Journal of the Nigerian Association of Mathematical Physics
Volume **19 (November, 2011), pp 439 – 452**

Optimisation of Transfer Function Models using Genetic Algorithms

A.C. Igboanugo[1]* and C. C. Nwobi-Okoye[2]

[1] Department of Production Engineering,
University of Benin, Benin City, Edo State, Nigeria
[2] Emeagwali Centre for Research,
Anambra State University, Uli, Nigeria

Abstract

Initial transfer function estimates usually require optimisation. In order to obtain a more efficient transfer function model, a manual estimation method is often employed. This manual estimation method, even though it results to better estimates of the transfer function model, it still does not give an optimum outcome. In order to obtain an optimum transfer function estimate, open source software based on genetic algorithm was developed. The software was developed with Visual Basic programming language. In order to test the software, a transfer function model was developed from data obtained from industry. The forecast obtained from the transfer function model has a MAPE of 99.82%. After optimisation using the developed software the MAPE of the new forecast was 99.84%. This shows a marginal improvement in forecast accuracy.

Keywords: Transfer function, Genetic Algorithm, Optimisation, Open Source, Software

Nomenclature, Symbols and Notations

v(B) = Transfer function
B = backshift operator
Y_t = process output
X_t = process input
y_t = differenced output series
x_t = differenced input series
a_t = error term/white noise
υ_k = impulse response weight at lag k
θ = moving average operator
$\boldsymbol{\varphi}$ = autoregressive operator
b = transfer function lag
ω = difference equation variable for input
δ = difference equation variable for output

Introduction

Initial estimates of transfer function models often require improvement in the estimate in order to increase its efficiency. Usually, a manual estimation method based on trial and error is often adopted. Such manual estimates are not efficient.

Efficient estimation of transfer function models leads to better forecasts, improved design of plants and equipment, improved plant maintenance and replacement, and better quality control.

Using manual methods in transfer function estimation improvement is rigorous, slow and often leads to errors. Beside these shortcomings, such methods are not very efficient.

Using optimisation approaches based on meta-heuristics is an attractive tool in efficiency improvement of transfer function estimates. This is because in metaheuristic randomness are deliberately introduced into the search process as a means of speeding convergence and making the algorithm less sensitive to modelling errors. Most of the methods in MetaHeuristics mimic natural processes [5], [7] and [8]. Some examples of MetaHeuristics methods include: genetic algorithm, Tabu Search, Ant Colony Optimisation etc. It has been reported that metaheuristics have become more standard in several fields of sciences, but their use in estimation and modelling in statistics appears to be still limited and transfer function modelling is no exception [9].

Corresponding author: A.C. Igboanugo, E-mail: dracigboanugo@yahoo.com, Tel. +2348033830934

Genetic algorithm was defined as an optimisation and search technique based on the principles of genetics and natural selection [5]. The method was developed by John Holland over the course of the 1960s and 1970s and finally popularised by his student, David Goldberg [5]. The landmark works of Holland and Goldberg can be found in [4] and [6].

Haupt and Haupt listed the advantages of genetic algorithm to include the following:

1. can handle optimisation with continuous and discrete variables;
2. does not require derivative information;
3. simultaneously searches from a wide sampling of the cost surface;
4. deals with a large number of variables;
5. it is well suited for parallel computers;
6. optimises variables with extremely complex cost surfaces;
7. provides a list of optimum variables, not just a single solution;
8. may encode the variables so that the optimisation is done with the encoded variables;
9. works with numerically generated data, experimental data, or analytical functions.

These advantages enumerated above make genetic algorithm a very good candidate for optimising transfer function models. The aim of this work, therefore, is to apply genetic algorithm to the optimisation of transfer function models so as to facilitate the attainment of the optimal outcome.

Methodology

Transfer Function Model Building

A sixty-five-month input-output data reflecting the production time series record is depicted as spectral plots in Figures 1 and 2 respectively. A close examination of the two shows the output is an after-image of the input. Besides, both are stochastically irregular. It is pertinent to stress that the company manufactures milk tin cans for commercial milk production. The inputs are sheet metal that is coated with tin metal after cans are produced. Both input and output materials are measured in kilograms.

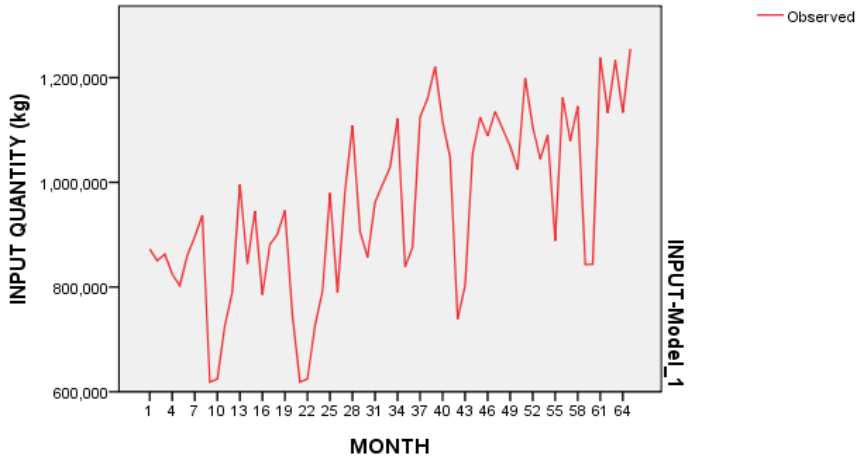

Figure 1: Spectral plot of the input time series

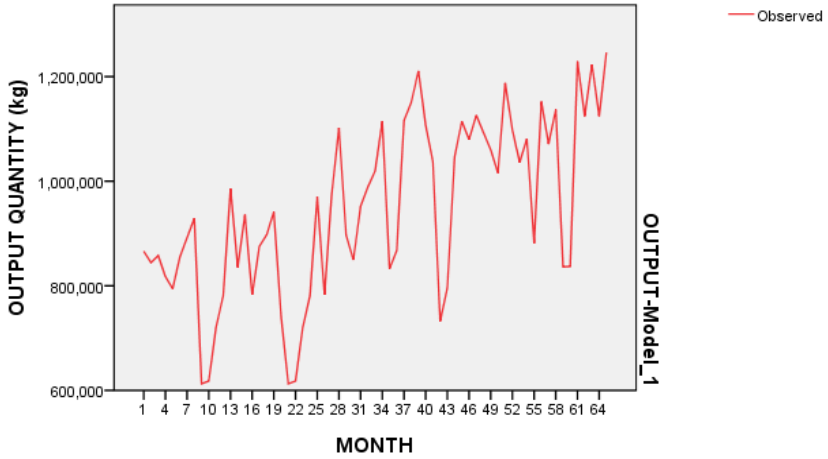

Figure 2: Spectral plot of the output time series

Since a production process is best described by a discrete transfer function model, for our analysis we assumed a transfer function model of the form:

$$Y_t = \delta^{-1}(B)\omega(B)X_{t-b} + N_t \quad (1)$$

The noise term, N_t, is represented by an ARIMA (p,d,q) process such that:

$$N_t = \varphi^{-1}(B)\theta(B)a_t \quad (2)$$

Here a_t is the white noise. Substituting equation (2) into (1), gives

$$Y_t = \delta^{-1}(B)\omega(B)X_{t-b} + \varphi^{-1}(B)\theta(B)a_t \quad (3)$$

Development of the Genetic Algorithm

The Model Parameters

As transfer function model parameters are continuous variables, the genetic algorithm method adopted was the continuous genetic algorithm. The model parameters are: **b**, **δ, ω, φ** and **θ**.

The Cost Function and Initial Population

(i) ***The Cost Function***

A close approximation to the maximum likelihood estimates of the parameters of the transfer function can be obtained by minimizing the conditional sum of squares function [1] and [2].

$$S_0(b,\delta,\omega,\varphi,\theta) = \sum_{t=1}^{n} a_t^2(b,\delta,\omega,\varphi,\theta \mid x_0, y_0, a_0) \quad (4)$$

Here x_0, y_0 and a_0 are starting values prior to the commencement of the series.

The cost function of the genetic algorithm was therefore determined by the least squares method, where the optimum model minimizes the least squares function.

(ii)The Initial Population

The initial population of the evolutionary system was kept at eight (8). The population that survives to the next generation is four (4), hence N_{keep}=4. Table 1 shows the initial population of the chromosomes after sorting and ranking. X, Y and Z in Table 1 represent the model parameters while C represents the least square estimate.

Table 1: Initial Chromosome Population

CHROMOSOMES			COST
X1	Y1	Z1	C1
X2	Y2	Z2	C2
X3	Y3	Z3	C3
X4	Y4	Z4	C4
X5	Y5	Z5	C5
X6	Y6	Z6	C6
X7	Y7	Z7	C7
X8	Y8	Z8	C8

C1>C2>C3>C4> C5>C6>C7>C8

Breeding and Mutation

(i) ***Breeding***

The breeding follows certain rules. A random number generator selects the pairs that would mate and breed the next generation. The pairing is done according to the rule in Table 2.

Table 2: Pairing Rule

ROWS	RND
R1	0.4
R2	0.7
R3	0.9
R4	1

According to the pairing rule in Table 2, if the random number value is less than 0.4, Row 1 (R1) is selected for pairing, if it lies between 0.4 and 0.7, Row 2 (R2) is selected, if it lies between 0.7 and 0.9, Row 3 (R3) is selected, if it lies between 0.9 and 1, Row 4 (R4) is selected. After pairing, the crossover point is selected by the random number generator. Assuming that after pairing R1, R2 and R1, R3 were selected for mating, Table 1 is changed as shown in Table 3.

Table 3: Initial Chromosome Population after Pairing

CHROMOSOMES			COST
X1	Y1	Z1	C1
X2	Y2	Z2	C2
X3	Y3	Z3	C3
X4	Y4	Z4	C4
X1	Y1	Z1	C1
X2	Y2	Z2	C2
X1	Y1	Z1	C1
X3	Y3	Z3	C3

A random number selects the crossover point. Assuming column 3 is selected as the crossover point; Table 3 is changed to Table 4 below.

Table 4: Initial Chromosome Population after Crossover

CHROMOSOMES			COST
X1	Y1	Z1	C1
X2	Y2	Z2	C2
X3	Y3	Z3	C3
X4	Y4	Z4	C4
X1	Y1	Z2	Cnew1
X2	Y2	Z1	Cnew2
X1	Y1	Z3	Cnew3
X3	Y3	Z1	Cnew4

New strains are introduced in the breeding population. A random number generator selects the column in the mating population where new strains will be introduced. Assuming column 2 is chosen, new strains are introduced according to the following formula [5].

$$T = RND$$

$$Ynew1 = Y1 - (T \times Y1) + (T \times Y2)$$

$$Ynew2 = Y2 + (T \times Y1) - (T \times Y2)$$

$$N = RND$$

$$Ynew3 = Y1 - (N \times Y1) + (N \times Y3)$$

$$Ynew4 = Y3 + (N \times Y1) - (T \times Y3)$$

After introducing new strains, the population will be as shown in Table 5.

Table 5: Population after Crossover and Introduction of new Strains

CHROMOSOMES			COST
X1	Y1	Z1	C1
X2	Y2	Z2	C2
X3	Y3	Z3	C3
X4	Y4	Z4	C4
X1	Ynew1	Z2	Cnew1
X2	Ynew2	Z1	Cnew2
X1	Ynew3	Z3	Cnew3
X3	Ynew4	Z1	Cnew4

(ii) Mutation

According to our methodology, 20 percent of the chromosome population are selected for mutation. Hence the number of chromosomes selected for mutation is 4. A random number generator selects the chromosomes that will undergo mutation. After mutation the new population could be as shown in Table 6.

Table 6: Population after Mutation

CHROMOSOMES			COST
Xnew2	Y1	Z1	Cnew5
X2	Y2	Z2	C2
X3	Y3	Znew1	Cnew6
X4	Ynew5	Z4	Cnew7
Xnew1	Ynew1	Z2	Cnew1
X2	Ynew2	Z1	Cnew2
X1	Ynew3	Z3	Cnew3
X3	Ynew4	Z1	Cnew4

Software Design Methodology

Choice of Programming Language

The programming language used was Visual Basic [3]. The language was chosen because it has rich graphic features and graphical software development tools. The user friendly integrated software development environment maximizes the programmer's productivity and minimizes errors during software development.

Software Architecture

The software consists of three modules namely: the main module and two sub modules. The software architecture is shown in figures 3 and 4.

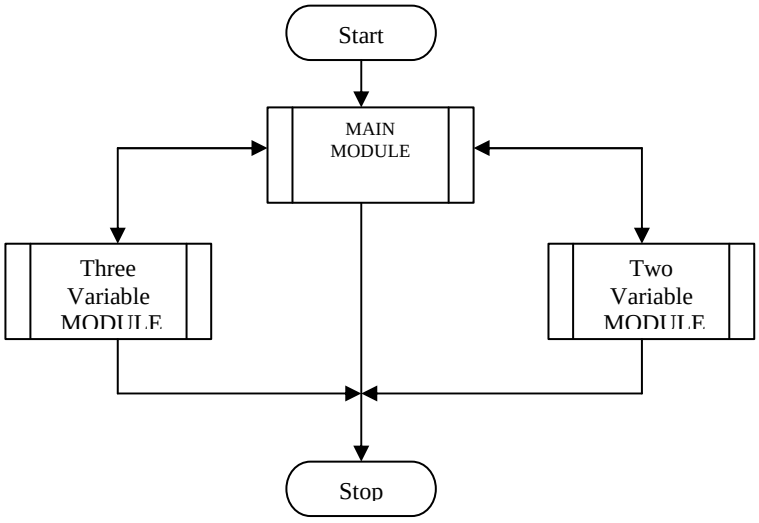

Fig. 3 Main software architecture

Figure 4 shows the genetic algorithm in the two and three variables modules. As shown in the algorithm, the evolution occurred in 4000 iterations. In order words 4000 generations were analysed before the optimum genetic combinations were selected.

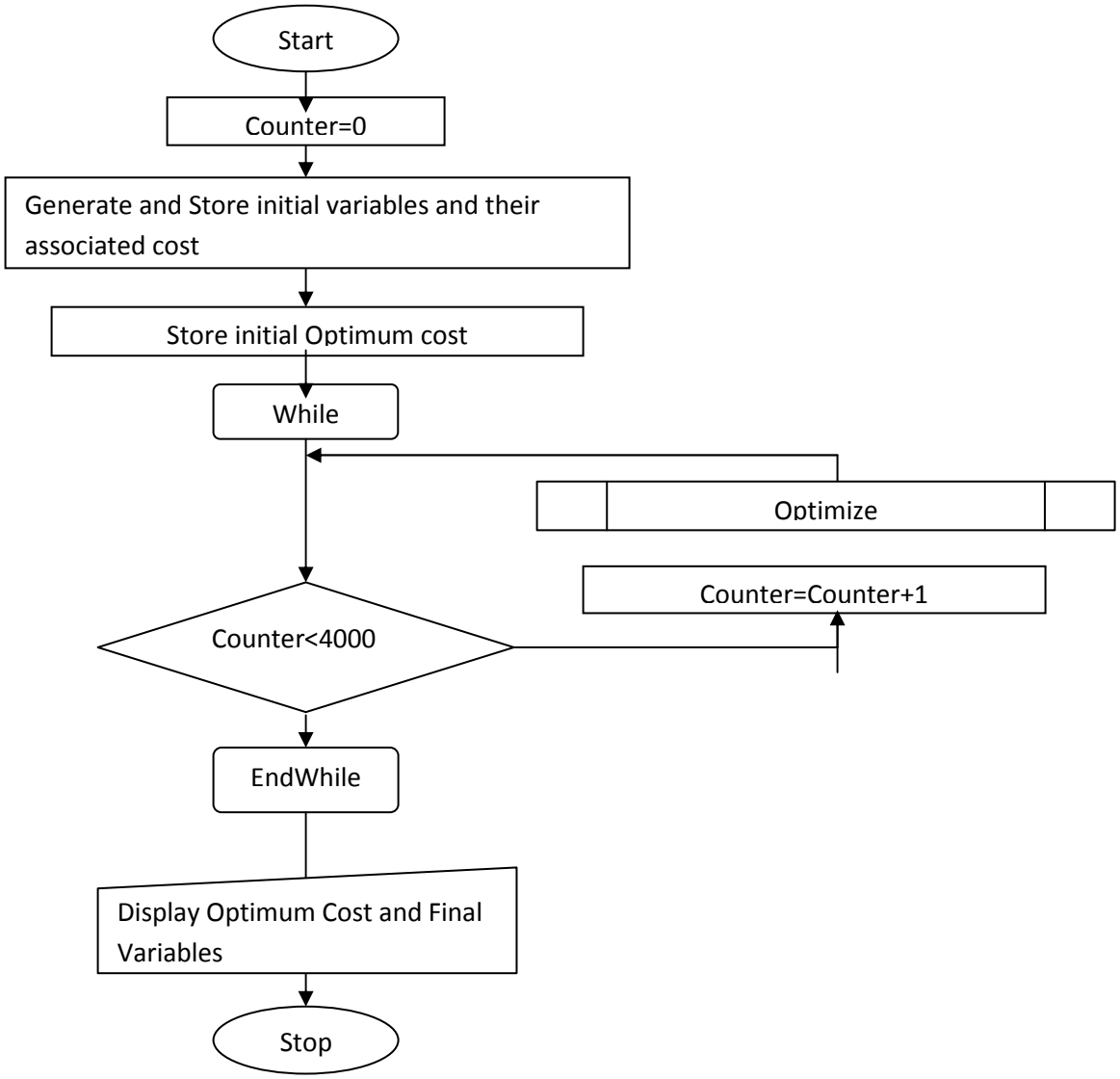

Fig. 4 Genetic Algorithm Flow Chart

Figure 5 shows the optimisation algorithm. The optimisation algorithm does the selection, breeding and mutation required to evolve from one generation to the next.

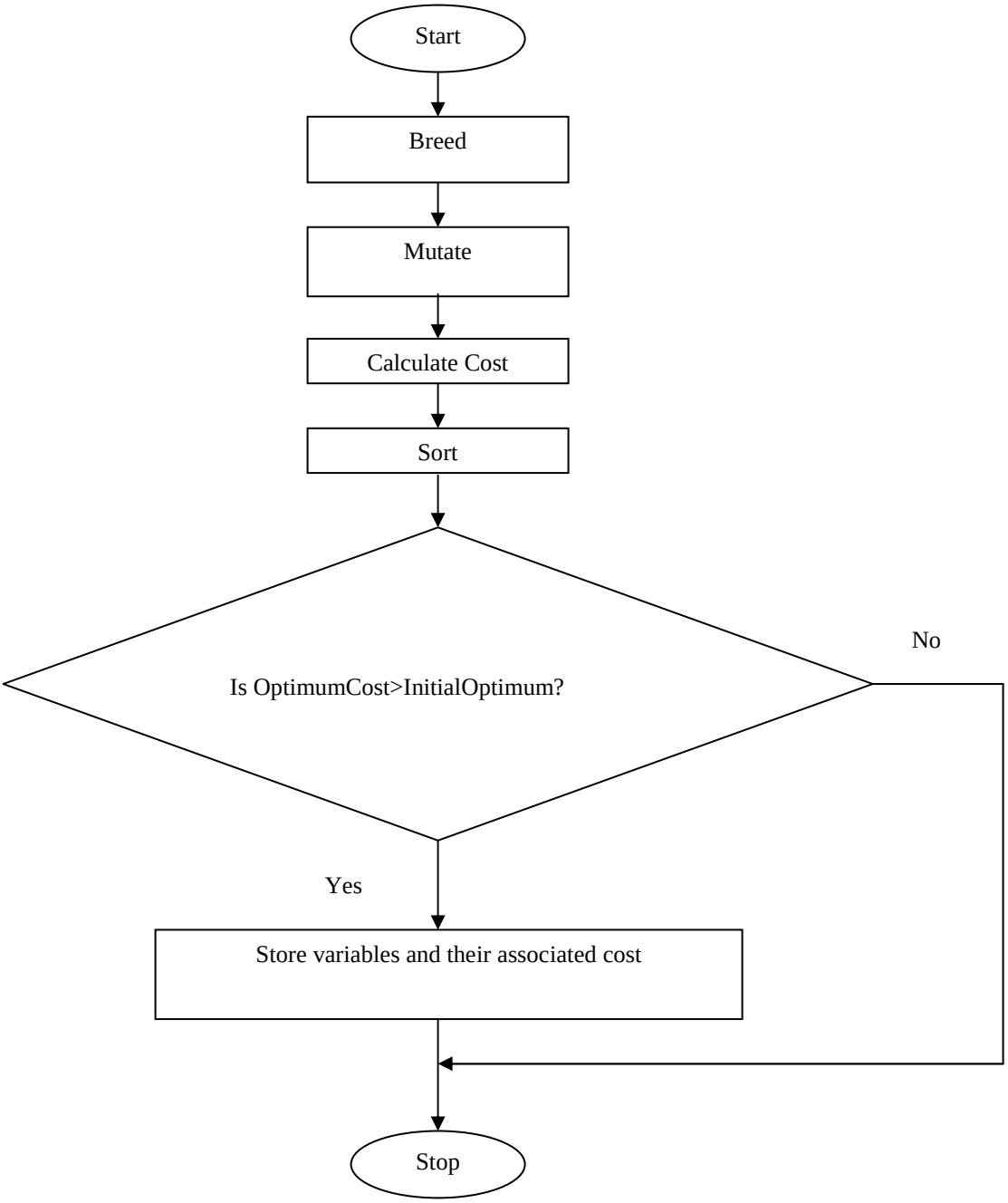

Fig. 5 Optimisation algorithm

Results Presentation and Analysis

After modelling, the following transfer function model was developed:

$$\hat{Y}_t = Y_{t-1} + J_{t-1} + 0.9932\,S_t - 0.6522\,e_{t-1} + 0.2e_{t-2} \quad (5)$$

This transfer function model is tentative and therefore requires optimisation to obtain the most efficient model. Consequently, the software was developed using visual basic programming language for optimising the model with genetic algorithm. Some of the software codes is shown in the appendix.

The user interfaces of the software developed for the optimisation are shown in figures 6, 7 and 8. The software developed as depicted in the figures under reference suggests that the optimisation can be applied to two-variable as well as three-variable problem situations. However, in this study, the latter case was adopted.

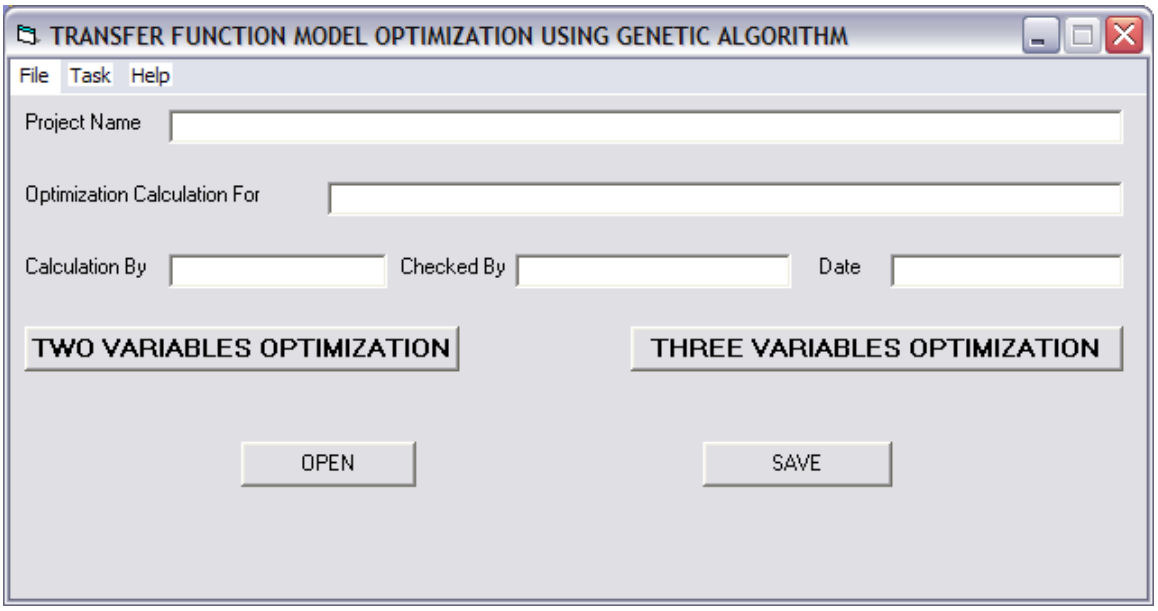

Fig. 6 Main Software Module

Examination of the two variable module shown in Figure 7 shows that it accepts two transfer function variables X and Y for optimisation. The data from the time series whose model is to be optimised are usually saved on the disk and retrieved prior to optimisation. The save and open buttons in Figure 7 does this. The three variable module shown in Figure 8 is similar to the two variable module except that it accepts three transfer function variables X, Y and Z.

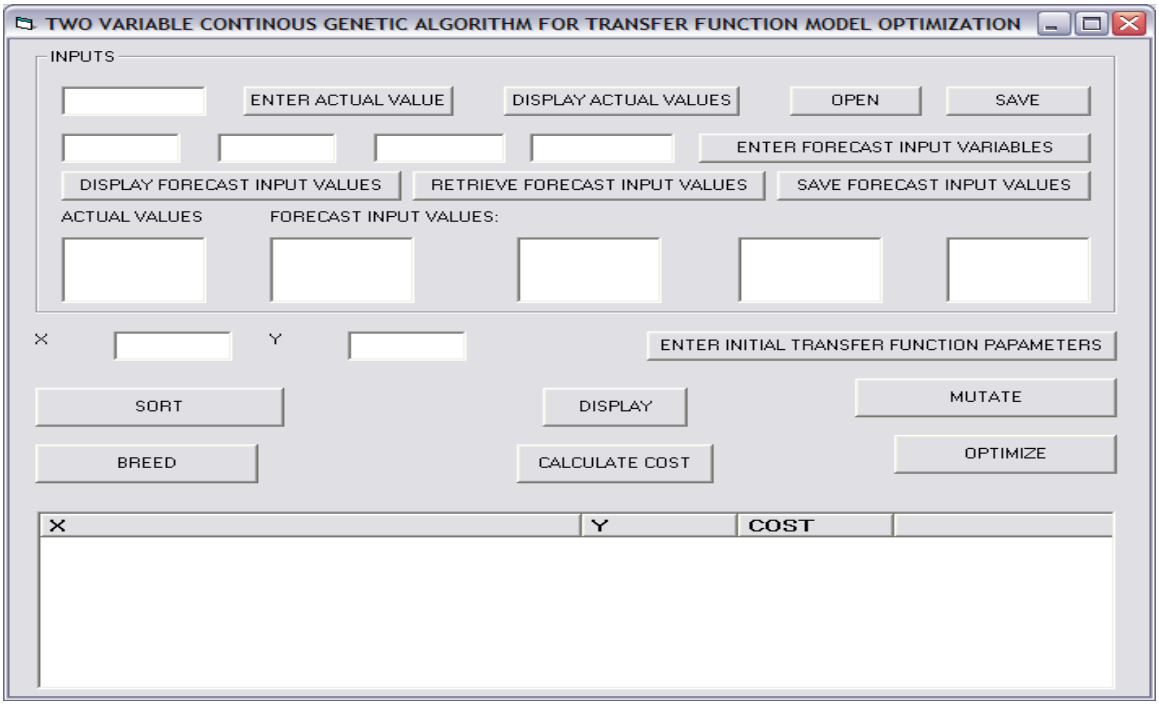

Fig. 7 Genetic algorithm module for two variables

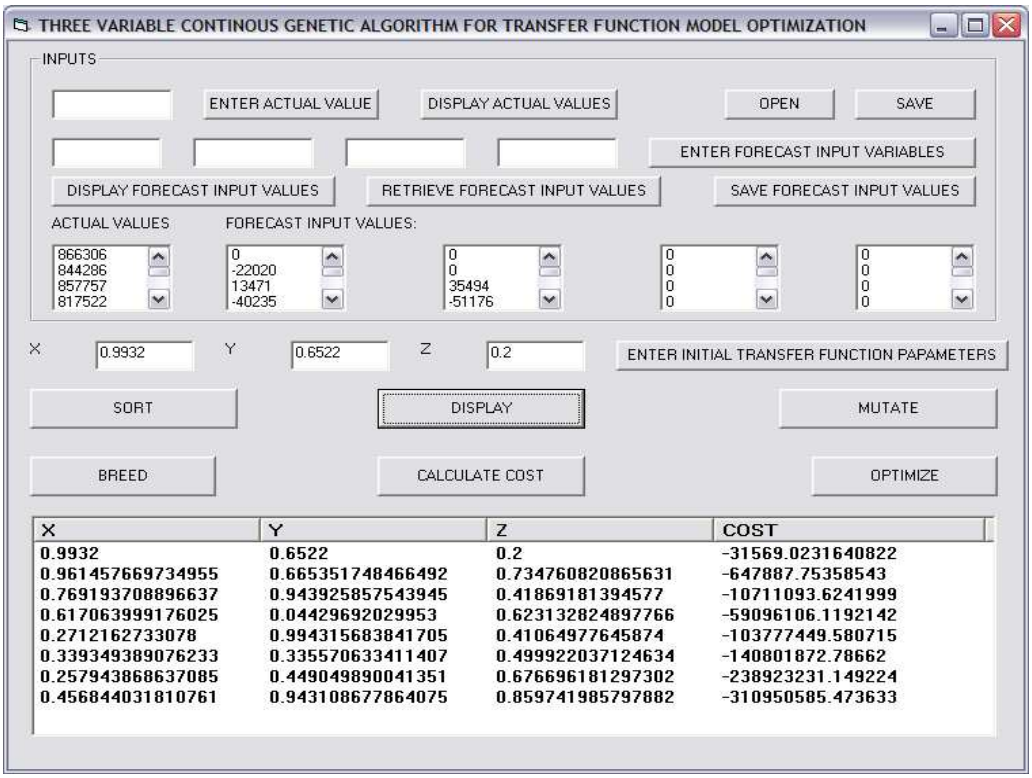

Fig. 8 Genetic algorithm module for three variables

Figure 6 shows the initial population of the transfer function variables and their initial costs. As shown in the figure the tentative transfer function parameter which are entered in the text boxes have the least cost.

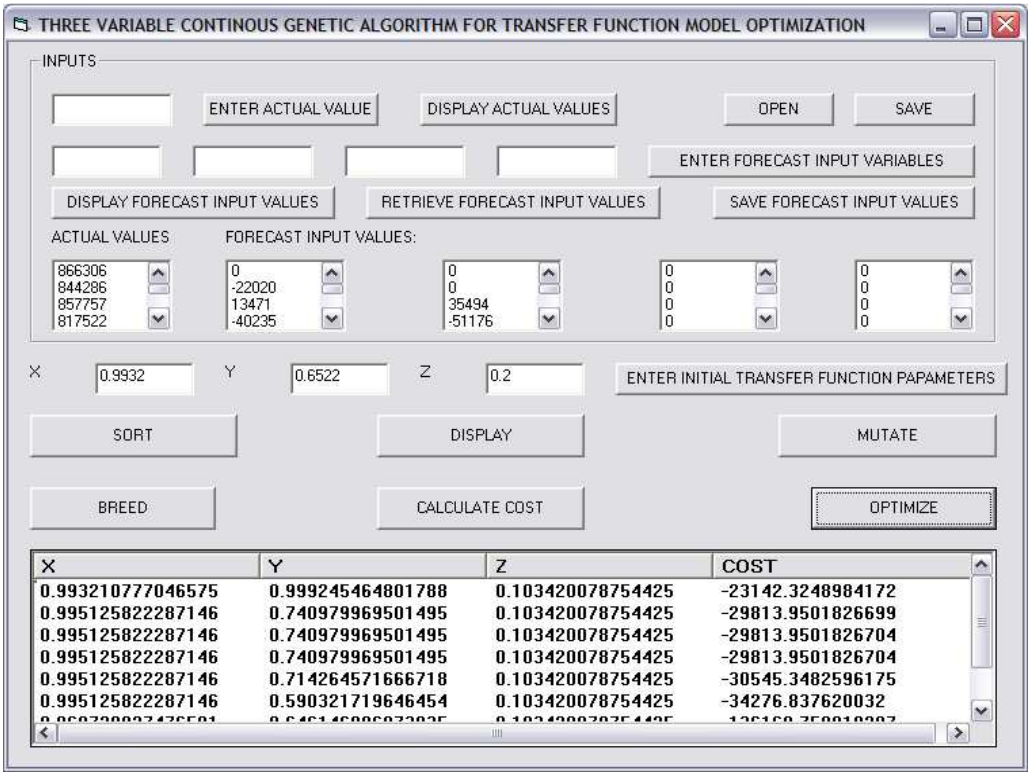

Fig. 9 The final population after optimisation

Figure 9 shows the population of the transfer function variables in the last generation and their costs. The last generation was obtained after 4000 iterations. As shown in the figure the least cost is 23142. The transfer function parameters that gave rise to the least cost are:

$$w_0 = 0.993211,\ \theta_1 = 0.99955 \text{ and } \theta_2 = 0.10342$$

After optimisation with the computer software, the optimum transfer function model was determined as:

$$\hat{Y}_t = Y_{t-1} + J_{t-1} + 0.993211\, S_t - 0.999245\, e_{t-1} + 0.10342\, e_{t-2} \quad (6)$$

The forecast obtained from the transfer function model in equation (5) has an MAPE of 99.82%. The forecast obtained from the optimised model shown in equation (6) above was 99.84%.

Discussion

We have been able to show here that tentative transfer function models developed by Box-Jenkins method are tentative and require more efficient estimation. Hence, optimisation is needed. We have demonstrated that genetic algorithm is a very good optimisation tool for optimising transfer function models.

Although the initial estimate of the transfer function resulted to excellent forecasts with 99.84% mean absolute percentage error (MAPE), the 0.02 percent increase in the MAPE after optimisation is remarkable considering the fact that the initial transfer function model estimate was excellent. This therefore confirms the fact that better transfer function models could be got through genetic algorithm.

The success of genetic algorithm in transfer function model optimisation necessitates the need to try other meta-heuristics methods such as Tabu Search and Ant Colony Optimisation. A comparison of the various meta-heuristics methods would reveal the best method for transfer function model optimisation [9].

Finally, the software developed in this work is open source and could be obtained from the authors and modified to suit different models.

6.0 Conclusion

Transfer function models are extremely useful and have wide application in industry. They are widely used in forecasting, plant design and redesign etc, hence excellent models results to excellent forecasts and designs. Hence, any technique, like the one demonstrated here, that results to better transfer function models results to better forecasts and design.

It is therefore expected that the optimisation software developed here would be used by engineers and scientists in solving industrial and scientific problems.

APPENDIX

SAMPLE SOFTWARE CODES

The Procedure for Breeding

```
Private Sub CmdBreed_Click()
   'Pairing
   Pair
   'Mating
   'Select crossover point
   Y = Rnd
   If Y < 0.166 Then
      j = 0
      k = 1
      l = 2
   ElseIf Y < 0.332 Then
      j = 0
```

```
        k = 2
        l = 1
    ElseIf Y < 0.498 Then
        j = 1
        k = 0
        l = 2
    ElseIf Y < 0.664 Then
        j = 1
        k = 2
        l = 0
    ElseIf Y < 0.83 Then
        j = 2
        k = 1
        l = 0
    ElseIf Y < 1 Then
        j = 2
        k = 0
        l = 1
    End If

    'Assign new values to the chromosomes prior to cross over
    ArrayPop(4, j) = ArrayPop(Pairs(0), j)
    ArrayPop(5, j) = ArrayPop(Pairs(1), j)
    ArrayPop(6, j) = ArrayPop(Pairs(2), j)
    ArrayPop(7, j) = ArrayPop(Pairs(3), j)

    ArrayPop(4, k) = ArrayPop(Pairs(0), k)
    ArrayPop(5, k) = ArrayPop(Pairs(1), k)
    ArrayPop(6, k) = ArrayPop(Pairs(2), k)
    ArrayPop(7, k) = ArrayPop(Pairs(3), k)
    'Crossover
    Swap ArrayPop(4, k), ArrayPop(5, k)
    Swap ArrayPop(6, k), ArrayPop(7, k)

    'Produce new strains in the chromosomes
    j1 = ArrayPop(4, l)
    j2 = ArrayPop(5, l)
    X = Rnd
    ArrayPop(4, l) = j1 - (X * j1) + X * j2
    ArrayPop(5, l) = j2 + (X * j1) - X * j2
    j1 = ArrayPop(6, l)
    j2 = ArrayPop(7, l)
    X = Rnd
    ArrayPop(6, l) = j1 - (X * j1) + X * j2
    ArrayPop(7, l) = j2 + (X * j1) - X * j2
End Sub
```

The Procedure for Pairing

```
Sub Pair()
    j = 0
```

```
k = 1
m = 0
p = 0
Do While m < 2
   For i = 1 To 2
      X = Rnd
      If X < 0.4 Then
        k = 0
      ElseIf X < 0.7 Then
        k = 1
      ElseIf X < 0.9 Then
        k = 2
      ElseIf X < 1 Then
        k = 3
      End If
      Y = Rnd
      If Y < 0.4 Then
        j = 0
      ElseIf Y < 0.7 Then
        j = 1
      ElseIf Y < 0.9 Then
        j = 2
      ElseIf Y < 1 Then
        j = 3
      End If
      If j = k Then Exit For
      'l = p + 1
      'v = p + 2
      Pairs(p) = k
      p = p + 1

      Pairs(p) = j
      p = p + 1
      m = m + 1
      If m > 1 Then Exit For
   Next i
      If m > 1 Then
           ' This condition tests this senario: 2,3,2,3
           If Pairs(0) = Pairs(2) And Pairs(1) = Pairs(3) Then
             m = m - 1
             p = 2
           End If
           ' This condition tests this senario: 2,3,3,2
           If Pairs(0) = Pairs(3) And Pairs(1) = Pairs(2) Then
             m = m - 1
             p = 2
           End If
      End If
Loop
End Sub
```

The Procedure for Mutation

```
Private Sub CmdMutate_Click()
    Dim MutateArray(3, 1) As Integer
    Dim MutateItems As New Collection
    j = 0
    k = 0
    m = 0
    Dim b As String

    Do While m < 4
        X = Rnd
            If X < 0.125 Then
          k = 0
        ElseIf X < 0.25 Then
          k = 1
        ElseIf X < 0.375 Then
          k = 2
        ElseIf X < 0.5 Then
          k = 3
        ElseIf X < 0.625 Then
          k = 4
        ElseIf X < 0.75 Then
          k = 5
        ElseIf X < 0.875 Then
          k = 6
        ElseIf X < 1 Then
          k = 7
        End If

        Y = Rnd

        If Y < 0.5 Then
          j = 0
        ElseIf Y < 1 Then
          j = 1
        End If
        f = CStr(k)
        t = CStr(j)
        b = f + t
        On Error Resume Next
        MutateItems.Add m, b
        MutateArray(m, 0) = k
        MutateArray(m, 1) = j
        m = m + 1
        If m <> MutateItems.Count Then
           m = m - 1
        End If
    Loop
    ArrayPop(MutateArray(0, 0), MutateArray(0, 1)) = Rnd
```

```
    ArrayPop(MutateArray(1, 0), MutateArray(1, 1)) = Rnd
      ArrayPop(MutateArray(2, 0), MutateArray(2, 1)) = Rnd
      ArrayPop(MutateArray(3, 0), MutateArray(3, 1)) = Rnd
End Sub
```

References

[1] Box, G.E.P., Jenkins G.M. and Reinsel G.C. (1994). Time Series Analysis Forecasting and Control. McGraw-Hill Inc., USA.

[2] DeLurgio, S.A. (1998). Forecasting Principles and Applications, 3rd Edn, McGraw-Hill, New York, USA.

[3] Evangelous P. (1998). Mastering Visual Basic 6.0, Sybex Inc., 1151 Marina Village Parkway, Alameda, CA 94501, USA.

[4] Goldberg, David E. (1989). Genetic Algorithms in Search Optimization and Machine Learning. Addison Wesley. pp. 41.

[5] Haupt, R.L and Haupt, S.E. (2004). Practical Genetic Algorithms. John Wiley & Sons, Inc., Hoboken, New Jersey, USA.

[6] Holland, J.H. (1975). Adptation in Natural Selection and Artificial Systems. University of Michigan Press, Ann Arbor, Michigan, USA.

[7] Sean L. (2009). Essentials of Metaheuristics. Available at http://cs.gmu.edu/!sean/book/metaheuristics/ [Accessed 15th August 2003]

[8] Spall, J. C. (2003). Introduction to Stochastic Search and Optimization. Wiley.

[9] Winker, P. and Gilli, M. (2004). Applications of optimization heuristics to estimation and modelling problems. Computational Statistics & Data Analysis 47 (2): 211-223.