Kalman Filter Neuron Training

Bull. Univ. Osaka Pref., Ser. B, Vol. 43,
大阪府立大学紀要, 農学・生物学, 1991

Kalman Filter Neuron Training

Haruhiko MURASE, Shuhei KOYAMA*, Nobuo HONAMI, and Takao KUWABARA*

Laboratory of Agricultural Machinery *Laboratory of Land
Development Engineering College of Agriculture

(Received, 1990)

Abstract

An attempt of implementing Kalman filter algorithm[1] in the procedure for training the neural network was made and evaluated. The Kalman filter neuron training program (KNT) was coded. The performance of Kalman filter in KNT was compared to commonly used neuron training algorithm. The study revealed that KNT requires much less calculation time to accomplish neuron training than commonly used other algorithms do. KNT also gave much smaller final error than any other algorithms tested in this study.

Introduction

Artificial intelligence (AI) is concerned with developing software systems that are capable of performing works that one would describe as intelligent if human did them. Neural nets are currently one of the hottest AI research areas, after having a period of inconspicuous research development for over a decade[2]. In many of the other AI research area, neural nets related studies are being conducted because neural modeling is a technique that can in principle be directed at a variety of different applications. One of most attractive aspects is that there are essential mechanisms that allow for incrementally improved performance. The neural modeling technique does not confine itself in AI research area. It has opened up a new way of describing nonlinear systems in the form that computers can handle.

Most of constitutive elements of biological functions and / or ecosystem are nonlinear in nature. Many attempts of nonlinear analysis in biological systems have been made for quite a number of years now, but because of the difficulty of the problem, mostly they ended up with the simplification of the problem by linear assumptions. The hierarchical neural net can be applicable to simulating nonlinear phenomena often found in the biological systems[3]. The learning process of the hierarchical neural net can be used as an algorithm for the nonlinear maltivariate analysis. Inversely the Kalman filter, which can be effectively utilized to such types of problems as nonlinear maltivariate analysis, inverse analysis and so forth, can possibly be an algorithm to train the neural net[1].

In this study, an attempt of implementing Kalman filter algorithm in the procedure for training the neural network was made and evaluated.

Neuron Training[3)]

Fig. 1 illustrates an example of a simple hierarchical neural net architecture. The neural net consists of the input edge layer, a middle layer and the output edge layer. In general the hierarchical neural net may have middle layers as many as it needs. In this example the input edge layer contains three input units. Each input unit is stimulated by input signal. The middle unit has also three hidden units. Three output units form the output edge layer. Each layer may contain necessary number of units in general. The input units project to the hidden units, and the hidden units project to the output units. There are no direct connections from the input units to the output units. The units are connected each other with an adjustable weight called synapse weight. The training procedure adjusts the weight so that each output unit gives a desired answer signal. Mathematical explanation of the process of signal (stimulus) transfer in the network might help understand the hierarchical neural net. The stimulus **T** as an input signal can be expressed in a vector form as $\mathbf{T}=(t_1,t_2,...,t_n)$. In Fig. 1, n is equal to 3. i-th component of the stimulus **T**, i.e. t_i, comes out from the input unit i is transferred to a hidden unit j through the synapse weight W_{ij}. Since each hidden unit has a summation function operating on inputs, the total input u_j received by the hidden unit j becomes

$$u_j=\sum_{i=1}^{n} W_{ij}t_i \tag{1}$$

The hidden unit i has also a transfer function that performs nonlinear transform on the total input u_i and then gives an output $f(u_i)$. $f(u_i)$ becomes the next input fed into the

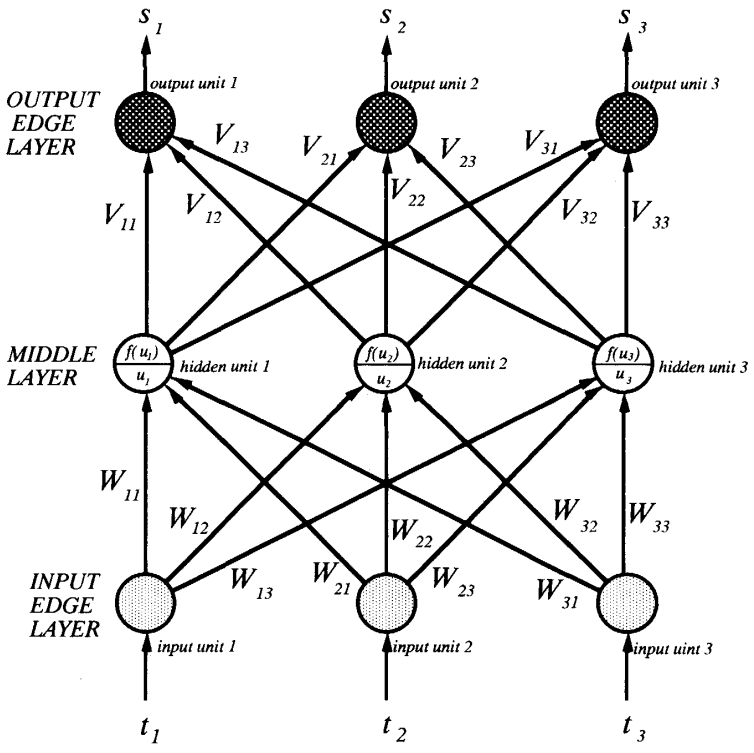

Fig. 1 Hieracrchical neural net architecture.

output unit j, which has a summation function, through another synapse weight V_{ij}. The total input received by the output unit j becomes directly its output s_j expressed as

$$s_j = \sum_{i=1}^{m} V_{ij} f(u_i) \tag{2}$$

The output signal is given in a vector form as $\mathbf{S} = (s_1, s_2, \ldots, s_m)$. After all what this neural network does is to perform a nonlinear transform on $\mathbf{T}$ as expressed in Eq. 3.

$$\mathbf{S} = F(\mathbf{T}) \tag{3}$$

Once those nonlinear functions of hidden units are specified, the behavior of the network can be identified by determining all synapse weights contained in the network. The neuron training is a procedure to determine optimal values of synapse weights by adjusting them step by step using known input data and their associated output data called teacher' s signal. The most common algorithm for this training procedure is the gradient descent method of back propagation, which is a version of the hill-climbing strategy. This technique can be considered as one of techniques used for parameter identification problems. Since the Kalman filter has many achievements on the engineering problems of parameter identification, it is worth trying to apply Kalman filter for the neuron training.

Kalman Filter[1)]

The Kalman filter technique will be adopted for the neuron training. A linear discreet system can be described by the following two equations

1) State equation

$$\{x\}_{k+1} = [A]\ \{x\}_k \tag{4}$$

$\{x\}$: state variable vector
$[A]$: system matrix
k : discreet time

2) Observation equation

$$\{y\}_k = \{h(\ \{x\}\ )\}_k \tag{5}$$

$\{y\}$: observation vector
$[h]$: observation matrix

In this study synapse weights should be considered as state variables. The discreet time can be taken as the iterative step,i.e., training iteration. Since synapse weight in this problem are independent upon time, $[A]$ in Eq. 4 must be the unit matrix. The state equation for this problem can then be written,

$$(W)_{k+1} = [I]\ \{W\}_k \tag{6}$$

where, (W) is a state variable vector of which components are synapse weights to be determined. Eq. 5 can be considered as a nonlinear observation equation. Eq. 5 may be expressed in simpler form using the sensitivity matrix $[H]$ as given by Eqs. 7, 8, and 9.

$$\{p\}_k = [H]_k \{W\}_k \tag{7}$$

where, $$(p)_k = (q) - \{F(\hat{W},t)\}_k + [H(\hat{W},t)]_k \{\hat{W}\}_k \tag{8}$$

$$[H_{ij}] = \frac{\partial F_i}{\partial W_j} \tag{9}$$

(q) : teacher's signal
$\hat{W}$: state variable estimated at one step prior to the present iteration.

The observation vector $\{p\}$ and matrix $[H]$ can be evaluated using a priori information.

Numerical Experiment

Numerical experiments have been conducted by considering the hierarchical neural net illustrated in Fig. 2. A three layered network is used for this test. Each layer consists of two units. Hidden unit H1 and H2 have the same type of transfer function called logistic function described by Eq. 10.

$$f(x) = \frac{1}{1+e^{-x}} \tag{10}$$

There are 8 synapse weights to be trained in this network. Eqs. 11 and 12 are the transfer

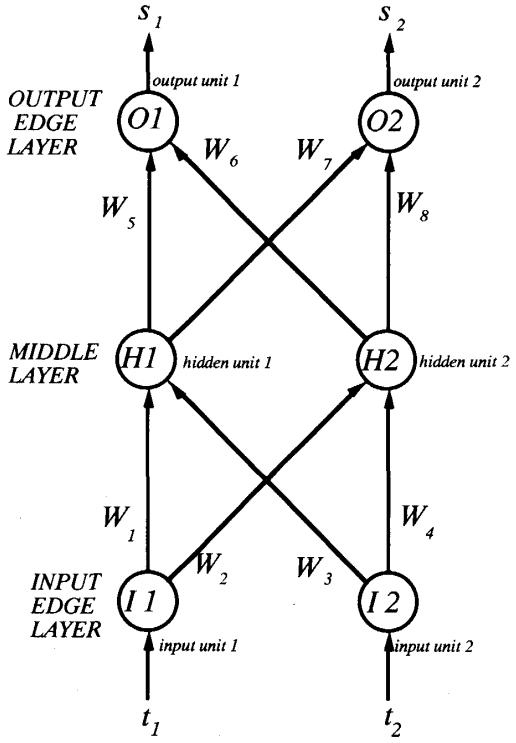

Fig. 2 Neural network used for numerical experiment.

functions of hidden unit H1 and H2, respectively.

$$F_1(t_1,t_2)=\frac{W_5}{1+e^{-(W_1t_1+W_3t_2)}}+\frac{W_6}{1+e^{-(W_2t_1+W_4t_2)}} \tag{11}$$

$$F_2(t_1,t_2)=\frac{W_7}{1+e^{-(W_1t_1+W_3t_2)}}+\frac{W_8}{1+e^{-(W_2t_1+W_4t_2)}} \tag{12}$$

The sensitivity matrix $[H]$ $((2\times n)\times 8)$, where n indicates a number of data sets used

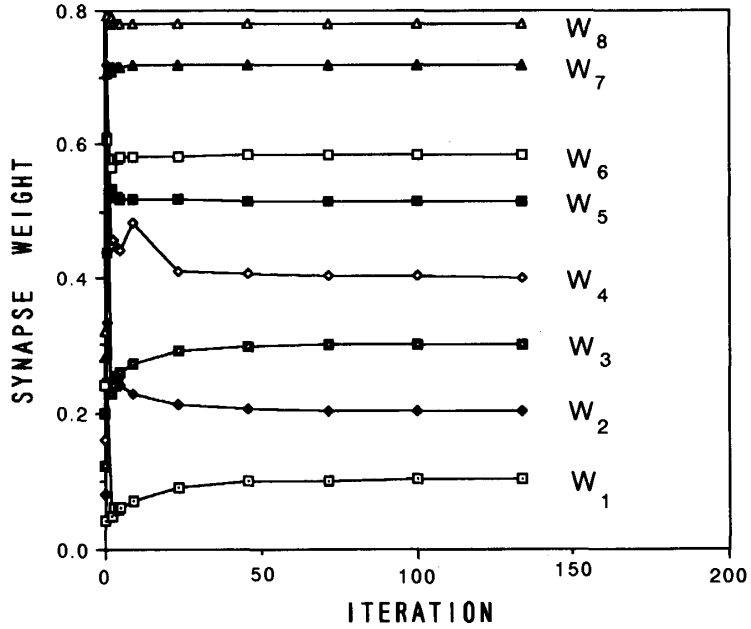

Fig. 3-aConvergence of synapse weights.

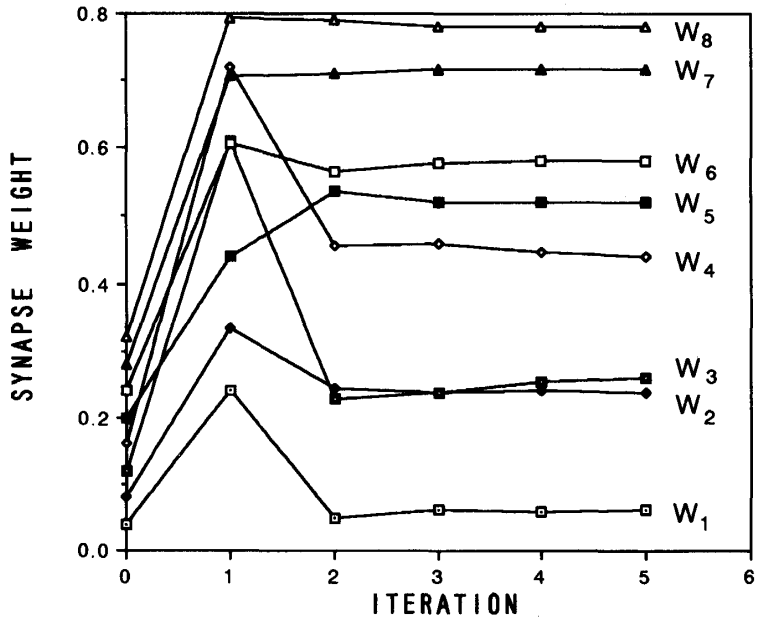

Fig. 3-bExpanded plot of convergence of estimates.

as teacher's signal, can be determined by differentiating Eqs. 11 and 12 with respect to synapse weights $W_1 \sim W_8$. The elements H_{11} and H_{12} of the matrix $[H]$, for example, are as follows;

$$H_{11}=\frac{\partial F_1(t)}{\partial W_1}=\frac{W_5 t_1 e^{-(W_1 t_1+W_3 t_2)}}{(1+e^{-(W_1 t_1+W_3 t_2)})^2}$$
$$H_{12}=\frac{\partial F_1(t)}{\partial W_2}=\frac{W_6 t_1 e^{-(W_2 t_1+W_4 t_2)}}{(1+e^{-(W_2 t_1+W_4 t_2)})^2} \tag{13}$$

Teacher's signal for the neuron training were generated by using Eqs. 11 and 12 with prescribed synapse weights W_1 through W_8 such as 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, respectively. Input values were generated by using the RND function of the computer language BASIC (NEC n88basic). Twenty sets of training data (teacher's signal) were provided.

Influence of Kalman filter parameters (variance of observation vector and diagonal elements of the estimation error covariance matrix) in filtering process on the total output error were examined. The total output error (e) was calculated using the following formula (Eq. 14), where o_i is a calculated output signal using i-th data set and q_i is i-th teacher's signal;

$$e=\sum_{i=1}^{n}(o_i-q_i)^2 \tag{14}$$

Twenty-four combinations of four different values (1×10^0, 1×10^{-2}, 1×10^{-5}, 1×10^{-9}) for the variance of observation vector and six magnitudes (0.01, 0.1, 1, 10, 100, 1000) for the

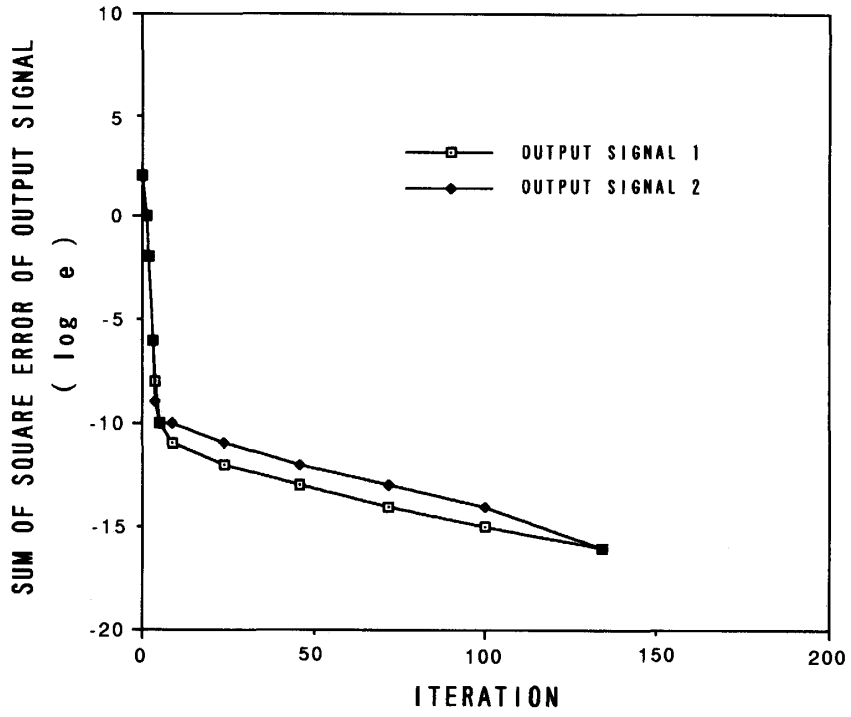

Fig. 4 Convergence of output error.

diagonal elements of the estimation error matrix were assigned and tested. Convergent characteristics of state variables (synapse weights) were also examined.

A comparison of the performance of this Kalman filter neuron training algorithm (KNT) and a neuron training program (PDP) developed by Brains Co. was made[4]. Forty data sets as teacher's signal were prepared. Those data were also generated using Eqs. 11 and 12 with random number inputs. Those data, however, contain a noise in the output signal. The format of the noise was $\pm 10\%$ random fluctuation of exact output value. In this numerical test, a random number between 0 and 1 was assigned to each initial value of state variables.

Results and Discussion

Table 1 capsulizes the test results regarding influence of the Kalman filter parameters on the output error. After 50 times of iterations, the sum of square error of output signal reached a value between 1×10^{-3} and 1×10^{-13} depending upon combination of the Kalman filter parameters. It seems that when the variance of observation vector is small, value of diagonal element of estimation error matrix results in smaller error.

Fig. 3—a shows the convergence of eight synapse weights. All estimates became close to the target values after 50 times of iteration. Fig. 3—b is an expanded plot showing initial fluctuation of estimates. A few initial iterations has already projected a right course of estimation.

The steep reduction of output error can be observed in Fig. 4. The first five iterations practically gave correct output signal with the error (e) less than 5.0×10^{-10}.

Table 2 highlights the outstanding performance of the kalman filter neuron training algorithm (KNT). PDP has a option to choose an minimization procedure from either

TABLE 1 INFLUENCE OF KALMAN FILTER PARAMETERS ON THE OUTPUT ERRORE AFTER 50 ITERATIONS

VARIANCE OF OBSERVATION VECTOR	DIAGONAL ELEMENT OF ESTIMATION ERROR MATRIX					
	0.01	0.1	1	10	100	1000
	SUM OF SQUARE ERROR OF OUTPUT SIGNAL					
1×10^{0}	1×10^{-3}	1×10^{-3}	1×10^{-5}	1×10^{-6}	1×10^{-7}	1×10^{-8}
1×10^{-2}	1×10^{-5}	1×10^{-7}	1×10^{-9}	1×10^{-9}	1×10^{-9}	1×10^{-9}
1×10^{-5}	1×10^{-7}	1×10^{-9}	1×10^{-9}	1×10^{-13}	1×10^{-12}	1×10^{-12}
1×10^{-9}	1×10^{-13}	1×10^{-12}	1×10^{-12}	1×10^{-8}	1×10^{-7}	1×10^{-6}

TABLE 2 COMPARISON OF NEURON TRAINING ALGORITHMS

SOFT WARE	NEURON TRAINING ALGORITHM	SUM OF SQUARE ERROR OF OUTPUT SIGNAL	ITERATION	CALCULATION TIME	COMPUTER LANGUAGE
KNT	Kalman Filter	3.386×10^{-2}	50	5 min	BASIC compiler
PDP	Steepest Descent Method	9.070×10^{0}	7576	30 min	C compiler
PDP	Conjugate Gradient Method	1.080×10^{-1}	2886	30 min	C compiler

the steepest gradient method or the conjugate gradient method. PDP is written by C compiler. KNT is intentionally coded using n88basic. This is because the BASIC program (see Appendix) can help those who are easily accessible to available personal computers understand rather sophisticated Kalman filter algorithm on a computer.

The most remarkable difference between PDP and KNT is the calculation time as shown in Table 2. Commonly used bench mark tests usually indicate that n88basic compiler needs more time to complete the test routine than C compiler. Even though such a handicap of the basic compiler, KNT needed only 5 minutes to reach a final output error. The final output error is defined as such a manner that iterative calculations no longer give appreciable reduction in sum of square error of output signal. The steepest gradient method of PDP did not attain any acceptable final output error within 30 minutes of calculation. PDP with the conjugate gradient method also gave about three times larger value of final output error than that calculated by KNT. The Kalman filter parameters chosen for this comparison were the followings; the variance of observation vector was 1×10^{-5}, and the diagonal element of estimation error matrix was 10.

The experimental results of this study showed that the application of Kalman filter algorithm to neuron training of hierarchical neural net as employed in this study is acceptable and effective.

Acknowledgement

A part of this research was supported by developmental scientific research found (Project number: 01860036) from the Ministry of Education, Science and Culture.

References

1) HRUHIKO, K., KOYAMA, S., and ISHIDA, R. (1990). *Introduction to inverse analysis.* Morikitashuppan. (*in Japanese*).
2) TELLO, E.R. (1989). *Object-oriented programing for artifitial intelligence.* Addison-Wesley Pub. Co. Inc.
3) HOSHI, T., HIRAFUJI, M. and HONJYO, T. (1990). *Expertsystems for biotechnology and agriculture.* Corona Pub. Co., Ltd. (*in Japanese*).
4) HORIUCHI, T. (1989). Development of neural net simulator using C Language. *Turing Machine.* Vol. 2., **5**, 50−61. (*in Japanese*).

APPENDIX

```
1000 '  ********************************************************
1010 '  *        KALMAN FILTER NEURON TRAINING PROGRAM         *
1020 '  *                                                      *
1030 '  *     Coded by Haruhiko Murase.     Oct. 1, 1990.      *
1040 '  ********************************************************
1050 '
1060 'SAVE"KALNEURO.BAS",A
1070 '
1080 OPEN "NEURO.DAT" FOR INPUT  AS #1
1090 DEFDBL  A-H,O-Z
1100 DIM W(10),FINP1(100),FINP2(100),OUT1(100),OUT2(100),WI(10)
1110 INPUT "RATE=";RATE
1120 IF RATE=0 THEN RATE=5
1130 R=VAL(RIGHT$(TIME$,2))
1140 RANDOMIZE R
1150 FOR I = 1 TO 8
1160 INPUT #1, WI(I):W(I)=RND(1)*RATE
1170 NEXT I
1180 FOR K=1 TO 100
1190 INPUT #1,FINP1(K),FINP2(K),OUT1(K),OUT2(K)
1200 NEXT K
1210 CLOSE
1220 OPEN "NR.DAT" FOR OUTPUT AS #1
1230 '
1240 CLS 3
1250 INPUT "NUMBER OF DATA POINTS=";NDP
1260 IF NDP=0 THEN NDP=20
1270 INPUT "R1=";R1
1280 IF R1=0 THEN R1=.00001
1290 INPUT "UU=";UU
1300 IF UU=0 THEN UU=10
1310 '
1320 PRINT #1,NDP,R1,UU
1330 FOR I = 1 TO 8
1340 PRINT #1,WI(I),W(I)
1350 NEXT I
1360 '
1370 N=NDP*2
1380 DIM HH(N,8),U(N,N),FF(N),YY(N),GG(N),H(N)
1390 '
1400 '   ------------ CALCULATION OF {h} ------------
1410 '
1420     FOR I = 1 TO NDP
1430     A=EXP(-(FINP1(I)*W(1)+FINP2(I)*W(3)))
1440     B=EXP(-(FINP1(I)*W(2)+FINP2(I)*W(4)))
1450     H(2*I-1)=W(5)/(1+A)+W(6)/(1+B)
1460     H(2*I)   =W(7)/(1+A)+W(8)/(1+B)
1470     NEXT I
1480 '
1490 '   ------------------- CALCULATION OF [H] -------------------
1500 '
1510     FOR I = 1 TO NDP
1520     HH(I*2-1,1)=0:HH(I*2-1,2)=0:HH(I*2-1,3)=0:HH(I*2-1,4)=0
1530     HH(I*2-1,5)=0:HH(I*2-1,6)=0:HH(I*2-1,7)=0:HH(I*2-1,8)=0
1540     HH(I*2,1)=0:HH(I*2,2)=0:HH(I*2,3)=0:HH(I*2,4)=0
1550     HH(I*2,5)=0:HH(I*2,6)=0:HH(I*2,7)=0:HH(I*2,8)=0
1560 '
1570     A=EXP(-(FINP1(I)*W(1)+FINP2(I)*W(3)))
1580     B=EXP(-(FINP1(I)*W(2)+FINP2(I)*W(4)))
1590     HH(I*2-1,1)=FINP1(I)*W(5)*A/((1+A)*(1+A))
1600     HH(I*2,1)  =FINP1(I)*W(7)*A/((1+A)*(1+A))
1610     HH(I*2-1,2)=FINP1(I)*W(6)*B/((1+B)*(1+B))
1620     HH(I*2,2)  =FINP1(I)*W(8)*B/((1+B)*(1+B))
1630     HH(I*2-1,3)=FINP2(I)*W(5)*A/((1+A)*(1+A))
1640     HH(I*2,3)  =FINP2(I)*W(7)*A/((1+A)*(1+A))
1650     HH(I*2-1,4)=FINP2(I)*W(6)*B/((1+B)*(1+B))
1660     HH(I*2,4)  =FINP2(I)*W(8)*B/((1+B)*(1+B))
1670     HH(I*2-1,5)=1/(1+A)
1680     HH(I*2,5)=0
1690     HH(I*2-1,6)=1/(1+B)
```

```
1700     HH(I*2,6)=0
1710     HH(I*2-1,7)=0
1720     HH(I*2,7)=1/(1+A)
1730     HH(I*2-1,8)=0
1740     HH(I*2,8)=1/(1+B)
1750     NEXT I
1760 '
1770 '   ------------ CALCULATION OF {Y} & OUTPUT ERROR -------------
1780 '
1790     ER1=0:ER2=0
1800     FOR I = 1 TO NDP
1810     SUM1=0:SUM2=0
1820     FOR J = 1 TO 8
1830     SUM1=SUM1+HH(I*2-1,J)*W(J)
1840     SUM2=SUM2+HH(I*2,J)*W(J)
1850     NEXT J
1860     YY(I*2-1)=OUT1(I)-H(I*2-1)+SUM1
1870     YY(I*2)  =OUT2(I)-H(I*2)  +SUM2
1880     NEXT I
1890 '
1900     FOR I = 1 TO NDP
1910     ER1=ER1+(OUT1(I)-H(I*2-1))^2
1920     ER2=ER2+(OUT2(I)-H(I*2))^2
1930     NEXT I
1940     PRINT ER1/NDP,ER2/NDP
1950 '
1960 '   -------------------- KALMAN FILTERING --------------------
1970 '
1980     GOSUB *KALMAN
1990 '
2000 '   ----------------------- PRINT OUT ------------------------
2010 '
2020     PRINT "ITERATION = ";ICOUNT:ICOUNT=ICOUNT+1:PRINT #1,ICOUNT
2030     FOR I = 1 TO 8
2040     PRINT USING "REF##.## ";WI(I);
2050     NEXT I
2060     PRINT
2070     FOR I = 1 TO 8
2080     PRINT USING "CAL##.########### ";W(I)
2090     PRINT #1,W(I);
2100     NEXT I
2110     PRINT:PRINT #1,
2120 '
2130     A$=INKEY$:IF A$="" THEN GOTO 1420
2140     CLOSE
2150     OPEN "W.DAT" FOR OUTPUT AS #1
2160     FOR I = 1 TO 8
2170     PRINT #1,W(I)
2180     NEXT I
2190     PRINT #1,NDP
2200 '
2210     CLOSE:STOP
2220     END
2230 '
2240 '   ---------------------- KALMAN FILTER --------------------
2250 '
2260     *KALMAN
2270 '    R1=1D-10
2280 '    UU=1
2290      FOR I = 1 TO N
2300      FOR J = 1 TO N
2310      U(I,J)=0
2320      IF I=J THEN  U(I,J)=UU
2330      NEXT J
2340      NEXT I
2350 '
2360      MXE=8
2370      FOR L = 1 TO N
2380      FOR J = 1 TO MXE
2390      YY(L)=YY(L)-HH(L,J)*W(J)
2400      NEXT J
2410      FOR  J = MXE TO 2 STEP -1
```

```
2420      FF(J)=HH(L,J)
2430      FOR I = 1 TO J-1
2440      FF(J)=FF(J)+U(I,J)*HH(L,I)
2450      NEXT I
2460      GG(J)=U(J,J)*FF(J)
2470      NEXT J
2480      FF(1)=HH(L,1)
2490      GG(1)=U(1,1)*FF(1)
2500      ALPHA=R1+GG(1)*FF(1)
2510      GAMMA=1!/ALPHA
2520      U(1,1)=R1*GAMMA*U(1,1)
2530      FOR J = 2 TO MXE
2540      BETA=ALPHA
2550      ALPHA=ALPHA+GG(J)*FF(J)
2560      LAMBDA=FF(J)*GAMMA
2570      GAMMA=1!/ALPHA
2580      U(J,J)=BETA*GAMMA*U(J,J)
2590      FOR I = 1 TO J-1
2600      BETA=U(I,J)
2610      U(I,J)=BETA-GG(I)*LAMBDA
2620      GG(I)=GG(I)+GG(J)*BETA
2630      NEXT I
2640      NEXT J
2650      YY(L)=YY(L)*GAMMA
2660      FOR J = 1 TO MXE
2670      W(J)=W(J)+GG(J)*YY(L)
2680      NEXT J
2690      NEXT L
2700      RETURN
```