Appendix A

A.1 Classical Optimization Methods

This appendix reviews the classical methods of calculus for finding a solution that maximizes or minimizes a function of a single variable. It is assumed that the functions considered possess continuous first and second derivatives everywhere.

Consider a function of a single variable, such as that shown in Figure A.1. A necessary condition for a particular solution $x = x^*$ to be either a minimum or a maximum is that

$$\frac{df(x)}{dx} = 0 \qquad \text{at } x = x^*. \tag{A.1}$$

Thus in Figure A.1, there are five solutions satisfying these conditions. To obtain more information about these five critical points, it is necessary to examine the second derivative. Thus, if

$$\frac{d^2 f(x)}{dx^2} > 0 \quad \text{at } x = x^*, \tag{A.2}$$

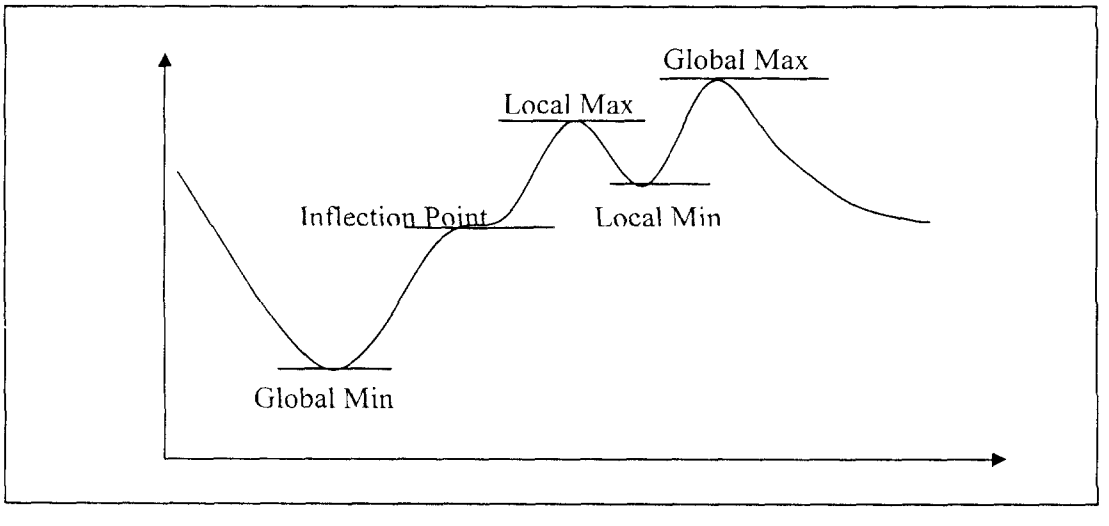


Figure A.1: A function having several maxima and minima.

then x^* must be at least a local minimum [that is, $f(x^*) \leq f(x)$ for all x sufficiently close to x^*]. To find a global minimum [i.e., a solution x^* suxh that $f(x^*) \leq f(x)$ for all x], it is necessary to compare the local minima and identify the one that yields the

smallest value of $f(x)$. If this value is less than $f(x)$ as $x \longrightarrow -\infty$ and as $x \longrightarrow +\infty$ (or at the endpoints of the function, if it is defined only over a finite interval), then this point is a global minimum as shown in Figure A.1. The global maximum is identified in an analogous way where the condition for x^* to be at least a local maximum is that

$$\frac{d^2 f(x)}{dx^2} < 0 \quad \text{at } x = x^*, \tag{A.3}$$

APPENDIX B

B.1 The Derivation OF The Extended Kalman Filter (EKF)

We will first present the derivation of Kalman Filter (KF) and then followed by the application of KF filtering technique into the weight updating algorithm of Multi Layer Perceptron Network(MLPN). The filtering technique of KF can be applied to solving problems such as optimal prediction, noise filtering and stochastic optimal control. Adaptive gain tuning capability is the main characteristic of the KF. KF was developed to optimally estimate state vector x(k) of a linear system by output y(k) with noisy measurements and process noise. The derivation of KF presented here is due to T. K. Chang [68, 69].

A linear system in its discrete form may be expressed as below

$$x(k+1) = A(k)x(k) + B(k)u(k) + e_1(k) \tag{B.1}$$

$$y(k) = C(k)x(k) + e_2(k) \tag{B.2}$$

where $e_1(k)$ is system noise and $e_2(k)$ is measurement noise. $e_1(k)$ and $e_2(k)$ are white noise, having zero mean and being independent with each other. $A(k)$ is called the transition matrix taking the state $x(k)$ from time k to time k+1 in the absence of the forcing function, $B(k)$ which acts upon the input, $u(k)$. $C(k)$ is the measurement matrix. Assume $A(k), B(k), C(k)$ and $y(k)$ are known, the criterion for the optimal estimation is the minimum covariance of the estimation error. Assume initial state $x(0)$ and noise $e_1(k)$ and $e_2(k)$ satisfying the following conditions:

$$E[x(0)] = \hat{x}(0) \tag{B.3}$$

$$E[e_1(k)] = 0 \tag{B.4}$$

$$E[e_2(k)] = 0 \tag{B.5}$$

$$E[e_1(i)e_2(j)^T] = 0 \tag{B.6}$$

$$E[e_1(i)e_1(j)^T] = E_1(i)\delta_{i,j} \tag{B.7}$$

$$E[e_2(i)e_2(j)^T] = E_2(i)\delta_{i,j} \tag{B.8}$$

$$E\{[x(0) - x_0(0)]e_1(k)^T\} = 0 \tag{B.9}$$

$$E\{[x(0) - x_0(0)]e_2(k)^T\} = 0 \tag{B.10}$$

where $\delta_{i,j}$ is the Kronecker delta,

$$\delta_{i,j} = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}$$ where $\neq$ means 'not equal'.

The estimation error is $e_k(k) = x(k) - \hat{x}(k)$ (B.11)

In the following discussion, at $k = n$, by given $y(0), y(1),..., y(n)$, where $k = 1,2,...,n$, KF is estimating optimal $\hat{x}(k)$ such that a positive definite matrix $E_k(k) = E[e_k(k)e_k(k)^T]$ is minimum.

The current estimation-type KF has the following form.

Prediction: $x_0(k+1) = A(k)\hat{x}(k) + B(k)u(k)$ (B.12)

Correction:

$$\hat{x}(k) = x_0(k) + K_k(k)[y(k) - C(k)x_0(k)] \tag{B.13}$$

$$= x_0(k) + K_k(k)[C(k)x(k) + e_2(k) - C(k)x_0(k)] \tag{B.14}$$

$$= x_0(k) + K_k(k)C(k)[x(k) - x_0(k)] + K_k(k)e_2(k) \tag{B.15}$$

where $x_0(k)$ is the prediction of $x(k)$ based on $y(0), y(1),..., y(k-1)$ information, and $\hat{x}(k)$ is the correction of $x_0(k)$ based on $y(k)$ information. Therefore, $e_k(k)$ can be rewritten as

$$e_k(k) = x(k) - x_0(k) - K_k(k)C(k)[x(k) - x_0(k)] - K_k(k)e_2(k) \tag{B.16}$$

$$= [I - K_k(k)C(k)][x(k) - x_0(k)] - K_k(k)e_2(k) \tag{B.17}$$

Then, $e_k(k+1)$ can be expressed as

$$e_k(k+1) = [I - K_k(k+1)C(k+1)][x(k+1) - x_0(k+1)] - K_k(k+1)e_2(k+1) \quad \text{(B.18)}$$

$$= [I - K_k(k+1)C(k+1)][A(k)\{x(k) - \hat{x}(k)\} + e_1(k)] - K_k(k+1)e_2(k+1) \quad \text{(B.19)}$$

$$= [I - K_k(k+1)C(k+1)[A(k)e_k(k) + e_1(k)] - K_k(k+1)e_2(k+1) \quad \text{(B.20)}$$

$$= L(k+1)e_k(k) + V(k+1) \quad \text{(B.21)}$$

where

$$L(k+1) = [I - K_k(k+1)C(k+1)]A(k) \quad \text{(B.22)}$$

$$V(k+1) = [I - K_k(k+1)C(k+1)]e_1(k) - K_k(k+1)e_2(k+1) \quad \text{(B.23)}$$

$$= L(k+1)A^{-1}(k)e_1(k) - K_k(k+1)e_2(k+1) \quad \text{(B.24)}$$

Notice that

$$e_k(k) = L(k)e_k(k-1) + V(k) \quad \text{(B.25)}$$

$$= L(k)[L(k-1)e_k(k-2) + V(k-1)] + V(k) \quad \text{(B.26)}$$

$$= L(k)L(k-1)...L(1)e_k(0) + L(2)V(1) + ... + L(k)V(k-1) + V(k) \quad \text{(B.27)}$$

and

$$E[V(i)V(j)^T]$$

$$= E\{[L(i)A^{-1}(i-1)e_1(i-1) - K_k(i)e_2(i)][L(j)A^{-1}(j-1)e_1(j-1) - K_k(j)e_2(j)]^T\} \quad \text{(B.28)}$$

$$= E\{L(i)A^{-1}(i-1)e_1(i-1)e_1(j-1)^T A^{-1}(j-1)^T L(j)^T \quad \text{(B.29)}$$

$$- K_k(i)e_2(i)e_1(j-1)^T A^{-1}(j-1)^T L(j)^T$$

$$- L(i)A^{-1}(i-1)e_1(i-1)e_2(j)^T K_k(j)^T + K_k(i)e_2(i)e_2(j)^T K(j)^T\}$$

$$= E\{[L(i)A^{-1}(i-1)e_1(i-1)e_1(j-1)^T A^{-1}(j-1)^T L(j)^T + K_k(i)e_2(i)e_2(j)^T K_k(j)^T\} \quad \text{(B.30)}$$

if $i = j$, then

$$E[V(i)V(j)^T] = L(i)A^{-1}(i-1)E_1(i-1)A^{-1}(i-1)^T L(i)^T + K_k(i)E_2(i)K(i)^T \quad \text{(B.31)}$$

if $i \neq j$, then

$$E[V(i)V(j)^T] = 0 \quad \text{(B.32)}$$

also

$$E[V(k)e_k(k-1)^T]$$

$$= E\{V(k)[L(k-1)L(k-2)...L(1)e_k(0) + L(2)V(1) + ... + L(k-1)V(k-2) + V(k-1)]^T\} \quad \text{(B.33)}$$

$$= [L(k-1)L(k-2)...L(1)]^T E[V(k)e_k(0)^T] \quad \text{(B.34)}$$

$$= 0 \quad \text{(B.35)}$$

by assuming that

$$e_k(0) = x(0) - \hat{x}(0) = x(0) - x_0(0) \tag{B.36}$$

Now, we are in the position to derive an expression for

$$E_k(k) = E[e_k(k)e_k(k)^T] \tag{B.37}$$
$$= E\{[L(k)e_k(k-1) + V(k)][L(k)e_k(k-1) + V(k)]^T\} \tag{B.38}$$
$$= E\{L(k)e_k(k-1)e_k(k-1)^T L(k)^T + V(k)e_k(k-1)^T L(k)^T \tag{B.39}$$
$$+ L(k)e_k(k-1)V(k)^T + V(k)V(k)^T\}$$
$$= L(k)E_k(k-1)L(k)^T + L(k)A^{-1}(k-1)E_1(k-1)A^{-1}(k-1)^T L(k)^T \tag{B.40}$$
$$+ K_k(k)E_2(k)K(k)^T$$
$$= L(k)A^{-1}(k-1)[A(k-1)E_k(k-1)A(k-1)^T + E_1(k-1)]A^{-1}(k-1)^T L(k)^T \tag{B.41}$$
$$+ K_k(k)E_2(k)K(k)^T$$
$$= L(k)A^{-1}(k-1)E_0(k)A^{-1}(k-1)^T L(k)^T + K_k(k)E_2(k)K_k(k)^T \tag{B.42}$$

where $E_0(k) = A(k-1)E_k(k-1)A(k-1)^T + E_1(k-1)$

$$E_k(k) = [I - K_k(k)C(k)]E_0(k)[I - K_k(k)C(k)]^T + K_k(k)E_2(k)K(k)^T \tag{B.43}$$
$$= E_0(k) - K_k(k)C(k)E_0(k) - E_0(k)C(k)^T K_k(k)^T \tag{B.44}$$
$$+ K_k(k)C(k)E_0(k)C(k)^T K_k(k)^T + K_k(k)E_2(k)K_k(k)^T$$
$$= E_0(k) - K_k(k)C(k)E_0(k) - E_0(k)C(k)^T K_k(k)^T + K_k(k)Q_k(k)K_k(k)^T \tag{B.45}$$

where $Q_k(k) = E_2(k) + C(k)E_0(k)C(k)^T$

$$E_k(k) = E_0(k) - K_k(k)C(k)E_0(k) + [K_k(k) - E_0(k)C(k)^T Q_k(k)^{-1}]Q_k(k)K_k(k)^T \tag{B.46}$$
$$= E_0(k) + [K_k(k) - E_0(k)C(k)^T Q_k(k)^{-1}]Q_k(k)K_k(k)^T \tag{B.47}$$
$$- [K_k(k) - E_0(k)C(k)^T Q_k(k)^{-1}]C(k)E_0(k) - E_0(k)C(k)^T Q_k(k)^{-1}C(k)E_0(k)$$
$$= E_0(k) - E_0(k)C(k)^T Q_k(k)^{-1}C(k)E_0(k) \tag{B.48}$$
$$+ [K_k(k) - E_0(k)C(k)^T Q_k(k)^{-1}][Q_k(k)K_k(k)^T - C(k)E_0(k)]$$
$$= E_0(k) - E_0(k)C(k)^T Q_k(k)^{-1}C(k)E_0(k) \tag{B.49}$$
$$+ [K_k(k) - E_0(k)C(k)^T Q_k(k)^{-1}]Q_k(k)[K_k(k)^T - Q_k(k)^{-1}C(k)E_0(k)]$$
$$= E_0(k) - E_0(k)C(k)^T Q_k(k)^{-1}C(k)E_0(k) + R_k(k)Q_k(k)R_k(k)^T \tag{B.50}$$

where
$$R_k(k) = K_k(k) - E_0(k)C(k)^T Q_k(k)^{-1} \tag{B.51}$$
$$R_k(k)^T = K_k(k)^T - Q_k(k)^{-1}C(k)E_0(k) \tag{B.52}$$

since $Q_k(k) = Q_k(k)^T$ (B.53)

$E_0(k) = E_0(k)^T$ (B.54)

Hence $E_k(k)$ is minimum when $R_k(k) = 0$ (B.55)

Hence $K_k(k) = E_0(k)C(k)^T Q_k(k)^{-1}$

$K_k(k) = E_0(k)C(k)^T[E_2(k) + C(k)E_0(k)C(k)^T]^{-1}$ (B.56)

Now

$E_{k\,\mathrm{min}imum}(k) = [I - E_0(k)C(k)^T Q_k(k)^{-1}C(k)]E_0(k) = [I - K_k(k)C(k)]E_0(k)$ (B.57)

By matrix inversion lemna

$[A - BCD]^{-1} = A^{-1} - A^{-1}B[C^{-1} + DA^{-1}B]^{-1}DA^{-1}$ (B.58)

$K_k(k) = E_0(k)C(k)^T\{E_2(k)^{-1} + E_2(k)^{-1}C(k)[E_0(k)^{-1} - C(k)^T E_2(k)^{-1}C(k)]^{-1}$ (B.59)
$C(k)^T E_2(k)^{-1}\}$

$= E_0(k)C(k)^T E_2(k)^{-1}\{I + C(k)[E_0(k)^{-1} - C(k)^T E_2(k)^{-1}C(k)]^{-1}$ (B.60)
$C(k)^T E_2(k)^{-1}\}$

$= E_0(k)C(k)^T E_2(k)^{-1}[I + C(k)E_0(k)C(k)^T E_2(k)^{-1}]^{-1}$ (B.61)

$K_k(k)[I + C(k)E_0(k)C(k)^T E_2(k)^{-1}] = E_0(k)C(k)^T E_2(k)^{-1}$ (B.62)

$K_k(k) = E_0(k)C(k)^T E_2(k)^{-1} - K_k(k)C(k)E_0(k)C(k)^T E_2(k)^{-1}$ (B.63)

$= [I - K_k(k)C(k)]E_0(k)C(k)^T E_2(k)^{-1}$ (B.64)

$= E_k(k)C(k)^T E_2(k)^{-1}$ (B.65)

Summary

System:

$x(k+1) = A(k)x(k) + B(k)u(k) + e_1(k)$ (B.66)

$y(k) = C(k)x(k) + e_2(k)$ (B.67)

Prediction:

$x_0(k+1) = A(k)\hat{x}(k) + B(k)u(k)$ (B.68)

$E_0(k+1) = A(k)E_k(k)A(k)^T + E_1(k)$ (B.69)

Correction:

$$\hat{x}(k) = x_0(k) + K_k(k)[y(k) - C(k)x_0(k)] \tag{B.70}$$

$$E_k(k) = [I - K_k(k)C(k)]E_0(k) \tag{B.71}$$

$$K_k(k) = E_0(k)C(k)^T[E_2(k) + C(k)E_0(k)C(k)^T]^{-1} \tag{B.72}$$

B.2 Development of EKF-Based Multi Layer Perceptron Network (MLPN)

In the previous section, we present the derivation of Extended Kalman Filter (EKF) algorithm. In order to apply this filtering algorithm for the estimation of the optimum state vector of Multi Layer Perceptron Network (MLPN), $\hat{x}(k)$ is replaced by the signal state vector of the MLPN, $\hat{\theta}(k)$. The other parameters such as $E_k(k)$, $K_k(k)$, $C(k)$, $E_0(k)$ and the predicted state vector $x_0(k)$ are replaced by $E_w(k)$, $K_w(k)$, $C_w(k)$, $E_w(k-1)$ and the previous state vector of the MLPN, $\hat{\theta}(k-1)$ respectively. $\hat{y}|_{k|k-1}$ is used to replace $C(k)x_0(k)$ in equation (B.70) since it is the output vector of the MLPN at time k with the previously estimated state vector, $\hat{\theta}(k-1)$. $R|_{k|k-1}$ is used to replace $E_2(k)$ in equation (B.72) since $E_2(k)$ cannot be determined. Hence, we have the following equations:

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K_w(k)[y(k) - \hat{y}|_{k|k-1}] \tag{B.73}$$

$$E_w(k) = [I - K_w(k)C_w(k)]E_w(k-1) \tag{B.74}$$

$$K_w(k) = E_w(k-1)C_w(k)^T[R|_{k|k-1} + C_w(k)E_w(k-1)C_w(k)^T]^{-1} \tag{B.75}$$

where

$\hat{y}|_{k|k-1} = \hat{g}[\hat{\theta}(k-1), x(k)]$ and $x(k)$ is used to denote the input vector to MLPN.

$$C_w(k) = \left.\frac{\partial \hat{y}(k)}{\partial \theta}\right|_{\theta=\hat{\theta}(k-1)} \tag{B.76}$$

$R|_{k|k-1}$ is an unknown priori error covariance matrix.

$$R|_{k|k-1} = R(k-1) + \frac{1}{k}\{[y(k) - \hat{y}|_{k|k-1}][y(k) - \hat{y}|_{k|k-1}]^T - R(k-1)\} \quad \text{(B.77)}$$

$$R(k) = R(k-1) + \frac{1}{k}\{[y(k) - \hat{y}(k)|_{\theta=\hat{\theta}(k)}][y(k) - \hat{y}(k)|_{\theta=\hat{\theta}(k)}]^T - R(k-1)\} \quad \text{(B.78)}$$

An EKF-based MLPN on-line training algorithm is presented here for adaptive network.

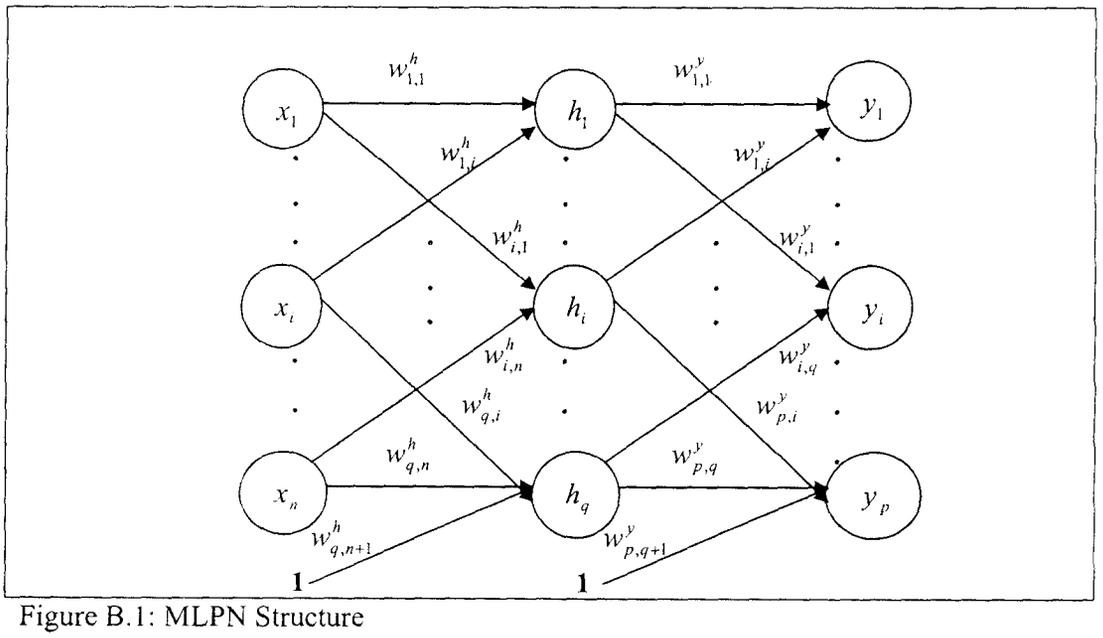


Figure B.1: MLPN Structure

$$\hat{y}(k) = W^y \begin{bmatrix} h(k) \\ 1 \end{bmatrix} \quad \text{(B.79)}$$

$$h(k) = f[z(k)] \quad \text{(B.80)}$$

$$z(k) = W^h \begin{bmatrix} x(k) \\ 1 \end{bmatrix} \quad \text{(B.81)}$$

where the input x(k) is a column vector of length n,

the hidden layer's output h(k) is a column vector of length p which can be expressed as

$$h(k) = \frac{1}{1 + \exp(-z(k))} \quad \text{(B.82)}$$

The weight, $w_{j,i}^{h}$ represents strength of the synaptic connection between the ith neuron of the input layer and the jth neuron of the hidden layer whereas the weight, $w_{j,i}^{y}$ represents strength of the synaptic connection between the ith neuron of the hidden layer and the jth neuron of the output layer as shown in Figure B.1.

The output layer's output y(k) is a column vector of length p, hence the MLPN consists of three layers only. In order to apply the KF algorithm, the weight matrices W^{h} and W^{y} of the MLPN are formulated as a parameter vector.

$$\theta = \begin{bmatrix} \theta^{h} \\ \theta^{y} \end{bmatrix} = \begin{bmatrix} (W_1^{h})^T \\ \cdot \\ \cdot \\ \cdot \\ (W_q^{h})^T \\ (W_1^{y})^T \\ \cdot \\ \cdot \\ \cdot \\ (W_p^{y})^T \end{bmatrix} = \begin{bmatrix} w_{1,1}^{h} \\ \vdots \\ w_{1,n+1}^{h} \\ \vdots \\ w_{q,1}^{h} \\ \vdots \\ w_{q,n+1}^{h} \\ w_{1,1}^{y} \\ \vdots \\ w_{1,q+1}^{y} \\ \vdots \\ w_{p,1}^{y} \\ \vdots \\ w_{p,q+1}^{y} \end{bmatrix} \quad \text{where } W^{h} = \begin{bmatrix} W_1^{h} \\ \cdot \\ \cdot \\ \cdot \\ W_q^{h} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{h} & \cdots & w_{1,n+1}^{h} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ w_{q,1}^{h} & \cdots & w_{q,n+1}^{h} \end{bmatrix} \quad \text{where } W^{y} = \begin{bmatrix} W_1^{y} \\ \cdot \\ \cdot \\ \cdot \\ W_p^{y} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{y} & \cdots & w_{1,q+1}^{y} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ w_{p,1}^{y} & \cdots & w_{p,q+1}^{y} \end{bmatrix} \tag{B.83}$$

Where (W_i^{h}) is the ith row vector in (W^{h}) and (W_i^{y}) is the ith row vector in (W^{y}). The MLPN model with the optimum parameter vector, θ can be described by the following equations,

$$\theta(k+1) = \theta(k) \tag{B.84}$$

$$y(k) = \hat{y}(k) + e_m(k) \tag{B.85}$$

where

$$\hat{y}(k) = \hat{g}[\theta(k), x(k)] = W^y\left[\frac{1}{1+\exp(-W^h x(k))}\right] \tag{B.86}$$

and $e_m(k) \in \Re^p$ is assumed to be a white noise vector and regarded as a modelling error. The EKF-based on-line learning algorithm for MLPN is repeated here for convenience:

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K_w(k)[y(k) - \hat{y}|_{k|k-1}] \tag{B.87}$$

$$E_w(k) = [I - K_w(k)C_w(k)]E_w(k-1) \tag{B.88}$$

$$K_w(k) = E_w(k-1)C_w(k)^T[R|_{k|k-1} + C_w(k)E_w(k-1)C_w(k)^T]^{-1} \tag{B.89}$$

where

$$\hat{y}|_{k|k-1} = \hat{g}[\hat{\theta}(k-1), x(k)] \tag{B.90}$$

$$C_w(k) = \left.\frac{\partial \hat{y}(k)}{\partial \theta}\right|_{\theta=\hat{\theta}(k-1)} \tag{B.91}$$

$R|_{k|k-1}$ is an unknown priori error covariance matrix.

$$R|_{k|k-1} = R(k-1) + \frac{1}{k}\{[y(k) - \hat{y}|_{k|k-1}][y(k) - \hat{y}|_{k|k-1}]^T - R(k-1)\} \tag{B.92}$$

$$R(k) = R(k-1) + \frac{1}{k}\{[y(k) - \hat{y}(k)|_{\theta=\hat{\theta}(k)}][y(k) - \hat{y}(k)|_{\theta=\hat{\theta}(k)}]^T - R(k-1)\} \tag{B.93}$$

At this stage, we need to introduce the Jacobian matrix since it will be used extensively in the later section.

The Jacobian of T is denoted by

$$J(u,v,w) = \frac{\partial(x,y,z)}{\partial(u,v,w)} = \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} & \frac{\partial x}{\partial w} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} & \frac{\partial y}{\partial w} \\ \frac{\partial z}{\partial u} & \frac{\partial z}{\partial v} & \frac{\partial z}{\partial w} \end{vmatrix} \tag{B.94}$$

where T is the transformation from $uvw-space$ to $xyz-space$ defined by the equations

$$x = x(u,v,w) \qquad , y = y(u,v,w) \qquad , z = z(u,v,w)$$

The $C_w(k) \in \Re^{p\times[q(n+1)+p(q+1)]}$ is the Jacobian matrix of model output, $\hat{y}(k)$ with respect to θ, and can be expressed as

$$C_w(k) = \frac{\partial \hat{y}(k)}{\partial \theta}\bigg|_{\theta=\hat{\theta}(k-1)} = \left[\frac{\partial \hat{y}(k)}{\partial \theta^h}\bigg|_{\theta=\hat{\theta}(k-1)} \quad \frac{\partial \hat{y}(k)}{\partial \theta^y}\bigg|_{\theta=\hat{\theta}(k-1)}\right] \tag{B.95}$$

From equation B.3, the term $\frac{\partial \hat{y}(k)}{\partial \theta^h}\Big|_{\theta=\hat{\theta}(k-1)}$ can be further expressed as

$$\frac{\partial \hat{y}(k)}{\partial \theta^h}\bigg|_{\theta=\hat{\theta}(k-1)} = \left[\frac{\partial \hat{y}(k)}{\partial h(k)}\frac{\partial h(k)}{\partial z(k)}\frac{\partial z(k)}{\partial \theta^h}\right]_{\theta=\hat{\theta}(k-1)} \quad \text{by using the chain rule.} \tag{B.96}$$

where

$$\frac{\partial \hat{y}(k)}{\partial h(k)}\bigg|_{\theta=\hat{\theta}(k-1)} = \begin{bmatrix} \hat{w}^{y}_{1,1}(k-1) & \cdots & \hat{w}^{y}_{1,q}(k-1) \\ \vdots & & \vdots \\ \hat{w}^{y}_{p,1}(k-1) & \cdots & \hat{w}^{y}_{p,q}(k-1) \end{bmatrix} \tag{B.97}$$

and

$$\frac{\partial h(k)}{\partial z(k)} = \left[\begin{matrix} \frac{\partial h_1(k)}{\partial z_1(k)} & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & \frac{\partial h_q(k)}{\partial z_q(k)} \end{matrix}\right]_{\theta=\hat{\theta}(k-1)} \quad \text{This is a diagonal matrix.} \tag{B.98}$$

where

$$\frac{\partial h_i(k)}{\partial z_i(k)}\bigg|_{\theta=\hat{\theta}(k-1)} = h_i(k)\big|_{\theta=\hat{\theta}(k-1)}\left[1-h_i(k)\big|_{\theta=\hat{\theta}(k-1)}\right], i = 1,\ldots,q \tag{B.99}$$

$$h_i(k)\big|_{\theta=\hat{\theta}(k-1)} = \frac{1}{1+\exp[-z_i(k)]\big|_{\theta=\hat{\theta}(k-1)}} \tag{B.100}$$

$$z_i(k)\Big|_{\theta=\hat{\theta}(k-1)} = \hat{W}_i^{h}(k-1)\begin{bmatrix} x(k) \\ 1 \end{bmatrix} \tag{B.101}$$

Proof: Let $h(k) = f[z(k)] = \dfrac{1}{1+\exp[-z(k)]}$ (B.102)

$$\frac{df[z(k)]}{dz(k)} = -(1+\exp[-z(k)])^{-2}.\exp[-z(k)].(-1) = \frac{\exp(-z(k))}{(1+\exp[-z(k)])^2} \tag{B.103}$$

since $\exp[-z(k)] = \dfrac{1}{f[z(k)]} - 1$ and $f[z(k)]^2 = \dfrac{1}{(1+\exp[-z(k)])^2}$

$$\frac{df[z(k)]}{dz(k)} = \left[\frac{1}{f[z(k)]} - 1\right] f[z(k)]^2 = f[z(k)](1 - f[z(k)]) \tag{B.104}$$

APPENDIX C

C.1 Matlab Source Code for Training of MLPN

```
function [Ew,Rk,sse,weight]=kfneunetnn(jinput, joutput, Hiddenlen,...
  dk,ki,Ew,Rk,weight,sse)

%       jinput is a matrix consisting of columns of input vector i.e. [1 0 0 1 ; 0 1 0 1]
%       jinput = [1001
%                 0101]
%       joutput is a matrix consisting of columns of desired  output vector

%       i.e. [1 1 0 0]

%       joutput = [1100]

%       The above is an example of the XOR (exclusive-OR) problem.

%       The truth table of the XOR logic is as shown below:
%       _____________________________
%       |x1     |x2     |y      |
%       |-------|-------|-------|
%       |0      |0      |0      |
%       |0      |1      |1      |
%       |1      |0      |1      |
%       |1      |1      |0      |
%       ---------------------------

%       Hiddenlen is a scalar value equal to

%       the number of hidden neurons in the hidden layer of the MLPN

%       The MLPN structure is as shown below:

%       Input           Hidden          Output
%       Layer           Layer           Layer

%                                 1st neuron
%                                /
%       o                 o                 o
%
%       o                 o                 o
%                          \
%       o                 o 2nd neuron      o
%                          \
%                                 3rd neuron
```

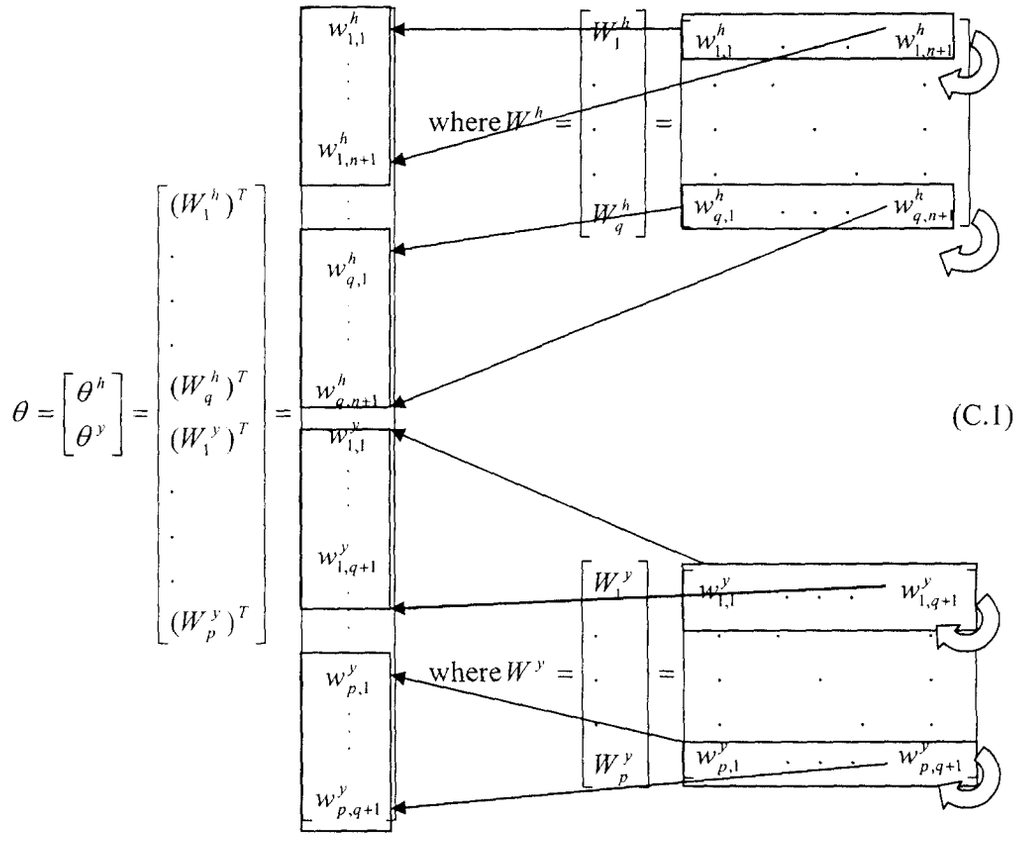
```
[Inputlen, NumberiPat]=size(jinput);
%       Inputlen is a scalar value equal to
%       the number of input neurons in the input layer of the MLPN
%       NumberiPat is the no. of input patterns/samples that are used for the training
[Outputlen, NumberoPat]=size(joutput);
%       Outputlen is a scalar value equal to
%       the number of output neurons in the input layer of the MLPN
%       NumberoPat is the no. of output patterns/targets that are used for the training
if nargin>3
%                                          1st argument   2nd argument   3rd argument
%                                                    \          /             /
%       function [Ew,Rk,sse,weight]=kfneunetnn(jinput, joutput, Hiddenlen,...
%         dk,ki,Ew,Rk,weight,sse)
%
%       If the no. of input argument (nargin) are more than 3, then
%       we assume that the parameters such as dk, ki, Ew, Rk, weight and sse
%       have been supplied by the user
else
%       Otherwise, the program will set all these parameters to their default values
  ki=0;
%       ki, the no. of training iterations that had been performed is set to default, 0
  dk=NumberiPat;
%       dk, the no. of patterns used for training is set equal to the no. of input patterns
  w=[Inputlen,Hiddenlen,Outputlen];
%       w, the MLPN structure is set to {Inputlen : Hiddenlen : Outputlen},e.g.{2:5:1}
```

```
calcu=w(3)*(w(2)+1)+w(2)*(w(1)+1);
%       calcu, the dimension of the Ew is set to p x ( q + 1 ) + q x ( n + 1 )
%       where
%       p = the no. of neurons in the output layer
%       q = the no. of neurons in the hidden layer
%       n = the no. of neurons in the input layer
Ew=eye(calcu);
%       p x ( q + 1 ) + q x ( n + 1 )
%               |--------------------|  ---
%               [1...0...0...0...0...0]  |
%               [0...1...0...0...0...0]  |
%       Ew =    [0...0...1...0...0...0]  |   p x ( q + 1 ) + q x ( n + 1 )
%               [0...0...0...1...0...0]  |
%               [0...0...0...0...1...0]  |
%               [0...0...0...0...0...1]  |
%                                        ---
%       Ew, the error covariance matrix for state estimation is set to default,
%       An Identity Matrix
Rk=smallrandn(w(3),w(3));
%       Rk, the error covariance matrix for measurement noise is set to default,
%       A  p x p Matrix made up of small random values with normal distribution
for i=1:2
   NumberInput=w(i)+1;
   weight{i}=smallrandn(w(i+1),NumberInput);
end
%       1st Loop: Creation of the hidden weight, W^h = w{1},  A q x ( n + 1 ) Matrix
%       that is made up of small random values with normal distribution
%       2nd Loop: Creation of the output weight, W^y = w{2},  A p x ( q + 1 ) Matrix
%       that is made up of small random values with normal distribution
```

```
end     % The ending of the above if – else – end statement.
xinput=jinput;          % xinput is assigned with the input patterns/samples, jinput

doutput=joutput;        % doutput is assigned with the output patterns/targets, joutput

figure(111), hold on    % A figure is created with no. 111 so as not to be overwritten

                        % incidentally! 'hold on' allows plotting on the same figure!

for k=1:NumberiPat      % Iterating from the 1st input pattern/sample to the last sample
  ki=ki+1;              % Incrementing the number of training iteration performed, ki
  x=xinput(:,k);        % x is assigned with the k-th input pattern/sample
  dyk=doutput(:,k);     % dyk is assigned with the k-th output pattern/target
  [yk,weight,Rk,Ew]=kalfilter(x,dyk,weight,Rk,Ew,k);

      %       Calling subroutine or function 'kalfilter' given at page xviii

  actualout=weight{1}*[x;1];

%       actualout is assigned with the output values from the hidden layer,

%       A q x 1 Vector

  activoutput=activation(actualout);
  realoutput(:,k)=weight{2}*[activoutput;1];

%       The k-th column of the realoutput,

%       A p x (number of input patterns) Matrix,

%       is assigned with the output values from the output layer at k-th iteration.

  ro=realoutput(:,k);

%       ro is assigned with the output values from the output layer at k-th iteration.


  e(:,k)=dyk-ro;

%       The cost function is calculated as follows:

%       dyk, the output patterns/targets

%       Minus

%       ro, the output values from the output layer at k-th iteration.
```

```
   sse(k)=sum(e(:,k).^2);

%       The sum square error is computed for each iteration, k.

   if k==1 & ki~=1     % If k equal to 1 and ki not equal to 1, then plot the following
      line([ki-1;ki],[sse(end);sse(1)]); %
   end %

   if mod(ki,dk)==0    % If all the input patterns has been used for the MLPN training
      dispsse=[ki-dk+1:ki;sse(k-dk+1:k)]'; % then plot the following
      line(dispsse(:,1),dispsse(:,2))
   end

end

************************************************

function [yk,weight,Rk,Ew]=kalfilter(x,dyk,weight,Rk,Ew,k)

% x contains a column vector of the input pattern/sample, e.g. [1;0;0;1].
% x = [1; 0; 0; 1]
% dyk contains a column vector of the output pattern/target, e.g. [1;0].

% dyk = [1; 0]
% weight is a cell consisting of w{1}, hidden weight and w{2}, output weight
% Rk is the error covariance matrix of the measurement noise.
% Ew is the error covariance matrix of the state estimation.
% k is an index value to keep track of the k-th input pattern/sample

NumberLayer=length(weight);

% NumberLayer is the number of layer for the MLPN excluding the input layer.

for i=1:NumberLayer % Iterate from i = (1) hidden layer to (2) output layer
   [b,c]=size(weight{i}); % At each layer, we determine
   out(i)=b; % the no. of output nodes, out(i) for the i-th layer, and
   in(i)=c; % the no. of input nodes, in(i) for the i-th layer including the bias.
end

x=[x;1]; % The input pattern/sample, x is appended with a 1 as the bias.
```

```
NumberInput=length(x); % The no. of input node for the hidden layer is determined

LayerInput=x; % The input values to the hidden layer, LayerInput is assigned with x

for i=1:NumberLayer % Iterate from i = (1) hidden layer to (2) output layer

   SOP{i}=weight{i}*LayerInput; % Compute the output of each layer
   LayerOutput{i}=activation(SOP{i}); % and apply the activation function (sigmoid)
   LayerInput=[LayerOutput{i};1]; % to obtained the bounded value between 0 and 1.

end

yk=SOP{end}; % we obtain the output values from the output layer before

% the sigmoidal function is applied at them

%       Initialize Weight-updating Algorithm Here.

dheta=[];       % we first initialize dheta, θ as an empty matrix, []
```



$$\theta = \begin{bmatrix} \theta^h \\ \theta^y \end{bmatrix} = \begin{bmatrix} (W_1^h)^T \\ \cdot \\ \cdot \\ \cdot \\ (W_q^h)^T \\ (W_1^y)^T \\ \cdot \\ \cdot \\ \cdot \\ (W_p^y)^T \end{bmatrix} \quad \text{(C.1)}$$

```
%       Then, we rearrange each row of the hidden weight matrix, W^h and
%       the output weight matrix, W^y into a state vector called dheta, θ
```

```
% the algorithm is given below
%**********************
for i=1:NumberLayer
   [NumberNeuron,temp]=size(weight{i});

   for j=1:NumberNeuron
      dheta=[dheta;(weight{i}(j,:)).'];
   end
end
% **********************
% for each layer of the MLPN (i.e. the hidden and output layer), we determine the
% no. of row in the corresponding weight matrix, then we transpose each of these
% row vectors into column vectors before joining them head to tail as shown above

HiddenOutput=[LayerOutput{1};1].';

% The output values from the hidden layer are appended with a 1 as the bias

LengthHidden=length(HiddenOutput);

% LengthHidden=the length of the hidden layer (including the bias) = q + 1

% where q is the no. of neurons in the hidden layer

p=out(end);

% p is the no. of neuron in the output layer
```

% we compute the derivatives of the MLPN output, $\hat{y}(k)$

```
% the formulas are given in Equation (4.51 & 4.52)

% the computational algorithm is given below
%**********************
A=HiddenOutput;

for i=1:p-1
   A=diagmx(A,HiddenOutput);
end
%**********************

ykDhetay=A;
```

% ykDhetay= the derivatives of the MLPN output, $\hat{y}(k)$ with respect to dheta-y, θ^y

```
HiZi=LayerOutput{1}.*(ones(LengthHidden-1,1)-LayerOutput{1});

% the derivatives of the hidden outputs, h_i(k) with respect to the net inputs, z_i(k)

% are given in Equation (4.46 & 4.47)

% the computational algorithm is given below
%**********************
B=HiZi(1);

for i=1:LengthHidden-2
   B=diagmx(B,HiZi(i+1));
end

%**********************

hizi=B;

%hizi=the derivatives of the hidden outputs, h_i(k) with respect to the net inputs, z_i(k)

InputLayer=x.';

% the derivatives of the net inputs, z_i(k) with respect to dheta-h, θ^h

% are given in Equation (4.50)

% the computational algorithm is given below

%**********************

C=InputLayer;

for i=1:LengthHidden-2
   C=diagmx(C,InputLayer);
end
%**********************

% C= the derivatives of the net inputs, z_i(k) with respect to dheta-h, θ^h

OutputWeight=weight{NumberLayer};
ykhk=OutputWeight;

%       ykhk=the derivatives of the MLPN output, ŷ(k) with respect to
%       the hidden outputs, h_i(k)
```

```
ykDhetah=ykhk(:,1:end-1)*hizi*C;

% ykDhetay= the derivatives of the MLPN output, ŷ(k) with respect to dheta-h, θ^h

% which are given in Equation (4.44)

Cw=[ykDhetah ykDhetay];

% Cw= the derivatives of the MLPN output, ŷ(k) with respect to dheta, θ

% which are given in Equation (4.43)

% Algorithm for updating dheta, θ is given below
%########################################

if k>1000
   k=1000;
end


Rkk=Rk+(1/k)*((dyk-yk)*(dyk-yk).'-Rk);

% The above expression follows from Equation (4.41)

% k value is limited to 1000 only as the (1/k) will approach 0 if k value is too large
Kw=Ew*Cw.'*inv(Rkk+Cw*Ew*Cw.');

% The above expression follows from Equation (4.39)

KC=Kw*Cw;

Iden=eye(length(KC));

Ew=(Iden-KC)*Ew;

% The above expression follows from Equation (4.38)

dheta=dheta+Kw*(dyk-yk);

% The above expression follows from Equation (4.37)

% Reverse of the process in Equation (C.1) to retrieve the weight vectors

% the computational algorithm is given below
%**********************
```

```
st=1; et=0;

for i=1:NumberLayer

   et=out(i)*in(i)+et;
   weight{i}=dheta(st:et);
   st=et+1;

end

for i=1:NumberLayer

   weight{i}=[reshape(weight{i},in(i),out(i))].';

end

%***********************

LayerInput=x;

for i=1:NumberLayer

   SOP{i}=weight{i}*LayerInput;
   LayerOutput{i}=activation(SOP{i});
   LayerInput=[LayerOutput{i};1];

end

yk=SOP{end};

% we compute the new outputs from the MLPN after updating of the weight vectors

Rk=Rk+(1/k)*((dyk-yk)*(dyk-yk).'-Rk);

% The above expression follows from Equation (4.42)

%########################################
% These are the subroutines or functions called by the main function

%***********************

function y=smallrandn(x1,x2)

y=0.1*randn(x1,x2);

%***********************
% The above function generates small random values with normal distribution
%***********************
```

```
function y=activation(x)

y=1./1+exp(-x);

%**********************

% The above function is the sigmoidal transfer function

% wich is used for bounding the hidden layer outputs

% to values between 0 and 1 i.e. [0,1].

% A function used for calling the EKF-based MLPN training algorithm

function [Ew,Rk,sse,weight,k]=testfilternn(jinput,...
  joutput,hidden,stopiter)

%       jinput is a matrix consisting of columns of input vector i.e. [1 0 0 1 ; 0 1 0 1]

%       jinput = [1001]
%                [0101]

%       joutput is a matrix consisting of columns of desired  output vector

%       i.e. [1 1 0 0]

%       joutput = [1100]

%       The above is an example of the XOR (exclusive-OR) problem.

%       The truth table of the XOR logic is as shown below:

%
%       ___________________________
%       |x1     |x2     |y      |
%       |-------|-------|-------|
%       |0      |0      |0      |
%       |0      |1      |1      |
%       |1      |0      |1      |
%       |1      |1      |0      |
%       ---------------------------
```

```
%       Hiddenlen is a scalar value equal to
%       the number of hidden neurons in the hidden layer of the MLPN
%       The MLPN structure is as shown below:

%       Input           Hidden          Output
%       Layer           Layer           Layer
%                         1st neuron
%                        /
%       o               o                o
%
%       o               o                o
%                        \
%       o               o 2nd neuron     o
%                        \
%                         3rd neuron

%       stopiter = the number of training iterations to be performed before stopping

figure(111),close

[Ew,Rk,sse,weight]=kfneunetnn(jinput,joutput,hidden);

[temp,num_pat]=size(jinput);

for k=num_pat:num_pat:stopiter

[Ew,Rk,sse,weight]=kfneunetnn(jinput,joutput,5,...
   num_pat,k,Ew,Rk,weight,sse);

end

zoom xon
```

% A function used for calling the EKF-based MLPN training algorithm online

```
function [Ew,Rk,sse,weight,k]=testfilteronline(jinput,...
   joutput,hidden,stopiter,Ew,Rk,sse,weight,k)

[temp,num_pat]=size(jinput);

[Ew,Rk,sse,weight]=kfneunetnn(jinput,joutput,hidden,...
   num_pat,k+num_pat,Ew,Rk,weight,sse);

for k=k+2*num_pat:num_pat:stopiter+k+num_pat

[Ew,Rk,sse,weight]=kfneunetnn(jinput,joutput,hidden,...
   num_pat,k,Ew,Rk,weight,sse);

end

zoom xon
```

% A function used for testing the performance of the MLPN

```
function [sse,ro]=testk(weight,target,input)
[temp,NumberiPat]=size(input);

for k=1:NumberiPat
   x=input(:,k)
   dyk=target(:,k);
   actualout=weight{1}*[x;1];
   activoutput=activation(actualout);
   realoutput(:,k)=weight{2}*[activoutput;1];
   ro(k)=realoutput(:,k);
   e(:,k)=dyk-ro(k);
   sse(k)=sum(e(:,k).^2);
end

function y=activation(x)
y=1./1+exp(-x);
```

C.2 Matlab Source Code for Classification of ECG signal with Hybrid Neural Fuzzy System

```
function varargout = gra10(varargin)

% GRA 10 Application M-file for gra10.fig

% FIG = GRA10 launch gra10 GUI

if nargin == 0 % Launch GUI

        fig=openfig(mfilename, 'reuse');

% Use system color scheme for figure:

set(fig, 'Color', get(0, 'defaultUicontrolBackgroundColor'));

% Generate a structure of handles to pass to callbacks,  and store it.

handles=guihandles(fig);

handles.button_state2=0;

handles.button_max2=1;

guidata(fig, handles);

if nargout > 0
        varargout{1}=fig;
end

elseif ischar(varargin{1}) % Invoke named subfunction or callback

try

        if (nargout)
          [varargout{1:nargout}]=feval( varargin{:}); % FEVAL switchyard
        else
          feval(varargin{:}); % FEVAL switchyard
        end

catch

        disp(lasterr);

end
```

```
% The subfunction name is composed using the object's Tag and the
% callback type separated by '_', e.g. 'slider2_Callback',
% 'figure1_CloseRequestFcn','axis1_ButtondownFcn'.
%
% H is the callback object's handle (obtained using GCBO).
%
% EVENTDATA is empty, but reserved for future use
%
% HANDLES is a structure containing handles of components in GUI using
% tags as fieldnames, e.g. handles.figure1, handles.slider2. This
% structure is created at GUI startup using GUIHANDLES and stored in
% the figure's application data using GUIDATA. A copy of the structure
% is passed to each callback. You can store additional information in this structure at %
GUI  startup, and you can change the structure during callbacks.
% Call guidata(h, handles) after changing your copy to replace the stored original so
% that subsequent callbacks see the updates. Type "help guihandles" and
% "help guidata" for more information.
% VARARGIN contains any extra arguments you have passed to the
% callback. Specify the extra arguments by editing the callback
% property in the inspector. By default, GUIDE sets the property to:
% <MFILENAME>('<SUBFUNCTION_NAME>',gcbo,[],guidata(gcbo))
%  Add any extra arguments after the last argument, before the final
% closing parenthesis.

Function varargout = source_edit_Callback(h, eventdata, handles, varargin)

Function varargout= store_edit_ Callback(h, eventdata, handles, varargin)

Function varargout=store_button_Callback(h, eventdata, handles, varargin)

Trfdata=num2str(get(handles.store_edit, 'String'));
Resultsum=handles.resultsum;
Save(trfdata, 'resultsum');

Function varargout=ok_button_Callback(h, eventdata, handles, varargin)
Data=get(handles.source_edit, 'String');
Load(data)
J12=y';
[lenb. lenc]=size(j12);
j=[j12;j12];
no1=j(1,:)-j(2,:);
for la=0:lenb-1
        o1(la+1,:)=no1;
        no1=j(la+2,:)-j(la+3,:);
        O2(la+1,:)=no1;
        O3(la+1,:)=o1(la+1,:)-o2(la+1,:);
end
```

```
Mo1=abs(o1);
Mo2=abs(o2);
Mo3=abs(o3);
Dmo1=mo1(:);
Dmo2=mo2(:);
Dmo3=mo3(:);
Max12=max(dmo1,dmo2);
Dmo4=dmo3>max12;
Mo4=reshape(dmo4,lenb,lenc);
Mo5=mo4.*mo3;

For fg=1:lenc

        Mo5fg(:,fg)=[mo5(end,fg);mo5(1:end-1,fg)];
        J12fg=j12(:,fg);
        Meanj=mean(j12fg);

        Varj=j12fg-meanj*ones(lenb,1);
        Varj=abs(varj);
        Varj(:,fg)=varj.*(mo5fg(:,fg)~=0);

        [md,mdI]=max(mo5fg(:,fg).*varj(:,fg));
        j12(:,fg)=[j12fg(mdI:end);j12fg(1:mdI-1)];
end

handles.y=j12';
handles.z=z;
guidata(h,handles)

function [nise,m,c]=normaly(ise)

Xmax=max(ise);
Xmin=min(ise);
Ymax=0.9;
Ymin=0.1;
Mc=inv([Xmax 1;Xmin 1])*[Ymax; Ymin];
M=mc(1);
C=mc(2);
Nise=ise*m+c;

Function varargout=exit_button_Callback(h, eventdata, handles, varargin)
Clf

Function varargout=mp_edit_Callback(h, eventdata, handles, varargin)
```

```
Function varargout=test_button_Callback(h, eventdata, handles, varargin)

Desired_output=str2double(get(handles.tar_edit, 'String'));
Save_weight=get(handles.weight_edit, 'String');
Save_weight2=get(handles.weight_edit2, 'String');
Save_weight3=get(handles.weight_edit3, 'String');
Load(save_weight)
Weight1=weight;
Load(save_weight2);
Weight2=weight;
Load(save_weight3);
Weight3=weight;
Data=get(handles.source_edit, 'String');
Load(data)
J12=y';
[lenb, lenc]=size(j12);
j=[j12;j12];
no1=j(1,:)-j(2,:);

for la=0:lenb-1

        o1(la+1,:)=no1;
        no1=j(la+2,:)-j(la+3,:);
        o2(la+1,:)=no1;
        o3(la+1,:)=o1(la+1,:)-o2(la+1,:);

end

mo1=abs(o1);
mo2=abs(o2);
mo3=abs(o3);
dmo1=mo1(:);
dmo2=mo2(:);
dmo3=mo3(:);
max12=max(dmo1,dmo2);
dmo4=dmo3>max12;
mo4=reshape(dmo4,lenb,lenc);
mo5=mo4.*mo3;
abc12=[0:10:100];
cba12=[0.5*ones(1,11); 1.5*ones(1,11); 2.5*ones(1,11); 3.5*ones(1,11)];
yqline3=0;
yqline2=0;
yqline=0;
yline=0;
yline2=0;
yline3=0;
xline=0;
xtest=0;
```

```
test_output2=0;
test_error2=0;
outpu=1:0.05:2.0;
outpusp=0:0.05:1.0;
outpusp4=-1:0.05:0.0;
outpusp5=1:0.05:2.0;
Outbar11=0;
Outbar12=0.5;
Outbar13=1;
Outbar14=-0.5;
Outbar15=1.5;
Outseita11=0.095;
Outseita12=0.25;
Outseita13=0.095;
Outk11=1;
Outk12=1;
Outk13=1;
Membe11=activfuzzy(outpu, outbar11, outk11, outseita11);
Membe12=activfuzzy(outpusp, outbar12, outk12, Desired_output);
Membe13=activfuzzy(outpu, outbar13, outk13, outseita13);
Membe14=activfuzzy(outpusp4, outbar14, outk12, Desired_output);
Membe15=activfuzzy(outpusp5, outbar15, outk12, Desired_output);
Outbar21=0;
Outbar22=0.5;
Outbar23=1;
Outbar24=-0.5;
Outbar25=1.5;
Outseita21=0.095;
Outseita22=0.25;
Outseita23=0.095;
Outk21=1;
Outk22=1;
Outk23=1;
Membe21=activfuzzy(outpu, outbar21, outk21, outseita21);
Membe22=activfuzzy(outpusp, outbar22, outk22, Desired_output);
Membe23=activfuzzy(outpu, outbar23, outk23, outseita23);
Membe24=activfuzzy(outpusp4, outbar24, outk22, Desired_output);
Membe25=activfuzzy(outpusp5, outbar25, outk22, Desired_output);
Outbar31=0;
Outbar32=0.5;
Outbar33=1;
Outbar34=-0.5;
Outbar35=1.5;
Outseita31=0;
Outseita32=0.25;
Outseita33=0.095;
Outk31=1;
Outk32=1;
Outk33=1;
```

```
Membe31=activfuzzy(outpu, outbar31, outk31, outseita31);
Membe32=activfuzzy(outpusp, outbar32, outk32, Desired_output);
Membe33=activfuzzy(outpu, outbar33, outk33, outseita33);
Membe34=activfuzzy(outpusp4, outbar34, outk32, Desired_output);
Membe35=activfuzzy(outpusp5, outbar35, outk32, Desired_output);
output22o=0;
output22p=0;
output22q=0;
output22r=0;
output22s=0;
output2o=0;
output2p=0;
output2q=0;
output2r=0;
output2s=0;
output22oa=0;
output22pa=0;
output22qa=0;
output22ra=0;
output22sa=0;
output2oa=0;
output2pa=0;
output2qa=0;
output2ra=0;
output2sa=0;

figure
subplot(2,1,2)
hold on
plot([1.5 1.5],[-1 4],'c-.')
plot([-0.5 3.5],[1.5 1.5],'c-.')
plot([3.5 3.5],[-1 4],'c.-')
plot([-0.5 3.5],[4 4],'c.-')
plot([-0.5 -0.5],[-1 4],'c.-')
plot([-0.5 3.5],[-1 -1],'c.-')
text(1,2,'STDP')
text(1.8,2,'NSR')
text(1.8,1,'SVT')
text(1,1,'TINV')

h1a=text(0,3.5,'0')
h2a=text(3,3.5,'0')
h3a=text(3,1,'0')
h4a=text(0,1,'0')

h1ab=text(0.4,3.5,'%')
h2ab=text(3.4,3.5,'%')
h3ab=text(3.4,1,'%')
h4ab=text(0.4,1,'%')
```

```
nsrn=0;
svtn=0;
stdpn=0;
stinvn=0;

for fg=1:lenc

   mo5fg(:,fg)=[mo5(end,fg);mo5(1:end-1,fg)];
   j12fg=j12(:,fg);
   meanj=mean(j12fg);
   subplot(2,2,1)
   cla
   hold on
   plot(j12fg)
   hline=line(1:lenb-1:lenb,meanj*ones(1,2));
   set(hline,'LineStyle','--','Color','c')
   varj=j12fg-meanj*ones(lenb,1);
   varj=abs(varj);
   varj(:,fg)=varj.*(mo5fg(:,fg)~=0);

   [md,mdI]=max(mo5fg(:,fg).* varj(:,fg));
   j12(:,fg)=[j12fg(mdI:end);j12fg(1:mdI-1)];

   output=weight1{1}*[j12(:,fg);1];
   output=1./(1+exp(-output));
   output2=weight1{2}*[output;1];
   member11=activfuzzy(output2,outbar11,outk11,outseita11);
   member13=activfuzzy(output2,outbar13,outk13,outseita13);
   if output2<0
      member12=activfuzzy(output2,outbar14,outk12,Desired_output);
   elseif output2>1
      member12=activfuzzy(output2,outbar15,outk12,Desired_output);
   else
      member12=activfuzzy(output2,outbar12,outk12,Desired_output);
   end
   aa1=[member11,member12,member13];

   output=weight2{1}*[j12(:,fg);1];
   output=1./(1+exp(-output));
   output22=weight2{2}*[output;1];
   member21=activfuzzy(output22,outbar21,outk21,outseita21);
   member23=activfuzzy(output22,outbar23,outk23,outseita23);
    if output22<0
      member22=activfuzzy(output22,outbar24,outk22,Desired_output);
   elseif output22>1
      member22=activfuzzy(output22,outbar25,outk22,Desired_output);
   else
      member22=activfuzzy(output22,outbar22,outk22,Desired_output);
   end
```

```
aa2=[member21,member22,member23];
[y2,i2]=sort(aa2);

output=weight3{1}*[j12(:,fg);1];
output=1./(1+exp(-output));
output23=weight3{2}*[output;1];
member31=activfuzzy(output23,outbar31,outk31,outseita31);
member33=activfuzzy(output23,outbar33,outk33,outseita33);
 if output23<0
   member32=activfuzzy(output23,outbar34,outk32,Desired_output);
elseif output23>1
   member32=activfuzzy(output23,outbar35,outk32,Desired_output);
else
   member32=activfuzzy(output23,outbar32,outk32,Desired_output);
end
aa3=[member31,member32,member33];

if abs(output23)<0.185
   aa(1,:)=[aa1(1).*aa2(1).*aa3(1)   1.0000   1.0000   1.0000   3.0000];
   aa(2,:)=[aa1(1).*aa2(1).*aa3(2)   1.0000   1.0000   2.0000   2.0000];
   aa(3,:)=[aa1(1).*aa2(1).*aa3(3)   1.0000   1.0000   3.0000   2.0000];
   aa(4,:)=[aa1(1).*aa2(2).*aa3(1)   1.0000   2.0000   1.0000   3.0000];
   aa(5,:)=[aa1(1).*aa2(2).*aa3(2)   1.0000   2.0000   2.0000   2.0000];
   aa(6,:)=[aa1(1).*aa2(2).*aa3(3)   1.0000   2.0000   3.0000   2.0000];
   aa(7,:)=[aa1(1).*aa2(3).*aa3(1)   1.0000   3.0000   1.0000   3.0000];
   aa(8,:)=[aa1(2).*aa2(1).*aa3(1)   2.0000   1.0000   1.0000   3.0000];
   aa(9,:)=[aa1(2).*aa2(1).*aa3(2)   2.0000   1.0000   2.0000   1.0000];
   aa(10,:)=[aa1(2).*aa2(1).*aa3(3)  2.0000   1.0000   3.0000   1.0000];
   aa(11,:)=[aa1(2).*aa2(2).*aa3(1)  2.0000   2.0000   1.0000   3.0000];
   aa(12,:)=[aa1(2).*aa2(2).*aa3(3)  2.0000   2.0000   3.0000   4.000];
   aa(13,:)=[aa1(2).*aa2(3).*aa3(1)  2.0000   3.0000   1.0000   3.0000];
   aa(14,:)=[aa1(2).*aa2(3).*aa3(2)  2.0000   3.0000   2.0000   4.000];
   aa(15,:)=[aa1(3).*aa2(1).*aa3(1)  3.0000   1.0000   1.0000   3.0000];
   aa(16,:)=[aa1(3).*aa2(2).*aa3(1)  3.0000   2.0000   1.0000   3.0000];
   aa(17,:)=[aa1(3).*aa2(2).*aa3(2)  3.0000   2.0000   2.0000   4.000];
   aa(18,:)=[aa1(3).*aa2(2).*aa3(3)  3.0000   2.0000   3.0000   4.000];
   aa(19,:)=[aa1(3).*aa2(3).*aa3(2)  3.0000   3.0000   2.0000   4.000];
   aa(20,:)=[aa1(3).*aa2(3).*aa3(3)  3.0000   3.0000   3.0000   4.000];

elseif abs(output2)<0.1

   aa(1,:)=[aa1(1).*aa2(1).*aa3(1)   1.0000   1.0000   1.0000   3.0000];
   aa(2,:)=[aa1(1).*aa2(1).*aa3(2)   1.0000   1.0000   2.0000   2.0000];
   aa(3,:)=[aa1(1).*aa2(1).*aa3(3)   1.0000   1.0000   3.0000   2.0000];
   aa(4,:)=[aa1(1).*aa2(2).*aa3(1)   1.0000   2.0000   1.0000   3.0000];
   aa(5,:)=[aa1(1).*aa2(2).*aa3(2)   1.0000   2.0000   2.0000   2.0000];
   aa(6,:)=[aa1(1).*aa2(2).*aa3(3)   1.0000   2.0000   3.0000   2.0000];
   aa(7,:)=[aa1(1).*aa2(3).*aa3(1)   1.0000   3.0000   1.0000   4.0000];
   aa(8,:)=[aa1(2).*aa2(1).*aa3(1)   2.0000   1.0000   1.0000   3.0000];
   aa(9,:)=[aa1(2).*aa2(1).*aa3(2)   2.0000   1.0000   2.0000   1.0000];
   aa(10,:)=[aa1(2).*aa2(1).*aa3(3)  2.0000   1.0000   3.0000   4.0000];
```

```
    aa(11,:)=[aa1(2).*aa2(2).*aa3(1)  2.0000  2.0000  1.0000  3.0000];
    aa(12,:)=[aa1(2).*aa2(2).*aa3(3)  2.0000  2.0000  3.0000  4.000];
    aa(13,:)=[aa1(2).*aa2(3).*aa3(1)  2.0000  3.0000  1.0000  4.0000];
    aa(14,:)=[aa1(2).*aa2(3).*aa3(2)  2.0000  3.0000  2.0000  4.000];
    aa(15,:)=[aa1(3).*aa2(1).*aa3(1)  3.0000  1.0000  1.0000  4.0000];
    aa(16,:)=[aa1(3).*aa2(2).*aa3(1)  3.0000  2.0000  1.0000  4.0000];
    aa(17,:)=[aa1(3).*aa2(2).*aa3(2)  3.0000  2.0000  2.0000  4.000];
    aa(18,:)=[aa1(3).*aa2(2).*aa3(3)  3.0000  2.0000  3.0000  4.000];
    aa(19,:)=[aa1(3).*aa2(3).*aa3(2)  3.0000  3.0000  2.0000  4.000];
    aa(20,:)=[aa1(3).*aa2(3).*aa3(3)  3.0000  3.0000  3.0000  4.000];

elseif abs(output22)<0.03

    aa(1,:)=[aa1(1).*aa2(1).*aa3(1)  1.0000  1.0000  1.0000  0.5000];
    aa(2,:)=[aa1(1).*aa2(1).*aa3(2)  1.0000  1.0000  2.0000  0.5000];
    aa(3,:)=[aa1(1).*aa2(1).*aa3(3)  1.0000  1.0000  3.0000  0.5000];
    aa(4,:)=[aa1(1).*aa2(2).*aa3(1)  1.0000  2.0000  1.0000  3.0000];
    aa(5,:)=[aa1(1).*aa2(2).*aa3(2)  1.0000  2.0000  2.0000  2.0000];
    aa(6,:)=[aa1(1).*aa2(2).*aa3(3)  1.0000  2.0000  3.0000  2.0000];
    aa(7,:)=[aa1(1).*aa2(3).*aa3(1)  1.0000  3.0000  1.0000  3.0000];
    aa(8,:)=[aa1(2).*aa2(1).*aa3(1)  2.0000  1.0000  1.0000  0.5000];
    aa(9,:)=[aa1(2).*aa2(1).*aa3(2)  2.0000  1.0000  2.0000  0.5000];
    aa(10,:)=[aa1(2).*aa2(1).*aa3(3)  2.0000  1.0000  3.0000  0.5000];
    aa(11,:)=[aa1(2).*aa2(2).*aa3(1)  2.0000  2.0000  1.0000  3.0000];
    aa(12,:)=[aa1(2).*aa2(2).*aa3(3)  2.0000  2.0000  3.0000  4.000];
    aa(13,:)=[aa1(2).*aa2(3).*aa3(1)  2.0000  3.0000  1.0000  3.0000];
    aa(14,:)=[aa1(2).*aa2(3).*aa3(2)  2.0000  3.0000  2.0000  4.000];
    aa(15,:)=[aa1(3).*aa2(1).*aa3(1)  3.0000  1.0000  1.0000  0.5000];
    aa(16,:)=[aa1(3).*aa2(2).*aa3(1)  3.0000  2.0000  1.0000  3.0000];
    aa(17,:)=[aa1(3).*aa2(2).*aa3(2)  3.0000  2.0000  2.0000  4.000];
    aa(18,:)=[aa1(3).*aa2(2).*aa3(3)  3.0000  2.0000  3.0000  4.000];
    aa(19,:)=[aa1(3).*aa2(3).*aa3(2)  3.0000  3.0000  2.0000  4.000];
    aa(20,:)=[aa1(3).*aa2(3).*aa3(3)  3.0000  3.0000  3.0000  4.000];

else
    aa(1,:)=[aa1(1).*aa2(1).*aa3(1)  1.0000  1.0000  1.0000  2.0000];
    aa(2,:)=[aa1(1).*aa2(1).*aa3(2)  1.0000  1.0000  2.0000  2.0000];
    aa(3,:)=[aa1(1).*aa2(1).*aa3(3)  1.0000  1.0000  3.0000  2.0000];
    aa(4,:)=[aa1(1).*aa2(2).*aa3(1)  1.0000  2.0000  1.0000  2.0000];
    aa(5,:)=[aa1(1).*aa2(2).*aa3(2)  1.0000  2.0000  2.0000  2.0000];
    aa(6,:)=[aa1(1).*aa2(2).*aa3(3)  1.0000  2.0000  3.0000  2.0000];
    aa(7,:)=[aa1(1).*aa2(3).*aa3(1)  1.0000  3.0000  1.0000  2.0000];
    aa(8,:)=[aa1(2).*aa2(1).*aa3(1)  2.0000  1.0000  1.0000  1.0000];
    aa(9,:)=[aa1(2).*aa2(1).*aa3(2)  2.0000  1.0000  2.0000  1.0000];
    aa(10,:)=[aa1(2).*aa2(1).*aa3(3)  2.0000  1.0000  3.0000  1.0000];
    aa(11,:)=[aa1(2).*aa2(2).*aa3(1)  2.0000  2.0000  1.0000  3.0000];
    aa(12,:)=[aa1(2).*aa2(2).*aa3(3)  2.0000  2.0000  3.0000  4.000];
    aa(13,:)=[aa1(2).*aa2(3).*aa3(1)  2.0000  3.0000  1.0000  4.0000];
    aa(14,:)=[aa1(2).*aa2(3).*aa3(2)  2.0000  3.0000  2.0000  4.000];
    aa(15,:)=[aa1(3).*aa2(1).*aa3(1)  3.0000  1.0000  1.0000  1.0000];
    aa(16,:)=[aa1(3).*aa2(2).*aa3(1)  3.0000  2.0000  1.0000  4.0000];
```

```
    aa(17,:)=[aa1(3).*aa2(2).*aa3(2)   3.0000   2.0000   2.0000   4.000];
    aa(18,:)=[aa1(3).*aa2(2).*aa3(3)   3.0000   2.0000   3.0000   4.000];
    aa(19,:)=[aa1(3).*aa2(3).*aa3(2)   3.0000   3.0000   2.0000   4.000];
    aa(20,:)=[aa1(3).*aa2(3).*aa3(3)   3.0000   3.0000   3.0000   4.000];

  end

  output22o=output22;
  output22p=output22o;
  output22q=output22p;
  output22r=output22q;
  output22s=output22r;
  output22oa=output22s;
  output22pa=output22oa;
  output22qa=output22pa;
  output22ra=output22qa;
  output22sa=output22ra;

  output2o=output2;
  output2p=output2o;
  output2q=output2p;
  output2r=output2q;
  output2s=output2r;
  output2oa=output2s;
  output2pa=output2oa;
  output2qa=output2pa;
  output2ra=output2qa;
  output2sa=output2ra;

  resultsam(:,1)=aa(:,1).*(2*(aa(:,end)==4));
  resultsam(:,2)=aa(:,1).*(2*(aa(:,end)==3));
  resultsam(:,3)=aa(:,1).*(1*(aa(:,end)==2));
  resultsam(:,4)=aa(:,1).*(1*(aa(:,end)==1));
  resultsim=sum(resultsam,2);
  resultsum(fg)=sum(resultsim)./sum(aa(:,1));

  resultsam2(:,1)=aa(:,1).*(2*(aa(:,end)==4));
  resultsam2(:,2)=aa(:,1).*(1*(aa(:,end)==3));
  resultsam2(:,3)=aa(:,1).*(2*(aa(:,end)==2));
  resultsam2(:,4)=aa(:,1).*(1*(aa(:,end)==1));
  resultsim2=sum(resultsam2,2);
  resultsum2(fg)=sum(resultsim2)./sum(aa(:,1));

  if resultsum(fg)>-0.5 & resultsum(fg)<1.5
    if resultsum2(fg)>-1 & resultsum2(fg)<4
      output2cg=[output2 output2q output2p output2s output2r output2o output2qa
output2pa output2sa output2ra output2oa];
      output22cg=[output22 output22q output22p output22s output22r output22o
output22qa output22pa output22sa output22ra output22oa];
      std2cg=std(output2cg,1,2);
      std22cg=std(output22cg,1,2);
```

```
      mstd2cg=mean(output2cg,2);
      mstd22cg=mean(output22cg,2);

      if abs(mstd2cg)>abs(mstd22cg)
         resultsum2(fg)=0.15*rand(1)+0.45;
      else
         resultsum2(fg)=(0.15*rand(1)+0.45)+2;
      end
   end
end

 if resultsum(fg)>-0.5 & resultsum(fg)<1.5
   if resultsum2(fg)>-1 & resultsum2(fg)<1.5
      stinvn=stinvn+1;
   end
end

if resultsum(fg)>-0.5 & resultsum(fg)<1.5
   if resultsum2(fg)>1.5 & resultsum2(fg)<4
      stdpn=stdpn+1;
   end
end

if resultsum(fg)>1.5 & resultsum(fg)<3.5
   if resultsum2(fg)>-1 & resultsum2(fg)<1.5
      svtn=svtn+1;
   end
end

if resultsum(fg)>1.5 & resultsum(fg)<3.5
   if resultsum2(fg)>1.5 & resultsum2(fg)<4
      nsrn=nsrn+1;
   end
end

subplot(222)
xline=[xline fg];
yline=[yline output2];
h1=line(xline(fg:fg+1),yline(fg:fg+1));
set(h1,'color','r')
yline2=[yline2 output22];
h2=line(xline(fg:fg+1),yline2(fg:fg+1));
set(h2,'color','g')
yline3=[yline3 output23];
h3=line(xline(fg:fg+1),yline3(fg:fg+1));

button_state2=get(handles.stop_toggle,'Value');
button_max2=get(handles.stop_toggle,'Max');
handles.button_state2=button_state2;
handles.button_max2=button_max2;
```

```
    if handles.button_state2==handles.button_max2;
      pause
    end
end
stdpp=num2str(stdpn/lenc*100);
svtp=num2str(svtn/lenc*100);
nsrp=num2str(nsrn/lenc*100);
stinvp=num2str(stinvn/lenc*100);

subplot(2,1,2)
set(h1a,'String',stdpp)
set(h2a,'String',nsrp)
set(h3a,'String',svtp)
set(h4a,'String',stinvp)

 subplot(2,1,2)
 plot(resultsum,resultsum2,'r+')

handles.resultsum=resultsum;
guidata(h,handles)

% --------------------------------------------------------------------
function varargout = weight_edit_Callback(h, eventdata, handles, varargin)

% --------------------------------------------------------------------
function varargout = stop_toggle_Callback(h, eventdata, handles, varargin)
button_state2=get(handles.stop_toggle,'Value');
button_max2=get(handles.stop_toggle,'Max');
handles.button_state2=button_state2;
handles.button_max2=button_max2;
guidata(h,handles);

% --------------------------------------------------------------------
function varargout = weight_edit2_Callback(h, eventdata, handles, varargin)

% --------------------------------------------------------------------
function varargout = weight_edit3_Callback(h, eventdata, handles, varargin)

% --------------------------------------------------------------------
function varargout = next_button_Callback(h, eventdata, handles, varargin)
fignorm=openfig('gui10','reuse');

% --------------------------------------------------------------------
function varargout = tar_edit_Callback(h, eventdata, handles, varargin)
```

```
% --------------------------------------------------------------------
function varargout = defuzzy_button_Callback(h, eventdata, handles, varargin)

% --------------------------------------------------------------------

function varargout = base_button_Callback(h, eventdata, handles, varargin)
trfdata=num2str(get(handles.store_edit,'String'));
load(trfdata)
handles.prodrule=prodrule;
handles.prodrules=prodrules;
guidata(h,handles)
lenprod=length(prodrule);
quadprod=lenprod/4;
prod=[prodrule.' prodrules];
prod1=prod(1:quadprod,:);
prod2=prod(quadprod+1:2*quadprod,:);
prod3=prod(2*quadprod+1:3*quadprod,:);
prod4=prod(3*quadprod+1:4*quadprod,:);
i=0;
lenpc=quadprod;
while ~isempty(prod1)
   i=i+1;
   prod11=abs(ones(lenpc,1)*prod1(1,2:end-1)-prod1(:,2:end-1));
   [sp,si]=sortrows(prod11);
   sum1=sum(sum(prod11,2)==0);
   si1=si(1:sum1);
   prod1e=prod1(si1,:);
   [spm,sim]=sortrows(prod1e);
   prod1c(i,:)=prod1e(sim(end),:);
   prod1(si1,:)=[];
   [lenpc,tempc]=size(prod1);
end
i=0;
lenpc=quadprod;
while ~isempty(prod2)
   i=i+1;
   prod11=abs(ones(lenpc,1)*prod2(1,2:end-1)-prod2(:,2:end-1));
   [sp,si]=sortrows(prod11);
   sum1=sum(sum(prod11,2)==0);
   si1=si(1:sum1);
   prod1e=prod2(si1,:);
   [spm,sim]=sortrows(prod1e);
   prod2c(i,:)=prod1e(sim(end),:);
   prod2(si1,:)=[];
   [lenpc,tempc]=size(prod2);
end
i=0;
lenpc=quadprod;
```

```
while ~isempty(prod3)

   i=i+1;
   prod11=abs(ones(lenpc,1)*prod3(1,2:end-1)-prod3(:,2:end-1));
   [sp,si]=sortrows(prod11);
   sum1=sum(sum(prod11,2)==0);
   si1=si(1:sum1);
   prod1e=prod3(si1,:);
   [spm,sim]=sortrows(prod1e);
   prod3c(i,:)=prod1e(sim(end),:);
   prod3(si1,:)=[];
   [lenpc,tempc]=size(prod3);

end

i=0;
lenpc=quadprod;

while ~isempty(prod4)

   i=i+1;
   prod11=abs(ones(lenpc,1)*prod4(1,2:end-1)-prod4(:,2:end-1));
   [sp,si]=sortrows(prod11);
   sum1=sum(sum(prod11,2)==0);
   si1=si(1:sum1);
   prod1e=prod4(si1,:);
   [spm,sim]=sortrows(prod1e);
   prod4c(i,:)=prod1e(sim(end),:);
   prod4(si1,:)=[];
   [lenpc,tempc]=size(prod4);

end

[pc1,temp]=size(prod1c);

[pc2,temp]=size(prod2c);

[pc3,temp]=size(prod3c);

[pc4,temp]=size(prod4c);

product=[prod1c;prod2c;prod3c;prod4c];

[pc,temp]=size(product);

i=0;

lenpc=pc;
```

```
while ~isempty(product)

   i=i+1;
   prod11=abs(ones(lenpc,1)*product(1,2:end-1)-product(:,2:end-1));
   [sp,si]=sortrows(prod11);
   sum1=sum(sum(prod11,2)==0);
   si1=si(1:sum1);
   prod1e=product(si1,:);
   [spm,sim]=sortrows(prod1e);
   productc(i,:)=prod1e(sim(end),:);
   product(si1,:)=[];
   [lenpc,tempc]=size(product);

end

trfdata=num2str(get(handles.base_edit,'String'));
save(trfdata,'productc')
handles.productc=productc;
guidata(h,handles)

% --------------------------------------------------------------------
function varargout = base_edit_Callback(h, eventdata, handles, varargin)
```