```
/**
 *                            "AgileSim_updated.java"
 *
 * This java program simulates demand, quantity and forecasting values for a choice
 * of safety factor, k = 0 & 0.84, as well as creating points which map the isovalue
 * line, overage and underage efficient frontiers using a sample of different values
 * for k.
 */

//The following imported class libraries are needed to execute a number of methods
//and procedures in the program.
//These libraries can be found in the API and Documentation on the Sun Developer
Network:
//http://java.sun.com/j2se/1.5.0/docs/api/.
import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class AgileSim_updated extends JFrame implements ActionListener{

      /**
       * Declaration of the objects used for the creation of the GUI,
       * as well as declaration of the variables used in the program.
       */

      JFrame frame;
      JPanel mainPanel, labelsPanel, lengthPanel,
                  dMeanPanel, kPanel, dSmoothPanel, typePanel,
                  expSmoothPanel, inputSelectPanel, runPanel;
      JTextField sampleSize, dMean, dSmooth, expSmooth;
      JComboBox safetyFactor, smoothType;
      JLabel sampleLabel, dMeanLabel, dSmoothLabel, expSmoothLabel,
                  kLabel, typeLabel;
      JButton runProgram;

      String[] kValues = {"Select a value", "0", "0.84"};
      String[] forecastMethods = {"Select a type", "Exponential smoothing",
                                  "Four point moving average"};
      String selectedkValue, selectedSmoothType;

      double R, C, M, Ve, V, cp, co1, cu1, co2, cu2;
      /**
       * R = Retailer's unit retail price,
       * M = Product's unit manufacturing cost,
       * Ve = Unit salvage cost obtainable from open market
       *          at the end of selling season,
       * cp = Unit contribution to profit,
       * co1 = Primal overage cost,
       * cu1 = Primal underage (goodwill) cost,
       * co2 = Dual overage (goodwill) cost,
       * cu2 = Dual underage cost.
       */

      //Constructor for the class "AgileSim_updated", which constructs the GUI.
      public AgileSim_updated(){

         //Create and set up the window for the GUI.
         frame = new JFrame("Agile Simulation");
         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    //Create the container panels for formatting the contents
    //of the GUI window.
    mainPanel = new JPanel(new BorderLayout());

    labelsPanel = new JPanel(new GridLayout(6, 1));
    labelsPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
    lengthPanel = new JPanel(new BorderLayout());
    dMeanPanel = new JPanel(new BorderLayout());
    kPanel = new JPanel(new BorderLayout());
    dSmoothPanel = new JPanel(new BorderLayout());
    typePanel = new JPanel(new BorderLayout());
    expSmoothPanel = new JPanel(new BorderLayout());
    inputSelectPanel = new JPanel(new GridLayout(6, 1));
    inputSelectPanel.setBorder(BorderFactory.createEmptyBorder(5,0,5,5));

    //Create the 'Run' button to execute the program.
    //An is action associated with this button so an 'Action Listener' is added.
    //For every subsequent object which has an action associated with them,
    //'Action Listeners' will be added.
    runProgram = new JButton("Run");
    runProgram.addActionListener(this);
    //Create the panel which contains the 'Run' button.
    runPanel = new JPanel(new BorderLayout());
    runPanel.setBorder(BorderFactory.createEmptyBorder(0,0,5,5));
    runPanel.add(runProgram, BorderLayout.PAGE_END);

    //Adds the contents of the panels (see class below).
    addContents();

    mainPanel.add(labelsPanel, BorderLayout.LINE_START);
    mainPanel.add(inputSelectPanel, BorderLayout.LINE_END);

    frame.getContentPane().add(mainPanel, BorderLayout.PAGE_START);
    frame.getContentPane().add(runPanel, BorderLayout.LINE_END);

    //Display the window.
    frame.pack();
    frame.setVisible(true);
}

//A 'void' method that performs some action other than returning a value,
//in this method contents are added to the container panels of the GUI
//and formatted.
private void addContents(){

    sampleLabel = new JLabel("Length of Life Cycle");
    sampleLabel.setHorizontalAlignment(SwingConstants.RIGHT);
    sampleSize = new JTextField(5);
    sampleSize.setHorizontalAlignment(SwingConstants.RIGHT);
    labelsPanel.add(sampleLabel);
    lengthPanel.add(sampleSize, BorderLayout.LINE_START);

    dMeanLabel = new JLabel("Initial Mean Demand");
    dMeanLabel.setHorizontalAlignment(SwingConstants.RIGHT);
    dMean = new JTextField(5);
    dMean.setHorizontalAlignment(SwingConstants.RIGHT);
    labelsPanel.add(dMeanLabel);
    dMeanPanel.add(dMean, BorderLayout.LINE_START);

    kLabel = new JLabel("Safety Factor");
```

```
   kLabel.setHorizontalAlignment(SwingConstants.RIGHT);
   safetyFactor = new JComboBox(kValues);
   safetyFactor.setSelectedIndex(0);
   //An action is associated with the selection of an item from this combo box.
   safetyFactor.addActionListener(this);
   labelsPanel.add(kLabel);
   kPanel.add(safetyFactor, BorderLayout.LINE_START);

   dSmoothLabel = new JLabel("Smoothing Factor for Predicted Demand");
   dSmoothLabel.setHorizontalAlignment(SwingConstants.RIGHT);
   dSmooth = new JTextField(5);
   dSmooth.setHorizontalAlignment(SwingConstants.RIGHT);
   labelsPanel.add(dSmoothLabel);
   dSmoothPanel.add(dSmooth, BorderLayout.LINE_START);

   typeLabel = new JLabel("Smoothing Type for Mu & Eta");
   typeLabel.setHorizontalAlignment(SwingConstants.RIGHT);
   smoothType = new JComboBox(forecastMethods);
   smoothType.setSelectedIndex(0);
   //An action is associated with the selection of an item from this combo box.
   smoothType.addActionListener(this);
   labelsPanel.add(typeLabel);
   typePanel.add(smoothType, BorderLayout.LINE_START);

   expSmoothLabel = new JLabel("Exponential Smoothing Factor");
   expSmoothLabel.setHorizontalAlignment(SwingConstants.RIGHT);
   expSmooth = new JTextField(5);
   expSmooth.setHorizontalAlignment(SwingConstants.RIGHT);
   expSmooth.setEditable(false);
   labelsPanel.add(expSmoothLabel);
   expSmoothPanel.add(expSmooth, BorderLayout.LINE_START);

   inputSelectPanel.add(lengthPanel);
   inputSelectPanel.add(dMeanPanel);
   inputSelectPanel.add(kPanel);
   inputSelectPanel.add(dSmoothPanel);
   inputSelectPanel.add(typePanel);
   inputSelectPanel.add(expSmoothPanel);

}

/**
 * The following methods are required in the calculations throughout the
 * executable part of the program (when the 'Run Program' button is clicked)
 */

//Method to return the Error Function used to calculate the
//cumulative density function for the standard normal distribution
public static double errorFunct(double x){
   double a = -(8*(Math.PI-3))/(3*Math.PI*(Math.PI-4));
   double numerator = (4/Math.PI)+a*Math.pow(x,2);
   double denominator = 1+a*Math.pow(x,2);
   double quotient = numerator/denominator;
   double exp = Math.exp(-Math.pow(x,2)*quotient);
   double erf = Math.sqrt(1-exp);
   return erf;
}

//Method to return the probability density function for the
//standard normal distribution
public static double NormalPDF(double x){
```

```
   double f = Math.sqrt(1/(2*Math.PI))*Math.exp(-0.5*Math.pow(x,2));
   return f;
}

//Method to return the cumluative density function for the
//standard normal distribution
public static double NormalCDF(double x){
   if(x>=0){
      double F = 0.5*(1+errorFunct(x/Math.sqrt(2)));
      return F;
   } else {
      double F = 0.5*(1-errorFunct(Math.abs(x)/Math.sqrt(2)));
      return F;
   }
}

//Method to return a suitable comparison function for the
//envelope rejection technique for generating a standard normal random variate
public static double ComparisonFunctPDF(double x){
   double g = 0.483641/(1+Math.pow(x,2));       //Cauchy distribution
   return g;
}

public static double Max(double[] A){
      double max = 0;
      for(int a = 0; a < A.length; a++){
         if(A[a]>max){
            max = A[a];
         }
      }
      return max;
}

public static int MaxIndex(double[] B){
   double max = 0;
   int index = 0;
   for(int b = 0; b < B.length; b++){
      if(B[b]>max){
         max = B[b];
         index = b;
      }
   }
   return index;
}

//The consequential results of the actions.
public void actionPerformed(ActionEvent event){

   //Values for cost parameters are set with the selection of a
   //value for 'Safety Factor'.
   if(event.getSource() == safetyFactor){
      JComboBox cb1 = (JComboBox)event.getSource();
      selectedkValue = (String)cb1.getSelectedItem();
      if(selectedkValue=="0"){
         R = 12;
         M = 6;
         Ve = 0;
         co1 = 5;
         cu2 = 1;
      } else if(selectedkValue=="0.84"){
         R = 12;
```

```
      C = 6;
      M = 4;
      Ve = 2;
      V = 4.5;
      co1 = C - V;
      cu2 = M - C + V - Ve;
   }
}

//The editable capability of the 'Exponential Smoothing Factor' text field
//in the GUI is determined by the selection of 'Smoothing Type for Mu & Eta'.
if(event.getSource() == smoothType){
   JComboBox cb2 = (JComboBox)event.getSource();
   selectedSmoothType = (String)cb2.getSelectedItem();
   if(selectedSmoothType=="Exponential smoothing"){
      expSmooth.setEditable(true);
   } else if(selectedSmoothType=="Four point moving average"){
      expSmooth.setText("");
      expSmooth.setEditable(false);
   }
}

else if(event.getSource() == runProgram){

   if(safetyFactor.getSelectedItem()=="Select a value"){
      JOptionPane.showMessageDialog(
                                 frame,
                                 "Please select value for safety factor",
                                 "Error",
                                 JOptionPane.INFORMATION_MESSAGE);
      return;

   } else if(smoothType.getSelectedItem()=="Select a type"){
      JOptionPane.showMessageDialog(
                                 frame,
                                 "Please select a smoothing type",
                                 "Error",
                                 JOptionPane.INFORMATION_MESSAGE);
      return;

   }

   if((sampleSize.getText().length()==0)||(dMean.getText().length()==0)
         ||(dSmooth.getText().length()==0)){

      JOptionPane.showMessageDialog(
                                 frame,
                                 "Please enter value(s) for missing”
                                 + “ paramter(s)",
                                 "Error (Missing Values)",
                                 JOptionPane.INFORMATION_MESSAGE);
      return;

   } else if((smoothType.getSelectedItem()=="Exponential smoothing")
               &&(expSmooth.getText().length()==0)){
      JOptionPane.showMessageDialog(
                                 frame,
                                 "Please enter value for missing paramter",
                                 "Error (Missing Values)",
                                 JOptionPane.INFORMATION_MESSAGE);
```

```
      return;

   } else {
      //The number of time points (changeable parameter)
      int SampleSize = Integer.parseInt(sampleSize.getText());
      //Normally (Standard) distributed random variable
      double[] X = new double[SampleSize];
      //Value of the cumulative density at the point of the random variable
      double[] P = new double[SampleSize];
      //Scaling for actual demand
      int[] QSTDnorm = new int[SampleSize];

      //Starting value for the expected demand
      double Dmean = Double.parseDouble(dMean.getText());
      //Scale parameter for the demand
      double DRandScale = 2;
      //Initial value for the safety factor in Brown's
      // allocation equation (1963)
      double SafetyFactor = Double.parseDouble(selectedkValue);
      //Exponential smoothing parameter for the predicted demand
      double Dsmooth = Double.parseDouble(dSmooth.getText());
      //Exponential smoothing parameter applied to Mu and Eta
      double ExpSmooth = 0;
      if(selectedSmoothType=="Exponential smoothing"){
         ExpSmooth = Double.parseDouble(expSmooth.getText());
      }

      //Arrays for the simulated values over the length of the life cycle of
      double[] Qhat = new double[SampleSize];      //forecasted quantity,
      double[] Dhat = new double[SampleSize];      //forecasted demand,
      double[] D = new double[SampleSize];         //actual demand,
      double[] Q = new double[SampleSize];         //actual quantity,
      double[] errorD = new double[SampleSize];    //demand forecasting error,
      double[] errorQ = new double[SampleSize];    //quantity forecasting error,
      double[] Mu = new double[SampleSize];        //expected difference in
                                                   //quantity and demand,
      double[] Eta = new double[SampleSize];       //expected volume of
                                                   //quantity and demand,
      double[] Error = new double[SampleSize];     //difference in
                                                   //forecasting errors,

      //standard deviation of forecasting errors
      double[] ErrorStdDev = new double[SampleSize];

      //Excel output
      try{
         FileWriter outputFile1 = new FileWriter("SimOutput.csv");
         PrintWriter out1 = new PrintWriter(
                                 new BufferedWriter(outputFile1));
         FileWriter outputFile2 = new FileWriter("EFoutput.csv");
         PrintWriter out2 = new PrintWriter(
                                 new BufferedWriter(outputFile2));
         if(selectedSmoothType=="Exponential smoothing"){
            out1.print("Expected volume (Eta) and difference (Mu)”
                           + “ exponentially smoothed; ");
            out1.println("Safety Factor (k) = " + SafetyFactor);
         } else if(selectedSmoothType=="Four point moving average"){
            out1.print("Expected volume (Eta) and difference (Mu) smoothed
                           + “over a four period moving average; ");
            out1.println("Safety Factor (k) = " + SafetyFactor);
         }
```

```
double sumD0 = 0;
double sumD = 0;
double MuD = 0;         //The mean demand
double sumE = 0;

//Excel output
out1.println(",");
out1.print(",,Forecast,,Actual,,Forecasting Errors,,Mix,Volume,”)
out1.print(“,Std Dev of Errors");
out1.println(",Period,Q(hat),D(hat),D,Q,Ed,Eq,Mu,Eta,E,SigmaE");

for(int s = 0; s < SampleSize; s++){
   X[s] = 0;
   //This loop repeats until a standard normal random variate
   //is generated using envelope rejection method
   while(X[s]==0){
      double R1 = Math.random();
      double X1 = Math.tan(Math.PI*(R1-0.5));
      if(Math.abs(X1)<=1){
         double R2 = Math.random();
         //Acceptance Rule
         double AcceptRatio1 = NormalPDF(X1)/ComparisonFunctPDF(X1);
         if(R2<=AcceptRatio1){
            X[s] = X1;
         }
      }
   }
   P[s] = NormalCDF(X[s]);

   if(s==0){
      //Initial values for
      Dhat[s] = Dmean;        //the forecasted demand,
      D[s] = Dmean;           //actual demand
      sumD0 += D[s];
      //Initial value for standard deviation of forecasting errors
      ErrorStdDev[s] = Math.sqrt(Dmean);
      //Initial value for the forecasted quantity using Brown's
      //allocation equation
      Qhat[s] = Dhat[s]+SafetyFactor*ErrorStdDev[s];
      QSTDnorm[s] = 1;
      //Initial value for the actual quantity using the cumulative
      //density from a standard normal randomly generated variate
      Q[s] = Qhat[s] + QSTDnorm[s]*P[s];
      //Initial value of Mu is calculated to be the difference in
      //actual quantity and demand which gives overage/underage
      Mu[s] = Q[s] - D[s];
      //Initial value of Eta is calculated to be the volume of
      //actual quantity and demand
      Eta[s] = Q[s] + D[s];
      //Subsequent simulated values in the sample
   } else {
      //Forecasted demand is simulated with exponentially smoothing
      //applied to it
      Dhat[s] = Dhat[s-1]+Dsmooth*errorD[s-1];
      double R3 = Math.random();
      if(s<8){
         //Scaling column whereby the demand increases
         //for 10 periods
         QSTDnorm[s] = QSTDnorm[s-1]+1;
         //Actual demand is a random variable from a standard
```

```
            //normal distribution and scaled appropriately
            D[s] = D[s-1]+QSTDnorm[s]*P[s]
                        +DRandScale*R3*Dhat[s-1]/Dhat[0];
         } else {
            double R4 = Math.random();
            //Scaling column whereby the demand decreases for another
            //10 periods after initially increases for 10
            QSTDnorm[s] = Math.max((QSTDnorm[s-1]-1),0);
            D[s] = D[s-1]-QSTDnorm[s]*P[s]+DRandScale*R3
                           -DRandScale*R4*Dhat[s-1]/Dhat[0];
         }
         if(D[s]<0){
            D[s] = 0;
         }
         if(s<5){
            sumD0 += D[s];
            MuD = sumD0/(s+1);
            ErrorStdDev[s] = Math.sqrt(MuD);
         } else {
            sumD += D[s];
            MuD = sumD/(s-4);
            double sum1 = 0;
            for(int n = s-5; n < s; n++){
               sum1 = sum1 + Error[n];
            }
            double Mean = sum1/5;
            double sum2 = 0;
            for(int n = s-5; n < s; n++){
               sum2 = sum2 + Math.pow((Error[n]-Mean),2);
            }
            ErrorStdDev[s] = Math.sqrt(sum2/4);
         }
         Qhat[s] = Dhat[s]+SafetyFactor*ErrorStdDev[s];
         double R5 = 1.5*Math.random()-1;
         Q[s] = D[s-1]+R5*ErrorStdDev[s-1];
         if(Q[s]<0){
            Q[s] = 0;
         }
         if(s==1){
            //Mu & Eta start by being calculated using a moving average
            Mu[s] = ((Q[s]-D[s])+(Q[s-1]-D[s-1]))/2;
            Eta[s] = ((Q[s]+D[s])+(Q[s-1]+D[s-1]))/2;
         } else if(s==2){
            Mu[s] = ((Q[s]-D[s])+(Q[s-1]-D[s-1])+(Q[s-2]-D[s-2]))/3;
            Eta[s] = ((Q[s]+D[s])+(Q[s-1]+D[s-1])+(Q[s-2]+D[s-2]))/3;
         } else {
            if(selectedSmoothType=="Exponential smoothing"){
               Mu[s] = ExpSmooth*(Q[s]-D[s])+(1-ExpSmooth)*Mu[s-1];
               Eta[s] = ExpSmooth*(Q[s]+D[s])+(1-ExpSmooth)*Eta[s-1];
            } else if(selectedSmoothType=="Four point moving average"){
               Mu[s] = ((Q[s]-D[s])+(Q[s-1]-D[s-1]
                           +(Q[s-2]-D[s-2])+(Q[s-3]-D[s-3]))/4;
               Eta[s] = ((Q[s]+D[s])+(Q[s-1]+D[s-1])
                           +(Q[s-2]+D[s-2])+(Q[s-3]+D[s-3]))/4;
            }
         }
      }
      errorD[s] = D[s]-Dhat[s];
      errorQ[s] = Q[s]-Qhat[s];
      Error[s] = errorQ[s]-errorD[s];
      if(s<=MaxIndex(Dhat)){
```

```
      sumE = sumE + Error[s];
   } else {
      sumE = sumE;
   }

}

//Excel output
for(int s = 0; s < SampleSize; s++){
   if(s>=5){
      if(s==MaxIndex(Dhat)){
         out1.print("*** " + (s-4) + "," + (s+1) + ","
                        + Qhat[s] + "," + Dhat[s]);
         out1.print("," + D[s] + "," + Q[s] + ","
                        + errorD[s] + "," + errorQ[s]);
         out1.println("," + Mu[s] + "," + Eta[s] + "," + Error[s]
                        + "," + ErrorStdDev[s] + ",***");
      } else {
         out1.print((s-4) + "," + (s+1) + ","
                        + Qhat[s] + "," + Dhat[s]);
         out1.print("," + D[s] + "," + Q[s] + ","
                        + errorD[s] + "," + errorQ[s]);
         out1.println("," + Mu[s] + "," + Eta[s] + ","
                        + Error[s] + "," + ErrorStdDev[s]);
      }
   } else if(s>13){
      out1.print("," + (s+1) + "," + Qhat[s] + ","
                     + Dhat[s] + "," + D[s] + "," + Q[s]);
      out1.print("," + errorD[s] + "," + errorQ[s] + ","
                     + Mu[s] + "," + Eta[s]);
      out1.println("," + Error[s] + "," + ErrorStdDev[s]);
   } else {
      out1.print("," + (s+1) + "," + Qhat[s] + ","
                     + Dhat[s] + "," + D[s] + "," + Q[s]);
      out1.print("," + errorD[s] + "," + errorQ[s] + ","
                     + Mu[s] + "," + Eta[s]);
      out1.println("," + Error[s] + "," + ErrorStdDev[s]);
   }
}

double meanE = sumE/MaxIndex(Dhat);
double innerSum = 0;
for(int s = 0; s <= MaxIndex(Dhat); s++){
   innerSum = innerSum + Math.pow((Error[s]-meanE),2);
}
double SigmaE = Math.sqrt(innerSum/(MaxIndex(Dhat)));

//Excel output
out1.println(",");
out1.println(",,,,Mean,,,,,,StDev");
out1.println(",,,,," + MuD + ",,,,,," + SigmaE);
out1.close();

cp = R - M;
cu1 = 0;
co2 = 0;

double c = -2;
int count = 0;
while(c<=2){
   c = c + 0.1;
```

```
   count++;
}

double[] k = new double[count];
double[] Profit = new double[count];
double[] Mu1 = new double[count];
double[] SigmaE1 = new double[count];
double[] MuUn = new double[count];
double[] SigmaEUn = new double[count];
double[] MuOv = new double[count];
double[] SigmaEOv = new double[count];

double kSum = -2;
for(int n = 0; n < count; n++){
   if(n==0){
      k[n] = kSum;                  //
   } else {                         //
      kSum = kSum + 0.1;            //Sample values for k
      k[n] = kSum;                  //
   }                                //
   if(k[n]==0){
      k[n] = 1.0E-50;
   }
   //This calculates the profit from the objective function
   //for each value of k
   Profit[n] = MuD*cp - SigmaE*((NormalPDF(k[n])
                  - k[n]*(1-NormalCDF(k[n])))*(cp+cu1+co2)
                  + (k[n]*NormalCDF(k[n])
                        +NormalPDF(k[n]))*(co1+cu2));
}

//This calculates the optimal profit
double OptProfit = Max(Profit);

double optOv =
      SigmaE*(k[MaxIndex(Profit)]*NormalCDF(k[MaxIndex(Profit)])
                  + NormalPDF(k[MaxIndex(Profit)]));
double optUn = SigmaE*(NormalPDF(k[MaxIndex(Profit)])
                  -  k[MaxIndex(Profit)]*(1 –
                              NormalCDF(k[MaxIndex(Profit)])));
//The isovalue line cuts the mu-axis at these two points
double MuNeg = -(optOv*(co1+cu2)/(cu1+co2+cp)+optUn);
double MuPos = optOv+optUn*(cu1+co2+cp)/(co1+cu2);

double ov = SigmaE*(SafetyFactor*NormalCDF(SafetyFactor)
                        +NormalPDF(SafetyFactor));
double un = SigmaE*(NormalPDF(SafetyFactor)
                        -SafetyFactor*(1-NormalCDF(SafetyFactor)));

out2.println(count);
out2.print("Overage Target with k = " + SafetyFactor + ",,,");
out2.println(ov + ",,The isovalue line cuts the mu-axis”
                  + “ at the following points");
out2.print("Underage Target with k = " + SafetyFactor + ",,,");
out2.println(un + ",,mu = ," + MuNeg + ", and mu = ," + MuPos);

out2.println(",");
out2.println("k,Profit,Mu,SigmaE,MuUn,SigmaEUn,MuOv,SigmaEOv");
out2.println(",," + MuNeg + "," + 0 + "," + -un + "," + 0);

for(int n = 0; n < count; n++){
```

```
            //SigmaE1 and Mu1 are the optimal points along the isovalue line
            //for each value of k
            SigmaE1[n] = (MuD*cp - OptProfit)/((NormalPDF(k[n])
                                - k[n]*(1 - NormalCDF(k[n])))*(cp+cu1+co2)
                                    + (k[n]*NormalCDF(k[n])
                                          + NormalPDF(k[n]))*(co1+cu2));
            Mu1[n] = k[n]*SigmaE1[n];
            //MuUn and SigmaEUn are the points along the underage efficient
            //frontier for each value of k
            MuUn[n] = (k[n]*un)/(NormalPDF(k[n])-k[n]*(1-NormalCDF(k[n])));
            SigmaEUn[n] = MuUn[n]/k[n];
            //MuOv and SigmaEOv are the points along the overage efficient
            //frontier for each value of k
            MuOv[n] = (k[n]*ov)/(k[n]*NormalCDF(k[n])+NormalPDF(k[n]));
            SigmaEOv[n] = MuOv[n]/k[n];
            //Excel output
            if(Mu1[n]>MuNeg && Mu1[n]<MuPos){
               if(SigmaEUn[n]<(SigmaE+1) && SigmaEUn[n]>0){
                  if(SigmaEOv[n]<(SigmaE+1) && SigmaEOv[n]>0){
                     out2.print(k[n] + "," + Profit[n] + "," + Mu1[n]
                                + "," + SigmaE1[n]);
                     out2.println("," + MuUn[n] + "," + SigmaEUn[n] + ","
                                + MuOv[n] + "," + SigmaEOv[n]);
                  } else {
                     out2.print(k[n] + "," + Profit[n] + "," + Mu1[n] + ","
                                + SigmaE1[n]);
                     out2.println("," + MuUn[n] + "," + SigmaEUn[n]);
                  }
               } else {
                  if(SigmaEOv[n]<(SigmaE+1) && SigmaEOv[n]>0){
                     if(n==(count-1)){
                        out2.print(k[n] + "," + Profit[n] + "," + MuPos
                                + "," + 0);
                        out2.println(",,," + ov + "," + 0);
                     } else {
                        out2.print(k[n] + "," + Profit[n] + "," + Mu1[n]
                                + "," + SigmaE1[n]);
                        out2.println(",,," + MuOv[n] + "," + SigmaEOv[n]);
                     }
                  } else {
                     out2.println(k[n] + "," + Profit[n] + "," + Mu1[n]
                                + "," + SigmaE1[n]);
                  }
               }
            }
         }
         out2.close();
      }
      catch(IOException e){
         System.out.println(e.getMessage());
      }

      JOptionPane.showMessageDialog(
                              frame,
                              "End of program run-time"
                              + " (Simulation run successful)",
                              "END",
                              JOptionPane.INFORMATION_MESSAGE);

   }
}
```

```
        }

        public static void main(String[] args){
           AgileSim_updated sim = new AgileSim_updated();
        }
}
```