// MagnumMult.cpp : Project for "A Computational Model of Industry Dynamics"

// This code generates and collects data from multiple (500) runs of the model

//

//

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*code written by: Myong-Hun Chang \*

//\* \*

//\*This code is intended for private research and not for public \*

//\*use. As such, the code is specifically designed for the narrow \*

//\*set of research questions and not for mass consumption. \*

//\*The code has few comments, no documentation, brittle user \*

//\*interfaces, minimal error-trapping and hardware specific I/O. \*

//\*The author will provide no support, no help debugging, and \*

//\*no evaluation of model output. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Completion Date: July 1, 2014

// NOTE

//

// The fluctuation in the decision environment for firms can occur in

// three different ways.

//

// 1. Change in the technological environment: This is controlled by

// two variables in the code, TRT and TRB. The rate of change in the

// technological environment is specified as (TRT/TRB). The "TRB" is

// set at 10,000 throughout the project. The rate of change is then

// controlled by appropriately choosing the value of TRT.

//

// 2. Change in the demand intercept, "a": This is controlled by

// PA and PAX, where the probability of a change in the demand

// intercept is specified as (PA/PAX). The project assumes the

// demand intercept to be fixed over time. As such, we set PA=0

// and PAX=1000 for all cases.

//

// 3. Change in the size of the market, "s": The fluctuation in market

// size, s, can take one of the two following modes: 1) Deterministic

// mode, where "s" follows a sine wave with its mean of S, amplitude

// of DS, and half-cycle of HP periods; 2) stochastic mode with a

// persistence parameter, PS. The deterministic mode is chosen by

// setting SM=0, while the stochastic mode is chosen by setting SM=1.

#include "stdafx.h"

#include <iostream>

#include <fstream>

#include <iomanip>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#define L 3 // total no. of bits = 32\*L (on a 32-bit machine)

#define LSTR 32 // length of a single bit-string

#define LDIM 1 // bit-length of a single task

#define S 4 // mean market size

#define DS 2 // market size fluctuates within [S-DS, S+DS]

#define SM 1 // mode of market fluctuation (sMode)

// SM=0: Deterministic fluctuation where "s" follows a sine wave

// with its mean at S, amplitude of DS, and half-cycle

// of HP periods.

// SM=1: Stochastic fluctuation where "s" takes a random value

// from [S-DS, S+DS] with probability, PS/PSX; Otherwise,

// "s" remains at S. PS needs to be specified only if SM=1.

#define PS 0.95 // probability that market size remains at the (t-1) level

#define RS 2 // range of the noise on s (market size): {-1/RS,+1/RS}

#define TS 6001 // start period for market fluctuation

// if TS>=T, then the market size stays constant at S

#define PI 3.14159 // pi for the sine function

#define HP 500 // duration of the half-cycle for market size

#define A 300 // mean demand intercept

#define DA 50 // demand intercept, a, fluctuates within [A-DA, A+DA]

#define PA 0 // probability that demand changes from t-1 to t = PA/PAX

#define PAX 1000 // ==> degree of demand turbulence

#define H 8 // serial tightness of the technology optimum

// hamming distance from the old tech. optimum of which

// a new tech. optimum must remain

#define TRT 1000 // probability that the tech. environment changes = TRT/TRB

#define TRB 10000 // ==> degree of technological turbulence

#define B 0.0 // start-up budget for each firm

#define T 5000 // length of the horizon

#define SB 0 // start of the historical data collection

#define SL 3001 // start of the leadership duration data collection

#define R 40 // no. of potential entrants

#define M 500 // total population size

#define FL 200 // fixed cost: floor value

#define FH 200 // fixed cost: ceiling value

#define I 0.0 // inertia (threshold wealth below which a firm exits

#define CN 100 // fixed cost of innovation

#define CM 50 // fixed cost of imitation

#define P 10000 // scaling factor for roulette wheel algorithm

#define AN 10 // A\_t for the new entrants

#define ABN 10 // A\_bar for the new entrants

#define BN 10 // B\_t for the new entrants

#define BBN 10 // B\_bar for the new entrants

// functions

double square(double y); // squares a number

void randBits(unsigned long s[], int numGroup); // generate a string of random bits

int sumBits(unsigned long b[], int numGroup); // sum all 1-bits in a string

int hamDist(unsigned long b1[], unsigned long b2[],

unsigned long xR[], int numGroup); // hamming distance between 2 strings

void trialBits(unsigned long inN[], unsigned long ouT[], unsigned int m1,

unsigned int m2, int numGroup, int lenGroup, int lenDim);

// take a bit string, randomly flip an adjacent substring

// of length lenDim, and return the resulting string

double bico(int n, int k); // binomial coefficient

double factln(int n);

double gammln(double xx);

void setBitsOne(unsigned long s[], int numGroup); // set all bits to 1

void flipOwnBit(unsigned long opT[], unsigned int m1, unsigned int m2,

int lenGroup); // flip one bit in a given bit-string

void displayBits(unsigned long value);

void observeBits(unsigned long iN[], unsigned long ouT[], unsigned long target[],

unsigned m1, unsigned m2, int numGroup, int lenGroup, int lenDim);

// observe the bit string of the target for imitation

int main()

{

int i, j, k, t, h, tid;

int length;

double mktSize; // market size in t

double mktSizeOld; // market size in t-1

int sMode; // market flux mode

unsigned long gPrev[L]; // tech. optimum in t-1

unsigned long gTrial[L]; // trial tech. optimum

unsigned long gCurr[L]; // tech. optimum in t

unsigned long xRx[L]; // bit-sequence for measuing hamming distance

int bitPosition[L\*LSTR];

unsigned d1, d2; // random position of a bit

unsigned b1, b2; // random position of a bit

double pHmass[H];

double cHmass[H];

double aPrev; // demand intercept in t-1

double aCurr; // demand intercept in t

int indic;

int rnd;

double rndScaled;

int hamm;

int yn;

int t\_div;

int divCount;

unsigned long techIndex[M][L]; // operating firms' technologies

double maxq; // maximum firm output for leadership determination

double minCost;

int interior;

double c\_hat;

int num\_active;

double max\_cost;

int down\_cand; // candidate incumbent for de-activation

double p\_trial; // trial market price

double p\_star; // equilibrium market price

int num1;

int num2;

int num3;

int numX;

double sumQ;

double cSurplus; // consumer surplus

double aggProfit; // aggregate profit

double c\_rd; // R&D expenditure

double sumposPi; // sum of positive profits used for roulette

// wheel algorithm

int wheelPi; // randomly chosen value on a roulette wheel

// from {1,...,P}

int rvl; // a rival chosen for imitation

double kPi; // sumposPi with self-imitation eliminated

double sumW; // markers the roulette wheel

double hhi; // herfindahl-hirschmann index

double pcm; // weighted industry price-cost margin

double wmc; // weighted industry marginal cost

double tpn; // aggregate probability of innovation (excl. new entrants)

double tpm; // aggregate probability of imitation (excl. new entrants)

double cpn; // conditional probability of innovation

double cpm; // conditional probability of imitation

double tcn; // aggregate expenditure on innovation

double tcm; // aggregate expenditure on imitation

int shift; // no. periods since the last technological shock

int countLead; // counter for leadership changes along the steady state

int countX; // counter for exits along the steady state

struct Firm {

short status; // firm's activity status

// 0: outsider

// 1: potential entrant

// 2: inactive incumbent

// 3: active incumbent

double budget; // firm's budget

unsigned long zCurr[L]; // firm's technology in t

unsigned long zPrev[L]; // firm's technology in t-1

unsigned long zTrial[L]; // firm's trial technology

double cExp; // firm's expected marginal cost

double cAct; // firm's actual marginal cost

double qTrial; // firm's trial output

double piTrial; // firm's trial profit

short tempStatus; // firm's temporary status

double q; // firm's equilibrium output

int age; // firm's age

double profit; // firm's equilibrium profit

double mktshr; // firm's market share

int prevDur; // firm's leadership duration in t-1

int currDur; // firm's leadership duration in t

int lead; // firm's leadership status (1=leader; 0=follower)

int A\_t; // attraction to R&D

int A\_bar; // attraction to status quo (No R&D)

int B\_t; // attraction to innovation

int B\_bar; // attraction to imitation

double alpha; // endogenous probability for R&D

double beta; // endogenous probability for innovation

int rd; // indicator: 1 = R&D; 0 = No R&D

int nm; // indicator: 1 = innovation; 0 = imitation

int adopt; // indicator: 1 = adopt; 0 = discard

double piPrev; // firm's profit in t-1

int obsX; // indicator: 1 = observed; 0 = unobserved

double fc; // firm-specific fixed cost

};

Firm m[M];

// open the output data files

std::ofstream fnum1out ("c:/MagnumData/sFixed/RD/mp000/num1.dat", std::ios::out);

if (!fnum1out) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fnum2out ("c:/MagnumData/sFixed/RD/mp000/num2.dat", std::ios::out);

if (!fnum2out) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fnum3out ("c:/MagnumData/sFixed/RD/mp000/num3.dat", std::ios::out);

if (!fnum3out) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fnumxout ("c:/MagnumData/sFixed/RD/mp000/numX.dat", std::ios::out);

if (!fnumxout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fnumDisTechout ("c:/MagnumData/sFixed/RD/mp000/disTech.dat", std::ios::out);

if (!fnumDisTechout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fdurationout ("c:/MagnumData/sFixed/RD/mp000/leader.dat", std::ios::out);

if (!fdurationout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fpout ("c:/MagnumData/sFixed/RD/mp000/p.dat", std::ios::out);

if (!fpout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fqout ("c:/MagnumData/sFixed/RD/mp000/q.dat", std::ios::out);

if (!fqout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fsumqout ("c:/MagnumData/sFixed/RD/mp000/sumq.dat", std::ios::out);

if (!fsumqout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream faout ("c:/MagnumData/sFixed/RD/mp000/a.dat", std::ios::out);

if (!faout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fsout ("c:/MagnumData/sFixed/RD/mp000/s.dat", std::ios::out);

if (!fsout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fxageout ("c:/MagnumData/sFixed/RD/mp000/XAGE.dat", std::ios::out);

if (!fxageout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fsxageout ("c:/MagnumData/sFixed/RD/mp000/SXAGE.dat", std::ios::out);

if (!fsxageout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fnleadout ("c:/MagnumData/sFixed/RD/mp000/NLEAD.dat", std::ios::out);

if (!fnleadout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fnxout ("c:/MagnumData/sFixed/RD/mp000/NEXIT.dat", std::ios::out);

if (!fnxout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fmcout ("c:/MagnumData/sFixed/RD/mp000/MC.dat", std::ios::out);

if (!fmcout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fprofitout ("c:/MagnumData/sFixed/RD/mp000/profit.dat", std::ios::out);

if (!fprofitout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fidout ("c:/MagnumData/sFixed/RD/mp000/ID.dat", std::ios::out);

if (!fidout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fsurvmcout ("c:/MagnumData/sFixed/RD/mp000/survMC.dat", std::ios::out);

if (!fsurvmcout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fsurvpiout ("c:/MagnumData/sFixed/RD/mp000/survPI.dat", std::ios::out);

if (!fsurvpiout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fsurvqout ("c:/MagnumData/sFixed/RD/mp000/survQ.dat", std::ios::out);

if (!fsurvqout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fsurvidout ("c:/MagnumData/sFixed/RD/mp000/survID.dat", std::ios::out);

if (!fsurvidout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fcsout ("c:/MagnumData/sFixed/RD/mp000/cSurplus.dat", std::ios::out);

if (!fcsout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fsumPiout ("c:/MagnumData/sFixed/RD/mp000/sumPi.dat", std::ios::out);

if (!fsumPiout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream falphaout ("c:/MagnumData/sFixed/RD/mp000/alpha.dat", std::ios::out);

if (!falphaout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fbetaout ("c:/MagnumData/sFixed/RD/mp000/beta.dat", std::ios::out);

if (!fbetaout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fageout ("c:/MagnumData/sFixed/RD/mp000/age.dat", std::ios::out);

if (!fageout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream ftpnout ("c:/MagnumData/sFixed/RD/mp000/tpn.dat", std::ios::out);

if (!ftpnout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream ftpmout ("c:/MagnumData/sFixed/RD/mp000/tpm.dat", std::ios::out);

if (!ftpmout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fcpnout ("c:/MagnumData/sFixed/RD/mp000/cpn.dat", std::ios::out);

if (!fcpnout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fcpmout ("c:/MagnumData/sFixed/RD/mp000/cpm.dat", std::ios::out);

if (!fcpmout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream ftcnout ("c:/MagnumData/sFixed/RD/mp000/tcn.dat", std::ios::out);

if (!ftcnout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream ftcmout ("c:/MagnumData/sFixed/RD/mp000/tcm.dat", std::ios::out);

if (!ftcmout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fshiftout ("c:/MagnumData/sFixed/RD/mp000/shift.dat", std::ios::out);

if (!fshiftout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fhhiout ("c:/MagnumData/sFixed/RD/mp000/hhi.dat", std::ios::out);

if (!fhhiout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fpcmout ("c:/MagnumData/sFixed/RD/mp000/pcm.dat", std::ios::out);

if (!fpcmout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream fwmcout ("c:/MagnumData/sFixed/RD/mp000/wmc.dat", std::ios::out);

if (!fwmcout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

std::ofstream ffcout ("c:/MagnumData/sFixed/RD/mp000/fc.dat", std::ios::out);

if (!ffcout) {

std::cerr << "File could not be opened." << std::endl;

exit(1);

}

// Iterate

for (tid = 1; tid < 501; tid++) {

std::cout << "replication: " << tid << std::endl;

////////////////////////////////////////////////////////////////////////

//////////////////////// STAGE 0: INITIALIZE //////////////////////////

////////////////////////////////////////////////////////////////////////

// set the random seed

srand(time(NULL));

// construct probability densities

pHmass[0] = bico(L\*LSTR, 1);

cHmass[0] = pHmass[0];

for (i = 1; i < H; i++) {

pHmass[i] = bico(L\*LSTR, i+1);

cHmass[i] = cHmass[i-1] + pHmass[i];

}

// generate the initial technological optimum

setBitsOne(gPrev, L);

// set the mode for market fluctuation

sMode = SM;

mktSizeOld = S\*1.0;

// generate the initial demand intercept: demand starts out at A and then fluctuates

// within [A-DA, A+DA]: Changes from previous A value with probability, PA/PAX

// and stays at the previous A value with 1-(PA/PAX)

aPrev = A\*1.0;

// generate the initial technology vector for each firm

for (i = 0; i < M; i++)

randBits(m[i].zPrev, L);

// initialize the tech index vector for each firm

for (i = 0; i < M; i++)

setBitsOne(techIndex[i],L);

// set other firm attributes

for (i = 0; i < M; i++) {

m[i].status = 0; // set the firm status

m[i].budget = 0.0; // set the firm budget

m[i].cExp = 0.0;

m[i].cAct = 0.0;

m[i].qTrial = 0.0;

m[i].piTrial = 0.0;

m[i].tempStatus = 0;

m[i].q = 0.0;

m[i].age = 0;

m[i].profit = 0.0;

m[i].mktshr = 0.0;

m[i].fc = 0.0;

m[i].currDur = 0;

m[i].prevDur = 0;

m[i].A\_t = AN;

m[i].A\_bar = ABN;

m[i].B\_t = BN;

m[i].B\_bar = BBN;

m[i].alpha = (m[i].A\_t\*1.0)/((m[i].A\_t + m[i].A\_bar)\*1.0);

m[i].beta = (m[i].B\_t\*1.0)/((m[i].B\_t + m[i].B\_bar)\*1.0);

m[i].rd = 0;

m[i].nm = 0;

m[i].adopt = 0;

m[i].obsX = 0;

m[i].piPrev = 0.0;

}

shift = 0; // no. periods since the last tech. shock

countLead = 0; // counter for leadership changes along the steady state

countX = 0; // counter for exits along the steady state

////////////////////////////////////////////////////////////////////////

//////////////////////////////// EVOLVE ////////////////////////////////

////////////////////////////////////////////////////////////////////////

for (t = 0; t < T; t++) {

////////////////////////////////////////////////////////////////////////

/////////////////////// STAGE 0: FLUCTUATION //////////////////////////

////////////////////////////////////////////////////////////////////////

// Set the Market Size

if (t >= TS) {

if (sMode == 0) {

mktSize = (S\*1.0) + (DS\*sin((PI/(HP\*1.0))\*(t\*1.0)));

}

else {

mktSize = ((1.0-PS)\*S\*1.0) + (PS\*mktSizeOld\*1.0)

+ (2.0/(RS\*1.0))\*(((rand()\*1.0)/(RAND\_MAX\*1.0)) - 0.5);

if (mktSize < 0) {

mktSize = 0;

}

mktSizeOld = mktSize;

}

}

else

mktSize = S\*1.0;

///// demand intercept changes with probability, (PA/PAX) //////////////

///// optimal technology vector changes within H hamming distance //////

///// with probability, (TRT/TRB) //////////////////////////////////////

////////////////////////////////////////////////////////////////////////

// retain the initial environment for t = 0

if (t == 0) {

aCurr = aPrev;

for (i = 0; i < L; i++)

gCurr[i] = gPrev[i];

shift = shift + 1;

}

// shake the environment for t > 0

else {

// shake the demand intercept

if (1+(int)(PAX\*1.0\*rand()/(RAND\_MAX+1.0)) > PA)

aCurr = A\*1.0;

else

aCurr = ((A-DA)\*1.0) + (DA\*2.0\*((rand()\*1.0)/(RAND\_MAX\*1.0)));

// shake the technological optimum

if ((1+(int)(TRB\*rand()/(RAND\_MAX+1.0))) < TRT) {

indic = 0;

shift = 0; // new tech shock

while (indic == 0) {

rnd = 1 + (int)(1.0\*RAND\_MAX\*rand()/(RAND\_MAX + 1.0));

rndScaled = rnd\*cHmass[H-1]/(RAND\_MAX\*1.0);

i = 0;

while (rndScaled > cHmass[i])

i = i + 1;

hamm = i + 1;

for (i = 0; i < hamm; i++) { // pick random positions

b1 = 1 + (int)(L\*1.0\*rand()/(RAND\_MAX + 1.0));

b2 = 1 + (int)(LSTR\*1.0\*rand()/(RAND\_MAX + 1.0));

bitPosition[i] = (b1 - 1)\*LSTR + b2;

if (i > 0) { // confirm distinct positions

j = 0;

while ((j < i) && (bitPosition[i] != bitPosition[j]))

j = j + 1;

if (j < i)

i = i - 1;

}

}

for (i = 0; i < L; i++)

gTrial[i] = gPrev[i];

for (i = 0; i < hamm; i++) {

d1 = (bitPosition[i]/LSTR) + 1;

d2 = bitPosition[i]%LSTR;

flipOwnBit(gTrial, d1, d2, LSTR);

}

for (i = 0; i < L; i++)

gCurr[i] = gTrial[i];

indic = 1;

} // close the while-loop

} // close the if-loop

else {

for (i = 0; i < L; i++)

gCurr[i] = gPrev[i];

shift = shift + 1; // increment the no. periods since last tech shock

}

} // end the goal-shaking routine

////////////////////////////////////////////////////////////////////////

//////////////////////// STAGE 1: INITIALIZE //////////////////////////

////////////////////////////////////////////////////////////////////////

//////////////////// An entrant starts out at age 1 ////////////////////

// wake up the potential entrants (R) and initialize their attributes

h = 0;

k = 0;

while (h < R && k < M) {

if (m[k].status == 0) {

m[k].status = 1; // select R outsiders to be potential entrants

randBits(m[k].zCurr, L); // assign technology to the potential entrant

m[k].fc = (FL\*1.0) + ((FH-FL)\*1.0\*((rand()\*1.0)/(RAND\_MAX\*1.0)));// set firm-specific fixed cost

// compute MC based on old technology optimum

m[k].cExp = 100.0\*(hamDist(m[k].zCurr, gCurr, xRx, L))/(L\*LSTR\*1.0);

m[k].budget = B\*1.0; // endow them with the initial budget

m[k].age = 1; // start the age counter

h = h + 1;

}

k = k + 1;

}

// check against insufficient pool of potential entrants

if (k >= M)

std::cout << "Error: The population is too small!" << std::endl;

////////////////////////////////////////////////////////////////////////

///////////////////// STAGE 2: ENTRY DECISION /////////////////////////

////////////////////////////////////////////////////////////////////////

// identifying the most efficient "inactive" firm

minCost = 100.0;

for (k = 0; k < M; k++) {

if (m[k].status == 2) {

if (m[k].cAct < minCost)

minCost = m[k].cAct;

}

}

// If the potential entrant is less efficient than the most efficient inactive firm, then

// he will be inactive upon entry. Otherwise, compute his expected output and profit

// based on his expected marginal cost and other "active" firms' actual marginal costs.

for (k = 0; k < M; k++) {

if (m[k].status == 1) {

if (m[k].cExp >= minCost) { // this firm is less efficient than the most efficient

// inactive firm

m[k].qTrial = 0.0;

m[k].piTrial = 0.0 - m[k].fc;

m[k].cAct = m[k].cExp;

}

else { // this firm is more efficient than the most efficient

// inactive firm

for (i = 0; i < M; i++) {

if (m[i].status == 3)

m[i].tempStatus = 1;

else

m[i].tempStatus = 0;

}

m[k].tempStatus = 1;

m[k].cAct = m[k].cExp;

// check for an interior equilibrium

// if none, de-activate the least efficient one

interior = 0;

while (interior == 0) {

c\_hat = 0.0;

num\_active = 0;

max\_cost = 0.0;

for (i = 0; i < M; i++) {

if (m[i].tempStatus == 1) {

c\_hat = c\_hat + m[i].cAct;

num\_active = num\_active + 1;

if (m[i].cAct > max\_cost) {

max\_cost = m[i].cAct;

down\_cand = i;

}

}

}

p\_trial = (aCurr + c\_hat)/((num\_active\*1.0) + 1.0);

interior = 1;

i = 0;

while ((i < M) && (interior == 1)) {

if (m[i].tempStatus == 1) {

m[i].qTrial = (p\_trial - m[i].cAct)\*(mktSize\*1.0);

if (m[i].qTrial < 0.0)

interior = 0;

}

i = i + 1;

} // close the while-loop

if (interior == 0) // if no equilibrium, the least efficient

m[down\_cand].tempStatus = 2; // firm is expected to shut down

} // close the while-loop

// check and see if k is one of the downed candidate

if (m[k].tempStatus == 2) {

m[k].qTrial = 0.0;

m[k].piTrial = 0.0 - m[k].fc;

}

else {

m[k].piTrial = (square(m[k].qTrial)/(mktSize\*1.0)) - m[k].fc;

}

} // close else

if (m[k].budget + m[k].piTrial <= 0.0) {// do not enter if the net wealth is

m[k].status = 0; // expected to be negative upon entry

m[k].age = 0; // switch his status to 0

m[k].cExp = 0.0;

m[k].cAct = 0.0;

m[k].fc = 0.0;

m[k].budget = 0.0;

}

m[k].tempStatus = 0;

m[k].piTrial = 0.0;

m[k].qTrial = 0.0;

} // close the if-loop

} // close the k-for-loop

// compute the no. of actual entrants

num1 = 0;

for (k = 0; k < M; k++) {

if (m[k].status == 1)

num1 = num1 + 1;

}

fnum1out << num1 << std::endl;

/////////////////////////////////////////////////////////////////

//////////////// STAGE 3: R&D BY INCUMBENTS ////////////////////

/////////////////////////////////////////////////////////////////

// activate the previously inactive incumbents

// all incumbents (but not the potential entrants) engage in innovation

for (i = 0; i < M; i++) {

if (m[i].status == 2)

m[i].status = 3;

}

// measure and output the aggregate R&D intensity

tpn = 0.0;

tpm = 0.0;

cpn = 0.0;

cpm = 0.0;

for (i = 0; i < M; i++){

if (m[i].status == 3) {

tpn = tpn + (m[i].alpha\*m[i].beta);

tpm = tpm + (m[i].alpha\*(1-m[i].beta));

cpn = cpn + m[i].beta;

cpm = cpm + (1-m[i].beta);

}

}

ftpnout << tpn << std::endl;

ftpmout << tpm << std::endl;

fcpnout << cpn << std::endl;

fcpmout << cpm << std::endl;

// let the incumbents (both previously inactive and active) decide on R&D

for (i = 0; i < M; i++) {

if (m[i].status == 3) {

// research with probability, alpha

if (1+(int)((m[i].A\_t + m[i].A\_bar)\*1.0\*rand()/(RAND\_MAX+1.0)) <= m[i].A\_t) {

m[i].rd = 1;

// R&D Subroutine

if (1 + (int)((m[i].B\_t + m[i].B\_bar)\*1.0\*rand()/(RAND\_MAX+1.0)) <= m[i].B\_t) {

m[i].nm = 1;

// Innovation Subroutine

d1 = 1 + (int)(L\*1.0\*rand()/(RAND\_MAX + 1.0));

d2 = 1 + (int)(((LSTR\*1.0)/(LDIM\*1.0))\*rand()/(RAND\_MAX + 1.0));

trialBits(m[i].zPrev, m[i].zTrial, d1, d2, L, LSTR, LDIM);

if (hamDist(gCurr, m[i].zTrial, xRx, L) < hamDist(gCurr, m[i].zPrev, xRx, L)) {

m[i].zCurr[d1-1] = m[i].zTrial[d1-1];

m[i].adopt = 1; // adopt the idea

}

else {

m[i].adopt = 0; // discard the idea

}

}

else {

m[i].nm = 0;

// Imitation Subroutine

// choose a rival to imitate, using a roulette wheel algorithm

wheelPi = 1 + (int)(P\*1.0\*rand()/(RAND\_MAX + 1.0));

if (m[i].obsX == 1)

kPi = sumposPi - m[i].profit;

else

kPi = sumposPi;

// carry out the imitation process only if there is someone to imitate

if (kPi > 0.0) {

sumW = 0.0;

for (j = 0; j < M; j++) {

if (j != i) {

if (m[j].obsX == 1) {

sumW = sumW + m[j].profit;

if ((wheelPi\*1.0) < (sumW/kPi)\*P\*1.0) {

rvl = j;

j = M; // terminate

}

}

}

}

// role the imitation dice

d1 = 1 + (int)(L\*1.0\*rand()/(RAND\_MAX + 1.0));

d2 = 1 + (int)(((LSTR\*1.0)/(LDIM\*1.0))\*rand()/(RAND\_MAX + 1.0));

observeBits(m[i].zPrev, m[i].zTrial, m[rvl].zPrev, d1, d2, L, LSTR, LDIM);

if (hamDist(gCurr, m[i].zTrial, xRx, L) < hamDist(gCurr, m[i].zPrev, xRx, L)) {

m[i].zCurr[d1-1] = m[i].zTrial[d1-1];

m[i].adopt = 1; // adopt the idea

}

else {

m[i].adopt = 0; // discard the idea

}

}

}

}

else {

m[i].rd = 0;

// No R&D

}

}

}

////////////////////////////////////////////////////////////////////////

////////////////// STAGE 4: COURNOT COMPETITION ///////////////////////

////////////////////////////////////////////////////////////////////////

//// all incumbents (previously active and inactive) plus the new //////

//// entrants engage in Cournot output competition: the equilibrium ///

//// is derived through an iterative process in which the least ////////

//// efficient firm is shut down until all firms produce non-negative //

//// quantities in equilibrium /////////////////////////////////////////

////////////////////////////////////////////////////////////////////////

// activate the new entrants

for (k = 0; k < M; k++) {

if (m[k].status == 1)

m[k].status = 3;

}

// compute the new marginal costs based on the new tech. optimum

for (k = 0; k < M; k++) {

if (m[k].status == 3)

m[k].cAct = 100.0\*(hamDist(m[k].zCurr, gCurr, xRx, L))/(L\*LSTR\*1.0);

else {

m[k].cAct = 0.0;

m[k].cExp = 0.0;

}

}

// check for an interior equilibrium solution

// if none, de-activate the least efficient active incumbent

interior = 0;

while (interior == 0) {

c\_hat = 0.0;

num\_active = 0;

max\_cost = 0.0;

for (k = 0; k < M; k++) {

if (m[k].status == 3) {

c\_hat = c\_hat + m[k].cAct;

num\_active = num\_active + 1;

if (m[k].cAct > max\_cost) {

max\_cost = m[k].cAct;

down\_cand = k;

}

}

}

p\_star = (aCurr + c\_hat)/((num\_active\*1.0) + 1.0);

interior = 1;

k = 0;

while ((k < M) && (interior == 1)) {

if (m[k].status == 3) {

m[k].q = (p\_star - m[k].cAct)\*(mktSize\*1.0);

if (m[k].q < 0.0)

interior = 0;

}

k = k + 1;

}

if (interior == 0) { // some firm is producing q < 0 in equilibrium

m[down\_cand].status = 2; // shut down the least efficient firm (down\_cand)

m[down\_cand].q = 0.0;

}

}

// compute the firms' profits and update their budget

sumQ = 0.0;

tcn = 0.0;

tcm = 0.0;

for (k = 0; k < M; k++) {

if (m[k].status >= 2) {

// assign the R&D expenditures

if ((m[k].rd == 1) && (m[k].nm == 1)) {

c\_rd = CN\*1.0;

tcn = tcn + CN\*1.0;

}

else if ((m[k].rd == 1) && (m[k].nm == 0)) {

c\_rd = CM\*1.0;

tcm = tcm + CM\*1.0;

}

else

c\_rd = 0.0;

// profit calculation and budget updating

m[k].profit = (square(m[k].q)/(mktSize\*1.0)) - m[k].fc - c\_rd;

m[k].budget = m[k].budget + m[k].profit;

sumQ = sumQ + m[k].q;

}

else

m[k].profit = 0.0;

}

// report the firm ages and their current endogenous probabilities given current attractions

for (k = 0; k < M; k++) {

if (m[k].status >= 2) {

if (t >= SB) {

fageout << m[k].age << std::endl;

falphaout << m[k].alpha << std::endl;

fbetaout << m[k].beta << std::endl;

ffcout << m[k].fc << std::endl;

}

}

}

// update the attraction levels for the next period

for (k = 0; k < M; k++) {

if (m[k].status >= 2) {

if ((m[k].rd == 1) && (m[k].nm == 1) && (m[k].adopt == 1)) {

m[k].A\_t = m[k].A\_t + 1;

m[k].B\_t = m[k].B\_t + 1;

}

else if ((m[k].rd == 1) && (m[k].nm == 1) && (m[k].adopt == 0)) {

m[k].A\_bar = m[k].A\_bar + 1;

m[k].B\_bar = m[k].B\_bar + 1;

}

else if ((m[k].rd == 1) && (m[k].nm == 0) && (m[k].adopt == 1)) {

m[k].A\_t = m[k].A\_t + 1;

m[k].B\_bar = m[k].B\_bar + 1;

}

else if ((m[k].rd == 1) && (m[k].nm == 0) && (m[k].adopt == 0)) {

m[k].A\_bar = m[k].A\_bar + 1;

m[k].B\_t = m[k].B\_t + 1;

}

}

// update endogenous choice probabilities

m[k].alpha = (m[k].A\_t\*1.0)/((m[k].A\_t + m[k].A\_bar)\*1.0);

m[k].beta = (m[k].B\_t\*1.0)/((m[k].B\_t + m[k].B\_bar)\*1.0);

}

// compute HHI, Price-Cost Margin, and Weighted Marginal Cost

hhi = 0.0;

pcm = 0.0;

wmc = 0.0;

for (k = 0; k < M; k++) {

if (m[k].status >= 2) {

hhi = hhi + ((m[k].q/sumQ)\*100.0)\*((m[k].q/sumQ)\*100.0);

pcm = pcm + ((p\_star - m[k].cAct)/p\_star)\*(m[k].q/sumQ);

wmc = wmc + (m[k].cAct)\*(m[k].q/sumQ);

}

}

// report the aggregate expenditures on innovation (tcn) and imitation (tcm)

ftcnout << tcn << std::endl;

ftcmout << tcm << std::endl;

// report HHI, PCM, and WMC

fhhiout << hhi << std::endl;

fpcmout << pcm << std::endl;

fwmcout << wmc << std::endl;

// compute consumer surplus and aggregate profit

cSurplus = 0.5\*(mktSize\*1.0)\*square(aCurr - p\_star);

aggProfit = 0.0;

for (k=0; k<M; k++) {

if (m[k].status >= 2)

aggProfit = aggProfit + m[k].profit;

}

// report market equilibrium (price and aggregate output)

fpout << p\_star << std::endl;

fsumqout << sumQ << std::endl;

// report welfare measures

fcsout << cSurplus << std::endl;

fsumPiout << aggProfit << std::endl;

// report firm-level data

for (k = 0; k < M; k++) {

if (m[k].status >= 2) {

if (t >= SB) {

fqout << m[k].q << std::endl;

fmcout << m[k].cAct << std::endl;

fprofitout << m[k].profit << std::endl;

fidout << m[k].status << std::endl;

}

}

}

// count the active and inactive incumbents

num2 = 0;

num3 = 0;

for (k = 0; k < M; k++) {

if (m[k].status == 2)

num2 = num2 + 1;

else if (m[k].status == 3)

num3 = num3 + 1;

}

// count and output the total number of operating firms

fnum2out << num2 << std::endl;

fnum3out << num3 << std::endl;

// compute the technological diversity index

yn = 0;

for (k = 0; k < M; k++) {

if (m[k].status >= 2) {

for (i = 0; i < L; i++)

techIndex[yn][i] = m[k].zCurr[i];

yn++;

}

}

if (yn != num2 + num3)

std::cout << "Error: No. of operating firms incorrect!" << std::endl;

t\_div = 0;

divCount = 0;

if (yn > 1) {

for (i = 0; i < yn - 1; i++) {

for (j = i + 1; j < yn; j++) {

t\_div = t\_div + hamDist(techIndex[i],techIndex[j],xRx,L);

divCount = divCount + 1;

}

}

}

else {

std::cout << "The market is empty or a monopoly!" << std::endl;

divCount = 1;

}

fnumDisTechout << (t\_div\*1.0)/(divCount\*1.0) << std::endl;

// calculate and report the leadership duration

for (k = 0; k < M; k++)

m[k].lead = 0;

maxq = 0.0;

for (k = 0; k < M; k++) {

if (m[k].status >= 2) {

if (m[k].q >= maxq)

maxq = m[k].q;

}

}

for (k = 0; k < M; k++) {

if (m[k].status >= 2) {

if (m[k].q == maxq) {

m[k].lead = 1;

m[k].currDur = m[k].prevDur + 1;

}

else

m[k].currDur = 0;

if (t - m[k].prevDur > SL) {

if ((m[k].prevDur > 0) && (m[k].currDur == 0))

{

fdurationout << m[k].prevDur << std::endl;

countLead = countLead + 1;

}

}

m[k].prevDur = m[k].currDur;

}

}

////////////////////////////////////////////////////////////////////////

///////////////////// STAGE 5: EXIT DECISION //////////////////////////

////////////////////////////////////////////////////////////////////////

// operating firms whose budget has fallen below $I exit the industry //

////////////////////////////////////////////////////////////////////////

// bankrupt firms exit the industry

numX = 0;

for (k = 0; k < M; k++) {

if (m[k].status >= 2) {

if (m[k].budget < I) {

// report the age of the firm at the time of exit

fxageout << m[k].age << std::endl;

// report the age of the firm at the time of exit if t>=3001

if (t >= SL-1)

{

fsxageout << m[k].age << std::endl;

countX = countX + 1; // raise the exit counter

}

m[k].status = 0;

m[k].budget = 0.0;

m[k].cExp = 0.0;

m[k].cAct = 0.0;

m[k].qTrial = 0.0;

m[k].piTrial = 0.0;

m[k].tempStatus = 0;

m[k].q = 0.0;

m[k].age = 0;

m[k].profit = 0.0;

m[k].mktshr = 0.0;

m[k].fc = 0.0;

m[k].A\_t = AN;

m[k].A\_bar = ABN;

m[k].B\_t = BN;

m[k].B\_bar = BBN;

m[k].alpha = (m[k].A\_t\*1.0)/((m[k].A\_t + m[k].A\_bar)\*1.0);

m[k].beta = (m[k].B\_t\*1.0)/((m[k].B\_t + m[k].B\_bar)\*1.0);

m[k].rd = 0;

m[k].nm = 0;

m[k].adopt = 0;

m[k].obsX = 0;

m[k].piPrev = 0.0;

numX = numX + 1;

}

}

}

fnumxout << numX << std::endl;

////////////////////////////////////////////////////////////////////////

////// STAGE 6: UPDATE THE TECHNOLOGIES, DEMAND, and AGES /////////////

////////////////////////////////////////////////////////////////////////

// update the technological optimum

for (i = 0; i < L; i++)

gPrev[i] = gCurr[i];

// update the market size

aPrev = aCurr;

// update the firm's technology and profit

for (k = 0; k < M; k++) {

if (m[k].status >= 2) {

for (i = 0; i < L; i++)

m[k].zPrev[i] = m[k].zCurr[i];

m[k].piPrev = m[k].profit;

}

}

// collect the data for the roulette wheel algorithm

sumposPi = 0.0;

for (k = 0; k < M; k++) {

if (m[k].profit > 0.0) {

m[k].obsX = 1;

sumposPi = sumposPi + m[k].profit;

}

else

m[k].obsX = 0;

}

// increment the incumbent firms' ages

for (k = 0; k < M; k++) {

if (m[k].status >= 2)

m[k].age = m[k].age + 1;

}

// report market size

fsout << mktSize << std::endl;

faout << aCurr << std::endl;

// report the marginal costs and profits of the survivors

for (k = 0; k < M; k++) {

if (m[k].status >= 2) {

fsurvmcout << m[k].cAct << std::endl;

fsurvpiout << m[k].profit << std::endl;

fsurvqout << m[k].q << std::endl;

fsurvidout << m[k].status << std::endl;

}

}

// report the no. periods since the last tech shock

fshiftout << shift << std::endl;

} // close the time loop

// report the no. leadership changes and the no. exits for the replication

fnleadout << countLead << std::endl;

fnxout << countX << std::endl;

} // end the replication (tid) iteration

// close the output files

fnum1out.close();

fnum2out.close();

fnum3out.close();

fnumxout.close();

fnumDisTechout.close();

fdurationout.close();

fpout.close();

fqout.close();

fsumqout.close();

faout.close();

fsout.close();

fxageout.close();

fsxageout.close();

fmcout.close();

fprofitout.close();

fidout.close();

fsurvmcout.close();

fsurvpiout.close();

fsurvqout.close();

fsurvidout.close();

fcsout.close();

fsumPiout.close();

falphaout.close();

fbetaout.close();

fageout.close();

ftpnout.close();

ftpmout.close();

fcpnout.close();

fcpmout.close();

ftcnout.close();

ftcmout.close();

fshiftout.close();

fhhiout.close();

fpcmout.close();

fwmcout.close();

ffcout.close();

fnleadout.close();

fnxout.close();

////////////////////////////////////////////////////////////////////////////////////////

// measure and report the processing time

length = clock()/CLOCKS\_PER\_SEC;

std::cout << std::endl << length << " seconds." << std::endl;

if (length < 60)

std::cout << "CPU Time: " << length << " seconds." << std::endl;

else if (length >= 60 && length < 3600)

std::cout << "CPU Time: " << length/60 << " minutes "

<< length % 60 << " seconds." << std::endl;

else

std::cout << "CPU Time: " << length/3600 << " hours "

<< (length % 3600)/60 << " minutes "

<< (length % 3600) % 60 << " seconds." << std::endl;

system("PAUSE");

return 0;

}

/////////////////////////////////////////////

// Functions //

/////////////////////////////////////////////

double square(double y)

{

return y\*y;

}

void randBits(unsigned long s[], int numGroup)

{

int j, k;

unsigned long randMask;

for (j = 0; j < numGroup; j++) {

s[j] = 0;

for (k = 0; k < 32; k++) {

randMask = 1 << k;

if ((1+(int)(2\*rand()/(RAND\_MAX+1.0))) == 1)

s[j] = s[j]^randMask;

}

}

}

int sumBits(unsigned long b[], int numGroup)

{

int j, k, sumTot, sum;

unsigned long sumMask;

sumTot = 0;

for (j = 0; j < numGroup; j++) {

sum = 0;

for (k = 0; k < 32; k++) {

sumMask = 1 << k;

sum = sum + ((b[j] & sumMask) >> k);

}

sumTot = sumTot + sum;

}

return sumTot;

}

int hamDist(unsigned long b1[], unsigned long b2[],

unsigned long xR[], int numGroup)

{

int j;

for (j = 0; j < numGroup; j++)

xR[j] = b1[j]^b2[j];

return sumBits(xR, numGroup);

}

void trialBits(unsigned long iN[], unsigned long ouT[],

unsigned m1, unsigned m2, int numGroup,

int lenGroup, int lenDim)

{

int j, k;

int bitLoc;

unsigned long trialMask;

for (j = 0; j < numGroup; j++)

ouT[j] = iN[j];

for (k = 0; k < lenDim; k++) {

bitLoc = (m2 - 1)\*lenDim + k + 1;

if ((1+(int)(1.0\*1000\*rand()/(RAND\_MAX+1.0)))

<= 500) { // flip the bit

trialMask = 1 << (lenGroup - bitLoc);

ouT[m1-1] = ouT[m1-1]^trialMask;

}

}

}

double bico(int n, int k)

{

double factln(int n);

return floor(0.5 + exp(factln(n)-factln(k)-factln(n-k)));

}

double factln(int n)

{

double gammln(double xx);

static double a[101];

if (n <= 1) return 0.0;

if (n <= 100) return a[n] ? a[n] : (a[n]=gammln(n+1.0));

else return gammln(n+1.0);

}

double gammln(double xx)

{

double x, y, tmp, ser;

static double cof[6]={76.18009172947146,-86.50532032941677,

24.01409824083091,-1.231739572450155,

0.1208650973866179e-2,-0.5395239384953e-5};

int j;

y = x = xx;

tmp = x+5.5;

tmp -= (x+0.5)\*log(tmp);

ser = 1.000000000190015;

for (j=0; j<=5; j++) ser += cof[j]/++y;

return -tmp+log(2.5066282746310005\*ser/x);

}

void setBitsOne(unsigned long s[], int numGroup)

{

int j, k;

unsigned long setMask;

for (j = 0; j < numGroup; j++) {

s[j] = 0;

for (k = 0; k < 32; k++) {

setMask = 1 << k;

s[j] = s[j]^setMask;

}

}

}

void flipOwnBit(unsigned long opT[], unsigned int m1, unsigned int m2,

int lenGroup)

{

unsigned long flipMask;

flipMask = 1 << (lenGroup - m2);

opT[m1 - 1] = opT[m1 - 1]^flipMask;

}

void displayBits(unsigned long value)

{

unsigned c;

unsigned long displayMask = 1 << 31;

std::cout << std::setw(7) << value << " = ";

for (c = 1; c <= 32; c++) {

std::cout << (value & displayMask ? '1' : '0');

value <<= 1;

if (c % 8 == 0)

std::cout << ' ';

}

std::cout << std::endl;

}

void observeBits(unsigned long iN[], unsigned long ouT[],

unsigned long target[], unsigned m1,

unsigned m2, int numGroup, int lenGroup,

int lenDim)

{

int j, k, bitLoc;

unsigned long trialMask;

for (j = 0; j < numGroup; j++)

ouT[j] = iN[j];

for (k = 0; k < lenDim; k++) {

bitLoc = (m2-1)\*lenDim + k + 1;

trialMask = 1 << (lenGroup-bitLoc);

if ((ouT[m1-1]&trialMask) != (target[m1-1]&trialMask))

{ // flip the bit

ouT[m1-1] = ouT[m1-1]^trialMask;

}

}

}