USING NEURAL NETWORKS FOR ESTIMATING CRUISE MISSILE RELIABILITY

THESIS

Donald L. Hoffman, Captain, USAF

AFIT/GOR/ENS/03-10

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOR/ENS/03-10

USING NEURAL NETWORKS FOR ESTIMATING CRUISE MISSILE RELIABILITY

THESIS

Presented to the Faculty

Department of Operations Research

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Donald L. Hoffman, BS

Captain, USAF

March 2003

AFIT/GOR/ENS/03-10

USING NEURAL N ETWORKS FOR
ESTIMATING CRUISE MISSILE
RELIABILITY

Donald L. Hoffman, BS
Captain, USAF

Approved:

Kenneth W. Bauer (Chairman) | 12 MAR 03
date

Stephen P. Chambal (Member) | 12 MAR 03
date

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Dr Kenneth Bauer, and thesis reader, Dr Stephen Chambal, for their guidance and support throughout the course of this thesis effort. I would also like to thank my sponsor, Mr. Al Montalvo, ACC/DONO for the continual support. I am further indebted to the many persons responsible for collecting the data over the years and those who took the time to explain the details of the collected measures. A special thanks is extended to Mike Bredehoeft and Major R. Nicole Benton for their insight and readiness to answer my many questions.

Donald L. Hoffman

Table of Contents

List of Figures

Page

List of Tables

Abstract

ACC believes its current methodology for predicting the reliability of its Air Launched Cruise Missile (ALCM) and Advanced Cruise Missile (ACM) stockpiles could be improved. They require a predictive model that delivers the best possible 24-month projection of cruise missile reliability using existing data sources, collection methods and software. It should be easily maintainable and developed to allow a layperson to enter updated data and receive an accurate reliability prediction. The focus of this thesis is to improve upon free flight reliability, although the techniques could also be applied to the captive carry portion of the missile reliability equation. The following steps were taken to ensure maximum accuracy in model results.

1. Add more detail to flight test reliability calculation.
2. Convert the ground test data into a usable form (reduce).
3. Engage in an exercise in feature selection.
4. Develop a Matlab model prototype.
5. Validate the model via problems with known solutions.
6. Apply an appropriate data fusion technique to the different network outputs (logistic regression, feed-forward and radial basis function).
7. Put the model into the form of a usable tool for the end-user.

The end product is the ALCM/ACM Reliability Estimation System (AARES), a VBA-based model that meets all user criteria.

USING NEURAL NETWORKS FOR ESTIMATING

CRUISE MISSILE RELIABILITY

I. Introduction

General Issue

United States Strategic Command (USSTRATCOM) conducts an annual Nuclear Weapon System Planning Factors Update to determine its ability to meet the Single Integrated Operational Plan (SIOP) commitment. USSTRATCOM requires the Navy, Space Command (SPACECOM) and Air Combat Command (ACC) to present a 24-month prediction of the reliability of the weapons systems of concern, along with a justification of the prediction methodology. ACC believes its current methodology for predicting the reliability of its Air Launched Cruise Missile (ALCM) and Advanced Cruise Missile (ACM) stockpiles could be improved. Consequently, ACC/DON was tasked with developing a new approach for meeting the STRATCOM requirement.

Problem Statement

ACC uses flight test results and an estimated degradation factor to compute current year cruise missile reliability. A simple logistic regression (discussed in Chapter 2) is performed to predict cruise missile reliability. Unfortunately, there are an extremely small number of annual flight tests (2-3 shots per year). As a result, the ACC method cannot be used with a great degree of confidence in its accuracy.

Objective

The goal of this thesis is to develop a predictive model that delivers a realistic 24-month reliability projection. The model should utilize existing data sources, collection methods and software. It should be easily maintainable and developed to allow a layperson to enter updated data and receive an accurate reliability prediction.

Background

The maintenance concept for cruise missiles does not lend itself to continuous data collection of missile status. ALCMs and ACMs are protected from the worst of the elements through storage in secured, structurally reinforced igloos. The majority of both stockpiles are stored mounted on common strategic rotary launchers (CSRL) or pylons, and generally referred to as "packages." Periodically, packages are pulled from storage for maintenance, testing and exercises. Results of the maintenance checks and tests are recorded by munitions personnel and forwarded to the depot at Oklahoma City, Air Logistics Center (OC-ALC) and ACC. Examples of pertinent test fields (notional) are shown in Table 1.

Table 1: Typical Input Data (notional)

	# Passed	# Failed	Total # Tested	Pass Rate
LLT Type A	167	15	182	92%
LLT Type B	16	2	18	89%
LPT Type A	230	8	238	97%
LPT Type B	13	11	24	54%
CSRL SIT	0	0	0	N/R
Pylon SIT	0	0	0	N/R
CSRL MIT	319	5	324	98%
Pylon MIT	380	19	399	95%
Level I Type A	159	50	209	76%
Level I Type B	15	22	37	41%
Level III Type B	0	0	0	N/R
INE Auto-Cal	124	15	139	89%

* see Appendix A for acronym definitions

Data is provided from Minot and Barksdale Integrated Maintenance Facilities (IMFs) as well as historical records from OC-ALC, ACC/LGWN and USSTRATCOM. The operational bases use the same basic maintenance concept, however, the manner in which the missiles are stored precludes certain tests – i.e. Minot does not store any ALCMs on pylons, therefore, no ALCM/Pylon test combinations are performed.

A Loaded Launcher Test/Loaded Pylon Test (LLT/LPT) Type A is run after building the package and to certify operational capability of the package. It is primarily a communication test and verifies that the aircraft will be able to communicate through the pylon/launcher and down to the missile. A LLT/LPT Type B is a retest of previous SIT or MIT failure. The test is identical to a LLT/LPT Type A and serves a similar purpose as a Level 1 except at the package level (as opposed to the individual missile level).

A MIT is a communication test between the aircraft and the missile and is normally performed after package upload onto the aircraft. The aircraft offensive avionics system (OAS) sends a command word to the missile and tells it to perform an internal built-in test (BIT) test on any components it has and report the results back to the aircraft. SITs are more involved and must be performed (per technical order) if a single missile swap occurs on the flight line. In addition to all the tests the MIT performs, a SIT commands the missile inertial navigation element (INE) to go into a Fine Align/Coarse Align. This test ensures that the inertial platform is able to align to an earth reference and can take 1-second updates from the aircraft. The SIT also performs a preflight test that actuates the elevons minutely to ensure the steering avionics are performing properly. Both tests are considered the last check on the weapon package prior to the aircrew accepting the aircraft as mission ready. Although MITs and SITs give a good first indication of missile health, detected faults must be verified with further testing via an electronic systems test set (ESTS) in the IMF.

Level 1 Type B is a deep cycle electronic test run by the ESTS as a verification of MIT, SIT or loaded launcher test/loaded pylon test (LLT/LPT) fault indication. When a memory dump from a previously mentioned test (LLT/LPT, MIT, SIT) indicates a problem in a missile area, the Level 1 Type B runs component BITs, interrogates components, and compares and validates proper responses to diagnose the problem down to the component level. Level 1 Type A's are identical to Type B's except they are run after a 72-month engine change or other periodic maintenance. Figure 1 illustrates the flow of events associated with the described ground maintenance tests.

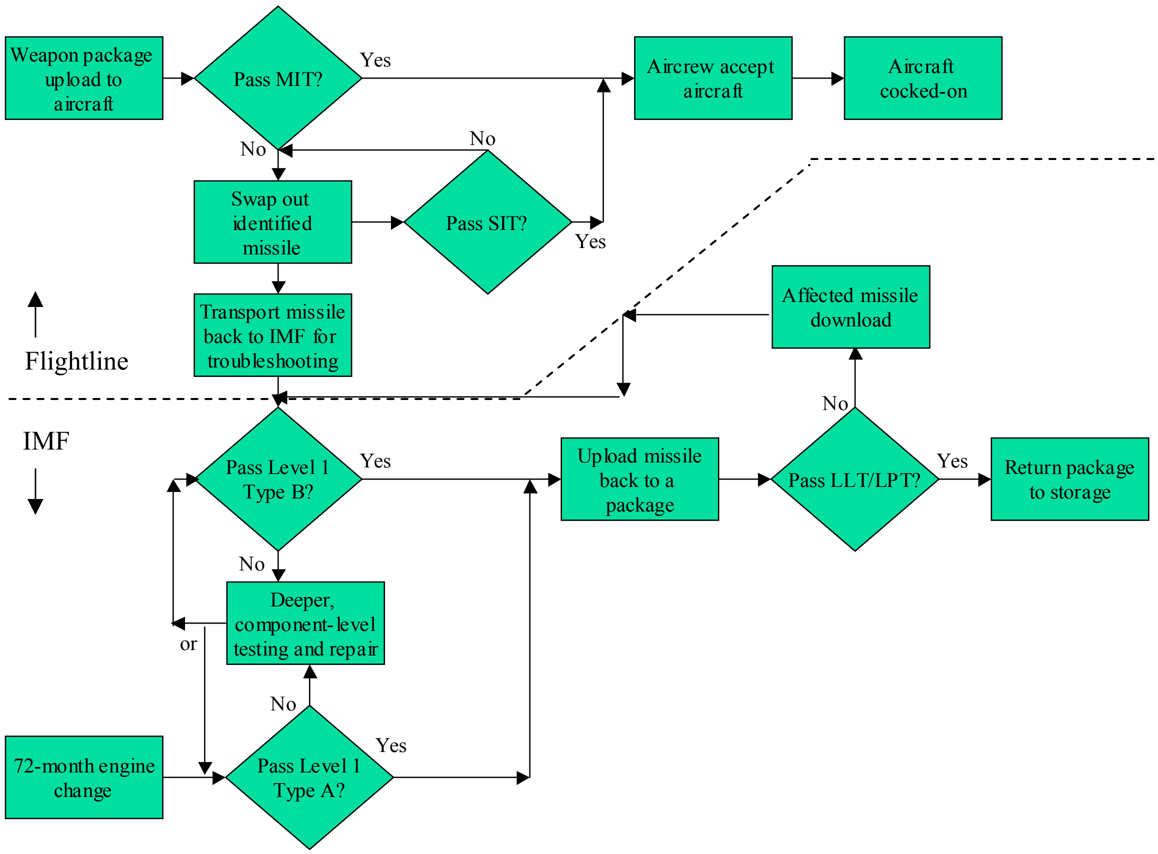


Figure 1: MIT/SIT - Level 1Maintenance Testing

INE auto-cals are performed in the IMF every 48-months and specifically check to ensure the INE is operating correctly and not drifting beyond tolerance limits. Due to the 7-hour test duration, auto-cals are normally performed on an entire package to reduce workload and expedite the maintenance schedule. Figure 2 illustrates typical INE auto-cal chain of events.

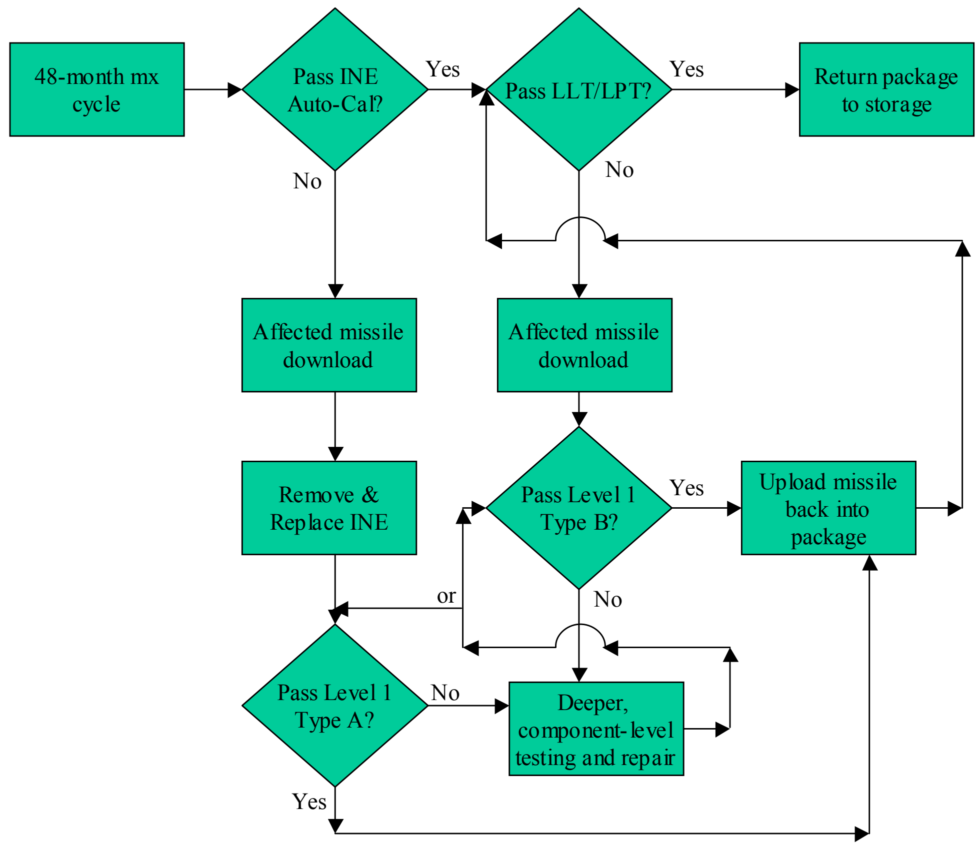


Figure 2: INE Auto-cal - Level 1Maintenance Events

Level 3 Type B testing is component level testing, run as a verification of faults identified in a Level 1 test – i.e. if a missile fault is identified down to a component during a Level 1 test, Level 3 testing will troubleshoot the identified component down to the subcomponent level.

Knowing the data available with which to improve upon the existing technique for determining missile reliability, the next logical step would be an overview of methodologies being used by other weapons communities, thereafter proceeding into a discussion on proposed steps to improve upon the existing cruise missile reliability computation.

II. Literature Review

Before engaging in an attempt to improve upon the current ACC methodology, one should consider (at the macro-level) other techniques being employed. Three other weapons communities are currently using valid methodologies for determining weapon system reliability. Although some concepts could be applied to cruise missiles, differences in weapon employment and maintenance concepts limit the extent to which the cruise missile community may use the ideas of others.

SLBM

The submarine launched ballistic missile (SLBM) community contracts the Johns Hopkins University Applied Physics Laboratory (JHU-APL) to calculate and track Trident II and Trident III reliability. All information contained in this section was derived from Appendix B, Methodology and Supporting Analysis, Trident II and Trident III Reliability Plan. Overall weapon system reliability (WSR) is calculated as follows:

$$WSR = LR \times FR \times RR \qquad \textbf{(1)}$$

where

LR = Launch Reliability

FR = Inflight Reliability

RR = Reentry Reliability

$$LR = CR \times PLA \times f(LI) \times f(LWA) \quad \textbf{(2)}$$

where

CR = Countdown Reliability

PLA = Post-launch Assessment

LI = Launch Interval

LWA = Launch Window Availability

$$FR = BR \times DR \quad \textbf{(3)}$$

where

BR = Boost Reliability

DR = Deployment Reliability

$$RR = RRS \times RRI \times RRB \quad \textbf{(4)}$$

where

RRS = Reentry Separation Reliability

RRI = Reentry Inflight Reliability

RRB = Reentry Burst Reliability

One should note that each sub-sub-reliability (eg. Launch Reliability) is further broken down at least one more level in the reliability plan -- discussion of which is beyond the scope of this thesis. The model uses inputs from a patrol test database [weapon system readiness tests (WSRTs), battle readiness tests (BRTs) and navigation accuracy tests (NATs)], surveillance tests and flight test results, as well as simulation results for components that cannot be exercised in the course of other testing.

TLAM

Information described in this section is derived from the SIOP Planning Factors Conference, October 2002. The Navy uses in-house contractors at Naval Surface Warfare Center (NSWC)-Corona for determining Tomahawk Land Attack Missile (TLAM) reliability. The reliability model developed consists of the following:

$$WSR = LR \times FR \times PR \qquad \textbf{(5)}$$

where

LR = Launch Reliability

FR = Inflight Reliability

PR = Payload Reliability

$$LR = PFR \times MR \times MA \qquad \textbf{(6)}$$

where

PFR = Platform Reliability

MR = Missile Reliability

MA = Missile Adjustment

$$FR = BR \times BA \times CR2 \times CA \quad \textbf{(7)}$$

where

BR = Boost Reliability

BA = Boost Adjustment

CR2 = Cruise Reliability

CA = Cruise Adjustment

$$PR = \Pr earm \times WAM \times NavyAF \,\&\, F \times DOE \quad \textbf{(8)}$$

where

Prearm = Warhead Prearm Reliability

WAM = Warhead Arming Module

AF&F = Arming Fuzing & Firing

DOE = Department of Energy Component Reliability

Downward adjustment factors shown in launch and inflight reliability equations stem from stockpile failures detected and attributed to the appropriate operational phase. Joint integrated laboratory tests (JILT), stockpile laboratory tests (SLT), functional ground tests (FGT) and flight tests serve as the primary data sources for the TLAM reliability model.

ICBM

The synopsis in this section is from the joint paper Weapon System Effectiveness for Legacy Systems, authored by Lindblad et al. As with SLBMs, the intercontinental ballistic missile (ICBM) system program office (SPO), TRW contractors and analysts at the JHU-APL have constructed an involved model to determine system reliability (see Figure 3).

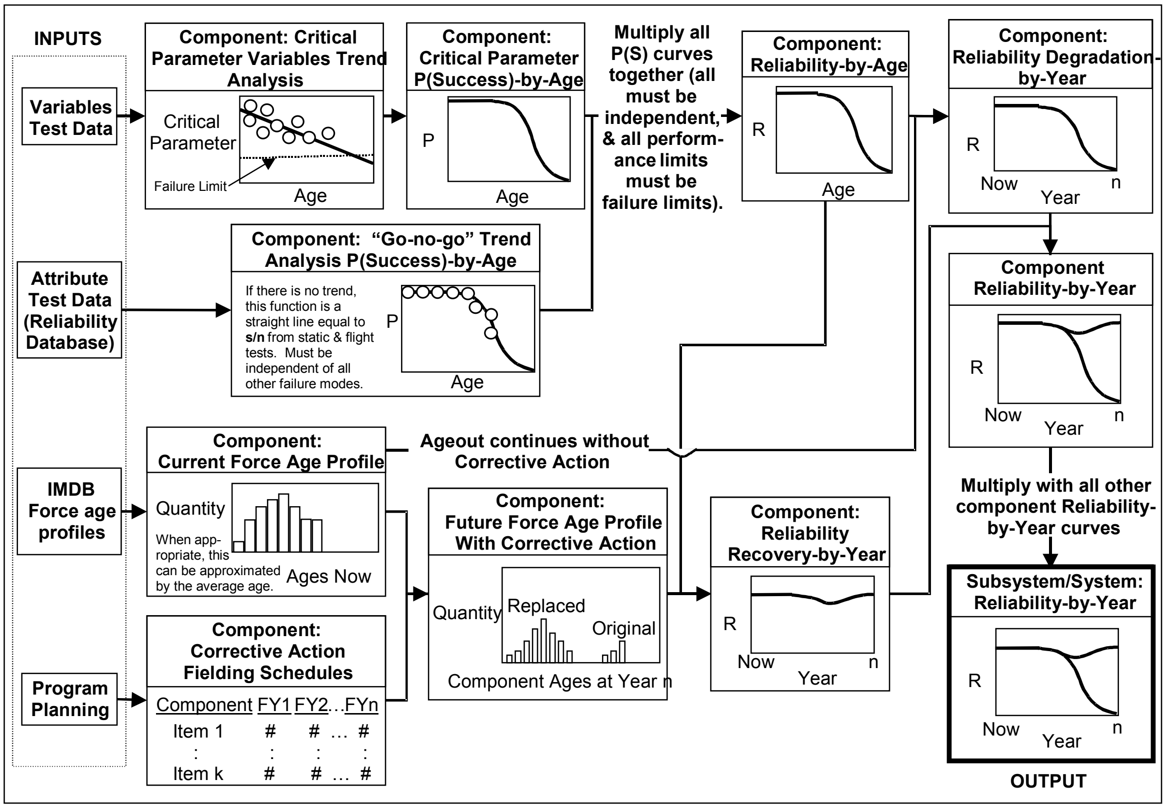


Figure 3: ICBM Reliability Model (Lindblad, 2001: 8)

Simplifying the model to some degree, the ICBM community uses ground tests, flight tests, simulated launches and DOE-provided warhead data as sources for traditional analytic models to determine reliability.

ALCM/ACM

The current reliability measures discussed in this section are sourced from interviews with subject matter experts at ACC (Quick, 2003) and OC-ALC (Bredehoeft, 2002), and briefings at the USSTRATCOM Planning Factors Conference, October 2002. As mentioned previously in Chapter 1, herein lie the problem and the reason for this thesis. With the exception of missile reliability, it is understood that all other components of the following equations have adequate sample sizes with copious amounts of data that has been reduced for use in classical analytic models, widely accepted within the weapons community.

$$WSR = CR2 \times MR \times WR \qquad \textbf{(9)}$$

where

CR2 = Carrier Reliability

MR = Missile Reliability

WR = Warhead Reliability

$$CR2 = AGR \times ASR \times WDR \times RSR \times ACR \qquad \textbf{(10)}$$

where

AGR = Aircraft Generation Reliability

ASR = Aircraft Systems Reliability

WDR = Weapon Delivery System Reliability

RSR = Release System Reliability

ACR = Aircrew Reliability

The National Nuclear Security Administration provides warhead reliability information (used in WSR calculation). All carrier data is collected from maintenance databases (updated weekly by maintenance organizations throughout ACC). With regard to missile reliability, ACC relies heavily upon the cruise missile SPO for reliability data. The calculation as follows:

$$MR = CCR \times FFR \times Degrade \quad \textbf{(11)}$$

where

CCR = Captive Carry Reliability

FFR = Free-flight Reliability

Captive carry and free flight data are collected in the course of flight testing. The cruise missile SPO provides the degrade factor shown in the missile reliability equation. (One should note here that this thesis focuses solely on improving the missile reliability determination -- in particular the determination for free-flight reliability; although the same steps could be applied to captive carry data for an analogous estimate).

The current methodology for predicting missile reliability involves regressing time against flight test results. For the purposes of demonstration, the notional data shown in Appendix B is used. The data is re-created in JMP where a logistic regression is performed using "FY" as the independent variable and "Result" as the dependent variable (response). The regression results are assumed to be a cumulative distribution function (CDF) for probability of failure with parameters:

Intercept	-2.8380919
Coefficient	0.23892478

Yielding

$$F(FY) = \frac{1}{(1+\exp(-(-2.8380919+.23892478\times FY)))} \quad \textbf{(12)}$$

By definition

$$R(FY) = 1 - F(FY) = \frac{1}{(1+\exp(-2.8380919+.23892478\times FY))} \quad \textbf{(13)}$$

Substituting the FY data into the equation results in the column labeled "Rel Est" in Appendix B. A plot of the derived reliability function is shown in Figure 4.

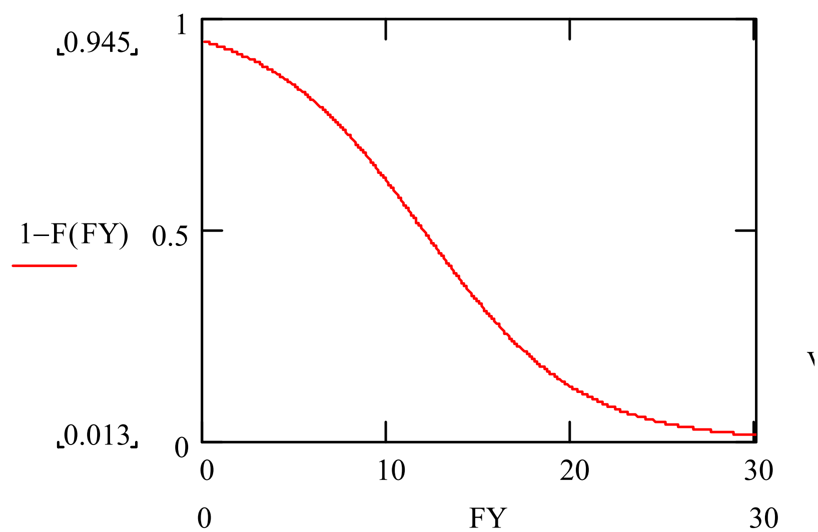


Figure 4: Flight Test Regression Plot

Predictive missile reliability can be calculated by inputting a value corresponding to the desired FY into the R(FY) equation. The assumption that a CDF results from the regression is supported by taking the derivative of F(FY) with respect to FY to get the probability density function (PDF) f(FY). Integrating a valid PDF over the applicable range should result in a value of one. The Mathcad results below show the derivative of F(FY) and the integration of f(FY). The integration solution (1) implies that the CDF interpretation with regard to the regression is not unreasonable.

$$\frac{d}{dFY}F(FY) \;\Big|\begin{array}{l}\text{simplify}\\ \text{float},4\end{array} \rightarrow \frac{.2389}{(1. + \exp(2.838 - .2389\,FY))^{2.}} \cdot \exp(2.838 - .2389\,FY)$$

$$\int_{-\infty}^{\infty} \frac{d}{dFY}F(FY)\, dFY \;\Big|\begin{array}{l}\text{simplify}\\ \text{float},4\end{array} \rightarrow 1. \qquad \textbf{(14)}$$

Although the other weapons communities have primarily opted to use analytic models for reliability predictions, a concerted effort into researching missile component reliabilities and corresponding tail-number histories would be necessary for developing a similar approach for cruise missiles. Statistical techniques that predict failures based upon the performance of a similar system could also be used. Unfortunately, analytic models rely upon assumptions about the nature of failures, development environments and probabilities of failure. Additionally, traditional reliability models demonstrate different predictive capabilities during the various phases of testing and work best with copious amounts of test data. The cruise missile community does not employ the maintenance concept nor have the data collection infrastructure to support such an effort. As a result, a traditional analytic model that predicts well under these circumstances seems infeasible.

In lieu of analytic models, neural networks could be used for reliability estimation and prediction using only failure histories. Although the weights developed by a network do not directly relate to particular reliability metrics (unlike analytic models), neural nets do not rely upon assumptions about the development environment or external parameters, nor do they require large amounts of data to make reasonable predictions.

In simplest terms, a neural network processes an input feature vector $\mathbf{x} = (x_1, \ldots x_N)$ along N branching nodes (Figure 5).

$$r_m = \sum_{n=1}^{N} w_{nm} \cdot X_n$$

$$y_m = f(r_m) = \frac{1}{1+e^{-r_m}}$$

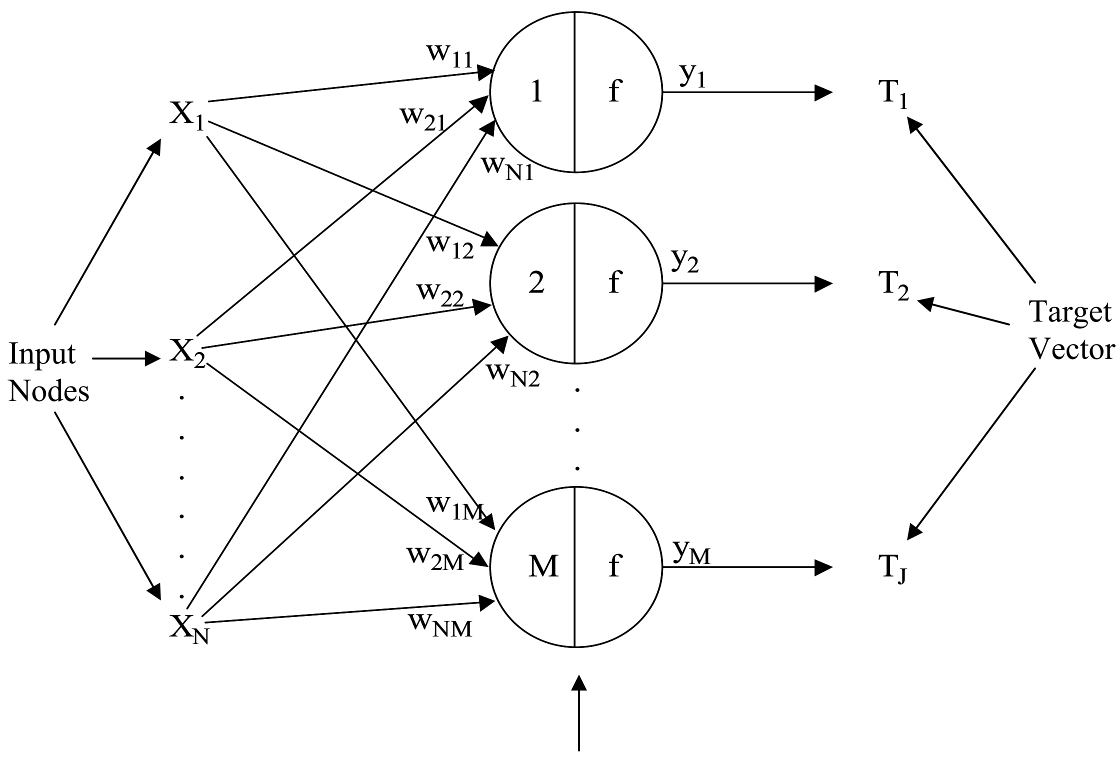


Figure 5: Simple Neural Network (Bauer, 2002)

The input nodes fan out to each perceptron (network node that performs operations upon N inputs and provides a single output) so as to allow input from each component of $\mathbf{x}$. Each incoming arrow has an associated weight (w_{nm}), indexed by the convention: input node associated with the x_n^{th} feature coming into the m^{th} perceptron. Each of the M perceptrons partitions the feature space in to two half-spaces, usually resulting in at least 2M half-spaces. Adjusting the weights (w_{nm}) determines the required convex regions that contain the desired multilinearly separable classes, as defined by the target vector (**T**). In other words, the network attempts to approximate the values in the target vector (**T**) using features contained in the input vector (**x**).

Karunanithi et al. in their IEEE journal article present a pertinent example of a neural network used to solve a reliability problem. In terms of a neural network mapping, reliability prediction can be stated as:

$$\underbrace{P:\{(I_k(t),O_k(t)),i_{k+h}(t+\Delta)\}}_{\text{System Failure History}} \rightarrow \underbrace{o_{k+h}(t+\Delta)}_{\text{Network Prediction}} \qquad \textbf{(15)}$$

where

$I_k(t)$	Set of sequential execution times
$O_k(t)$	Set of corresponding observed accumulated faults
$i_{k+h}(t+\Delta)$	Desired future test session
$o_{k+h}(t+\Delta)$	Corresponding cumulative faults
Δ	Cumulative execution time of h consecutive future test sessions

By adjusting network neurons' weights via training, the network can be used to predict the total number of faults at the end of a future test session $k + h$, merely by inputting $i_{k+h}(t+\Delta)$. A network's predictive ability can be determined by what it learns and in what sequence. Generalization training can be described as relating each input i_t at time t with an output o_t – so the network learns to model the relationship between the input and output variables *relative to the same time period* (Figure 6).

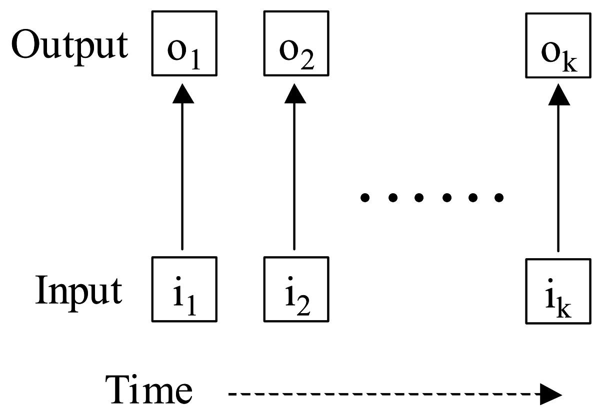


Figure 6: Generalization Training

Prediction training is similar to generalization training, except i_t at time t is associated with the value of the output variable o_{t+k} at time. So the network learns to predict outputs *relative to the n^{th} time period* (Figure 7).

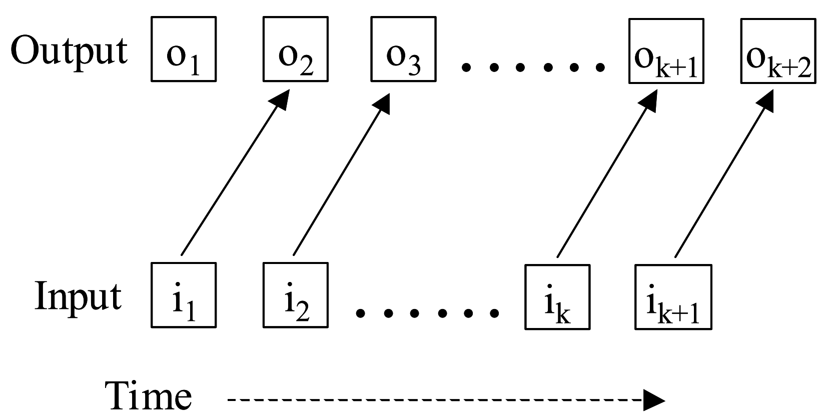


Figure 7: Prediction Training

Training a network is usually accomplished via a supervised learning algorithm, where network weights are adjusted using a quantified error feedback. Back-propagation is the most common supervised learning algorithm. Using an iterative approach, back-propagation calculates the sum-squared error between desired outputs and the network-generated outputs and uses the gradient of the sum-squared error to adapt network weights in an effort to reduce the error measure in future epochs. The network is considered to be trained when the squared error drops below a specified threshold.

To test the contention that neural nets can work as well or better than analytic models, Karunanithi et al used the following example. A typical feed-forward network was trained on a software failure dataset. Total test and debugging time was 46 days with a cumulative 266 faults over the time period. Since logistic-function units were used in the network, data was scaled down to a suitable range (0.1 to 0.9). For the purpose of the experiment, minimum training-set size started at three data points (time increments) and incremented up to 45 data points (time increments) in steps of two. A prediction average was taken over fifty trials at each set size with different random seeds used to initialize the weights for each trial. The overall purpose of the experiment was to predict cumulative endpoint errors at various points of time prior to the actual dataset endpoint (46). Table 2 shows the experiment results by way of comparison. Results are in terms of relative prediction error using the formula:

$$\text{RPE} = (\text{predicted faults} - \text{actual faults}) / \text{actual faults} \qquad \textbf{(16)}$$

Table 2: Endpoint Relative Prediction Error Results

Average and Maximum Errors in Endpoint Predictions						
Model	Average Error			Maximum Error		
	1st Half	2nd Half	Overall	1st Half	2nd Half	Overall
FFN Generalization	7.34	1.19	3.36	10.48	2.85	10.48
FFN Prediction	6.25	1.10	2.92	8.69	3.18	8.69
Logarithmic	21.59	6.16	11.61	35.75	13.48	35.75
Inverse Polynomial	11.97	5.65	7.88	20.36	11.65	20.36
Exponential	23.81	6.88	12.85	40.85	15.25	40.85
Power	38.30	6.39	17.66	76.52	15.64	76.52
Delayed S-shape	43.01	7.11	19.78	54.52	22.38	54.52

First Half is the model's average prediction error in the first half of the experiment.
Second Half is the model's average prediction error in the second half of the experiment.
Overall is the model's average prediction error for the entire duration of the experiment.

The results show accurate neural network endpoint predictions in early and late stages of the experiment. A similar experiment was conducted to show next-step prediction accuracy with results shown in Table 3.

Table 3: Next-Step Relative Prediction Error Results

Average and Maximum Errors in Next-Step Predictions						
Model	Average Error			Maximum Error		
	1st Half	2nd Half	Overall	1st Half	2nd Half	Overall
FFN Generalization	8.61	2.40	4.59	17.51	4.95	17.51
FFN Prediction	8.02	3.05	4.80	17.74	6.64	17.74
Logarithmic	4.94	2.31	3.24	5.95	7.56	7.56
Inverse Polynomial	4.76	2.24	3.13	6.34	7.83	7.84
Exponential	5.70	2.33	3.52	10.17	7.42	10.17
Power	4.59	2.44	3.20	8.59	7.12	8.59
Delayed S-shape	6.17	2.12	3.55	13.24	7.98	13.24

In this case, the data shows neural nets having prediction errors only slightly greater than traditional analytic models. As illustrated by the example, neural networks can be used to approximate reliability at different points in time using failure histories. Furthermore, the prediction errors realized by the networks are less than or comparable to traditional analytic models.

As a practical, although modified, application of the previous article in this thesis, neural networks are used for predicting cruise missile reliability (for this thesis, free-flight reliability prediction is the focus). Selected ground test results (features) are run through different types of neural networks with notional free flight test results as the target. Once generated, the different network outputs are fused into a single number representing the model's estimate of free flight reliability per year.

Logistic Regression.

Widely used in statistics, logistic regression can be visualized using Figure 8 (Bauer, 2002).

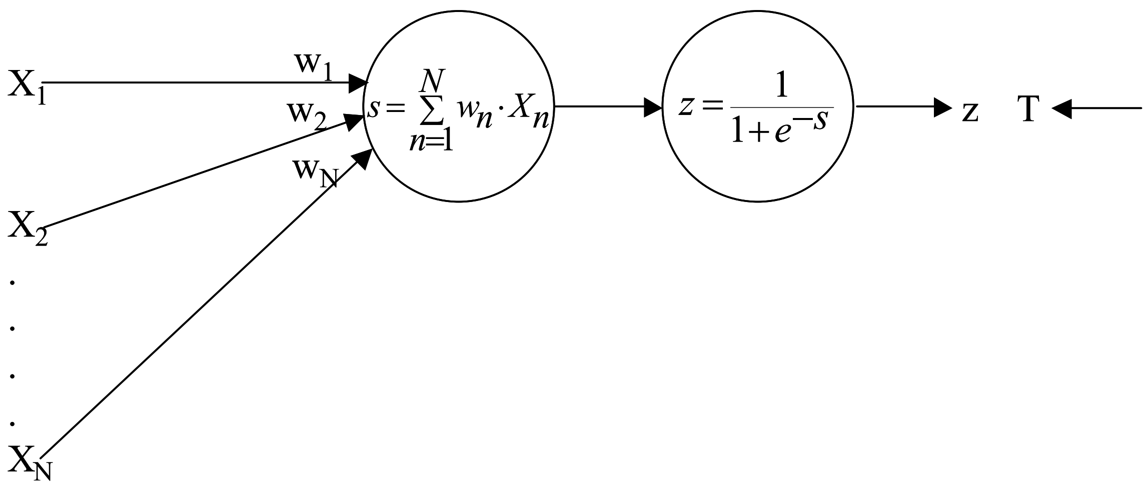


Figure 8: Logistic Regression Network

Model features (X_n) are multiplied by an initial draw of random weights (w_n) and summed (s). The sum (s) is put through a 'squashing function' and an output (z) results. By calculating the sum-squared error between desired outputs (T) and the network-generated outputs (z), network weights (w) are adjusted iteratively in the direction opposite the gradient of the sum-squared error. The process continues until changes in the sum of squared error are reduced below a specified threshold.

Feed-Forward Neural Network.

Taking the logistic regression network a step further, feed-forward neural networks (FFN) use an additional layer of hidden neurodes to approximate the target vector (Figure 9 – Looney, 1977: 84).

$$r_m = \sum_{n=1}^{N} w_{nm} \cdot X_n \qquad s_j = \sum_{m=1}^{M} u_{mj} \cdot y_m$$

$$y_m = f(r_m) = \frac{1}{1+e^{-r_m}} \qquad z_j = g(s_j) = \frac{1}{1+e^{-s_j}}$$

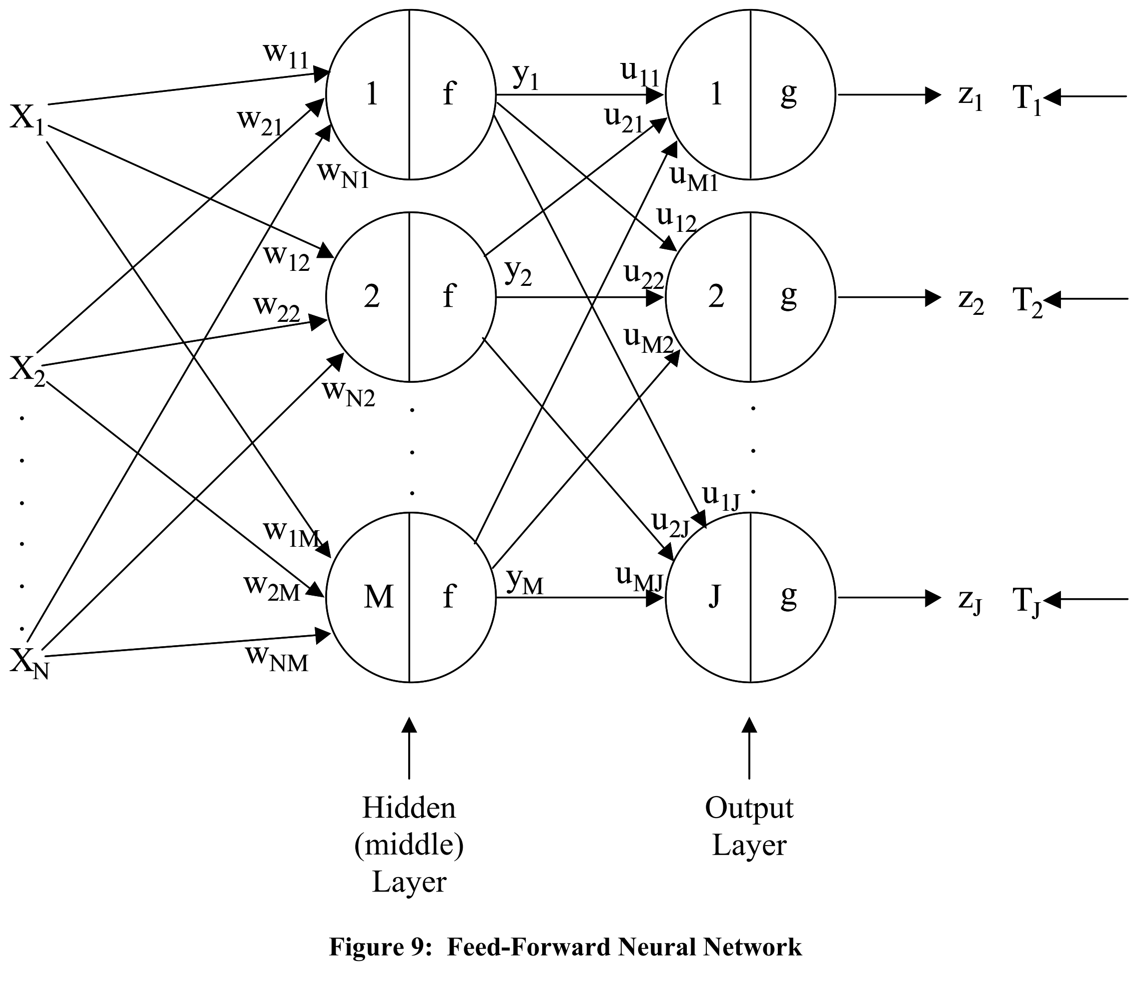


Figure 9: Feed-Forward Neural Network

At each neurode (m) in the middle (hidden) layer, model features (X_n) are multiplied by respective weights (w_{nm}) and summed (r_m). The middle layer sums (r_m) are put through the 'squashing functions' (f) to get middle layer outputs (y_m). At each output neurode (j), middle layer outputs (y_m) are multiplied by upper layer weights (u_{mj}) and summed (s_j). The upper layer sums (s_j) are put through another set of 'squashing functions' (g) to get network outputs (z_j). Upper and middle layer weights are trained using a supervised training algorithm – back-propagation. As described by Karunanithi et al, back-propagation iteratively calculates sum of squared errors between desired outputs (T_j) and network outputs (z_j). Upper and middle layer weights are adjusted in the direction opposite the gradient of the sum of squared errors. As with logistic regression, training continues until changes in the total sum of squared error drop below a specified threshold.

Radial Basis Function Network.

A visualization of the third and final type of neural network used in the model can be seen in Figure 10 (Looney, 1977: 96).

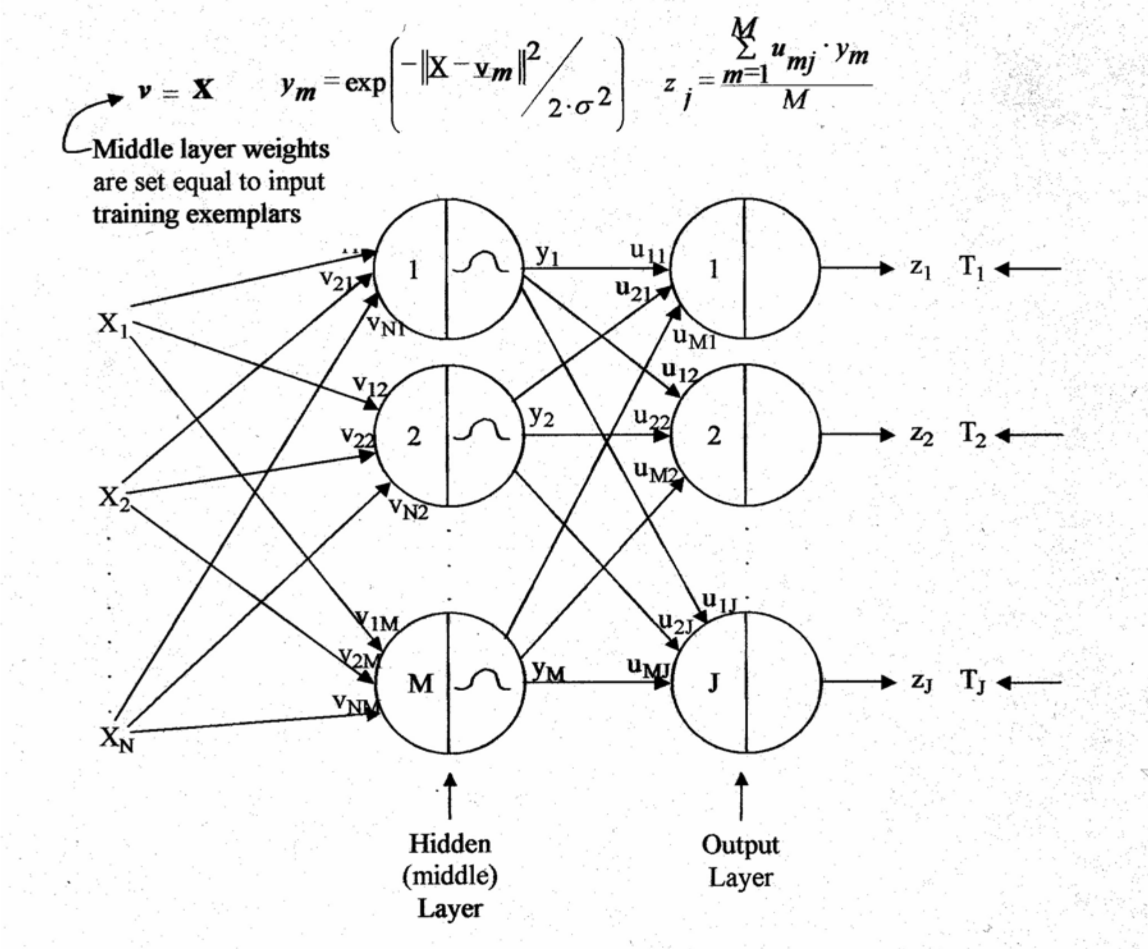


Figure 10: Radial Basis Function Network

A radial basis function network (RBFN) differs from the previously described feed-forward neural network in the activation functions and the way they are used. Different paradigms are used when training a RBF network (Looney, 1977: 98). In the simplest case, network weights at the middle and output layers are initially set and remain fixed – i.e. no training. The second paradigm deigns that the middle layer weights remain fixed and only the output layer weights are trained. The third and most flexible design allows for training of both the middle and output layer weights. The particular network allows for training of both the middle and output layer weights. The particular network used in the model is designed according to the second paradigm, in that the matrix of weights at the middle hidden layer (v_{nm}) is initially set equal to the matrix of input training exemplars (X_{nq}) and then not adjusted further. Only the weights at the output layer (u_{mj}) are trained to reduce the sum of squared error for the network. Hidden layer neurodes number the same as the number of input exemplars (M=Q), with each neurode having the same number of components (N) as the input vectors' features. Put another way, "The *center vector* $\mathbf{v_m} = (v_{1m},\ldots, v_{Nm})$ at the *m*th hidden neurode has *N* components to match the input feature vector." (Looney, 1977: 96) A spread parameter (σ) is calculated using the formula:

$$\sigma = \frac{1}{(2 \cdot M)^{1/N}} \qquad \textbf{(17)}$$

As exemplar vectors (**X**) 'proceed' through the network, the square of its' distance from the center vector ($\mathbf{v_m}$) is calculated. The idea being, the neurode activation function will react more strongly as **X** is closer to the center vector of the particular neurode, with $\mathbf{X} = \mathbf{v}_m$ resulting in the strongest response. Middle layer outputs y_m are calculated as shown in Figure 10. At each upper layer output neurode, initial weights (u_{mj}) are set by a random draw, multiplied by the appropriate middle layer outputs, summed, and divided by *M* to attain a model output (z_j). Upper layer weights are adjusted via supervised training (similar to the previously discussed FFN) until changes in total sum of squared error drops below a specified threshold.

Generalized Ensemble Method.

When faced with three network outputs and desiring only one, a method for combining the outputs becomes necessary. Ideally, it is desirable to combine the outputs in such a manner as to reduce the mean squared error as compared to any single network. Each network in the model develops differently since the randomly generated initial weights result in different starting locations and the model uses three different classes of networks. These facts in conjunction with the gradient search method potentially cause each network to point to a different local minimum in the error space. The local minima are important as they capture different performance areas of the data set. Therefore, when the results of different networks are combined, more information is captured and the performance of the model is increased. The generalized method for combining the different network outputs is referred to as generalized ensemble method (GEM). (Perrone and Cooper: 7-8) The generalized ensemble method entails combining N networks (*$f_i(x)$*) such that $f_{GEM}(x) \equiv \sum_{i=1}^{i=N} \alpha_i f_i(x) = f(x) + \sum_{i=1}^{i=N} \alpha_i m_i(x)$. The α_i's must satisfy the constraint $\sum \alpha_i = 1$, and m_i is defined as the difference between the network $f_i(x)$ and the true, unknown function $f(x)$. Perrone and Cooper define a correlation matrix C_{ij} as $E[m_i(x)m_j(x)]$ and propose minimizing the $MSE[f_{gem}]$ by minimizing $\sum_{i,j} \alpha_i \alpha_j C_{ij}$.

Furthermore, the authors state that $\alpha_i = \frac{\sum_j C_{ij}^{-1}}{\sum_k \sum_j C_{kj}^{-1}}$ will minimize the desired MSE. Put simply, the correlation matrix between the different networks allows calculation of "weights" to be applied to the output of each net. Simply summing the weighted outputs of each network produces a new model that reduces the MSE of the overall model. This result stems from different parts of the error space being captured by the different networks, but combining the networks allows the capture of more of the error space than any single model.

Using the tools and techniques described in this section, it becomes possible to develop a model for determining and predicting free flight reliability using a ground test database, three neural networks and a fusion of network outputs.

III. Methodology

As with the models developed by other agencies, the objective of this thesis is to create a more detailed, easily maintainable model that accurately predicts cruise missile reliability. It should be noted that the focus of this thesis is to improve upon free flight reliability, although the techniques could also be applied to the captive carry portion of the missile reliability equation. The steps taken in the course of this thesis ensure maximum accuracy in model results.

1. As the other weapons communities have done, develop a good target vector for the networks by adding more definition to cruise missile flight test reliability calculations.

2. Convert the ground test data into a usable form (reduce).

3. Engage in an exercise in feature selection.

4. Develop a Matlab model prototype.

5. Validate the model via problems with known solutions.

6. Apply an appropriate data fusion technique to the different network outputs (logistic regression, feed-forward and radial basis function).

7. Put the model into the form of a usable tool for the end-user – convert the model into visual basic for applications (VBA) and save into a MS Excel worksheet containing the database.

Add Definition to Flight Test Reliability

To attain valid outputs from a model, valid targets must be used. Therefore, an examination of the inflight portion of the mission is in order. During reliability testing, "Methods exercising all product operational modes should be described." and "…the effective use of test resources and the validity of the data collected require that a degree of rigor be included such that the product is operated and stresses as intended…" (Morris: 255-256) A review of the technical order (TO) for AGM-129 (ACM -- TO 21-AG129-2-1: 1-30 – 1-34), and conversations with subject matter experts reveals some natural break points in the course of a mission that can be used to further define the operational modes of the missile. During captive carry the missile has two identifiable phases: transit and prelaunch. The transit phase includes the time after the aircrew has accepted the aircraft but prior to prelaunch. Prelaunch phase begins with missile warm-up and extends up to (but not including) missile separation. The flight phase of the missile is broken down into three phases: transition to cruise, cruise and endgame. Transition to cruise begins with missile separation and ends after the missile separation maneuver is completed. The cruise phase begins with the missile flying to the first waypoint and ends prior to the warhead arming maneuver. Endgame begins with the warhead arming maneuver and terminates with missile detonation. Figure 11 illustrates the sequence of events for a typical mission.

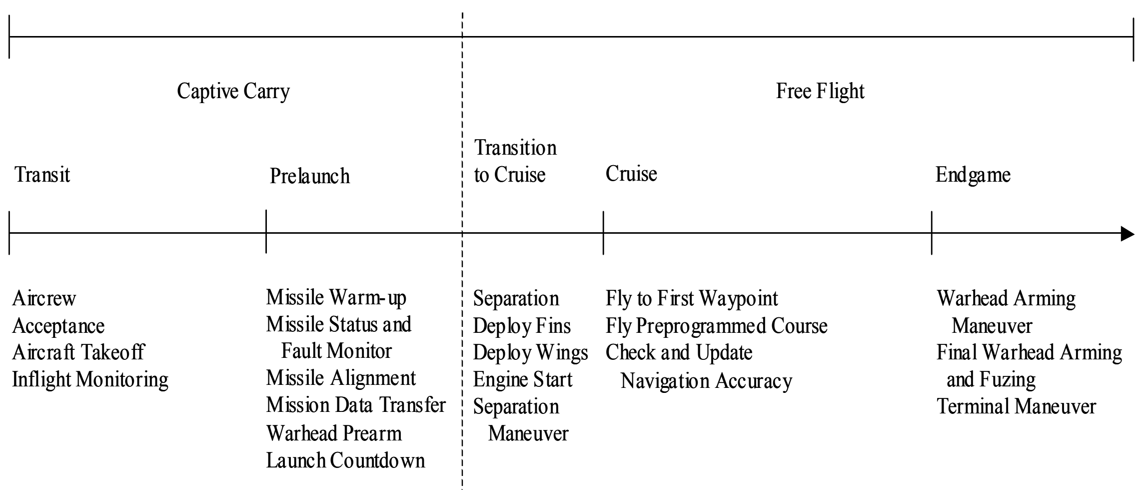


Figure 11: Mission Sequence (TO 21-AG129-2-1: 1-30 – 1-34)

Each flight test missile uses a telemetry kit to provide the ground station with missile status. Flight test failures are investigated fully until a causative factor for the failure is identified. As a result, the mission phase where a failure-causing fault occurs is readily identifiable. Using the natural breakpoints in the mission profile, more detailed reliability equations for missile reliability (equation 11) become evident.

$$CCR = CCTR \times CCPR \quad \textbf{(18)}$$

where

CCTR = Captive Carry Transit Reliability

CCPR = Captive Carry Prelaunch Reliability

$$FFR = FFTR \times FFCR \times FFER \quad \textbf{(19)}$$

where

FFTR = Free Flight Transition to Cruise Reliability

FFCR = Free Flight Cruise Reliability

FFER = Free Flight Endgame Reliability

Data Reduction

The data being considered for use in the model is standardized into pass rates per month using the simple formula:

$$PassRate = \frac{\#\_missiles\_passed\_test}{\#\_missiles\_tested} \quad \textbf{(20)}$$

The pass rates for MITs and SITs are adjusted for false negatives using Level 1 Type B results. Missiles passing Type B testing are credited back to the MIT and SIT pass rates in proportion to the number of missiles undergoing test.

$$MIT\_proportion = \frac{\#\_missiles\_failing\_MIT}{\#\_missiles\_failing\_MIT + \#\_missiles\_failing\_SIT} \quad \textbf{(21)}$$

$$TypeB\_MIT\_adjustment = \#\_missiles\_passed\_TypeB \times MIT\_proportion \quad \textbf{(22)}$$

$$MIT\_PassRate = \frac{\#\_missiles\_passed\_MIT + TypeB\_MIT\_adjustment}{\#\_missiles\_tested\_via\_MIT} \quad \textbf{(23)}$$

$$SIT\_proportion = \frac{\#\_missiles\_failing\_SIT}{\#\_missiles\_failing\_MIT + \#\_missiles\_failing\_SIT} \quad \textbf{(24)}$$

$$TypeB\_SIT\_adjustment = \#\_missiles\_passed\_TypeB \times SIT\_proportion \quad \textbf{(25)}$$

$$SIT\_PassRate = \frac{\#\_missiles\_passed\_SIT + TypeB\_SIT\_adjustment}{\#\_missiles\_tested\_via\_SIT} \quad \textbf{(26)}$$

Another consideration is whether to use monthly data or annual averages. When making the decision, one should first consider continuity of the data. Analysis of the data reveals MITs are primarily run in the course of exercises and aircraft generations – i.e. they are not accomplished every month. Using the monthly averages would cause considerable gaps in the database and render the test unusable as a feature. As a second matter of course, missile MIT failures will result in Level 1 Type B re-testing to verify faults. In some cases, the Type B verification is not run in the same month as when the MIT fault was realized; or the missile testing "bleeds-over" into another month. In that case, the Type B adjustment to the MIT pass rate would not be credited to the appropriate month. Annual averages alleviate the "bleed-over" problem by using the raw numbers accumulated over the course of the year and making the adjustments at year's end. As a final note, STRATCOM only requires annual numbers (rates per FY) for their planning factors.

Model Feature Selection

Once again, one should note that this thesis focuses solely on the free flight portion of the missile reliability equation, but the same feature selection techniques can be applied toward developing an analogous model for captive carry reliability. In developing the neural networks for predicting free flight reliability, pertinent features must be selected from a ground test database (database synopsis presented in Appendix C). Using all the available tests may give a more precise estimate of the desired reliability, however running the entire set of input features through the model could be time consuming as well as unnecessary. Ideally, a feature set that adequately represents the underlying structure of the data while providing an accurate estimate of the chosen reliability is desirable. The database compiled previously is comprised of numerous ground test results conducted on Air Launched Cruise Missiles compiled over 13 years (FY1990 through FY2002). The few empty data fields (years where tests of that nature were not performed – SIT testing primarily) are filled in by interpolation estimates. Changes in the manner of tracking the test data also result in using estimates for certain fields – LLT/LPT Types A and B primarily. Test definitions and feature selection techniques can be used to reduce the number of ground tests to be used as inputs in the model. The selected inputs are then validated against subject matter expert opinion. Table 4 summarizes the data fields available as potential model features.

Table 4: Database Summary

GROUND TEST	DESCRIPTION
Loaded Launcher Test / Loaded Pylon Test (LLT/LPT) Type A	After package build-up; run to certify operational capability of package; communication test primarily – will the aircraft be able to communicate through the pylon/launcher and down to the missile
LLT/LPT Type B	Identical to Type A except run to verify previous SIT or MIT failure
Missile Interface Test (MIT)	Communication test between the aircraft and the missile -- normally performed after package upload onto the aircraft.
Systems Interface Test (SIT)	More involved test than MIT; must be performed (per technical order) if a single missile swap occurs on the flight line
Level I Test, Type A	Run after a 72-month engine change or other periodic maintenance; deep cycle electronic test run by the ground test set
Level I Test, Type B	Identical to Type A except run as a verification of MIT, SIT or LLT/LPT fault indication -- when a memory dump from a previously mentioned test indicates a problem in a missile area, the Level 1 Type B runs component BITs, interrogates components, and compares and validates proper responses to diagnose the problem down to the component level.
Level III Test, Type B	Run after a Level 1 test indicates a problem with a specific component – diagnoses problem down to sub-component level
INE Auto-Calibrations	Performed every 48 months – specifically checks to ensure INE is operating correctly and not drifting beyond tolerance limits

By definition, Type B testing only occurs as a result of a Type A test failure. Therefore, all Type B testing is excluded from the model except for use as an adjustment factor. The remaining tests of interest include, LLT/LPT Type A, SIT, MIT, Level 1 Type A and INE Auto-cal. Additionally, previous year flight test results are added to the list of possible features, now totaling six potentials. Two techniques are used for feature selection: factor analysis and backwards-selection logistic regression. All flight test data (previous year results only used for factor analysis; previous and current year results used for backwards-selection logistic regression) used in both approaches are notional for classification purposes. Table 5 illustrates the input matrix used for both techniques. Shaded fields denote estimated data.

Table 5: Input Matrix – Potential Features

FY	LLT A	SIT	MIT	Lvl 1 A	INE	Prev Yr	Flt Test
90	96.03%	88.95%	93.88%	82.66%	94.10%	67.00%	75.00%
91	95.63%	96.34%	96.84%	81.87%	95.60%	75.00%	75.00%
92	95.32%	98.79%	99.10%	78.63%	97.45%	75.00%	50.00%
93	93.98%	93.64%	98.18%	79.57%	95.15%	50.00%	67.00%
94	93.13%	96.74%	98.75%	80.43%	95.42%	67.00%	75.00%
95	94.44%	94.90%	96.84%	81.22%	95.37%	75.00%	50.00%
96	95.04%	84.62%	99.00%	79.07%	96.94%	50.00%	67.00%
97	95.00%	100.00%	97.96%	78.05%	94.39%	67.00%	75.00%
98	95.09%	93.72%	98.65%	79.58%	93.72%	75.00%	100.00%
99	94.97%	91.18%	97.67%	73.49%	93.14%	100.00%	100.00%
00	95.48%	100.00%	99.37%	83.46%	96.48%	100.00%	100.00%
01	96.19%	100.00%	99.21%	71.10%	90.65%	100.00%	75.00%
02	92.06%	94.91%	99.46%	55.15%	84.13%	75.00%	100.00%

estimated data

A factor analysis is performed to investigate underlying dimensions of the data set. Using SAS to perform the factor analysis on the matrix of potential features (columns 2-7 of Table 5), the resulting eigenvalues suggested a 3-factor model as appropriate (Kaiser's Criterion). A Varimax rotation was applied to see how the features loaded with the following results (Table 6 -- full SAS factor analysis output available in Appendix D):

Table 6: Factor Analysis Results (abbreviated)

Eigenvalues of the Correlation Matrix: Total = 6 Average = 1

	Eigenvalue	Difference	Proportion	Cumulative
1	**2.57942941**	0.93849633	0.4299	0.4299
2	**1.64093307**	0.64655285	0.2735	0.7034
3	**0.99438022**	0.49333963	0.1657	**0.8691**
4	0.50104060	0.25566604	0.0835	0.9526
5	0.24537455	0.20653240	0.0409	0.9935
6	0.03884215		0.0065	1.0000

3 factors will be retained by the NFACTOR criterion.

Rotated Factor Pattern

	Factor1	Factor2	Factor3
LLTA	**0.60440**	0.55597	0.39962
SIT	0.06236	0.58353	**0.64407**
MIT	-0.19761	0.00203	**0.88377**
Level1A	**0.95825**	-0.02802	-0.18447
INE	**0.97243**	-0.12875	-0.00015
PrevYr	-0.16770	**0.92043**	0.10361

Variance Explained by Each Factor

Factor1	Factor2	Factor3
2.3002360	1.5141744	1.4003323

Final Communality Estimates: Total = 5.214743

LLTA	SIT	MIT	Level1A	INE	PrevYr
0.83410138	0.75922666	0.82009747	0.95306195	0.96220513	0.88605012

Communality estimates suggest that a 3-factor model design adequately explains the majority of the variance in the individual variables and, therefore is appropriate.

Running across the columns with regard to each feature, the maximum values are circled and boldface. Each maximum value is grouped with the others in the column and an analysis of the groupings reveals corresponding categories. Table 7 shows a translation of the factor analysis results into categories. As a rule of thumb, the model should include one of the relevant features under each of the factor columns.

Table 7: 3-Factor Analysis Breakdown

	Factor 1	Factor 2	Factor 3
Category	IMF Testing	Flight Testing	On-Acft Testing
Relevant Features	Level 1 Type A INE Auto-cal LLT/LPT Type	Previous Year Flight Test	SIT MIT

A backwards-selection logistic regression is run on the same data shown in Table 5, with the code utilized shown in Appendix E. Columns 2-7, along with a bias column, were used as features with the last column serving as the target. After examining the absolute value of the resultant weights, and removing from the model the feature corresponding to the weight smallest in magnitude, the model is re-run. Table 8 shows the results of the backwards-selection regression with shaded elements to show the features eliminated and the model formed as a result. In the first case, all the features (6) are included in the regression. The calculated weights are shown in the first data row of Table 8. In this case, the weight associated with the SIT feature (shaded) has the smallest magnitude – so it is removed from the model. The logistic regression code is run again with only the bias, level 1, INE, LLT A, Prev Yr and MIT features (5) included. From the second run, the LLT A feature has the smallest associated weight and so it is eliminated from the next run. The process continues until only three features remain, as suggested by the factor analysis. Feature elimination is also tempered with judgment based upon factor analysis results. Total error is tracked to verify only minor changes occurring as the features are eliminated.

Table 8: Backwards-Selection Logistic Regression Results

	Weights						
Factor		IMF Testing			Flt Test	On-Acft Testing	
Error	Bias	Level 1	INE	LLT A	PrevYr	SIT	MIT
0.2766	0.4755	-1.4109	-0.7661	0.2540	2.5231	-0.1943	0.6601
0.2792	0.4001	-1.3285	-0.7275	0.2010	2.4317		0.5675
0.2796	0.4535	-1.2879	-0.6740		2.4400		0.6172
0.2831	0.2300	-1.4920			2.3869		0.3995

Plots of the backwards-selection regression results (model outputs from 6, 5, 4 and 3 feature networks) are shown in Figure 12. For the sake of comparison, repeated regression traces are shown as solid lines with the notional flight test results displayed as a dashed line. As shown, the LogReg results closely overlay each other; making it seem as if only one plot is shown.

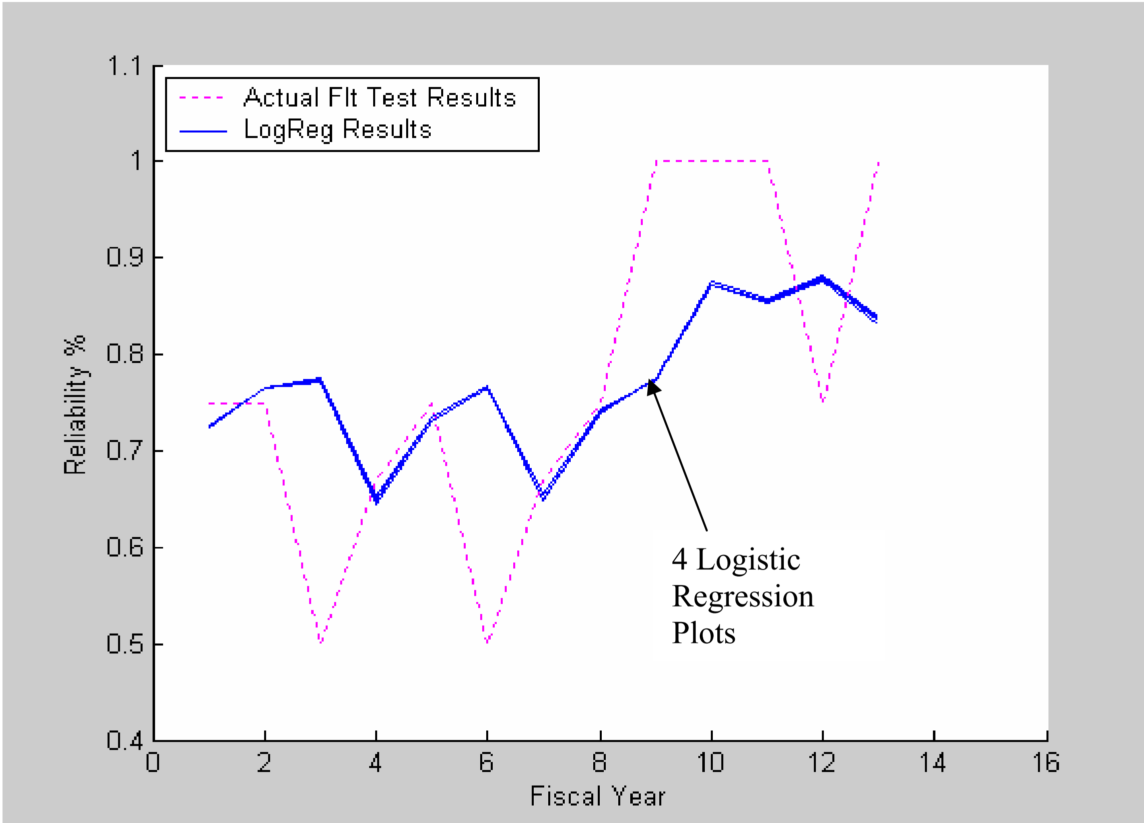


Figure 12: 3-Factor Backwards Regression Results

Error statistics from Table 8 and the log-reg plot from Figure 12 show little change with the removal of the selected features. Therefore, the feature selection results suggest the following features for use in the neural network: Level 1 Type A, MIT, and Previous Year Flight Test. The three features also happen to coincide with subject matter expert opinion (Bredehoeft, 2002), lending validity to the feature selection techniques used.

Using the aforementioned rationale, with notional flight test data included, a matrix of input vectors results as illustrated by Table 9:

Table 9: Missile Test Data

	ALCM Model Features			Target
FY	MIT	Level 1 A	Prev Yr	Flt Test
1990	93.88%	82.66%	67.00%	75.00%
1991	96.84%	81.87%	75.00%	75.00%
1992	99.10%	78.63%	75.00%	50.00%
1993	98.18%	79.57%	50.00%	67.00%
1994	98.75%	80.43%	67.00%	75.00%
1995	96.84%	81.22%	75.00%	50.00%
1996	99.00%	79.07%	50.00%	67.00%
1997	97.96%	78.05%	67.00%	75.00%
1998	98.65%	79.58%	75.00%	100.00%
1999	97.67%	73.49%	100.00%	100.00%
2000	99.37%	83.46%	100.00%	100.00%
2001	99.21%	71.10%	100.00%	75.00%
2002	99.46%	55.15%	75.00%	100.00%

Matlab Prototype

With the preparatory work completed, it is now possible to develop a model to predict the desired reliability. Although the final version is a standalone model, written in VBA and nested in the same MS Excel workbook as the database, the majority of the development and validation is Matlab. The code is presented in full in Appendix F.

For developmental purposes, the matrix of input values (Table 9, columns 2 – 5) is hard coded into the file. The user sets the number of years upon which the networks will train as well as the number of out-years to predict. The same matrix is used in each network in turn – logistic regression, feed-forward neural network and radial basis function network (Figure 13).

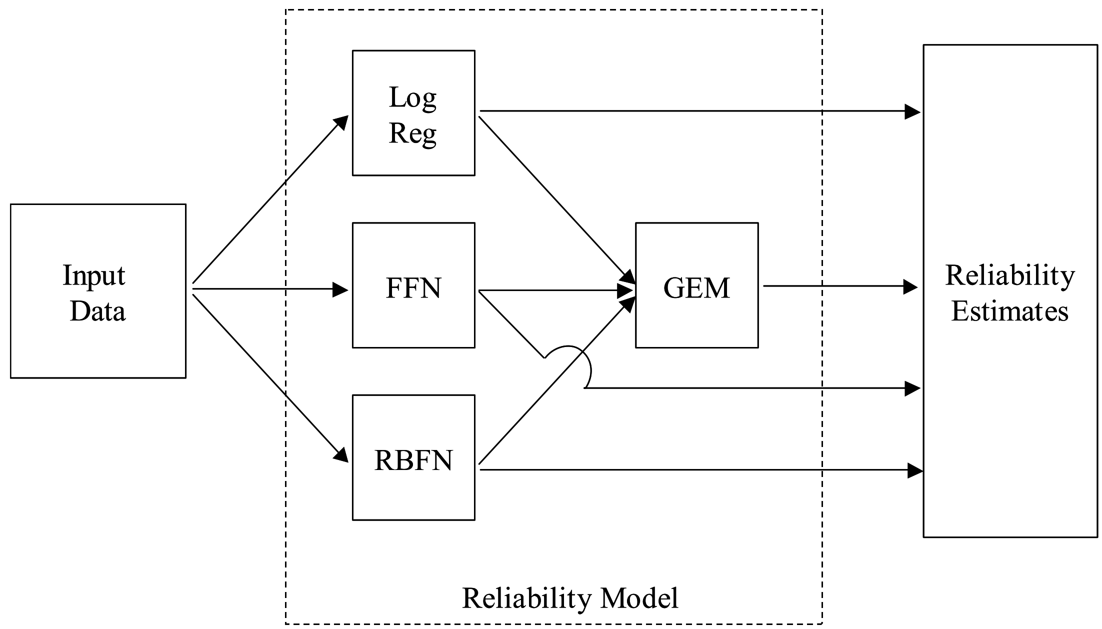


Figure 13: Reliability Model Block Diagram

Using training algorithms given in class notes (Bauer, 2002) and the Looney text (Looney, 1977: 99-100, 125), the different networks train and generate outputs. The weights developed in training are used to run the remaining exemplars through the networks and generate prediction outputs. Training and prediction outputs are presented graphically along with the target vector for the sake of comparison (Figures 14 and 15). The cluster of traces running through the center of each chart suggests similar estimate and predictive outputs from the different networks in the model. The numerical model results are also displayed in tabular format (Tables 10 and 11).

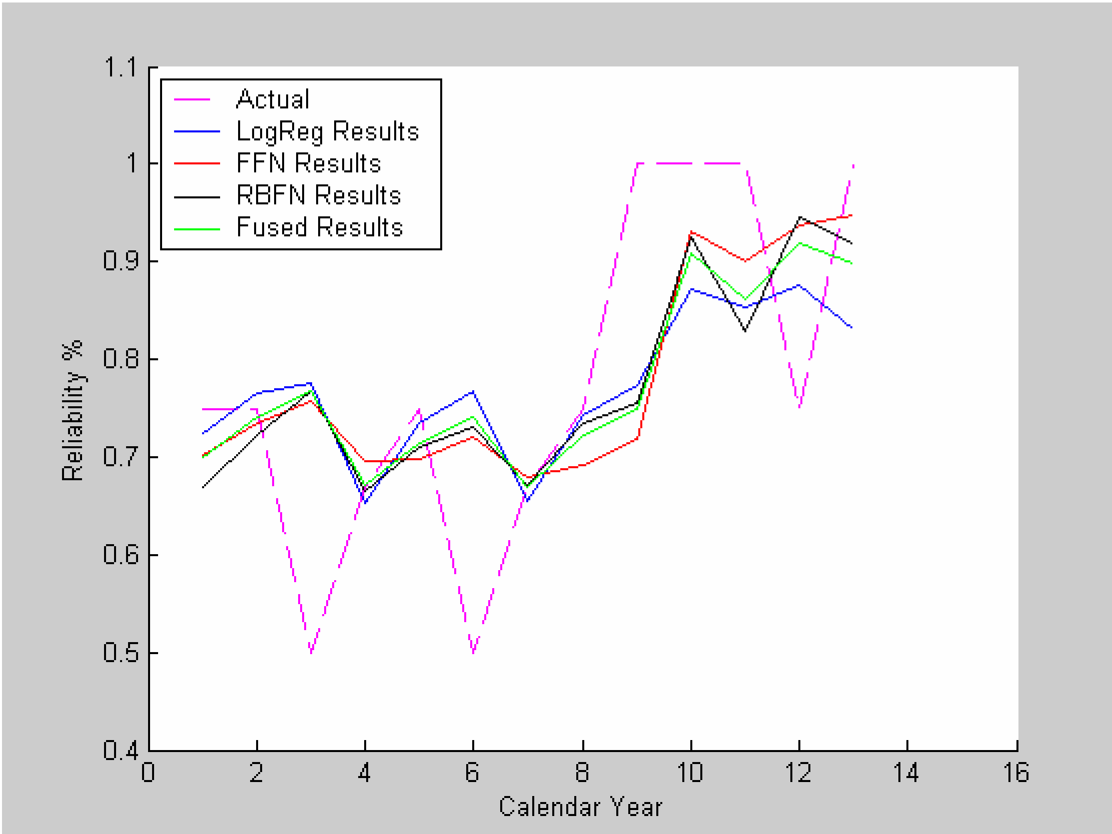
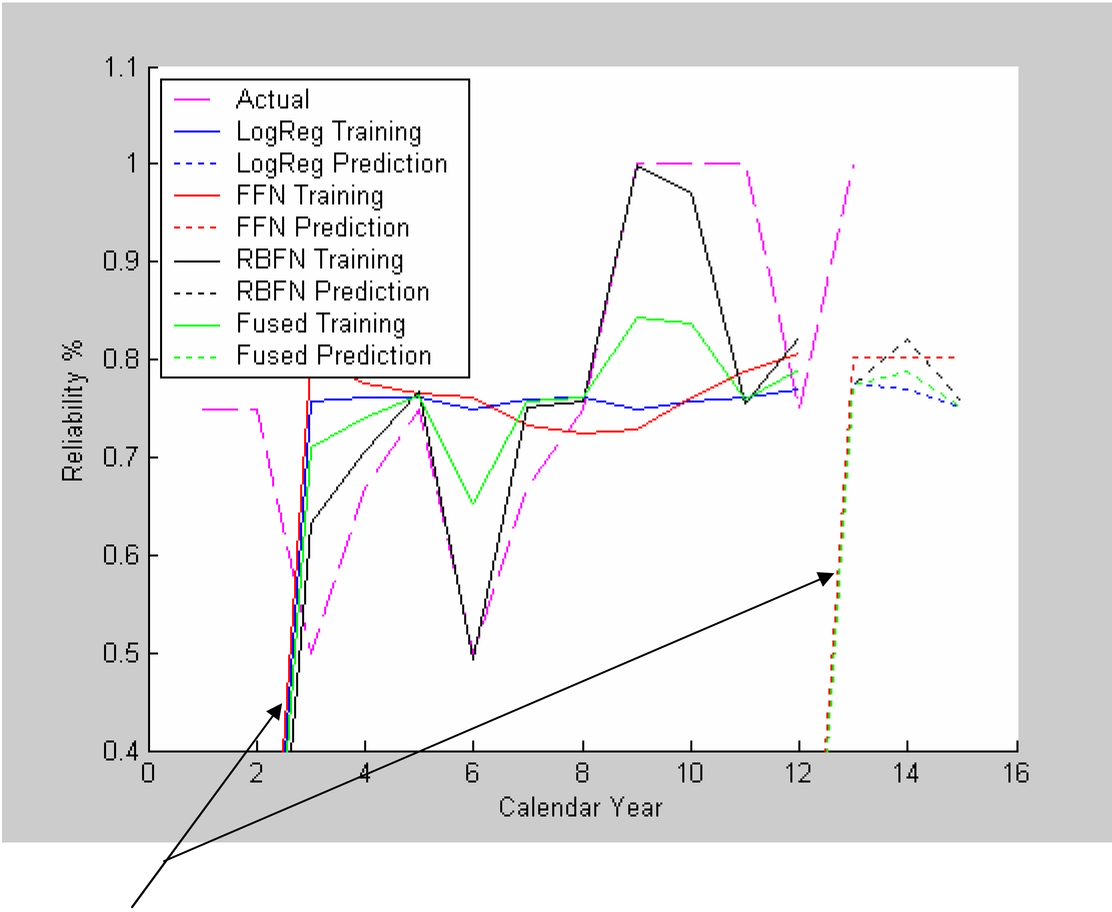


Figure 14: Current Year Reliability Estimates

Table 10: Current Year Reliability Estimates

	FY90	**FY91**	**FY92**	**FY93**	**FY94**	**FY95**	**FY96**
Z_{LR}	0.7253	0.7659	0.7761	0.6521	0.7357	0.7677	0.6545
Z_{FF}	0.7026	0.7340	0.7574	0.6949	0.6970	0.7209	0.6801
Z_{RBF}	0.6698	0.7232	0.7664	0.6654	0.7103	0.7310	0.6717
Z_{GEM}	0.6995	0.7413	0.7667	0.6707	0.7145	0.7400	0.6687
	FY97	FY98	**FY99**	**FY00**	**FY01**	**FY02**	
Z_{LR}	0.7419	0.7732	0.8711	0.8543	0.8757	0.8313	
Z_{FF}	0.6917	0.7175	0.9302	0.8998	0.9381	0.9473	
Z_{RBF}	0.7345	0.7547	0.9249	0.8298	0.9456	0.9193	
Z_{GEM}	0.7227	0.7486	0.9085	0.8614	0.9195	0.8988	



* Trace dropoffs due to Matlab graphing limitations.

Figure 15: 24-month Reliability Prediction

Table 11: 24-month Reliability Prediction

	FY90	**FY91**	**FY92**	**FY93**	**FY94**	**FY95**	**FY96**	**FY97**
$\mathbf{Z_{LR}}$			0.7564	0.7619	0.7618	0.7494	0.7586	0.7615
$\mathbf{Z_{FF}}$			0.8009	0.7751	0.7646	0.7621	0.7332	0.7251
$\mathbf{Z_{RBF}}$			0.6262	0.6973	0.7583	0.4836	0.7436	0.7457
$\mathbf{Z_{GEM}}$			0.7140	0.7426	0.7604	0.6519	0.7583	0.7565

	FY98	**FY99**	**FY00**	**FY01**	**FY02**	**FY03**	**FY04**
$\mathbf{Z_{LR}}$	0.7497	0.7568	0.7620	0.7701	0.7761	0.7700	0.7501
$\mathbf{Z_{FF}}$	0.7279	0.7620	0.7873	0.8069	0.8026	0.8024	0.8013
$\mathbf{Z_{RBF}}$	0.9889	0.9593	0.7457	0.8154	0.7808	0.8294	0.7655
$\mathbf{Z_{GEM}}$	0.8329	0.8303	0.7562	0.7830	0.7791	0.7562	0.7750

Code Validation

Although the Matlab code follows the higher-level training algorithms as previously discussed, the code must be validated against a problem with a known answer to determine if it is performing correctly.

The full validation code is presented in Appendix G. For the logistic regression network, a set of 30 data points is randomly drawn over the range [1,10] and a target vector is developed using the logistic function: $t(x) = \dfrac{1}{1 + \exp(-(\beta_o + \beta_1 \cdot x))}$. The network trains on the first 20 points and predicts on the last 10 points. Both sets of data are plotted to show coincidence. If the network is coded properly, the network training and prediction outputs should plot a line that is near identical to the input data set and produce weights such that $\beta_o = -1.5$ and $\beta_1 = 0.6$. Figure 16 shows the results of the logistic regression verification code. The network results plot easily matches the target values and the calculated weights are w = -1.4999   0.6000, supporting the contention that the code logic is performing as expected.

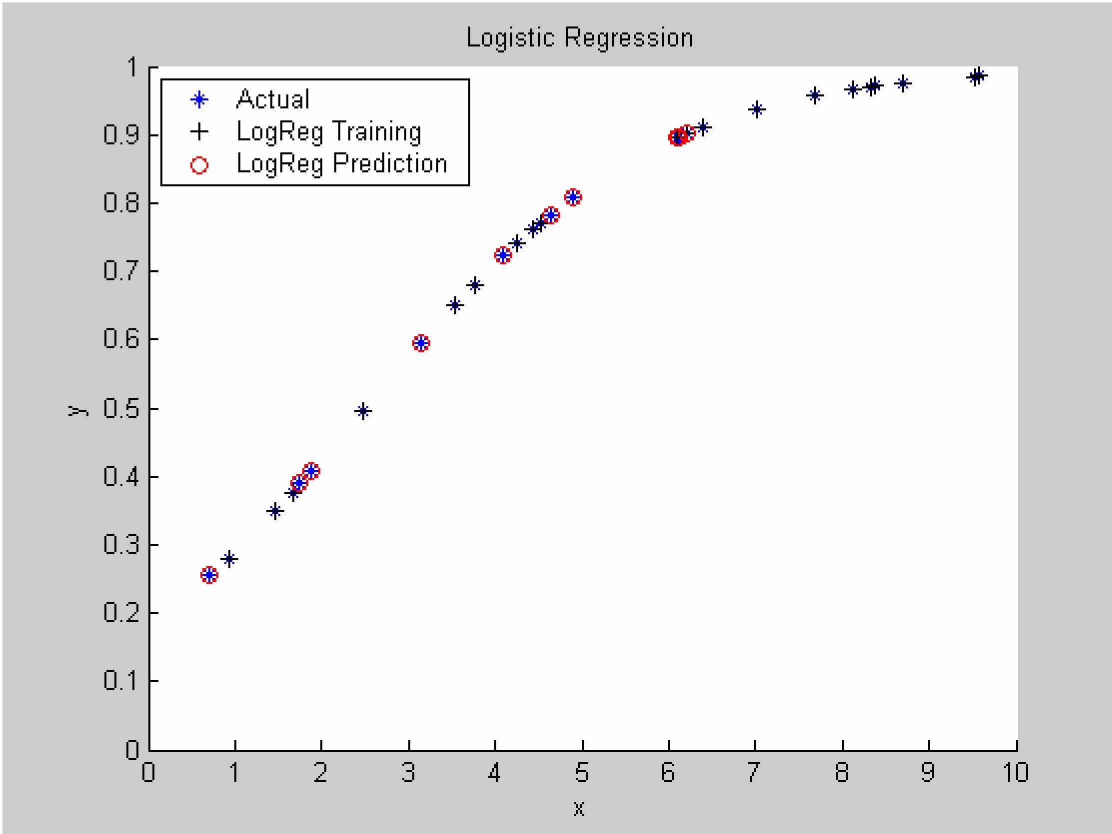


Figure 16: Logistic Regression Validation

The other two networks (feed forward and radial basis function) are another matter. The code for the feed forward network and the radial basis function network is robust enough to be used for classification as well as estimation, so the XOR problem serves as a means for verification. The code presented in Appendix G is identical to the model in Appendix F except the input matrix consists of two columns of uniformly generated numbers between [-1, 1]. The columns correspond to X and Y Cartesian coordinates (Figure 17).

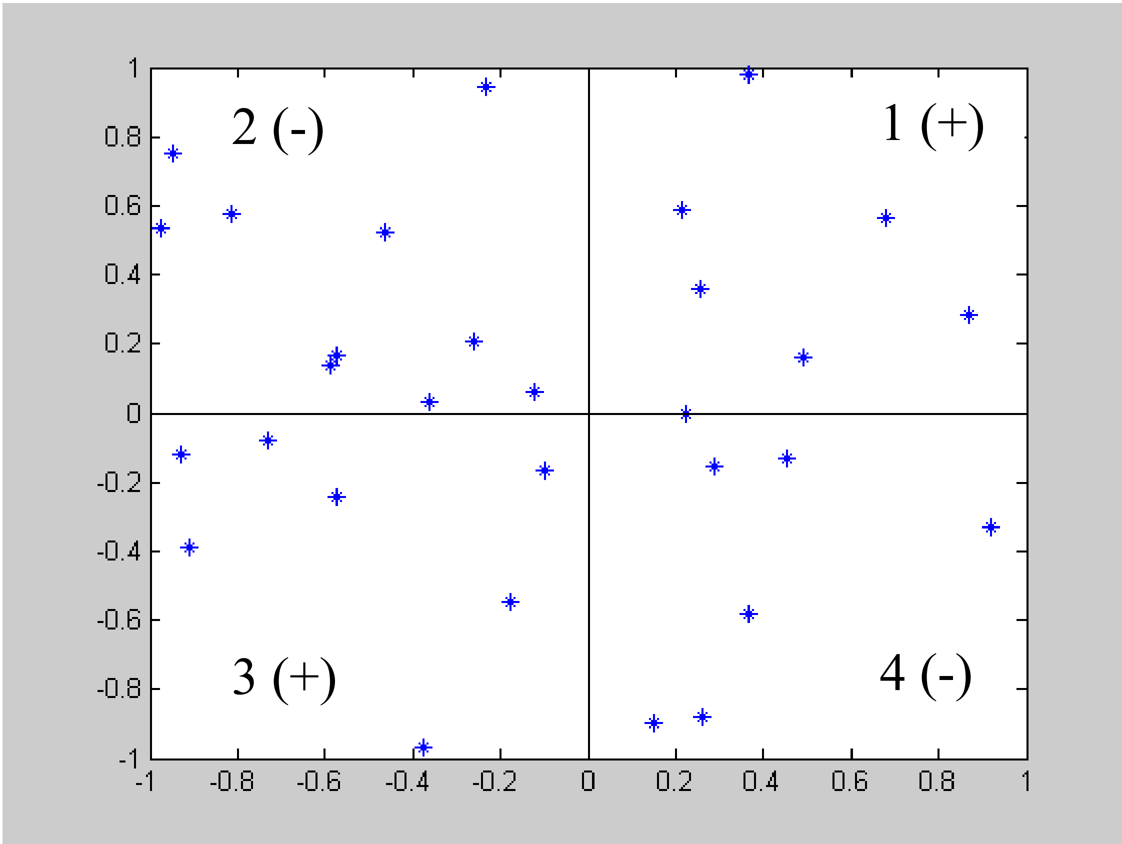


Figure 17: Random Input Data Classification

A corresponding target vector is generatied based upon the categorization of the data into two classes: (0,1) for quad 1 or 3 membership, (1,0) for quad 2 or 4 membership. A confusion matrix is calculated at the end of the code as a measure of classification accuracy. As a naming convention, quad 1 or 3 membership is given as positive while quad 2 or 4 membership is given as negative. Results from the confusion matrices are shown in Table 12.

Table 12: Network Verification Confusion Matrices

Actual \ Output	**Example**	
Actual	Pos	Neg
Pos	True Pos	False Neg
Neg	False Pos	True Neg
Actual \ Output	**FF Training Results**	
Actual	Pos	Neg
Pos	10	0
Neg	0	10
Actual \ Output	**FF Test Results**	
Actual	Pos	Neg
Pos	5	1
Neg	0	4
Actual \ Output	**RBF Training Results**	
Actual	Pos	Neg
Pos	11	0
Neg	0	9
Actual \ Output	**RBF Test Results**	
Actual	Pos	Neg
Pos	5	1
Neg	1	3

If the networks are coded and functioning properly, the confusion matrices will load heaviest in the 'true positive' and 'true negative' cells. The confusion matrices produced by the validation codes support the contention that the code for the feed forward and radial basis function networks are coded, training and predicting properly.

Fusion

The model generates three outputs that need to be fused into a single estimate of free-flight reliability. Per the generalized ensemble method, network outputs are combined into a single output matrix from which a matrix of correlation coefficients is generated. Using the formulae described in Chapter 2 of this document, the model calculates weights that are applied to the network outputs and then summed to provide a single estimate of reliability. Figure 18 illustrates an example of the GEM method as applied to the outputs generated by the model from the matrix of model inputs (Table 9).

Table 13: Training Outputs

FY	LR	FF	RBF
90	0	0	0
91	0	0	0
92	77.64%	49.86%	60.37%
93	78.27%	79.35%	74.50%
94	78.26%	88.23%	73.89%
95	76.73%	74.34%	47.69%
96	77.85%	77.55%	79.10%
97	78.24%	69.67%	72.20%
98	76.75%	73.02%	97.27%
99	77.67%	85.00%	92.95%
00	78.28%	86.88%	89.91%
01	79.36%	92.58%	90.94%
02	79.96%	88.33%	89.07%

Table 14: Correlation Matrix

1	0.532873	0.370005
0.532873	1	0.557086
0.370005	0.557086	1

$$\alpha_i = \frac{\sum_j C_{ij}^{-1}}{\sum_k \sum_j C_{kj}^{-1}} \quad \textbf{(27)}$$

Table 15: GEM Weights

	LR	FF	RBF
α_i	0.35427	0.296639	0.349091

Multiply elements in each column by the associated weight and add across the rows.

e.g. $R_{FY01} = .7936 \times .35427 + .9258 \times .296639 + .9094 \times .349091 = .7940$ **(28)**

Table 16: Fused Outputs

FY	90	91	92	93	94	95	96	97
Z_{GEM}	0	0	.6337	.7728	.7969	.6588	.7820	.7359

FY	98	99	00	01	02	03	04	
Z_{GEM}	.8281	.8518	.8489	.8733	.8562	.8711	.7940	* Predictions

Figure 18: Generalized Ensemble Method (24-month Prediction Example)

Conversion to VBA

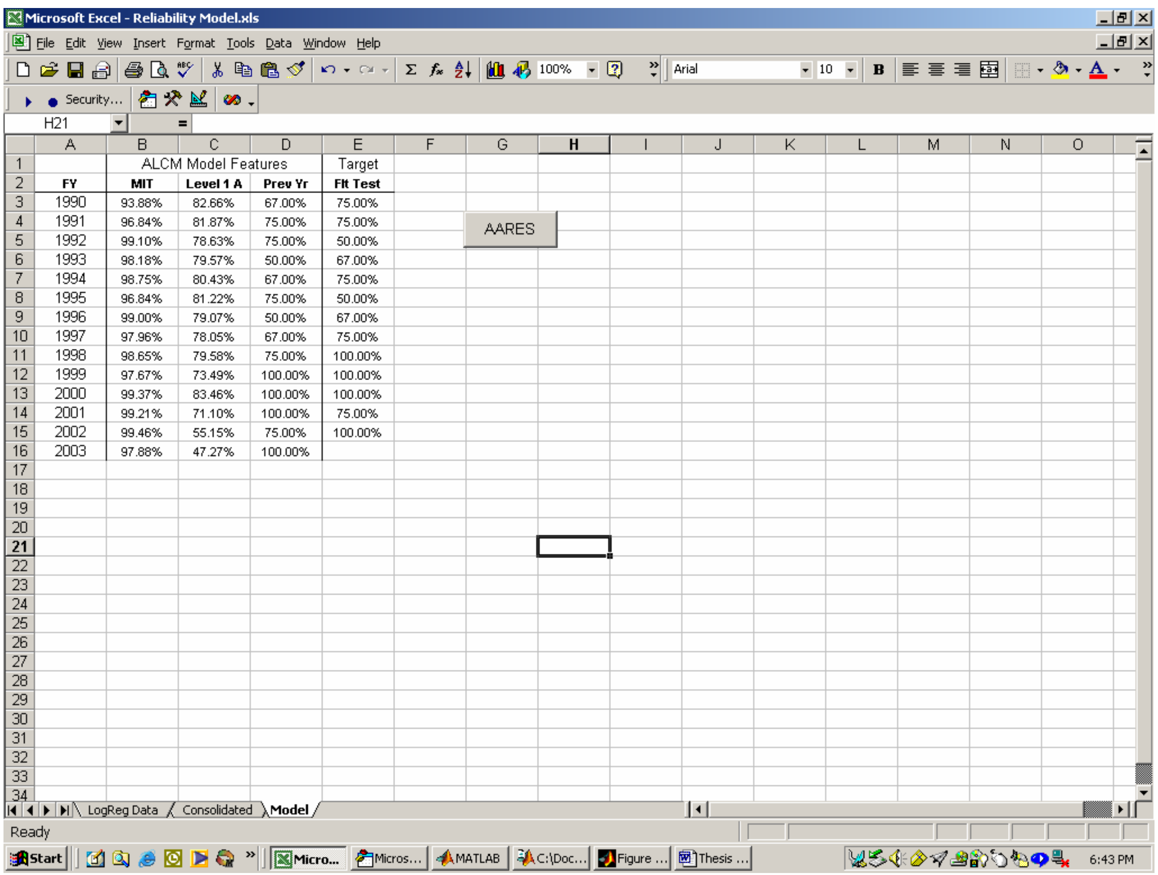
Once the model logic is determined and validated, the code is converted into VBA and nested in the worksheet containing the missile ground test database. In the course of conversion, the name ALCM/ACM Reliability Estimation System (AARES) was selected for the model. The full version of the VBA code is presented in Appendix H. The majority of the conversion consists of syntax changes and partitioning the Matlab code into major subroutines and adding a graphical user interface as well as other utility subroutines as listed below.

1. GUI – collects user input parameters
2. Main – calls all other subroutines based upon GUI inputs
3. Capture – captures model input exemplars and target vector
4. Logistic Regression Network – calculates reliability estimates and presents them in tabular format
5. Feed-Forward Neural Network – calculates reliability estimates and presents them in tabular format
6. Radial Basis Function Network – calculates reliability estimates and presents them in tabular format
7. Fusion – fuses selected network outputs into a single number per year and presents them in tabular format
8. Error – calculates sum of squared errors (SSE), mean squared errors (MSE) and root mean squared errors (RMSE) of each network output
9. Charting – presents a graphical representation of the model outputs

IV. Model Adequacy

As stated previously in Chapter 1, the user desires a simple-to-use, standalone model that uses existing data and data collection, and provides a single estimate of cruise missile reliability up to 24 months in the future.

The user starts on the worksheet containing the features selected from an existing ground test database, and flight test results collected over the past 13 years. On the worksheet is a single button that starts the model and brings up the GUI (Figure 18).



	ALCM Model Features			Target
FY	MIT	Level 1 A	Prev Yr	Flt Test
1990	93.88%	82.66%	67.00%	75.00%
1991	96.84%	81.87%	75.00%	75.00%
1992	99.10%	78.63%	75.00%	50.00%
1993	98.18%	79.57%	50.00%	67.00%
1994	98.75%	80.43%	67.00%	75.00%
1995	96.84%	81.22%	75.00%	50.00%
1996	99.00%	79.07%	50.00%	67.00%
1997	97.96%	78.05%	67.00%	75.00%
1998	98.65%	79.58%	75.00%	100.00%
1999	97.67%	73.49%	100.00%	100.00%
2000	99.37%	83.46%	100.00%	100.00%
2001	99.21%	71.10%	100.00%	75.00%
2002	99.46%	55.15%	75.00%	100.00%
2003	97.88%	47.27%	100.00%	

Figure 19: Model Starting Worksheet

Pressing the "AARES" button brings up the dialog box that allows the user to select the level of user interaction desired: Custom or Quick Estimate (Figure 19).

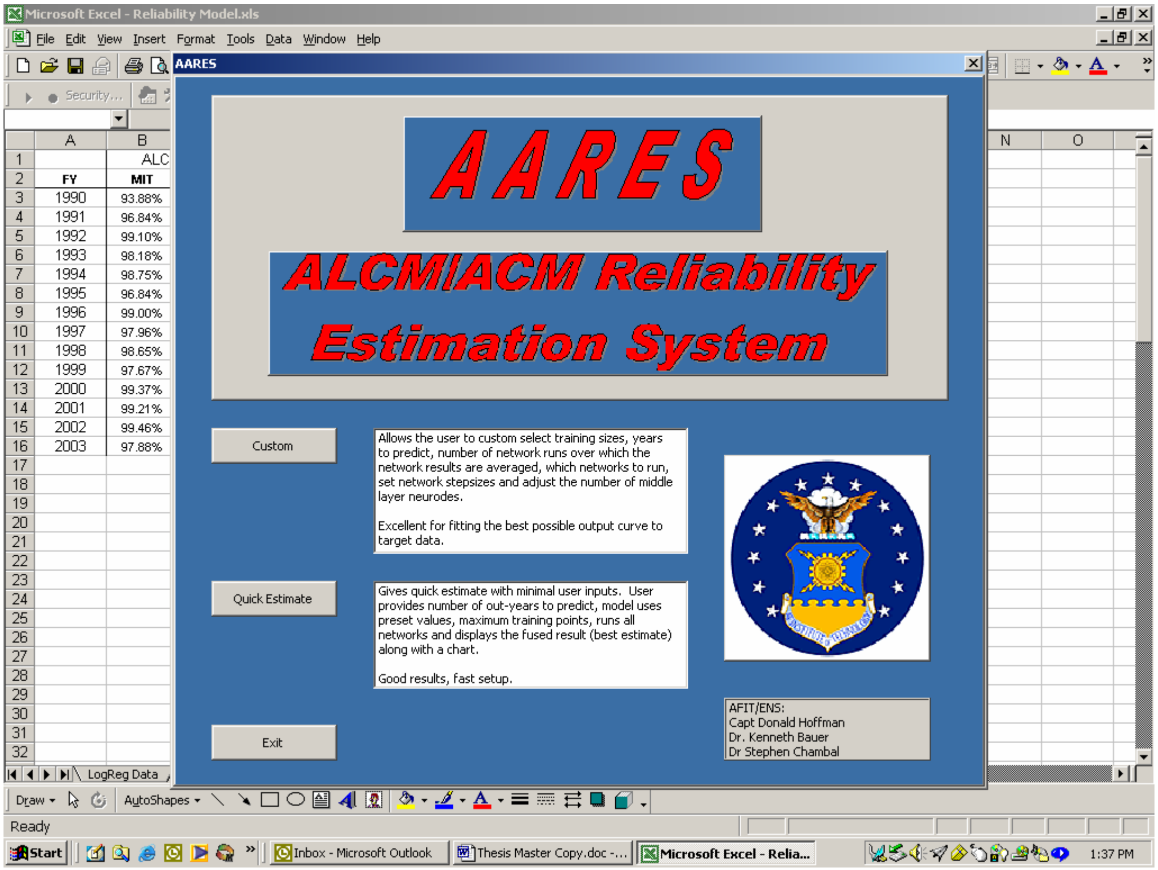


Figure 20: User Interaction Dialog Box

"Custom" allows the user to set parameters for training, out-year prediction, runs over which to average, networks to use and associated stepsize, and number of middle layer neurodes for the FFN (if selected). Instructions for entering data are included in dialog box. Preset values are present in the input windows, pull-downs appear for entering the years for training and out-year prediction, and placing the cursor over an empty input box prompts a "pop-up" suggestion for entering a parameter. Checks are in place to ensure the user selects at least one network and enters appropriate input box values (non-negative, numeric, ranging between 0 and 1, etc…see Figure 20).

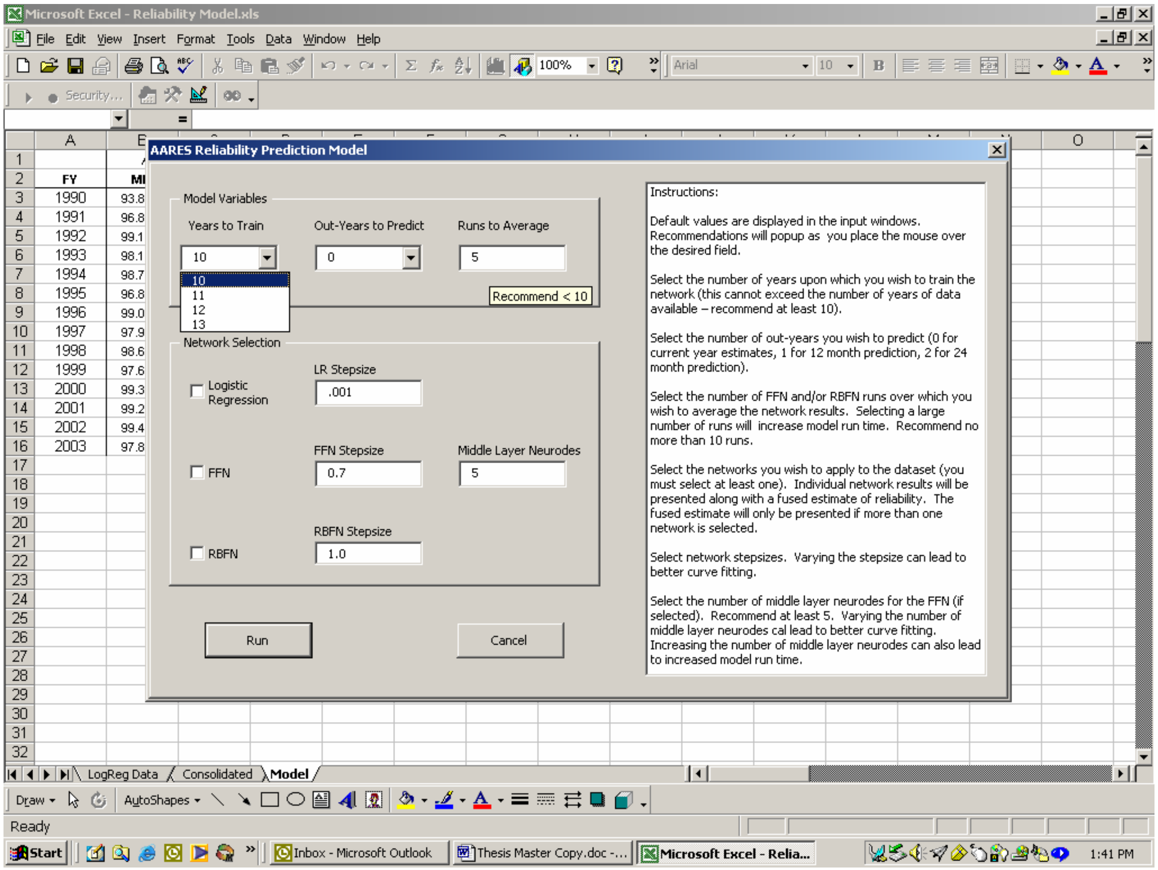


Figure 21: Model Custom GUI

"Quick Estimate" allows the user to get a desired reliability estimate with minimal input. The only required input is out-year prediction; all other values are preset in the code based upon best estimates divined in the course of model design (Figure 21).

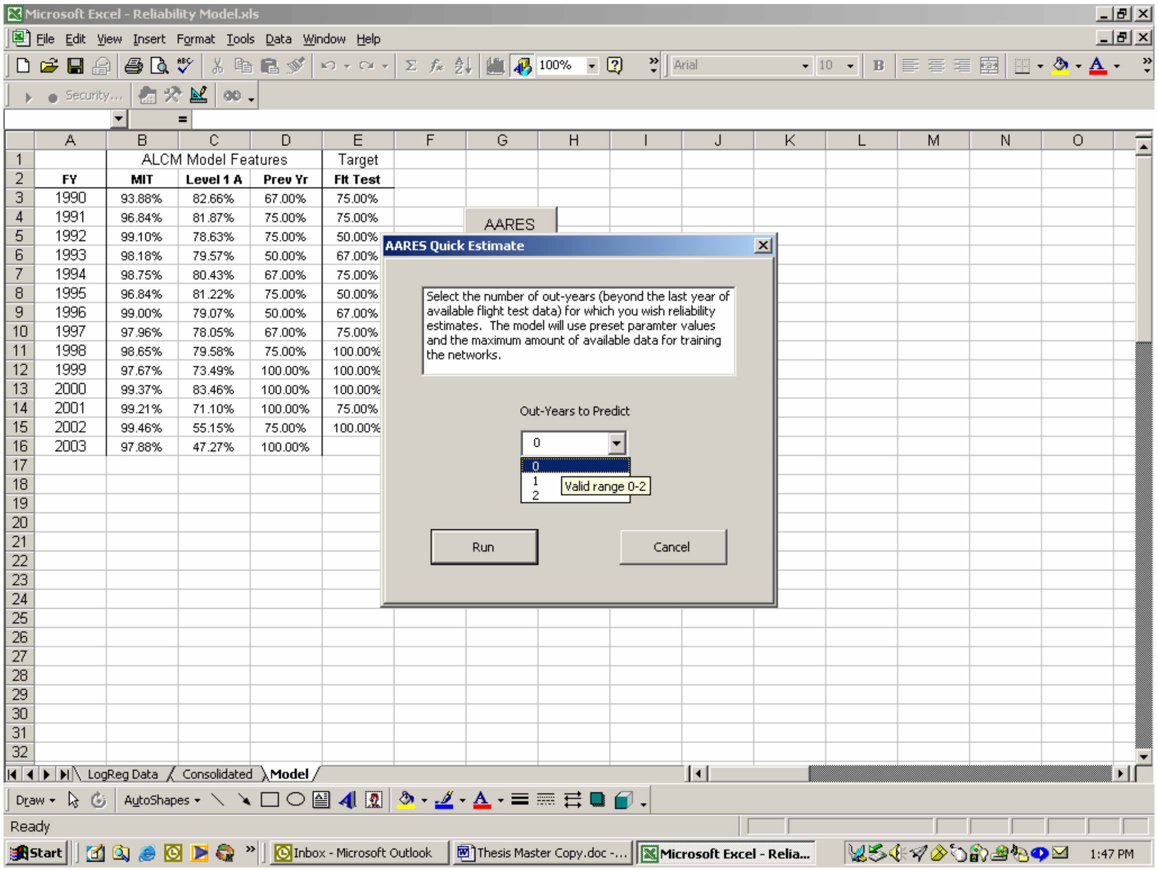


Figure 22: Quick Estimate Input Dialog Box

After all inputs have been entered, the user presses the "Run" button and the model calculates reliability estimates based upon the inputs. If the "Custom" option is selected, reliability estimates are presented in tabular format along with error estimates and a chart presenting a graphical representation of the model outputs (Figure 20).

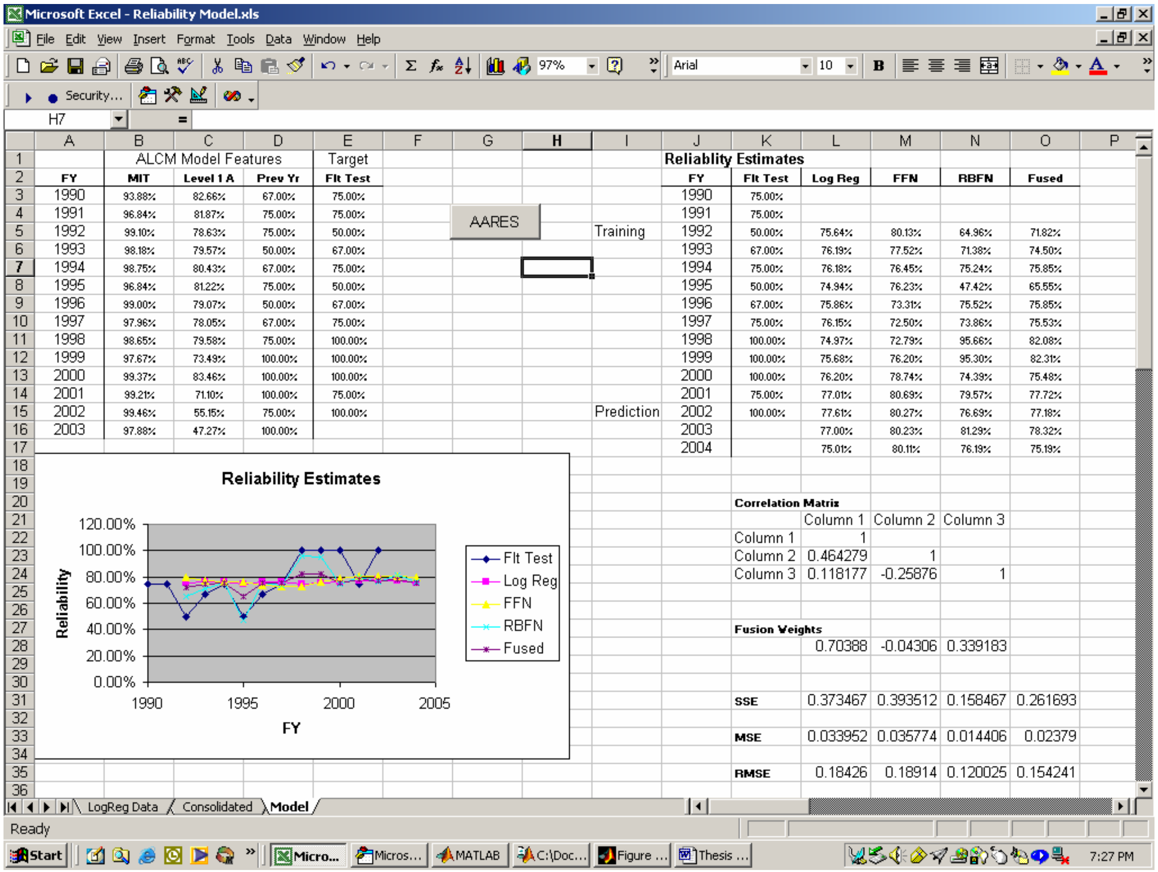


Figure 23: AARES Model Outputs – Custom

For best results, the user should select one network, start with the suggested model parameters, and observe the model-generated chart and error values. The user should then vary the stepsize to fit the best curve to the target vector. Once the user has followed this procedure for each network, he/she can make the decision on which output (logistic regression, feed forward, radial basis function or fused output) gives the best reliability estimate. In most cases, the fused output should give the best overall estimate.

If the "Quick Estimate" is desired, the model runs as if all networks were selected in the "Custom" option and default values were used. At the end of the run, the model presents a full-size chart with text in the upper-right corner displaying the desired reliability estimate (Figure 23).

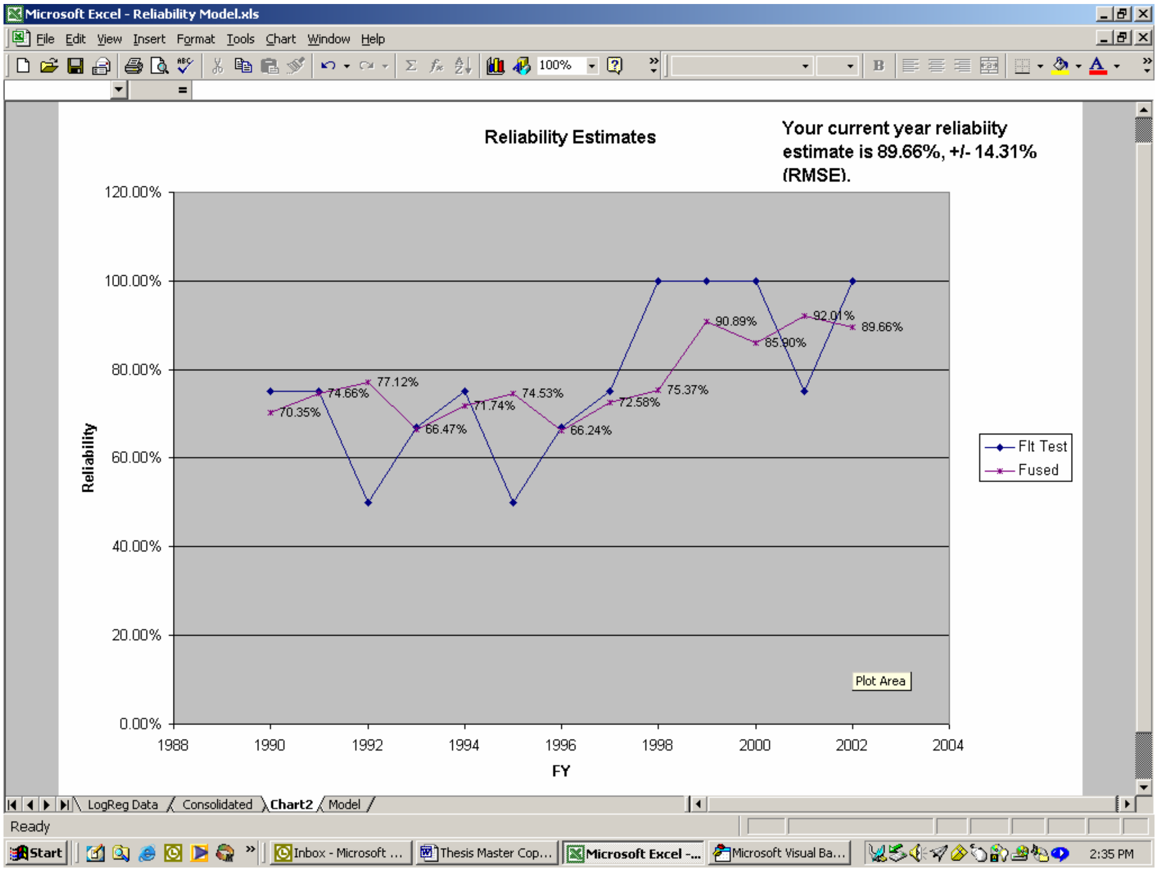


Figure 24: AARES Model Outputs – Quick Estimate

As designed, the AARES model meets all the criteria set by the user (easy to use, standalone, existing data sources, single answer reliability estimate, 24-month prediction capability).

V. Conclusions

As stated in previously, the focus of this thesis is estimating ALCM free flight reliability. Following the steps as listed in Chapter 3 should produce equally accurate results when using ACM flight test results or captive carry test results for either missile. It is merely a matter of compiling the database, selecting proper features, then applying the AARES VBA code to the data.

With regard to maintenance, the user will be required to maintain the ground and flight test database. The pass rates must be present on the "model" worksheet for AARES to capture them and calculate the estimates. Simply "paste linking" the values into the worksheet as with previous years will suffice. The model will self-adjust and capture the new values as they are added. A new year of data will not be captured, however, until flight test results have been added.

Furthermore, the VBA code has room for expansion. The current version utilizes three neural networks: logistic regression, feed forward and radial basis function. Dozens more exist, and once properly coded and validated, additional neural network subroutines could be added at the user's discretion.

One should note that AARES does not use time (FY) as an explicit model feature. In the course of development, some experimentation using FY as a feature was performed, but feature selection techniques eliminated the variable from consideration. Furthermore, the scale of the variable is different from the rest of the model features – resulting in poor estimates and large errors. As a result, FY has not been included in the model other than as a label for the x-axis. Instead, the model relies upon past ground and flight test pass rates to estimate reliability. If the user truly desires to have time included in the model, it becomes merely a matter of adding another column and making some minor code edits. AARES self-adjusts to feature size as it does to exemplars.

As a final note, the estimates produced by the AARES model are generated by statistically sound techniques, but the model suffers from the same shortcoming as previous logistic regression efforts: lack of validation data. Specifically, the cruise missile program simply does not have enough annual flight test events to provide a representative sample of the stockpile and thus generate a truly representative target vector for the model. AARES alleviates the problem by using numerous ground tests as model features for estimating free flight reliability, however until the number of shots per year increases, the model outputs cannot be validated.

Appendix A: Acronyms

AARES	ALCM/ACM Reliability Estimation System
ACC	Air Combat Command
ACM	Advanced Cruise Missile
ACR	Aircrew Reliability
AF&F	Arming, Fuzing and Firing
AGM-86	ALCM
AGM-129	ACM
AGR	Aircraft Generation Reliability
ALCM	Air Launched Cruise Missile
ASR	Aircraft Systems Reliability
Auto-Cal	Automatic Calibration
BA	Boost Adjustment
BIT	Built-In Test
BR	Boost Reliability
BRT	Battle Readiness Test
CA	Cruise Adjustment
CCPR	Captive Carry Prelaunch Reliability
CCR	Captive Carry Reliability
CCTR	Captive Carry Transit Reliability
CDF	Cumulative Distribution Function
CR	Countdown Reliability
CR1	Cruise Reliability

CR2	Carrier Reliability
CSRL	Common Strategic Rotary Launcher
DOE	Department of Energy
D/R	Decoder/Receiver
DR	Deployment Reliability
FFCR	Free Flight Cruise Reliability
FFER	Free Flight Endgame Reliability
FFN	Feed-forward Neural Network
FFR	Free Flight Reliability
FFTR	Free Flight Transition to Cruise Reliability
FGT	Functional Ground Test
GEM	Generalized Ensemble Method
GUI	Graphical User Interface
ICBM	Intercontinental Ballistic Missile
IMF	Integrated Maintenance Facility
INE	Inertial Navigation Element
JHU-APL	Johns Hopkins University Applied Physics Laboratory
JILT	Joint Integrated Lab Test
LI	Launch Interval
LLT	Loaded Launcher Test
LPT	Loaded Pylon Test
LR	Launch Reliability

LR	Logistic Regression
LWA	Launch Window Availability
MA	Missile Adjustment
MIT	Missile Interface Test
MR	Missile Reliability
MSE	Mean Squared Error
NAT	Navigation Accuracy Test
NSWC	Naval Surface Warfare Center
OAS	Offensive Avionics System
OC-ALC	Oklahoma City Air Logistics Center
PFR	Platform Reliability
PLA	Post-launch Assessment
PR	Payload Reliability
RBFN	Radial Basis Function Network
RMSE	Root Mean Squared Error
RR	Reentry Reliability
RRB	Reentry Burst Reliability
RRI	Reentry Inflight Reliability
RRS	Reentry Separation Reliability
RSR	Release System Reliability
SIOP	Single Integrated Operational Plan
SIT	System Interface Test
SLBM	Sub-launched Ballistic Missile

SLT	Stockpile Lab Test
SPACECOM	Space Command
SPO	System Program Office
SSE	Sum of Squared Errors
TLAM	Tomahawk Land Attack Missile
TO	Technical Order
VBA	Visual Basic for Applications
USSTRATCOM	United States Strategic Command
WAM	Warhead Arming Monitor
WDR	Weapon Delivery System Reliability
WSR	Weapon System Reliability
WSRT	Weapon System Readiness Test

Appendix B: Notional Flight Test Data

				Intercept	-2.8380919	
				Coeff	0.23892478	
FY	Result	Number	Relobs		Rel Est	Log Reg Results
1	1	1			0.93	0.069192042
1	1	1			0.93	0.069192042
1	1	1			0.93	0.069192042
1	1	1			0.93	0.069192042
1	1	1			0.93	0.069192042
1	1	1			0.93	0.069192042
1	1	1			0.93	0.069192042
1	1	1			0.93	0.069192042
1	1	1			0.93	0.069192042
1	1	1	1		0.93	0.069192042
2	1	1			0.91	0.086255093
2	1	1			0.91	0.086255093
2	1	1			0.91	0.086255093
2	1	1			0.91	0.086255093
2	1	1			0.91	0.086255093
2	0	1			0.91	0.086255093
2	1	1			0.91	0.086255093
2	1	1	0.88		0.91	0.086255093
3	1	1			0.89	0.107042068
3	1	1			0.89	0.107042068
3	1	1			0.89	0.107042068
3	0	1			0.89	0.107042068
3	1	1	0.8		0.89	0.107042068
4	1	1			0.87	0.132114276
4	1	1			0.87	0.132114276
4	1	1			0.87	0.132114276
4	1	1	1		0.87	0.132114276
5	1	1			0.84	0.161993723
5	1	1			0.84	0.161993723
5	0	1			0.84	0.161993723
5	1	1	0.75		0.84	0.161993723
6	1	1			0.80	0.197096161
6	1	1			0.80	0.197096161
6	1	1			0.80	0.197096161
6	0	1	0.75		0.80	0.197096161
7	1	1			0.76	0.237647885
7	1	1			0.76	0.237647885
7	0	1	0.67		0.76	0.237647885
8	0	1			0.72	0.28359598
8	1	1	0.5		0.72	0.28359598
9	1	1			0.67	0.334529581
9	1	1	1		0.67	0.334529581
10	0	1			0.61	0.389635627
10	1	1			0.61	0.389635627
10	1	1	0.67		0.61	0.389635627

Appendix C: Ground Test Data

CY	LLT A	LLT B	SIT	MIT	Level 1 A	Level 1 B	Level 3 B	INE	Prev Yr	Flt Test
90	96.03%	58.70%	88.95%	93.88%	82.66%	27.00%	33.33%	94.10%	67.00%	75.00%
91	95.63%	52.83%	96.34%	96.84%	81.87%	16.00%	33.33%	95.60%	75.00%	75.00%
92	95.32%	36.11%	98.79%	99.10%	78.63%	33.63%	25.00%	97.45%	75.00%	50.00%
93	93.98%	82.50%	93.64%	98.18%	79.57%	29.52%	33.33%	95.15%	50.00%	67.00%
94	93.13%	73.83%	96.74%	98.75%	80.43%	24.00%	46.15%	95.42%	67.00%	75.00%
95	94.44%	81.82%	94.90%	96.84%	81.22%	23.68%	64.52%	95.37%	75.00%	50.00%
96	95.04%	92.56%	84.62%	99.00%	79.07%	40.00%	84.00%	96.94%	50.00%	67.00%
97	95.00%	80.95%	100.00%	97.96%	78.05%	20.37%	45.45%	94.39%	67.00%	75.00%
98	95.09%	76.92%	93.72%	98.65%	79.58%	38.33%	N/R	93.72%	75.00%	100.00%
99	94.97%	77.38%	91.18%	97.67%	73.49%	37.78%	100.00%	93.14%	100.00%	100.00%
00	95.48%	66.67%	100.00%	99.37%	83.46%	47.37%	16.67%	96.48%	100.00%	100.00%
01	96.19%	35.48%	100.00%	99.21%	71.10%	28.95%	0.00%	90.65%	100.00%	75.00%
02	92.06%	N/R	#DIV/0!	99.46%	55.15%	34.78%	50.00%	84.13%	75.00%	100.00%

estimated data

N/R none recorded

Appendix D: SAS Factor Analysis Output

The FACTOR Procedure
Initial Factor Method: Principal Components

Prior Communality Estimates: ONE

Eigenvalues of the Correlation Matrix: Total = 6 Average = 1

	Eigenvalue	Difference	Proportion	Cumulative
1	2.57942941	0.93849633	0.4299	0.4299
2	1.64093307	0.64655285	0.2735	0.7034
3	0.99438022	0.49333963	0.1657	0.8691
4	0.50104060	0.25566604	0.0835	0.9526
5	0.24537455	0.20653240	0.0409	0.9935
6	0.03884215		0.0065	1.0000

3 factors will be retained by the NFACTOR criterion.

Factor Pattern

		Factor1	Factor2	Factor3
LLTA	LLTA	0.68941	0.48964	-0.34507
SIT	SIT	-0.26020	0.78118	0.28509
MIT	MIT	-0.56918	0.29065	0.64160
Level1A	Level1A	0.94058	0.13506	0.22390
INE	INE	0.87664	0.11866	0.42382
PrevYr	PrevYr	-0.24349	0.82106	-0.39067

Variance Explained by Each Factor

Factor1	Factor2	Factor3
2.5794294	1.6409331	0.9943802

Final Communality Estimates: Total = 5.214743

LLTA	SIT	MIT	Level1A	INE	PrevYr
0.83410138	0.75922666	0.82009747	0.95306195	0.96220513	0.88605012

The FACTOR Procedure
Rotation Method: Varimax

Orthogonal Transformation Matrix

	1	2	3
1	0.89460	-0.05157	-0.44389
2	0.24031	0.89299	0.38056
3	0.37676	-0.44712	0.81126

Rotated Factor Pattern

		Factor1	Factor2	Factor3
LLTA	LLTA	0.60440	0.55597	-0.39962
SIT	SIT	0.06236	0.58353	0.64407
MIT	MIT	-0.19761	0.00203	0.88377
Level1A	Level1A	0.95825	-0.02802	-0.18447
INE	INE	0.97243	-0.12875	-0.00015
PrevYr	PrevYr	-0.16770	0.92043	0.10361

Variance Explained by Each Factor

Factor1	Factor2	Factor3
2.3002360	1.5141744	1.4003323

Final Communality Estimates: Total = 5.214743

LLTA	SIT	MIT	Level1A	INE	PrevYr
0.83410138	0.75922666	0.82009747	0.95306195	0.96220513	0.88605012

Appendix E: MATLAB Logistic Regression Code

```
clc
clear
% input matrix
%MIT  Level 1A Prev Yr Flt Test
x=[0.9388  0.8266  0.6700  0.7500
0.9684   0.8187   0.7500   0.7500
0.9910   0.7863   0.7500   0.5000
0.9818   0.7957   0.5000   0.6700
0.9875   0.8043   0.6700   0.7500
0.9684   0.8122   0.7500   0.5000
0.9900   0.7907   0.5000   0.6700
0.9796   0.7805   0.6700   0.7500
0.9865   0.7958   0.7500   1.0000
0.9767   0.7349   1.0000   1.0000
0.9937   0.8346   1.0000   1.0000
0.9921   0.7110   1.0000   0.7500
0.9946   0.5515   0.7500   1.0000];

%number of exemplars upon which to train
tr = 13;
% number of out-years to predict
yr = 0;

% logistic regression (instantaneous)
% output training vector
z=[];
% output prediction vector
zvr=[];
% weight vector
w=[];
% weight gradient vector
dw=[];
%sets nfeat = to the number of columns
nfeat=size(x,2);

% zero out weights
for ii=1:nfeat
   w(ii)=0;
end

%adds a bias column of 1's to the left of side of matrix x
x=[ones(size(x,1),1) x];
%sets number of iterations for code to run through
iter=1000;
%sets stepsize = .001
stepsize=.001;
% used as a comparator to know when to stop increasing iterations
prevtoterr = 1;
% parameter that tells the code when to stop (when decreases in toterr become very small)
toterr = 0;
% transpose x matrix to keep with Looney convention
x=x';
```

```
% loops through with increasing number of iterations until graph stabilizes
% and converges -- when toterr changes very little
while abs(prevtoterr-toterr) > .001
   prevtoterr=toterr;
   for i=1:iter
      toterr=0.0; % zeros out total error
      for ii=1:nfeat
         dw(ii)=0; % zeros out dw, differential of the error
      end
      for j=1+yr:tr+yr %j runs from 1 down the number of rows
         z(j)=0.0; % initializes Yhatj at zero (estimated value)
         for k=1:nfeat % runs from 1 across the number of columns
            z(j)=z(j)+w(k)*x(k,j-yr); % sets Yhat = previous_Yhat + weight*current x_value,  x_value
changes across the columns
         end % does this across the columns
         z(j)=(1./(1.+exp(-1.0*z(j)))); % call the sigmoid file and do it's thing with the z_matrix element
         for l=1:nfeat %l runs across the columns
            dw(l)=(z(j)-x(nfeat+1,j))*z(j)*(1.-z(j))*x(l,j-yr); % cumes all the differentials of the errors
            w(l)=w(l)-stepsize*dw(l); % steps in the direction opposite the error, converges toward the "true"
weights/b_knot and b_one
         end
         toterr=toterr+(z(j)-x(nfeat+1,j))^2; % cumes total error per iteration
      end
   end
   toterr;
   % sets number of iterations to run through next depending upon changes
   % in toterr
   if abs(prevtoterr-toterr)>.01
      iter = iter+1000;
   else
      iter = iter+500;
   end
end

% plot the regression and the flight test results
axis([0 16 .5 1.1])
xlabel('Calendar Year')
ylabel('Reliability %')
hold on
plot(x(nfeat+1,:),'m --' )
plot(z,'b')

% logreg prediction code
if tr < size(x,2)
   for n=tr+1+yr:size(x,2)+yr
      zvr(n)=0.0;
      for k=1:nfeat
         zvr(n) = zvr(n) + w(k)*x(nfeat+1,n-yr);
      end % end k loop
      zvr(n)=1/(1+exp(-(zvr(n))));
   end % end n loop
   plot(zvr,'b :')
end % end year check
```

Appendix F: Matlab Reliability Model Code

```
clc
clear
% input matrix
%MIT  Level 1A Prev Yr Flt Test
x=[0.9388  0.8266  0.6700  0.7500
0.9684   0.8187   0.7500   0.7500
0.9910   0.7863   0.7500   0.5000
0.9818   0.7957   0.5000   0.6700
0.9875   0.8043   0.6700   0.7500
0.9684   0.8122   0.7500   0.5000
0.9900   0.7907   0.5000   0.6700
0.9796   0.7805   0.6700   0.7500
0.9865   0.7958   0.7500   1.0000
0.9767   0.7349   1.0000   1.0000
0.9937   0.8346   1.0000   1.0000
0.9921   0.7110   1.0000   0.7500
0.9946   0.5515   0.7500   1.0000];

%number of exemplars upon which to train
tr = 13;
% number of out-years to predict
yr = 0;

% logistic regression (instantaneous)
% output training vector
z=[];
% output prediction vector
zvr=[];
% weight vector
w=[];
% weight gradient vector
dw=[];
%sets nfeat = to the number of columns
nfeat=size(x,2);

% zero out weights
for ii=1:nfeat
   w(ii)=0;
end

%adds a bias column of 1's to the left of side of matrix x
x=[ones(size(x,1),1) x];
%sets number of iterations for code to run through
iter=1000;
%sets stepsize = .001
stepsize=.001;
% used as a comparator to know when to stop increasing iterations
prevtoterr = 1;
% parameter that tells the code when to stop (when decreases in toterr become very small)
toterr = 0;
% transpose x matrix to keep with Looney convention
x=x';
```

```
% loops through with increasing number of iterations until graph stabilizes
% and converges -- when toterr changes very little
while abs(prevtoterr-toterr) > .001
   prevtoterr=toterr;
   for i=1:iter
      toterr=0.0; % zeros out total error
      for ii=1:nfeat
         dw(ii)=0; % zeros out dw, differential of the error
      end
      for j=1+yr:tr+yr %j runs from 1 down the number of rows
         z(j)=0.0; % initializes Yhatj at zero (estimated value)
         for k=1:nfeat % runs from 1 across the number of columns
            z(j)=z(j)+w(k)*x(k,j-yr); % sets Yhat = previous_Yhat + weight*current x_value,  x_value
changes across the columns
         end % does this across the columns
         z(j)=(1./(1.+exp(-1.0*z(j)))); % call the sigmoid file and do it's thing with the z_matrix element
         for l=1:nfeat %l runs across the columns
            dw(l)=(z(j)-x(nfeat+1,j))*z(j)*(1.-z(j))*x(l,j-yr); % cumes all the differentials of the errors
            w(l)=w(l)-stepsize*dw(l); % steps in the direction opposite the error, converges toward the "true"
weights/b_knot and b_one
         end
         toterr=toterr+(z(j)-x(nfeat+1,j))^2; % cumes total error per iteration
      end
   end
   toterr;
   % sets number of iterations to run through next depending upon changes
   % in toterr
   if abs(prevtoterr-toterr)>.01
      iter = iter+1000;
   else
      iter = iter+500;
   end
end

% plot the regression and the flight test results
axis([0 16 .5 1.1])
xlabel('Calendar Year')
ylabel('Reliability %')
hold on
plot(x(nfeat+1,:),'m --' )
plot(z,'b')

% logreg prediction code
if tr < size(x,2)
   for n=tr+1+yr:size(x,2)+yr
      zvr(n)=0.0;
      for k=1:nfeat
         zvr(n) = zvr(n) + w(k)*x(k,n-yr);
      end % end k loop
      zvr(n)=1/(1+exp(-(zvr(n))));
   end % end n loop
   plot(zvr,'b :')
end % end year check
```

```
% reset input matrix, strip off bottom row of flight test results
flttest=x(nfeat+1,:);
x(nfeat+1,:)=[];
nfeat=size(x,1);
ncols=size(x,2);

% average of output runs
zzz=[];
% average of prediction runs
zvv=[];
% lower layer output matrix
zz=[];
% verification output matrix
zv=[];

% loop through a few times to get an average of the output values
for count=1:10
% set stepsize
nu=.7;
% upper layer output row vector
y=[];
% middle layer weights matrix
w=[];
% upper layer weights matrix
u=[];
% middle layer summations weight gradients
dw=[];
% matrix of targets -- flight test results
t=flttest;
% number of midddle layer neurodes
M=5;
% number of output layer neurodes
J=size(t,1);
% number of inputs (features)
N=size(x,1);
% number of exemplars to run through
Q=tr;
% set number of iterations
iter=1500;
% setting initial weights
for m=1:M
   for n=1:N
      w(n,m)=unifrnd(-0.2, 0.2);
   end
   for j=1:J
      u(m,j)=unifrnd(-0.2, 0.2);
   end
end % end m loop, setting initial weights

prevtoterr=1;
toterr=0;
while abs(prevtoterr-toterr)>.001
   prevtoterr=toterr;
   % initialize iterations
```

```
for i=1:iter
    toterr=0.0;
    % run down the rows of exemplars
    for q=1+yr:Q+yr
        % zero out outputs
        for j=1:J
            zz(j,q,count)=0;
        end % end j loop, zero out outputs
        for n=1:N
            for m=1:M
                dw(n,m)=0;
            end % end m loop
        end % end n loop, zero out summation portion of middle layer weight gradients
        for m=1:M
            %calculate middle layer outputs
            y(m)=0.0;
            for n=1:N
                y(m) = y(m) + w(n,m)*x(n,q-yr);
            end % end n loop, sum across middle layer prior to squashing
            % calculate sigmoid of middle layer outputs -- squash 'em
            y(m)=1/(1+exp(-(y(m))));
        end % end m loop, middle layer outputs
        % calculate outputs
        for j=1:J
            for m=1:M
                zz(j,q,count) = zz(j,q,count) + u(m,j)*y(m);
            end % end m loop, sum across the outputs prior to squashing
            % calculate sigmoid of outputs -- squash 'em
            zz(j,q,count)=1/(1+exp(-(zz(j,q,count))));
        end % end j loop, new output loop
        % adjust weights
        for m=1:M
            % calculate new upper layer weights
            for j=1:J
                u(m,j) = u(m,j) + nu*((t(j,q) - zz(j,q,count))*zz(j,q,count)*(1 - zz(j,q,count))*y(m));
            end % end j loop, uppper layer weight update
            % calculate summation portion of gradient for middle layer
            for n=1:N
                for j=1:J
                    dw(n,m) = dw(n,m) + (t(j,q) - zz(j,q,count))*(zz(j,q,count)*(1 - zz(j,q,count)))*u(m,j);
                end % end j loop cume portion of middle layer weight gradient
                % calculate middle layer weights
                w(n,m) = w(n,m) + nu*dw(n,m)*(y(m)*(1 - y(m))*x(n,q-yr));
            end % end n loop middle layer weight adjustments
        end % end m loop, weight adjustments
        % calculate SSE
        for j=1:J
            toterr=toterr+(zz(j,q,count)-t(j,q))^2;
        end % end toterr cume loop
    end % end q loop number of exemplars on which to train
end %end iteration loop
if abs(prevtoterr-toterr)>.005
    iter = iter+100;
else
```

```
        iter = iter+50;
    end % end iteration step-check loop
end % end .001 while loop

% verify weights developed during training -- attempt to predict current year or out-year flight
% test results within data set
if tr < size(x,2)
    for q=Q+1+yr:size(x,2)+yr
        for j=1:J
            zv(j,q,count)=0.0;
        end
        for m=1:M
            y(m)=0.0;
            for n=1:N
                y(m) = y(m) + w(n,m)*x(n,q-yr);
            end
            y(m)=1/(1+exp(-(y(m))));
            for j=1:J
                zv(j,q,count) = zv(j,q,count) + u(m,j)*y(m);
            end
        end
        for j=1:J
            zv(j,q,count)=1/(1+exp(-(zv(j,q,count))));
        end
    end % end verification loop
end % end prediction test
end  % end count loop

% calculate average of the runs and display
zzz = mean(zz,3);
plot(zzz,'r')
hold on

if tr < size(x,2)
    zvv = mean(zv,3);
    plot(zvv,'r :')
end

% RBFN code
% set output vectors
zrb=[];
zrbt=[];
zzrb=[];
zzrbt=[];

% loop through a few times and get an averaqe
for count=1:10
% set stepsize
nu=1.0;
% upper layer output row vector
y=[];
% middle layer neurode centers
v=[];
% upper layer weights matrix
```

```
u=[];
% middle layer summations weight gradients
dw=[];
% summation matrix for distance calculation
addup=[];
% number of inputs (features)
N=size(x,1);
% number of output layer neurodes
J=size(t,1);
% number of exemplars to run through
Q=tr;
% number of midddle layer neurodes
M=Q;
% set number of iterations
iter=100;
%compute single spread parameter
sigma=1/((2*M)^(1/N));
%sigma = 0.9;
% setting initial weights, neurode centers, and neurode spread parameters
for m=1:M
    for j=1:J
        u(m,j)=unifrnd(-0.5, 0.5);
    end % end J loop
end % end m loop, setting initial weights
v=x;

% used as a comparator to know when to stop increasing iterations
prevtoterr = 1.0;
% parameter that tells the code when to stop (when decreases in toterr become very small)
toterr = 0;
% calculate difference vector
for q=1+yr:Q+yr
    for m=1:M
        distnc=0;
        for n=1:N
            distnc = distnc + (x(n,q-yr)-v(n,m))^2;
        end
        addup(m,q) = distnc;
    end
end

% compute y(m,q)
for q=1+yr:Q+yr
    for m=1:M
        if q == m
            y(m,q)=1;
        else
            y(m,q)=exp(-(addup(m,q))/(2*(sigma^2)));
        end % end if test
    end % end m loop
end % end q loop

% train the network
while abs(prevtoterr-toterr)>.00001
```

```
    prevtoterr=toterr;
    % initialize iterations
    for i=1:iter
        toterr=0;
        for m=1:M
            for j=1:J
                du(m,j)=0;
                for q=1+yr:Q+yr
                    dw(j,q)=0;
                end % end q loop
            end % end j loop
        end % end m loop
        % compute new outputs
        for q=1+yr:Q+yr
            for j=1:J
                for m=1:M
                    dw(j,q) = dw(j,q) + (u(m,j)*y(m,q));
                end % end m loop
            end % end j loop
        end % end new output loops
        for q=1+yr:Q+yr
            for j=1:J
                zrb(j,q,count) = dw(j,q)/M;
            end % end j loop
        end % end q loop
        % SSE calculation
        for q=1+yr:Q+yr
            for j=1:J
                toterr = toterr + ((t(j,q) - zrb(j,q,count))^2);
            end % end j loop
        end % end error calculation
        if toterr<prevtoterr
            nu=nu*1.04;
        else
            nu=nu*0.92;
        end % end new stepsize check
        % adjust weights
        for m=1:M
            for j=1:J
                for q=1+yr:Q+yr
                    du(m,j) = du(m,j) + ((t(j,q) - zrb(j,q,count))*y(m,q));
                end % end q loop
            end % end j loop
        end % end m loop
        for m=1:M
            for j=1:J
                u(m,j) = u(m,j) + ((2*nu)/M)*du(m,j);
            end % end j loop
        end % end m loop
    end % end iteration loop
end % end tolerance loop

% test middle layer outputs
ytest=[];
```

```
% verify test data
if tr < size(x,2)
   for q=Q+1+yr:size(x,2)+yr
      % zero out output matrix
      for j=1:J
         zrbt(j,q,count)=0;
      end % end j loop
      % calculate distances from center
      for m=1:M
         distnc=0;
         for n=1:N
            distnc = distnc + (x(n,q-yr)-v(n,m))^2;
         end % end n loop
         addup(m,q) = distnc;
      end % end m loop
   end % end q loop
   % compute ytest(m,q)
   for q=Q+1+yr:size(x,2)+yr
      for m=1:M
         ytest(m,q)=exp(-(addup(m,q))/(2*(sigma^2)));
      end % end m loop
   end % end q loop
   % compute outputs
   for q=Q+1+yr:size(x,2)+yr
      for j=1:J
         adduys=0;
         for m=1:M
            adduys=adduys+u(m,j)*ytest(m,q);
         end % end m loop
         zrbt(j,q,count)=adduys/M;
      end % end j loop
   end % end q loop
end % end prediction test
end % end count loop

zzrb=mean(zrb,3);
plot(zzrb,'k');

if tr < size(x,2)
   zzrbt=mean(zrbt,3);
   plot(zzrbt,'k :')
end

% fuse the outputs from the three nets
% set up matrices and strip off any zero rows
Z=[z' zzz' zzrb']
for i=1:yr
   Z(1,:)=[];
end

corrZ=corrcoef(Z)
denomalpha=0;
alpha=[];
```

```
Zgem=[];
ZZgem=[];

for i=1:size(corrZ,2)
   for j=1:size(corrZ,1)
      denomalpha=denomalpha+(1/corrZ(i,j));
   end
end

for i=1:size(corrZ,2)
   numalpha=0;
   for j=1:size(corrZ,1)
      numalpha=numalpha+(1/corrZ(i,j));
   end
   alpha(i)=numalpha/denomalpha;
end

for q=1:size(Z,1)
   Zgem(q)=0;
   for i=1:size(Z,2)
      Zgem(q)=Zgem(q)+alpha(i)*Z(q,i);
   end
end

% add offset back into fused results vector
if yr > 0
   for i=1:yr
      Zgem=[zeros(size(Zgem,1),1),Zgem];
   end
end

Zgem
plot(Zgem, 'g')

% calculate fused prediction
if tr < size(x,2)
   ZZ=[zvr' zvv' zzrbt']
   for i=1:tr+yr
      ZZ(1,:)=[];
   end

   for q=1:size(ZZ,1)
      ZZgem(q)=0;
      for i=1:size(ZZ,2)
         ZZgem(q)=ZZgem(q)+alpha(i)*ZZ(q,i);
      end
   end

   % add offset back into fused results vector
   for i=1:tr+yr
      ZZgem=[zeros(size(ZZgem,1),1),ZZgem];
   end
   ZZgem
   plot(ZZgem, 'g :')
```

```
end % end if check

% calculate RMSEs of the two methods
sumrmselr=0;
sumrmseff=0;
sumrmserb=0;
sumrmseZ=0;
rmselr=0;
rmseff=0;
rmserb=0;
rmseZ=0;
% SSE of training points
for q=1+yr:tr+yr
   sumrmselr = sumrmselr + (z(q)-t(1,q-yr))^2;
   sumrmseff = sumrmseff + (zzz(q)-t(1,q-yr))^2;
   sumrmserb = sumrmserb + (zzrb(q)-t(1,q-yr))^2;
   sumrmseZ = sumrmseZ + (Zgem(q)-t(1,q-yr))^2;
end
% SSE of prediction points
if tr < size(x,2)
   for q=tr+1+yr:size(x,2)+yr
      sumrmselr = sumrmselr + (zvr(q)-t(1,q-yr))^2;
      sumrmseff = sumrmseff + (zvv(q)-t(1,q-yr))^2;
      sumrmserb = sumrmserb + (zzrbt(q)-t(1,q-yr))^2;
      sumrmseZ = sumrmseZ + (ZZgem(q)-t(1,q-yr))^2;
   end
end
rmselr = sqrt(sumrmselr/size(t,2))
rmseff = sqrt(sumrmseff/size(t,2))
rmserb = sqrt(sumrmserb/size(t,2))
rmseZ = sqrt(sumrmseZ/size(t,2))

% add a legend to the graph
if tr < size(x,2)
   legend('Actual', 'LogReg Training', 'LogReg Prediction', 'FFN Training', 'FFN Prediction', 'RBFN
Training', 'RBFN Prediction', 'Fused Training', 'Fused Prediction', 2)
else
   legend('Actual', 'LogReg Results', 'FFN Results', 'RBFN Results','Fused Results', 2);
end
```

Appendix G: Matlab Validation Code

Logistic Regression Validation

```
clc
clear
% input matrix
% generate training data points and populate into matrix
x=unifrnd(0,1,30,1);

% gin up a simple relationship between x and y
for i=1:size(x,1)
   t(i)=x(i);
end
t=t';
x=[x t];

% number of out-years to predict
yr = 0;
tr=20;

% logistic regression (instantaneous)
% output training vector
z=[];
% output prediction vector
zvr=[];
% weight vector
w=[];
% weight gradient vector
dw=[];
%sets nfeat = to the number of columns
nfeat=size(x,2);
% zero out weights
for ii=1:nfeat
   w(ii)=0;
end

%adds a bias column of 1's to the left of side of matrix x
x=[ones(size(x,1),1) x];
%sets number of iterations for code to run through
iter=1000;
%sets stepsize = .001
stepsize=.001;
% used as a comparator to know when to stop increasing iterations
prevtoterr = 1;
% parameter that tells the code when to stop (when decreases in toterr become very small)
toterr = 0;
% transpose x matrix to keep with Looney convention
x=x';

% loops through with increasing number of iterations until graph stabilizes
% and converges -- when toterr changes very little
while abs(prevtoterr-toterr) > .00000001
   prevtoterr=toterr;
   for i=1:iter
```

```
        toterr=0.0; % zeros out total error
        for ii=1:nfeat
            dw(ii)=0; % zeros out dw, differential of the error
        end
        for j=1+yr:tr+yr %j runs from 1 down the number of rows
            z(j)=0.0; % initializes Yhatj at zero (estimated value)
            for k=1:nfeat % runs from 1 across the number of columns
                z(j)=z(j)+w(k)*x(k,j-yr); % sets Yhat = previous_Yhat + weight*current x_value,  x_value
changes across the columns
            end % does this across the columns
            z(j)=(1./(1.+exp(-1.0*z(j)))); % call the sigmoid file and do it's thing with the z_matrix element
            for l=1:nfeat %l runs across the columns
                dw(l)=(z(j)-x(nfeat+1,j))*z(j)*(1.-z(j))*x(l,j-yr); % cumes all the differentials of the errors
                w(l)=w(l)-stepsize*dw(l); % steps in the direction opposite the error, converges toward the "true"
weights/b_knot and b_one
            end
            toterr=toterr+(z(j)-x(nfeat+1,j))^2; % cumes total error per iteration
        end
    end
    toterr;
    % sets number of iterations to run through next depending upon changes
    % in toterr
    if abs(prevtoterr-toterr)>.01
        iter = iter+1000;
    else
        iter = iter+500;
    end
end


w
toterr

% logreg prediction code
if tr < size(x,2)
    for n=tr+1+yr:size(x,2)+yr
        zvr(n)=0.0;
        for k=1:nfeat
            zvr(n) = zvr(n) + w(k)*x(k,n-yr);
        end % end k loop
        zvr(n)=1/(1+exp(-(zvr(n))));
    end % end n loop
end % end year check

% plot the regression and the flight test results
axis([0 1 0 1])
xlabel('x')
ylabel('y')
title('Logistic Regression')
hold on

plot(x(2,1),t(1,1),'b *')
plot(x(2,1),z(1),'k +')
plot(x(2,tr+1),zvr(tr+1),'r o')
```

```
legend('Actual','LogReg Training','LogReg Prediction', 2)
for i=1:size(x,2)
   plot(x(2,i),t(i,1),'b *')
end
for i=1:tr
   plot(x(2,i),z(i),'k +')
end
for i=tr+1:size(x,2)
   plot(x(2,i),zvr(i),'r o')
end
```

<u>Feed Forward Validation</u>

```
clc
clear
% input matrix
% generate training data points and populate into matrix
x=unifrnd(-1,1,30,2);

% plot the points
for i=1:size(x,1)
   plot(x(i,2),x(i,1),'*')
   hold on
end

% create target vector based upon discriminator lines
% let (0,1) denote quad 2-4 membership, let (1,0) denote quad 1-3 membership
T=[];

for i=1:size(x,1)
   if (x(i,1) > 0 & x(i,2) > 0) | (x(i,1) < 0 & x(i,2) < 0)
      T(i,1)=1;
   else
      T(i,2)=1;
   end
end

% adds a bias column of 1's to the left of side of matrix x
x=[ones(size(x,1),1) x];

% number of exemplars upon which to train
tr = 20;
% number of years to predict ahead (0=current year estimates)
yr = 0;
% transpose input matrix,
x=x';
% average of output runs
zzz=[];
% average of prediction runs
zvv=[];
% lower layer output matrix
zz=[];
% verification output matrix
zv=[];
```

```
% loop through a few times to get an average of the output values
for count=1:1
% set stepsize
nu=.01;
% upper layer output row vector
y=[];
% middle layer weights matrix
w=[];
% upper layer weights matrix
u=[];
% middle layer summations weight gradients
dw=[];
% matrix of targets -- flight test results
t=T';
% number of midddle layer neurodes
M=2*size(t,1);
% number of output layer neurodes
J=size(t,1);
% number of inputs (features)
N=size(x,1);
% number of exemplars to run through
Q=tr;
% set number of iterations
iter=1000;
% setting initial weights
for m=1:M
   for n=1:N
      w(n,m)=unifrnd(-0.2, 0.2);
   end
   for j=1:J
      u(m,j)=unifrnd(-0.2, 0.2);
   end
end % end m loop, setting initial weights

prevtoterr=1;
toterr=0;
while abs(prevtoterr-toterr)>.01
   prevtoterr=toterr;
   % initialize iterations
   for i=1:iter
      toterr=0.0;
      % run down the rows of exemplars
      for q=1+yr:Q
         % zero out outputs
         for j=1:J
            zz(j,q,count)=0;
         end % end j loop, zero out outputs
         for n=1:N
            for m=1:M
               dw(n,m)=0;
            end % end m loop
         end % end n loop, zero out summation portion of middle layer weight gradients
         for m=1:M
            %calculate middle layer outputs
```

```
            y(m)=0.0;
            for n=1:N
               y(m) = y(m) + w(n,m)*x(n,q-yr);
            end % end n loop, sum across middle layer prior to squashing
            % calculate sigmoid of middle layer outputs -- squash 'em
            y(m)=1/(1+exp(-(y(m))));
         end % end m loop, middle layer outputs
         % calculate outputs
         for j=1:J
            for m=1:M
               zz(j,q,count) = zz(j,q,count) + u(m,j)*y(m);
            end % end m loop, sum across the outputs prior to squashing
            % calculate sigmoid of outputs -- squash 'em
            zz(j,q,count)=1/(1+exp(-(zz(j,q,count))));
         end % end j loop, new output loop
         % adjust weights
         for m=1:M
            % calculate new upper layer weights
            for j=1:J
               u(m,j) = u(m,j) + nu*((t(j,q) - zz(j,q,count))*zz(j,q,count)*(1 - zz(j,q,count))*y(m));
            end % end j loop, uppper layer weight update
            % calculate summation portion of gradient for middle layer
            for n=1:N
               for j=1:J
                  dw(n,m) = dw(n,m) + (t(j,q) - zz(j,q,count))*(zz(j,q,count)*(1 - zz(j,q,count)))*u(m,j);
               end % end j loop cume portion of middle layer weight gradient
               % calculate middle layer weights
               w(n,m) = w(n,m) + nu*dw(n,m)*(y(m)*(1 - y(m))*x(n,q-yr));
            end % end n loop middle layer weight adjustments
         end % end m loop, weight adjustments
         % calculate SSE
         for j=1:J
            toterr=toterr+(zz(j,q,count)-t(j,q))^2;
         end % end toterr cume loop
      end % end q loop number of exemplars on which to train
   end %end iteration loop
   if abs(prevtoterr-toterr)>.005
      iter = iter+1000;
   else
      iter = iter+500;
   end % end iteration step-check loop
end % end .01 tolerance while loop

% verify weights developed during training -- attempt to predict current year or out-year flight
% test results within data set
if tr < size(x,2)
   for q=Q+1:size(x,2)+yr
      for j=1:J
         zv(j,q,count)=0.0;
      end
      for m=1:M
         y(m)=0.0;
         for n=1:N
            y(m) = y(m) + w(n,m)*x(n,q-yr);
```

```
        end
        y(m)=1/(1+exp(-(y(m))));
        for j=1:J
          zv(j,q,count) = zv(j,q,count) + u(m,j)*y(m);
        end
      end
      for j=1:J
        zv(j,q,count)=1/(1+exp(-(zv(j,q,count))));
      end
    end % end verification loop
  end % end prediction test
end  % end count loop

% calculate average of the runs and display
toterr;
zzz = mean(zz,3);

% calculate training confusion matrix
% recall (0,1) denotes quad 2-4 membership, (1,0) denotes quad 1-3 membership
% let 1-3 membership be 'Positive', and 2-4 membership be 'Negative'
TPtr = 0;
FPtr = 0;
TNtr = 0;
FNtr = 0;
TPver = 0;
FPver = 0;
TNver = 0;
FNver = 0;

for q=1:Q
  if zzz(1,q,count)>zzz(2,q,count)
    if t(1,q) == 1
      TPtr = TPtr + 1;
    else
      FPtr = FPtr + 1;
    end
  else
    if t(2,q) == 1
      TNtr = TNtr + 1;
    else
      FNtr = FNtr + 1;
    end
  end
end

for q=Q+1:size(x,2)
  if zv(1,q,count)>zv(2,q,count)
    if t(1,q) == 1
      TPver = TPver + 1;
    else
      FPver = FPver + 1;
    end
  else
    if t(2,q) == 1
```

```
      TNver = TNver + 1;
    else
      FNver = FNver + 1;
    end
  end
end

postr = [TPtr FPtr];
negtr = [FNtr TNtr];

disp('  FF Training Results')
disp('     Pos    Neg')
disp(postr)
disp(negtr)

posver = [TPver FPver];
negver = [FNver TNver];

disp('  FF Test Results')
disp('     Pos    Neg')
disp(posver)
disp(negver)
```

Radial Basis Function Validation Code

```
%clc
clear
% input matrix
% generate training data points and populate into matrix
x=unifrnd(-1,1,30,2);

% plot the points
for i=1:size(x,1)
  plot(x(i,2),x(i,1),'*')
  hold on
end

% create target vector based upon discriminator lines
% let (0,1) denote quad 2-4 membership, let (1,0) denote quad 1-3 membership
t=[];

for i=1:size(x,1)
  if (x(i,1) > 0 & x(i,2) > 0) | (x(i,1) < 0 & x(i,2) < 0)
    t(i,1)=1;
  else
    t(i,2)=1;
  end
end

% adds a bias column of 1's to the left of side of matrix x
x=[ones(size(x,1),1) x];

% number of exemplars upon which to train
tr = 20;
```

```
% number of years ahead to predict
yr=0;
% transpose input and target matrices,
x=x';
t=t';
% set stepsize
nu=1.0;
% upper layer output row vector
y=[];
% middle layer neurode centers
v=[];
% upper layer weights matrix
u=[];
% middle layer summations weight gradients
dw=[];
% summation matrix for distance calculation
addup=[];
% number of inputs (features)
N=size(x,1);
% number of output layer neurodes
J=size(t,1);
% number of exemplars to run through
Q=tr;
% number of midddle layer neurodes
M=Q;
% set number of iterations
iter=100;
%compute single spread parameter
sigma=1/((2*M)^(1/N));
%sigma = 0.065;
% setting initial weights, neurode centers, and neurode spread parameters
for m=1:M
   for j=1:J
      u(m,j)=unifrnd(-0.5, 0.5);
   end % end J loop
end % end m loop, setting initial weights
v=x;

% used as a comparator to know when to stop increasing iterations
prevtoterr = 1.0;
% parameter that tells the code when to stop (when decreases in toterr become very small)
toterr = 0;
% calculate difference vector
for q=1:Q
   for m=1:M
      distnc=0;
      for n=1:N
         distnc = distnc + (x(n,q)-v(n,m))^2;
      end
      addup(m,q) = distnc;
   end
end

% compute y(m,q)
```

```
for q=1:Q
   for m=1:M
      if q == m
         y(m,q)=1;
      else
         y(m,q)=exp(-(addup(m,q))/(2*(sigma^2)));
      end % end if test
   end % end m loop
end % end q loop

while abs(prevtoterr-toterr)>.000001
   prevtoterr=toterr;
   % initialize iterations
   for i=1:iter
      toterr=0;
      for m=1:M
         for j=1:J
            du(m,j)=0;
            for q=1:Q
               dw(j,q)=0;
            end % end q loop
         end % end j loop
      end % end m loop
      % compute new outputs
      for q=1:Q
         for j=1:J
            for m=1:M
               dw(j,q) = dw(j,q) + (u(m,j)*y(m,q));
            end % end m loop
         end % end j loop
      end % end new output loops
      for q=1:Q
         for j=1:J
            z(j,q) = dw(j,q)/M;
         end % end j loop
      end % end q loop
      for q=1:Q
         for j=1:J
            toterr = toterr + ((t(j,q)-z(j,q))^2);
         end % end j loop
      end % end error calculation
      if toterr<prevtoterr
         nu=nu*1.04;
      else
         nu=nu*0.92;
      end % end new stepsize check
      % adjust weights
      for m=1:M
         for j=1:J
            for q=1:Q
               du(m,j)=du(m,j)+((t(j,q)-z(j,q))*y(m,q));
            end % end q loop
         end % end j loop
      end % end m loop
```

```
        for m=1:M
            for j=1:J
                u(m,j) = u(m,j)+((2*nu)/M)*du(m,j);
            end % end j loop
        end % end m loop
    end % end iteration loop
end % end tolerance loop

% test output matrix
zvrb=[];
% test middle layer outputs
ytest=[];

% verify test data
if tr < size(x,2)
    for q=Q+1:size(x,2)+yr
        % zero out output matrix
        for j=1:J
            zvrb(j,q)=0;
        end % end j loop
        % calculate distances from center
        for m=1:M
            distnc=0;
            for n=1:N
                distnc = distnc + (x(n,q-yr)-v(n,m))^2;
            end % end n loop
            addup(m,q) = distnc;
        end % end m loop
    end % end q loop
    % compute ytest(m,q)
    for q=Q+1:size(x,2)+yr
        for m=1:M
            ytest(m,q)=exp(-(addup(m,q))/(2*(sigma^2)));
        end % end m loop
    end % end q loop
    % compute outputs
    for q=Q+1:size(x,2)+yr
        for j=1:J
            adduys=0;
            for m=1:M
                adduys=adduys+u(m,j)*ytest(m,q);
            end % end m loop
            zvrb(j,q)=adduys/M;
        end % end j loop
    end % end q loop
end % end test code

% calculate training confusion matrix
% recall (0,1) denotes quad 2-4 membership, (1,0) denotes quad 1-3 membership
% let 1-3 membership be 'Positive', and 2-4 membership be 'Negative'
TPtr = 0;
FPtr = 0;
TNtr = 0;
FNtr = 0;
```

```
TPver = 0;
FPver = 0;
TNver = 0;
FNver = 0;

for q=1:Q
   if z(1,q)>z(2,q)
      if t(1,q) == 1
         TPtr = TPtr + 1;
      else
         FPtr = FPtr + 1;
      end
   else
      if t(2,q) == 1
         TNtr = TNtr + 1;
      else
         FNtr = FNtr + 1;
      end
   end
end

for q=Q+1:size(x,2)
   if zvrb(1,q)>zvrb(2,q)
      if t(1,q) == 1
         TPver = TPver + 1;
      else
         FPver = FPver + 1;
      end
   else
      if t(2,q) == 1
         TNver = TNver + 1;
      else
         FNver = FNver + 1;
      end
   end
end

postr = [TPtr FPtr];
negtr = [FNtr TNtr];

disp('  RBF Training Results')
disp('     Pos    Neg')
disp(postr)
disp(negtr)

posver = [TPver FPver];
negver = [FNver TNver];

disp('  RBF Test Results')
disp('     Pos    Neg')
disp(posver)
disp(negver)
```

Appendix H: VBA Reliability Model (AARES) Code

Custom GUI

```
Private Sub Cancel_Click()
   Unload Me
   End
End Sub
Private Sub Run_Click()
Dim tr As Integer, yr As Integer, stepsizeLR As Double, nuFF As Double, nuRB As Double, _
MFF As Integer, agg As Integer, tag As Integer
' Capture the value of the years to train listbox
With TrYr
   If .ListIndex <> -1 Then
      tr = TrYr.Value
   Else
      MsgBox "Select the number of years to train the network."
      Exit Sub
   End If
End With

'Capture value of out-year prediction listbox
With OutYear
   If .ListIndex <> -1 Then
      yr = OutYear.Value
   Else
      MsgBox "Select the number of out-years to predict."
      .SetFocus
      Exit Sub
   End If
End With

'Capture value of number of runs over which to average FFN and RBFN
With Average
   If .Value = "" Or Not IsNumeric(.Value) Or .Value <= 0 Then
      MsgBox "Enter a number of runs over which to average results."
      .SetFocus
      Exit Sub
   Else
   agg = Average.Value
   End If
End With

'Check to ensure at least one network selected
If LR.Value = False And FFN.Value = False And RBFN.Value = False Then
   MsgBox "You must select at least one network."
   Exit Sub
End If

'Capture which networks to run and associated parameters
With LR
   If .Value = True Then
      With TextBox1
         If .Value = "" Or Not IsNumeric(.Value) Or .Value <= 0 Or .Value > 1 Then
            MsgBox "Enter a LR stepsize between 0.0 and 1.0."
```

```
            .SetFocus
            Exit Sub
        Else
        stepsizeLR = TextBox1
        End If
    End With
  End If
End With

With FFN
  If .Value = True Then
    With TextBox2
      If .Value = "" Or Not IsNumeric(.Value) Or .Value <= 0 Or .Value > 1 Then
        MsgBox "Enter a FFN stepsize between 0.0 and 1.0."
        .SetFocus
        Exit Sub
      End If
    End With
    With TextBox4
      If .Value = "" Or Not IsNumeric(.Value) Or .Value <= 0 Then
        MsgBox "Enter the number of middle layer neurodes."
        .SetFocus
        Exit Sub
      Else
      nuFF = TextBox2
      MFF = TextBox4
      End If
    End With
  End If
End With

With RBFN
  If .Value = True Then
    With TextBox3
      If .Value = "" Or Not IsNumeric(.Value) Or .Value <= 0 Or .Value > 1 Then
        MsgBox "Enter a RBFN spread between 0.0 and 1.0."
        .SetFocus
        Exit Sub
      Else
      nuRB = TextBox3
      End If
    End With
  End If
End With

tag = 0

Unload Me

' kick back over to the main program, transfer the arguments
Call Sheet2.Main(tr, yr, stepsizeLR, nuFF, nuRB, MFF, agg, tag)
End Sub

Private Sub TrYr_DropButtonClick()
```

```
End Sub

Private Sub UserForm_Initialize()

'Populate the TrYr listbox
If TrYr.ListIndex = -1 Then
    For i = 10 To 13
        TrYr.AddItem (i)
    Next i
End If

'Populate the out-year prediction listbox
If OutYear.ListIndex = -1 Then
    For i = 0 To 2
        OutYear.AddItem (i)
    Next i
End If
End Sub
```

Quick Estimate GUI

```
Private Sub Cancel_Click()

Unload Me
End

End Sub
Private Sub Run_Click()
Dim tr As Integer, yr As Integer, stepsizeLR As Double, nuFF As Double, nuRB As Double, _
MFF As Integer, agg As Integer, tag As Integer

tag = 1

With OutYear
    If .ListIndex <> -1 Then
        yr = OutYear.Value
    Else
        MsgBox "Select the number of out-years to predict."
        .SetFocus
        Exit Sub
    End If
End With

tr = 11
stepsizeLR = 0.001
nuFF = 0.7
nuRB = 1
MFF = 5
agg = 5

Unload Me

Call Sheet2.Main(tr, yr, stepsizeLR, nuFF, nuRB, MFF, agg, tag)
```

```
End Sub

Private Sub UserForm_Initialize()
'Populate the out-year prediction listbox
If OutYear.ListIndex = -1 Then
   For i = 0 To 2
      OutYear.AddItem (i)
   Next i
End If
End Sub
```

AARES Logic

```
Option Explicit
Option Base 1

Dim i As Integer, j As Integer, k As Integer, l As Integer, ii As Integer, _
iter As Integer, n As Integer, m As Integer, q As Integer, _
prevtoterr As Double, toterr As Double, count As Integer, X() As Double, _
t() As Double, ncols As Integer, nrows As Integer, agg As Integer, _
nn As Integer, mm As Integer, qq As Integer, jj As Integer, sumcount As Double, _
zlr() As Double, zzlr() As Double, zff() As Double, zzff() As Double, _
zvff() As Double, zzvff() As Double, zrb() As Double, zzrb() As Double, _
zvrb() As Double, zzvrb() As Double, cc As Integer, rr As Integer, marker As Integer, _
ZGem() As Double, corrZ() As Double, kk As Integer

Sub Main(tr, yr, stepsizeLR, nuFF, nuRB, MFF, agg, tag)

Call Capture

' if doing the quick estimate, get maximum training points
If tag = 1 Then
   tr = UBound(X, 2) - yr
End If

' check to ensure not training beyond prediction capability
If tr + yr > UBound(X, 2) Then
   MsgBox "Sum of Training Years and Out-Year Prediction must be <= " & UBound(X, 2)
   UserInputs.Show
End If

' get parameters to place model results
With Range("A2")
   cc = Range(.Offset(0, 0), .End(xlToRight)).Columns.count + 4
End With
With Range("E2")
   rr = Range(.Offset(1, 0), .End(xlDown)).Rows.count
End With

' copy over FY column -- will use for x-axis on charts
With Range("A2")
   For j = 0 To rr
      .Offset(j, 0).Copy
      .Offset(j, cc).PasteSpecial (xlPasteFormats)
      .Offset(j, cc).PasteSpecial (xlPasteValues)
```

```
        .Offset(j, 4).Copy
        .Offset(j, cc + 1).PasteSpecial (xlPasteFormats)
        .Offset(j, cc + 1).PasteSpecial (xlPasteValues)
    Next j
    .Offset(-1, cc).Value = "Reliablity Estimates"
    .Offset(-1, cc).Characters.Font.Size = 10
    .Offset(-1, cc).Characters.Font.Bold = True
    For j = tr + 1 + yr To UBound(X, 2) + yr
        .Offset(j, cc).Value = .Offset(j - 1, cc).Value + 1
        .Offset(j, cc).Borders(xlEdgeRight).LineStyle = xlContinuous
        .Offset(j, cc).HorizontalAlignment = xlCenter
    Next j
End With

marker = 0

If stepsizeLR <> 0 Then
    Call LogReg(tr, yr, stepsizeLR)
End If

If nuFF <> 0 Then
    Call FFNN(tr, yr, nuFF, MFF, agg)
End If

If nuRB <> 0 Then
    Call RBFNN(tr, yr, nuRB, agg)
End If

If marker > 1 Then
    Call Fusion(tr, yr)
End If

Call errors(tr, yr)

If tag = 1 Then
    Call QuickChart(yr)
Else
    Call Chart
End If

End Sub

Sub LogReg(tr, yr, stepsizeLR)
' logistic regression (instantaneous)

' strip off bottom row of flight test results from input matrix and set as target vector
ReDim t(1, UBound(X, 2))
For i = 1 To UBound(X, 2)
    t(1, i) = X(UBound(X, 1), i)
Next i

'sets nfeat = to the number of columns
Dim nfeat As Integer
nfeat = UBound(X, 1) - 1
```

```
' output training vector
ReDim zlr(tr + yr) As Double
' output prediction vector
ReDim zvlr(UBound(X, 2) + yr) As Double
' weight vector
ReDim w(nfeat) As Double
' weight gradient vector
ReDim dw(nfeat) As Double

'variable to index where to display data
marker = marker + 1

' zero out weights
For ii = 1 To nfeat
    w(ii) = 0
Next ii

'sets number of iterations for code to run through
iter = 1000
' used as a comparator to know when to stop increasing iterations
prevtoterr = 1
' parameter that tells the code when to stop (when decreases in toterr become very small)
toterr = 0

' loops through with increasing number of iterations until graph stabilizes
' and converges -- when toterr changes very little
Do While Abs(prevtoterr - toterr) > 0.001
    prevtoterr = toterr
    For i = 1 To iter
        toterr = 0 ' zeros out total error
        For ii = 1 To nfeat
            dw(ii) = 0 ' zeros out dw, differential of the error
        Next ii
        For j = 1 + yr To tr + yr 'j runs from 1 down the number of rows
            zlr(j) = 0 ' initializes zlr(j) at zero (estimated value)
            For k = 1 To nfeat ' runs from 1 across the number of columns
                zlr(j) = zlr(j) + w(k) * X(k, j - yr) ' sets Yhat = previous_Yhat + weight*current x_value,  x_value
changes across the columns
            Next k ' does this across the columns
            zlr(j) = (1 / (1 + Exp(-1 * zlr(j)))) ' call the sigmoid file and do it's thing with the z_matrix element
            For l = 1 To nfeat 'l runs across the columns
                dw(l) = (zlr(j) - X(nfeat + 1, j)) * zlr(j) * (1 - zlr(j)) * X(l, j - yr) ' cumes all the differentials of the
errors
                w(l) = w(l) - stepsizeLR * dw(l) ' steps in the direction opposite the error, converges toward the
"true" weights/b_knot and b_one
            Next l
            toterr = toterr + ((zlr(j) - X(nfeat + 1, j)) ^ 2) ' cumes total error per iteration
        Next j
    Next i
    ' sets number of iterations to run through next depending upon changes
    ' in toterr
    If Abs(prevtoterr - toterr) > 0.01 Then
        iter = iter + 1000
    Else
```

```
        iter = iter + 500
    End If
Loop

' logreg prediction code
If tr < UBound(X, 2) Then
    For n = tr + 1 + yr To UBound(X, 2) + yr
        zvlr(n) = 0
        For k = 1 To nfeat
            zvlr(n) = zvlr(n) + w(k) * X(k, n - yr)
        Next k ' end k loop
        zvlr(n) = 1 / (1 + Exp(-(zvlr(n))))
    Next n ' end n loop
End If ' end year check

With Range("k2")
    .Offset(0, marker) = "Log Reg"
    .Offset(0, 0).Copy
    .Offset(0, marker).PasteSpecial (xlPasteFormats)
    For ii = 1 + yr To tr + yr
        .Offset(ii, marker) = zlr(ii)
        .Offset(ii, marker).HorizontalAlignment = xlCenter
        .Offset(ii, marker).NumberFormat = "##.00%"
        .Offset(ii, marker).Characters.Font.Size = 8
    Next ii
    For ii = 1 + tr + yr To UBound(X, 2) + yr
        .Offset(ii, marker) = zvlr(ii)
        .Offset(ii, marker).HorizontalAlignment = xlCenter
        .Offset(ii, marker).NumberFormat = "##.00%"
        .Offset(ii, marker).Characters.Font.Size = 8
    Next ii
    If marker = 1 Then
        .Offset(1 + yr, -2) = "Training"
            If tr < UBound(X, 2) Then
                .Offset(1 + tr + yr, -2) = "Prediction"
            End If
    End If
End With

End Sub

Sub FFNN(tr, yr, nuFF, MFF, agg)
Randomize

' strip off bottom row of flight test results from input matrix and set as target vector
ReDim t(1, UBound(X, 2))
For i = 1 To UBound(X, 2)
    t(1, i) = X(UBound(X, 1), i)
Next i

'variable to index where to display data
marker = marker + 1
' number of runs to and then average together
'agg = 2
```

```
' number of inputs (features)
n = UBound(X, 1) - 1
' number of midddle layer neurodes
m = MFF
' number of output layer neurodes
j = UBound(t, 1)
' average of output runs
ReDim zzff(j, tr + yr) As Double
' average of prediction runs
ReDim zzvff(j, UBound(X, 2) + yr) As Double
' lower layer output matrix
ReDim zff(j, tr + yr, agg) As Double
' verification output matrix
ReDim zvff(j, UBound(X, 2) + yr, agg) As Double

' loop through a few times to get an average of the output values
For count = 1 To agg
' upper layer output row vector
ReDim Y(m) As Double
' middle layer weights matrix
ReDim w(n, m) As Double
' upper layer weights matrix
ReDim u(m, j) As Double
' middle layer summations weight gradients
ReDim dw(n, m) As Double

' set number of iterations
iter = 1500
' setting initial weights
For mm = 1 To m
    For nn = 1 To n
        w(nn, mm) = (0.4 * Rnd) - 0.2
    Next nn
    For jj = 1 To j
        u(mm, jj) = (0.4 * Rnd) - 0.2
    Next jj
Next mm ' end m loop, setting initial weights

prevtoterr = 1
toterr = 0
Do While Abs(prevtoterr - toterr) > 0.001
    prevtoterr = toterr
    ' initialize iterations
    For i = 1 To iter
        toterr = 0
        ' run down the rows of exemplars
        For qq = 1 + yr To tr + yr
            ' zero out outputs
            For jj = 1 To j
                zff(jj, qq, count) = 0
            Next jj ' end j loop, zero out outputs
            For nn = 1 To n
                For mm = 1 To m
                    dw(nn, mm) = 0
```

```
            Next mm ' end m loop
         Next nn ' end n loop, zero out summation portion of middle layer weight gradients
         For mm = 1 To m
            'calculate middle layer outputs
            Y(mm) = 0
            For nn = 1 To n
               Y(mm) = Y(mm) + w(nn, mm) * X(nn, qq - yr)
            Next nn ' end n loop, sum across middle layer prior to squashing
            ' calculate sigmoid of middle layer outputs -- squash 'em
            Y(mm) = 1 / (1 + Exp(-(Y(mm))))
         Next mm ' end m loop, middle layer outputs
         ' calculate outputs
         For jj = 1 To j
            For mm = 1 To m
               zff(jj, qq, count) = zff(jj, qq, count) + u(mm, jj) * Y(mm)
            Next mm ' end m loop, sum across the outputs prior to squashing
            ' calculate sigmoid of outputs -- squash 'em
            zff(jj, qq, count) = 1 / (1 + Exp(-(zff(jj, qq, count))))
         Next jj ' end j loop, new output loop
         ' adjust weights
         For mm = 1 To m
            ' calculate new upper layer weights
            For jj = 1 To j
               u(mm, jj) = u(mm, jj) + nuFF * ((t(jj, qq) - zff(jj, qq, count)) * zff(jj, qq, count) * (1 - zff(jj, qq,
count)) * Y(mm))
            Next jj ' end j loop, uppper layer weight update
            ' calculate summation portion of gradient for middle layer
            For nn = 1 To n
               For jj = 1 To j
                  dw(nn, mm) = dw(nn, mm) + (t(jj, qq) - zff(jj, qq, count)) * (zff(jj, qq, count) * (1 - zff(jj,
qq, count))) * u(mm, jj)
               Next jj ' end j loop cume portion of middle layer weight gradient
               ' calculate middle layer weights
               w(nn, mm) = w(nn, mm) + nuFF * dw(nn, mm) * (Y(mm) * (1 - Y(mm)) * X(nn, qq - yr))
            Next nn ' end n loop middle layer weight adjustments
         Next mm ' end m loop, weight adjustments
         ' calculate SSE
         For jj = 1 To j
            toterr = toterr + (zff(jj, qq, count) - t(jj, qq)) ^ 2
         Next jj ' end toterr cume loop
      Next qq ' end q loop number of exemplars on which to train
   Next i 'end iteration loop
   If Abs(prevtoterr - toterr) > 0.005 Then
      iter = iter + 100
   Else
      iter = iter + 50
   End If ' end iteration step-check loop
Loop ' end .001 while loop

' verify weights developed during training -- attempt to predict current year or out-year flight
' test results within data set
If tr < UBound(X, 2) Then
   For qq = tr + 1 + yr To UBound(X, 2) + yr
      For jj = 1 To j
```

```
            zvff(jj, qq, count) = 0
        Next jj
        For mm = 1 To m
            Y(mm) = 0
            For nn = 1 To n
                Y(mm) = Y(mm) + w(nn, mm) * X(nn, qq - yr)
            Next nn
            Y(mm) = 1 / (1 + Exp(-(Y(mm))))
            For jj = 1 To j
                zvff(jj, qq, count) = zvff(jj, qq, count) + u(mm, jj) * Y(mm)
            Next jj
        Next mm
        For jj = 1 To j
            zvff(jj, qq, count) = 1 / (1 + Exp(-(zvff(jj, qq, count))))
        Next jj
    Next qq ' end verification loop
End If ' end prediction test
Next count  ' end count loop

' calculate average of the training runs and display
For jj = 1 To j
    For qq = 1 + yr To tr + yr
        sumcount = 0
        For count = 1 To agg
            sumcount = sumcount + zff(jj, qq, count)
        Next count
        zzff(jj, qq) = sumcount / UBound(zff, 3)
    Next qq
Next jj
'MsgBox "Training " & tr & " Out-year " & yr & " stepsize " & nuFF
' calculate average of prediction runs
If tr < UBound(X, 2) Then
    For jj = 1 To j
        For qq = tr + 1 + yr To UBound(X, 2) + yr
            sumcount = 0
            For count = 1 To agg
                sumcount = sumcount + zvff(jj, qq, count)
            Next count
            zzvff(jj, qq) = sumcount / UBound(zvff, 3)
        Next qq
    Next jj
End If

'present calculated estimates in worksheet
With Range("k2")
    .Offset(0, marker) = "FFN"
    .Offset(0, 0).Copy
    .Offset(0, marker).PasteSpecial (xlPasteFormats)
    For jj = 1 To UBound(t, 1)
        For ii = 1 + yr To tr + yr
            .Offset(ii, marker) = zzff(jj, ii)
            .Offset(ii, marker).HorizontalAlignment = xlCenter
            .Offset(ii, marker).NumberFormat = "##.00%"
            .Offset(ii, marker).Characters.Font.Size = 8
```

```
        Next ii
        For ii = 1 + tr + yr To UBound(X, 2) + yr
            .Offset(ii, marker) = zzvff(jj, ii)
            .Offset(ii, marker).HorizontalAlignment = xlCenter
            .Offset(ii, marker).NumberFormat = "##.00%"
            .Offset(ii, marker).Characters.Font.Size = 8
        Next ii
    Next jj
    If marker = 1 Then
        .Offset(1 + yr, -2) = "Training"
            If tr < UBound(X, 2) Then
                .Offset(1 + tr + yr, -2) = "Prediction"
            End If
    End If
End With

End Sub

Sub RBFNN(tr, yr, nuRB, agg)
' RBFN code
Randomize

' strip off bottom row of flight test results from input matrix and set as target vector
ReDim t(1, UBound(X, 2))
For i = 1 To UBound(X, 2)
    t(1, i) = X(UBound(X, 1), i)
Next i

'variable to index where to display data
marker = marker + 1
' number of runs to and then average together
'agg = 2
' number of inputs (features)
n = UBound(X, 1) - 1
' number of midddle layer neurodes
m = tr
' number of output layer neurodes
j = UBound(t, 1)

' set output vectors
ReDim zrb(j, tr + yr, agg) As Double
ReDim zvrb(j, UBound(X, 2) + yr, agg) As Double
ReDim zzrb(j, tr + yr) As Double
ReDim zzvrb(j, UBound(X, 2) + yr) As Double

' summation variables for use in code
Dim adduys As Double
Dim distnc As Double

' loop through a few times and get an averaqe
For count = 1 To agg
' upper layer output row vector
ReDim Y(m, tr + yr) As Double
' middle layer neurode centers
```

```
ReDim v(n, m) As Double
' upper layer weights matrix
ReDim u(m, j) As Double
' upper layer weights gradients
ReDim du(m, j) As Double
' middle layer summations weight gradients
ReDim dw(j, tr + yr) As Double
' summation matrix for distance calculation
ReDim addup(m, UBound(X, 2) + yr) As Double

' set number of iterations
iter = 100
'compute single spread parameter
Dim sigma As Double
sigma = 1 / ((2 * m) ^ (1 / n))
' setting initial weights, neurode centers
For mm = 1 To m
    For jj = 1 To j
        u(mm, jj) = (0.5 * Rnd) - 0.5
    Next jj ' end J loop
    For nn = 1 To n
        v(nn, mm) = X(nn, mm)
    Next nn ' end n loop
Next mm ' end m loop, setting initial weights

' used as a comparator to know when to stop increasing iterations
prevtoterr = 1
' parameter that tells the code when to stop (when decreases in toterr become very small)
toterr = 0
' calculate difference vector
For qq = 1 + yr To tr + yr
    For mm = 1 To m
        distnc = 0
        For nn = 1 To n
            distnc = distnc + (X(nn, qq - yr) - v(nn, mm)) ^ 2
        Next nn
        addup(mm, qq) = distnc
    Next mm
Next qq

' compute y(m,q)
For qq = 1 + yr To tr + yr
    For mm = 1 To m
        If qq = mm Then
           Y(mm, qq) = 1
       Else
           Y(mm, qq) = Exp(-(addup(mm, qq)) / (2 * (sigma ^ 2)))
       End If ' end if test
    Next mm ' end m loop
Next qq ' end q loop

' train the network
Do While Abs(prevtoterr - toterr) > 0.00001
    prevtoterr = toterr
```

```
    ' initialize iterations
    For i = 1 To iter
        toterr = 0
        For mm = 1 To m
            For jj = 1 To j
                du(mm, jj) = 0
                For qq = 1 + yr To tr + yr
                    dw(jj, qq) = 0
                Next qq ' end q loop
            Next jj ' end j loop
        Next mm ' end m loop
        ' compute new outputs
        For qq = 1 + yr To tr + yr
            For jj = 1 To j
                For mm = 1 To m
                    dw(jj, qq) = dw(jj, qq) + (u(mm, jj) * Y(mm, qq))
                Next mm ' end m loop
            Next jj ' end j loop
        Next qq ' end new output loops
        For qq = 1 + yr To tr + yr
            For jj = 1 To j
                zrb(jj, qq, count) = dw(jj, qq) / m
            Next jj ' end j loop
        Next qq ' end q loop
        ' SSE calculation
        For qq = 1 + yr To tr + yr
            For jj = 1 To j
                toterr = toterr + ((t(jj, qq) - zrb(jj, qq, count)) ^ 2)
            Next jj ' end j loop
        Next qq ' end error calculation
        If toterr < prevtoterr Then
            nuRB = nuRB * 1.04
        Else
            nuRB = nuRB * 0.92
        End If ' end new stepsize check
        ' adjust weights
        For mm = 1 To m
            For jj = 1 To j
                For qq = 1 + yr To tr + yr
                    du(mm, jj) = du(mm, jj) + ((t(jj, qq) - zrb(jj, qq, count)) * Y(mm, qq))
                Next qq ' end q loop
            Next jj ' end j loop
        Next mm ' end m loop
        For mm = 1 To m
            For jj = 1 To j
                u(mm, jj) = u(mm, jj) + ((2 * nuRB) / m) * du(mm, jj)
            Next jj ' end j loop
        Next mm ' end m loop
    Next i ' end iteration loop
Loop ' end tolerance loop

' test middle layer outputs
ReDim ytest(m, UBound(X, 2) + yr) As Double
```

```
' verify test data
If tr < UBound(X, 2) Then
    For qq = tr + 1 + yr To UBound(X, 2) + yr
        ' zero out output matrix
        For jj = 1 To j
            zvrb(jj, qq, count) = 0
        Next jj ' end j loop
        ' calculate distances from center
        For mm = 1 To m
            distnc = 0
            For nn = 1 To n
                distnc = distnc + (X(nn, qq - yr) - v(nn, mm)) ^ 2
            Next nn ' end n loop
            addup(mm, qq) = distnc
        Next mm ' end m loop
    Next qq ' end q loop
    ' compute ytest(m,q)
    For qq = tr + 1 + yr To UBound(X, 2) + yr
        For mm = 1 To m
            ytest(mm, qq) = Exp(-(addup(mm, qq)) / (2 * (sigma ^ 2)))
        Next mm ' end m loop
    Next qq ' end q loop
    ' compute outputs
    For qq = tr + 1 + yr To UBound(X, 2) + yr
        For jj = 1 To j
            adduys = 0
            For mm = 1 To m
                adduys = adduys + u(mm, jj) * ytest(mm, qq)
            Next mm ' end m loop
            zvrb(jj, qq, count) = adduys / m
        Next jj ' end j loop
    Next qq ' end q loop
End If ' end prediction test
Next count ' end count loop

' calculate average of the training runs and display
For jj = 1 To j
    For qq = 1 + yr To tr + yr
        sumcount = 0
        For count = 1 To agg
            sumcount = sumcount + zrb(jj, qq, count)
        Next count
        zzrb(jj, qq) = sumcount / UBound(zrb, 3)
    Next qq
Next jj
'MsgBox "Training " & tr & " Out-year " & yr & " stepsize " & nuFF
' calculate average of prediction runs
If tr < UBound(X, 2) Then
    For jj = 1 To j
        For qq = tr + 1 + yr To UBound(X, 2) + yr
            sumcount = 0
            For count = 1 To agg
                sumcount = sumcount + zvrb(jj, qq, count)
            Next count
```

```
        zzvrb(jj, qq) = sumcount / UBound(zvrb, 3)
      Next qq
    Next jj
End If

'present calculated estimates in workwheet
With Range("k2")
    .Offset(0, marker) = "RBFN"
    .Offset(0, 0).Copy
    .Offset(0, marker).PasteSpecial (xlPasteFormats)
    For jj = 1 To UBound(t, 1)
      For ii = 1 + yr To tr + yr
        .Offset(ii, marker) = zzrb(jj, ii)
        .Offset(ii, marker).HorizontalAlignment = xlCenter
        .Offset(ii, marker).NumberFormat = "##.00%"
        .Offset(ii, marker).Characters.Font.Size = 8
      Next ii
      For ii = tr + 1 + yr To UBound(X, 2) + yr
        .Offset(ii, marker) = zzvrb(jj, ii)
        .Offset(ii, marker).HorizontalAlignment = xlCenter
        .Offset(ii, marker).NumberFormat = "##.00%"
        .Offset(ii, marker).Characters.Font.Size = 8
      Next ii
    Next jj
    If marker = 1 Then
      .Offset(1 + yr, -2) = "Training"
        If tr < UBound(X, 2) Then
          .Offset(1 + tr + yr, -2) = "Prediction"
        End If
    End If
End With
End Sub

Sub Fusion(tr, yr)
' fuse the outputs from the selected nets
Dim denomalpha As Double
denomalpha = 0
'ReDim ZZGem(UBound(x, 2) + yr - tr) As Double
Dim numalpha As Double
Dim CM As Range
Dim PL As Range

'generate correlation matrix and display on worksheet
With Range("J2")
    j = Range(.Offset(0, 1), .End(xlToRight)).Columns.count
    ii = Range(.Offset(0, 0), .End(xlDown)).Rows.count
    Range(.Offset(yr + 1, 2), .Offset(tr + yr, j)).Select
    Range(.Offset(yr + 1, 2), .Offset(tr + yr, j)).Name = "CM"
    Range(.Offset(ii + 3, 1), .Offset(ii + 3, 1)).Name = "PL"
    Application.Run "ATPVBAEN.XLA!Mcorrel", ActiveSheet.Range("CM"), _
      ActiveSheet.Range("PL"), "C", False
    .Offset(ii + 2, 1) = "Correlation Matrix"
    .Offset(ii + 2, 1).Characters.Font.Size = 8
    .Offset(ii + 2, 1).Characters.Font.Bold = True
```

```
ReDim corrZ(j - 1, j - 1) As Double

'put worksheet correlation matrix into an array
For jj = 1 To j - 1
   For kk = 1 To j - 1
      corrZ(kk, jj) = Range(.Offset(kk + ii + 3, jj + 1), .Offset(kk + ii + 3, jj + 1)).Value
   Next kk
Next jj

'MsgBox "corrZ " & corrZ(1, 1) & "  " & corrZ(1, 2) & "  " & corrZ(2, 1) & "  " & corrZ(2, 2)

ReDim alpha(j - 1) As Double

' make matrix symmetrical for ease of use, sum up inverse of elements for denominator
For jj = 1 To j - 1
   For kk = 1 To j - 1
      If corrZ(kk, jj) = 0 Then
         corrZ(kk, jj) = corrZ(jj, kk)
      End If
      'MsgBox "corrZ " & corrZ(kk, jj)
   Next kk
Next jj

For jj = 1 To j - 1
   For kk = 1 To j - 1
      'MsgBox "corrZ " & corrZ(kk, jj)
      denomalpha = denomalpha + (1 / corrZ(kk, jj))
   Next kk
Next jj

' calculate numerator and weights, display on worksheet
.Offset(ii + 6 + UBound(corrZ, 1), 1) = "Fusion Weights"
.Offset(ii + 6 + UBound(corrZ, 1), 1).Characters.Font.Size = 8
.Offset(ii + 6 + UBound(corrZ, 1), 1).Characters.Font.Bold = True
For jj = 1 To UBound(corrZ, 1)
   numalpha = 0
   For kk = 1 To UBound(corrZ, 2)
      numalpha = numalpha + (1 / corrZ(jj, kk))
   Next kk
   alpha(jj) = numalpha / denomalpha
   .Offset(ii + 7 + UBound(corrZ, 1), 1 + jj) = alpha(jj)
Next jj

'Calculate fused outputs and display on worksheet
.Offset(0, j + 1) = "Fused"
.Offset(0, j + 1).HorizontalAlignment = xlCenter
.Offset(0, j + 1).Characters.Font.Size = 8
.Offset(0, j + 1).Characters.Font.Bold = True
.Offset(0, j + 1).Borders(xlEdgeBottom).LineStyle = xlContinuous
.Offset(0, j + 1).Borders(xlEdgeLeft).LineStyle = xlContinuous

ReDim ZGem(ii - 1) As Double
For kk = 1 + yr To ii - 1
   ZGem(kk) = 0
```

```
    For jj = 1 To UBound(corrZ, 1)
        ZGem(kk) = ZGem(kk) + alpha(jj) * .Offset(kk, jj + 1).Value
    Next jj
    .Offset(kk, j + 1) = ZGem(kk)
    .Offset(kk, j + 1).HorizontalAlignment = xlCenter
    .Offset(kk, j + 1).NumberFormat = "##.00%"
    .Offset(kk, j + 1).Characters.Font.Size = 8
Next kk

End With

End Sub

Sub errors(tr, yr)

' calculate the errors of the selected methods
' first capture the outputs and put into a matrix
Dim ncols As Integer, nrows As Integer, Z() As Double

If marker = 1 Then
    qq = 3
    Else
    qq = 14
End If

With Range("J2")
    ncols = Range(.Offset(0, 2), .End(xlToRight)).Columns.count
    nrows = Range(.Offset(1, 0), .End(xlDown)).Rows.count
    ReDim Z(nrows, ncols) As Double
    For nn = 1 To nrows
        For mm = 1 To ncols
            Z(nn, mm) = .Offset(nn, 1 + mm).Value
        Next mm
    Next nn

    ReDim sse(ncols) As Double, mse(ncols) As Double, rmse(ncols) As Double
    For mm = 1 To ncols
        For nn = 1 + yr To UBound(X, 2)
            sse(mm) = sse(mm) + (t(1, nn) - Z(nn, mm)) ^ 2
        Next nn
        mse(mm) = sse(mm) / (UBound(X, 2) - yr)
        rmse(mm) = mse(mm) ^ (1 / 2)
    Next mm

    .Offset(nrows + qq, 1) = "SSE"
    .Offset(nrows + qq, 1).Characters.Font.Size = 8
    .Offset(nrows + qq, 1).Characters.Font.Bold = True
    .Offset(nrows + qq + 2, 1) = "MSE"
    .Offset(nrows + qq + 2, 1).Characters.Font.Size = 8
    .Offset(nrows + qq + 2, 1).Characters.Font.Bold = True
    .Offset(nrows + qq + 4, 1) = "RMSE"
    .Offset(nrows + qq + 4, 1).Characters.Font.Size = 8
    .Offset(nrows + qq + 4, 1).Characters.Font.Bold = True
```

```
    For mm = 1 To ncols
        .Offset(nrows + qq, 1 + mm) = sse(mm)
        .Offset(nrows + qq + 2, 1 + mm) = mse(mm)
        .Offset(nrows + qq + 4, 1 + mm) = rmse(mm)
    Next mm

End With

End Sub

Private Sub GoBabyGo_Click()

' clear old model results
With Range("I1")
    Range(.Offset(0, 0), .Offset(100, 50)).Clear
End With

Worksheets("Model").ChartObjects.Delete

SnappyIntro.Show

End Sub
Sub Capture()

' collect the number of years worth of flight test data
With Range("E2")
    ncols = Range(.Offset(1, 0), .End(xlDown)).Rows.count
End With

'collect the number of features
With Range("B2")
    nrows = Range(.Offset(0, 0), .End(xlToRight)).Columns.count
    ' add a row of ones across the top and take the transpose of the input matrix
    ReDim X(nrows + 1, ncols) As Double
    For j = 1 To ncols
        X(1, j) = 1
        For i = 1 To nrows
            X(i + 1, j) = .Offset(j, i - 1).Value
        Next i
    Next j
End With

End Sub

Sub Chart()
Dim ncols As Integer, nrows As Integer

With Range("J2")

    ncols = Range(.Offset(0, 0), .End(xlToRight)).Columns.count - 1
    nrows = Range(.Offset(0, 0), .End(xlDown)).Rows.count - 1
    Charts.Add
    ActiveChart.ChartType = xlXYScatterLines
```

```
    ActiveChart.SetSourceData Source:=Sheets("Model").Range(.Offset(0, 0), .Offset(nrows, ncols)),
PlotBy:= _
        xlColumns
    ActiveChart.Location Where:=xlLocationAsObject, Name:="Model"
    With ActiveChart
        .HasTitle = True
        .ChartTitle.Characters.Text = "Reliability Estimates"
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "FY"
        .Axes(xlValue, xlPrimary).HasTitle = True
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Reliability"
    End With
    ActiveChart.ApplyDataLabels Type:=xlDataLabelsShowNone, LegendKey:=False
    ActiveChart.Axes(xlCategory).Select
    With ActiveChart.Axes(xlCategory)
        .MinimumScale = 1990
        .MaximumScaleIsAuto = True
        .MinorUnitIsAuto = True
        .MajorUnitIsAuto = True
        .Crosses = xlCustom
        .CrossesAt = 1990
        .ReversePlotOrder = False
        .ScaleType = xlLinear
        .DisplayUnit = xlNone
    End With

End With

With ChartObjects(1)
    .Left = 0
    .Top = 214
End With
End Sub

Sub QuickChart(yr)
Dim nrows As Integer, ncols As Integer

With Range("J2")
    ncols = Range(.Offset(0, 0), .End(xlToRight)).Columns.count - 1
    nrows = Range(.Offset(0, 0), .End(xlDown)).Rows.count - 1
    Charts.Add
    ActiveChart.ChartType = xlXYScatterLines
    ActiveChart.SetSourceData Source:=Sheets("Model").Range(.Offset(0, 0), .Offset(nrows, ncols)), _
        PlotBy:=xlColumns
    ActiveChart.SeriesCollection(4).Delete
    ActiveChart.SeriesCollection(3).Delete
    ActiveChart.SeriesCollection(2).Delete
    ActiveChart.Location Where:=xlLocationAsNewSheet
    With ActiveChart
        .HasTitle = True
        .ChartTitle.Characters.Text = "Reliability Estimates"
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "FY"
        .Axes(xlValue, xlPrimary).HasTitle = True
```

```
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Reliability"
    End With
    ActiveChart.ApplyDataLabels Type:=xlDataLabelsShowValue, LegendKey:=False
    ActiveChart.SeriesCollection(2).DataLabels.Select
    Selection.AutoScaleFont = True
    With Selection.Font
        .Name = "Arial"
        .FontStyle = "Regular"
        .Size = 8
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
        .Background = xlAutomatic
    End With
    ActiveChart.SeriesCollection(1).Select
    ActiveChart.SeriesCollection(1).ApplyDataLabels Type:=xlDataLabelsShowNone, _
        AutoText:=True, LegendKey:=False

    If yr = 0 Then
        ActiveChart.Shapes.AddTextbox(msoTextOrientationHorizontal, 475, 5, _
        200, 45).Select
        Selection.Characters.Text = "Your current year reliabiity estimate is " & Round(.Offset(nrows,
ncols).Value * 100, 2) & _
        "%, +/- " & Round(.Offset(nrows + 18, ncols) * 100, 2) & "% (RMSE)."
        Selection.AutoScaleFont = False
    With Selection.Characters(Start:=1, Length:=70).Font
        .Name = "Arial"
        .FontStyle = "Bold"
        .Size = 12
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
    End With
    Else
        ActiveChart.Shapes.AddTextbox(msoTextOrientationHorizontal, 475, 5, _
        200, 45).Select
        Selection.Characters.Text = "Your " & yr & " year reliability prediction is " & Round(.Offset(nrows,
ncols).Value * 100, 2) & _
        "%, +/- " & Round(.Offset(nrows + 18, ncols) * 100, 2) & "% (RMSE)."
        Selection.AutoScaleFont = False
    With Selection.Characters(Start:=1, Length:=70).Font
        .Name = "Arial"
        .FontStyle = "Bold"
        .Size = 12
        .Strikethrough = False
        .Superscript = False
```

```
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
    End With
    End If
End With

End Sub
```

Bibliography

Bauer, Kenneth W. Class handout, OPER 785, Applied Multivariate Data Analysis 2, School of Engineering and Management, Air Force Institute of Technology, Wright-Patterson AFB OH, November 2002.

Bredehoeft, Michael R.; OC-ALC/PSMRT, Personal Interview, 12 August 2002, 25 October 2002

Department of the Air Force, TO 21-AG129-2-1, "AGM-129 Advanced Cruise Missile Operations Manual."

Karunanithi, Nachimuthu; Darrell Whitley, and Yashwant K. Malaiya, "Using Neural Networks in Reliability Prediction", *IEEE Software*, 9, #4: 53-59, (Jul/Aug 1992).

Lindblad, David, Mark Bringhurst, Rakesh Dewan, and Noriene Jee. "Weapon System Effectiveness for a Legacy System," *Second Biennial Forum on Weapon System Effectiveness*, 27-29 March 2001, Johns Hopkins, Applied Physics Laboratory, Laurel MD

Looney, Carl G. *Pattern Recognition Using Neural Networks, Theory and Algorithms for Engineers and Scientists*, Oxford University Press, 1977

Morris, Seymour and others. *Reliability Toolkit: Commercial Practices Edition*

Perrone, Michael P. and Leon N. Cooper, "When Networks Disagree: Ensemble Methods for Hybrid Neural Networks," October 27, 1992

Quick, David M., Major, ACC/DRYS, Personal Interview, 6-15 January 2003

SIOP Planning Factors Conference, Offutt AFB NE, 15-16 October 2002

Strategic Systems Department, Johns Hopkins, Applied Physics Laboratory; *Appendix B, Methodology and Supporting Analysis, Trident II and Trident III Reliability Plan*, 2002

Vita

Captain Donald Hoffman enlisted in the Air Force in November 1986. After completing basic training and attending technical school, he was stationed at Scott AFB, IL and assigned to the 375 Consolidated Aircraft Maintenance Squadron. He worked on the flightline as a crew chief and flight mechanic on C-9As until his honorable discharge in 1991 at the rank of staff sergeant. After completing his baccalaureate degree programs at St Louis University and working in industry for a year, Captain Hoffman was accepted to officer training school and commissioned a second lieutenant in March 1995. His first assignment was Barksdale AFB, LA working as a maintenance officer assigned to the 2nd Munitions Squadron and later the 11th Bomb Squadron. Donald's second assignment was Eglin AFB, FL working for Detachment 2, Air Force Operational Test and Evaluation Center as a Deputy for Logistics and Test Director. While there he completed his Master's of Business Administration (MBA) degree; applied and was accepted into the Operations Research master's program at AFIT. Upon graduation, he will be assigned to the USAFE Warrior Preparation Center at Einseidlerhof AB, Germany.

REPORT DOCUMENTATION PAGE							Form Approved OMB No. 074-0188
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**							
1. REPORT DATE *(DD-MM-YYYY)* 14-03-2003			**2. REPORT TYPE** **Master's Thesis**				**3. DATES COVERED** *(From – To)* Sep 2002 - Mar 2003
4. TITLE AND SUBTITLE USING NEURAL NETWORKS FOR ESTIMATING CRUISE MISSILE RELIABILITY							**5a. CONTRACT NUMBER** **5b. GRANT NUMBER** **5c. PROGRAM ELEMENT NUMBER**
6. AUTHOR(S) Hoffman, Donald, L., Captain, USAF							**5d. PROJECT NUMBER** **5e. TASK NUMBER** **5f. WORK UNIT NUMBER**
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Building 640, WPAFB OH 45433-7765							**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GOR/ENS/03-10
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) ACC/DONO Attn: Mr. Al Montalvo 205 Dodd Blvd; Suite 101 DSN: 574-6415 LAFB VA 23655 e-mail: AL.MONTALVO@LANGLEY.AF.MIL							**10. SPONSOR/MONITOR'S ACRONYM(S)** **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**
12. DISTRIBUTION/AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.							
13. SUPPLEMENTARY NOTES							
14. ABSTRACT ACC believes its current methodology for predicting the reliability of its Air Launched Cruise Missile (ALCM) and Advanced Cruise Missile (ACM) stockpiles could be improved. They require a predictive model that delivers a realistic 24-month projection of cruise missile reliability using existing data sources, collection methods and software. It should be easily maintainable and developed to allow a layperson to enter updated data and receive an accurate reliability prediction. The focus of this thesis is to improve upon free flight reliability, although the techniques could be applied to the captive carry portion of the missile reliability equation also. The end product is the ALCM/ACM Reliability Estimation System (AARES), a VBA-based model that meets all user criteria.							
15. SUBJECT TERMS Cruise Missile, Neural Networks, Feed Forward Neural Nets, Radial Basis Function Network, Logistic Regression, ALCM, ACM, Reliabilty							
16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Stephen P. Chambal, Capt, USAF (ENS)		
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	121	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 4314; e-mail: Stephen.Chambal@afit.edu		

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18