for learnSens (application sensitivity) and selesctionSens
(strength of selection) should accept a transformation of the values that
appear in the paper due to the way calculation was made in the simulation.
Here 3.3 is replaced with 0.1.

popsz=100; %population size
generations=5000;%number of generations
runs=100; %number of repeats of the simulation
selectionSens=0.1; %sensitivity of proportional selection (1 is exponential,
e is insensitive, completely random. larger values flip the direction of the
transformation)
mutrate=0.1; %mutation rate
LGene=(0:0.1:1); %Learning gene, the probability of individual learning
complemented by social
LGeneProb=repmat(1/length(LGene),1,length(LGene)); %probability of each
allele in the Lgene
InnovIntGene=(0:0.1:1); %Innovation intensity gene: the intensity is the
standard deviation from the mean of all behaviors
InnovIntGeneProb=repmat(1/length(InnovIntGene),1,length(InnovIntGene));
%probability of each allele in the InnovIntGene
SEGene=[0,1,2]; %social enhancement gene. 0: no effect of copying high
magnitude innovation; 1: probability of innovating on top of copied behavior
is 1; 2: probability is squre root of allele in InnovIntGene
SEGeneProb=repmat(1/length(SEGene),1,length(SEGene)); %probability of each
allele in SEGene
LT=10; %time steps = learning opportunities
demonstratorFrac=###; %fraction of innovators to be copied by social learners
learnWeight=0.1; %fraction of time dedicated to learning: detrmines how much
weight payoffs from learning get in the total payoff
learnSens=0.1; %learning sensitivity, the factor by which e is devided when
calculating the weight of each acquired behavior. when set to e, there's no
sensitivity
meanPayoff=0; %set the mean payoff for behaviors distribution, respective to
innovativeness intensity alleles
highMagnitude=1; %threshold for innovation to be considered high magnitude
(currently set to represent 1 standard devitation from the mean in the
positive direction)
SLpenalty=0; %social learning penalty: percentage deducted from value of
socially learned behaviors
geneNum=3;
popgen=zeros(popsz,geneNum); %the current population's genetic data
trackpopgen=zeros(popsz,geneNum*generations,runs); %save genetic data through
generations
trackmeanpayoff=zeros(runs,generations); %track how the mean top behavior
changes across generations

for r=1:runs

popgen=[randsample(LGene,popsz,true,LGeneProb)',randsample(InnovIntGene,popsz
,true,InnovIntGeneProb)',randsample(SEGene,popsz,true,SEGeneProb)'];
%generate agents genetics
    for g=1:generations
        trackpopgen(:,g*geneNum-geneNum+1:g*geneNum,r)=popgen; %store
population's genetics
```

```
        popknow=repmat(meanPayoff,popsz,LT); %each agent's knowledge/
repertior of behaviors. used both for social learning and for estimating
fitness
        innovProb=popgen(:,1); %copy genetics of innovation frequency gene
(LGene)
        inspired=[];
        for t=1:LT %learning time steps
            Innovators=binornd(1,innovProb); %find who is innovating and who
is a social learner

popknow(Innovators==1,t)=normrnd(repmat(meanPayoff,sum(Innovators),
1),popgen(Innovators==1,2)); %generate new behaviors based on individuals'
innovation intensity
            if isempty(inspired)==0 %if there are inspired copiers
                ifinspired=Innovators(inspired); %find if inspired copiers
innovated or not (list of 1 and 0)
                enhanced=inspired(find(ifinspired));%list of innovative
copiers
                popknow(enhanced,t)=popknow(enhanced,t)
+popknow(enhanced,t-1); %use new innovation by former copier to change copied
behavior
            end
            [sortedpopknow,innovRank]=sort(popknow(:,t),'descend'); %sort
population according to the knowledge in this step
            filterInnov=innovRank(Innovators(innovRank)==1); %create a rank
of innovators only
            roleModels=filterInnov(1:round(sum(Innovators)*demonstratorFrac),
1); %determine who are the top innovators to be copied
            if isempty(roleModels) %if the population of innovators is too
small relative to the demonstrators fraction
                if length(filterInnov)>0
                    roleModels=filterInnov(1);  %if there is at least one
innovator make sure there's at least one innovator
                elseif isempty(filterInnov)
                roleModels=innovRank(1); %if there are no innovators pick a
random individual
                end
            end
            if length(roleModels)<2
                copyfrom=repmat(roleModels,popsz-sum(Innovators),1); %if
there's only one innovator then generate a list with only this innovator
            else
                copyfrom=randsample(roleModels,popsz-sum(Innovators),true);
%generate a list of innovators to copy from the length of social learners
            end
            popknow(Innovators==0,t)=popknow(copyfrom,t)-
SLpenalty*abs(popknow(copyfrom,t)); %copy the best innovation into the
knowledge of social learners
            innovProb=popgen(:,1); %copy genetics of innovation frequency
gene
            copiers=find(Innovators==0); %find all copiers in this turn
            inspired=copiers(popknow(copiers,t)>highMagnitude); %find the
individuals who copied high magnitude innovations
            propSE=inspired(find(popgen(inspired,3)==2)); %find inspired
copiers with proportional enhancement allele
```

```
            fullSE=inspired(find(popgen(inspired,3)==1)); %find inspired
copiers with full enhancement allele
            innovProb(propSE,1)=sqrt(popgen(propSE,1));
            innovProb(fullSE,1)=1;
        end

        %total payoff calculations
        learnPayoff=sum(popknow,2); %summarize payoffs of all behaviors
acquired by each agent
        sumPayoffExp=sum((exp(1)/learnSens).^popknow,2); %summarize exponents
to calculate the proportion of time each behavior is used
        sumPayoffExpRep=repmat(sumPayoffExp,1,LT); %a matrix of repeats of
the exponents of cumulative payoff to be used to calculate weighted payoff
        weightedPayoff=(((exp(1)/learnSens).^(popknow))./
sumPayoffExpRep).*popknow; %calculate weighted payoff: exponent of each
behavior relative to the sum of all exponents, times the value of the
behavior itself
        weightedPayoffSum=sum(weightedPayoff,2); %sum weighted payoffs per
agent
        totalPayoff=learnWeight*learnPayoff+(1-
learnWeight)*weightedPayoffSum; %sum learning payoff and weighted payoff by
ratio of learning and application during life

        %selection and reproduction
        %proportional selection
        totalPayoffDeducted=totalPayoff-max(totalPayoff); %deduct maximum
payoff to allow exponent calculations for large numbers
        totalPayoffExp=(exp(1)/selectionSens).^totalPayoffDeducted;
%transform total payoffs into positive scale
        sumTotalPayoffExp=repmat(sum(totalPayoffExp),popsz,1); %find the
denominator to calculate reproduction probabilities
        reproProb=totalPayoffExp./sumTotalPayoffExp; %calculate reproduction
probabilty for each agent
        reproProbVector=cumsum(reproProb); %generate a scale for to draw
offspring
        newpop=zeros(popsz,geneNum); %open matrix for next generation
        parentsList=zeros(popsz,1); %open array to create list of parents
        for offspring=1:popsz
            randSpring=rand; %draw a random number betweeb 0 and 1
            parent=find(reproProbVector>randSpring,1,'first'); %find the
first agent who has a larger value in the cumulative probabilities vector
            newpop(offspring,:)=popgen(parent,:); %copy the genetic data of
the parent
            parentsList(parent,1)=1; %update that this agent reproduced
        end
        popgen=newpop;
        trackmeanpayoff(r,g)=mean(max(popknow(find(parentsList),:),[],2));
%store mean payoff

        %mutation
        ismut1=binornd(1,mutrate);
        if ismut1
            mutant1=randsample(popsz,1); %randomly pick an agent
            popgen(mutant1,1)=randsample(LGene,1); %draw an allele from the
pool
```

```
        end
        ismut2=binornd(1,mutrate);
        if ismut2
            mutant2=randsample(popsz,1); %randomly pick an agent
            popgen(mutant2,2)=randsample(InnovIntGene,1); %draw an allele
from the pool
        end
        ismut3=binornd(1,mutrate);
        if ismut3
            mutant3=randsample(popsz,1); %randomly pick an agent
            popgen(mutant3,3)=randsample(SEGene,1); %draw an allele from the
pool
        end
    end
end
```