NASA/TM–2001-209622

A Closed-Loop Optimal Neural-Network Controller to Optimise Rotorcraft Aeromechanical Behaviour
Volume 1: Theory and Methodology

Jane Anne Leyland

March 2001

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the Lead Center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at *http://www.sti.nasa.gov*

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA Access Help Desk at (301) 621-0134

- Telephone the NASA Access Help Desk at (301) 621-0390

- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320

NASA/TM–2001-209622

A Closed-Loop Optimal Neural-Network Controller to Optimise Rotorcraft Aeromechanical Behaviour
Volume 1: Theory and Methodology

Jane Anne Leyland
Ames Research Center, Moffett Field, California

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

March 2001

Available from:

NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
(703) 487-4650

Table of Contents

Table of Contents (Continued)

ABSTRACT

A Closed-Loop Optimal Neural-Network Controller
to
Optimise Rotorcraft Aeromechanical Behaviour

by

Jane Anne Leyland, PhD/AMES
NASA-Ames Research Centre
Moffett Field, California

Previous development and design of closed-loop controllers to optimise rotorcraft aeromechanical behaviour focused on the simple "standard" closed-loop controller which employs an actively updated linear plant model (i.e., a single system matrix) to model the rotorcraft and simplified pseudo-optimal methods to determine the control. A recent development was the use of modern constrained optimisation techniques rather than the commonly used pseudo-optimal methods to determine the optimal control subject to constraints for a linear plant model. One promising controller scheme which is of interest to analysts at this time utilises a "neural-network" scheme to provide a general non-linear model the plant. Accordingly a closed-loop optimal neural-network controller was developed which employs a general non-linear neural-network function rather than a linear function to model the plant. Modern constrained optimisation methods are used to determine/update the constants in the neural-network plant model as well as in the determination of the optimal control vector.

Current data is read, weighted, and added to a sliding data window. When the specified maximum data window length (i.e., the number of data sets allowed in the data window) is exceeded, the oldest data set is purged and the remaining data sets are re-weighted. This procedure provides at least four additional degrees-of-freedom in addition to the size and geometry of the neural-network itself with which to optimise the overall operation of the controller (e.g., the update of the non-linear neural-network plant model and the determination of the optimal control). These additional degrees-of-freedom are: 1. the maximum length of the sliding data

window, 2. the frequency of neural-network updates, 3. the weighting of the individual data sets within the sliding window, and 4. the maximum number of optimisation iterations used for the neural-network updates.

Cases run to date indicate that the controller is operating as planned, but that the controller performance as measured by the rate of convergence of the neural-network parameters is slow. This is due to the fact that the determination of the neural-network parameters by minimisation of an error metric of the neural-network function values is an ill-posed problem with multiple solutions for these parameters. Elimination of multiple solutions with corresponding acceleration of convergence appears to be possible with the addition of a regularisation functional to the error metric performance index.

1.0 INTRODUCTION

Given the predicted growth in air transportation, the potential exists for significant market niches for rotary wing subsonic vehicles. Technological advances which optimise rotorcraft aeromechanical behaviour can contribute significantly to both their commercial and military development, acceptance, and sales. Examples of the optimisation of rotorcraft aeromechanical behaviour which are of interest include the minimisation of vibration and/or loads. The reduction of rotorcraft vibration and loads is an important means to extend the useful life of the vehicle and to improve its ride quality. Although vibration reduction can be accomplished by using passive dampers and/or tuned masses, active closed-loop control has the potential to reduce vibration and loads throughout a wider flight regime whilst requiring less additional weight to the aircraft than that obtained by using passive methods. It is emphasised that the analysis described herein is applicable to all those rotorcraft aeromechanical behaviour optimisation problems for which the relationship between the harmonic control vector and the measurement vector can be adequately described by a neural-network model.

Previous development and design of closed-loop controllers to optimise rotorcraft aeromechanical behaviour focused on the simple "standard" closed-loop controller which employs an actively updated linear plant model (i.e., a single system matrix) to model the rotorcraft and simplified pseudo-optimal methods to determine the control. A recent development (Reference 1) was the use of modern constrained optimisation techniques (References 2 through 8) rather than the commonly used pseudo-optimal methods to determine the optimal control subject to constraints for a linear plant model. One promising controller scheme which is of interest to analysts at this time utilises a "neural-network" scheme to provide a general non-linear model of the plant. Accordingly a closed-loop optimal neural-network controller was developed which employs a general non-linear neural-network function rather than a linear function to model the plant. The modern constrained optimisation methods described in References 2 through 8 are used to determine/update the constants in the neural-network plant model and to determine the optimal control vector by employing the IMSL main driver routines DNCONF and/or DNCONG and their subroutines as described in Reference 9.

Current data is read, weighted, and added to a sliding data window. When the specified maximum data window length (i.e., the number of data sets allowed in the data window) is exceeded, the oldest data set is purged and the remaining data sets are re-weighted. This procedure provides at least four additional degrees-of-freedom in addition to the size and geometry of the neural-network itself with which to optimise the overall operation of the controller (i.e., the update of the non-linear neural-network function plant model and the determination of the optimal control). These additional degrees-of-freedom are: 1. the maximum length of the sliding data window, 2. the frequency of the neural-network updates, 3. the weighting of the individual data sets within the sliding window, and 4. the maximum number of optimisation iterations used for the neural-network updates.

2.0 TECHNICAL

A typical general closed-loop controller which is the reference controller for this study is discussed first, noting the two forms of systems models which are of interest for rotorcraft aeromechanical behaviour problems. Next, the proposed optimal closed-loop neural-network (N^2) controller is presented. The analytic non-linear neural-network function $f_{N^2}(\bullet)$ or more specifically $f_{N^2}^{I_I - I_2 - I_3 - \cdots\cdots - I_K - J_K}(\bullet)$ when the neurone distribution (i.e., the number of nodes per layer) is defined, and an example geometrical schematic is presented for the 3-5-3-2 neural-network function $f_{N^2}^{3-5-3-2}(\bullet)$. Several neural-network node filters are presented and the "sliding window" of data acquisition is explained. The optimisation method used to update the neural-network parameters and the control vector is discussed, and various sources of trajectory data are identified. The stand-alone optimal neural-network controller system which was developed during this study is described and the results to date of using this controller system are discussed. Lastly, conclusions and recommendations are presented.

2.1 General Closed-Loop Controller

The general controller scheme assumes that the measured behaviour (i.e., the measurement state vector, the measurement vector, the Z-vector, etc.) of a physical system (i.e., the rotorcraft, the plant, etc.) can be completely controlled by means of an appropriate system control vector (i.e., the control vector, the θ-vector, etc.). A schematic representation of this fundamental relationship appears in the upper part of Figure 1. The general controller uses this relationship together with a mathematical model of it to estimate the control vector to be used in a future duty cycle which will satisfy some criteria. The relationship between the control vector, the mathematical model of the rotorcraft, and the measurement state vector is schematically shown in the lower part of Figure 1.

The general closed-loop controller (see Figure 2) is comprised of two parts: 1) the operating rotorcraft plant which generates the measurement vector for the currently specified control vector, and 2) the controller itself which estimates the control vector which will satisfy some criteria to be used in a future duty cycle. This latter function uses the mathematical model of the rotorcraft to estimate the new control vector. The parameters of the model can be updated during the trajectory if an appropriate update scheme is available. The new estimated control vector is then input to the operating rotorcraft to be used in a future duty cycle. This looping process is continued until completion of the last duty cycle when the operation of the controller is terminated.

2.1.1 Systems Models of a Controlled Response

Mathematical models of the control vector - operating rotorcraft - measurement vector relationship used in the general closed-loop controller to estimate the control vector can be conveniently placed into one of two categories: 1) fixed form system models, and 2) free-form system models. As these category names suggest, the fixed form models are rigid and not too flexible even though their parameters can sometimes be updated during controller operation, and consequently they might not be suitable for experimental applications. The free form system models are not rigid and can be quite flexible, and consequently they can be quite amenable to experimental applications.

2.1.1.1 Fixed Form Systems Models

The fixed form systems models have a rigid mathematical function form/shape. Although this form/shape might be adjusted or distorted by appropriate selection of the values of the model parameters either initially or during the trajectory by a parameter identification process, the basic function shape is what it is and cannot be substantially changed by the model parameters. Examples of fixed form models include:

$$Z = T\theta + Z_0 \qquad \text{Linear (Simplistic)}$$

$$Z = \theta\theta^T A_2 + A_1\theta + Z_0 \qquad \text{Non-Linear (Quadratic)}$$

$$Z = B\,Tanh(A\theta) + Z_0 \qquad \text{Non-Linear (Hyperbolic Tangent)}$$

2.1.1.2 Free Form Systems Models

The free form systems models do not have a rigid mathematical function form/shape. The form/shape can be changed substantially by appropriate selection of the values of the model parameters either initially or during the trajectory by a parameter identification process. That is, the model is one for which the representing function(s) can be made to approximate operating rotorcraft relationship as closely as required at a finite number of points by appropriately selecting the values of the model parameters. Examples of free form models include:

$$Z = f_{SF}(\theta, C) \qquad \text{Surface Fit Functions}$$

$$Z = f_{N^2}(\theta, C) = f_{N^2}^{\,I_1 - I_2 - I_3 - \cdots - I_K - J_K}(\theta, C) \qquad \text{Neural-Network Functions}$$

where C is the attenuation coefficient matrix.

$I_1 - I_2 - I_3 - \cdots - I_K - J_K$ defines the number of origin and destination nodes for each neural-network layer. The convention used here uses the superscript chain to specify the number of available node positions at the origin side (i.e., the left side) of each layer, except for the last superscript value in the chain which denotes the number of available node positions at the destination side (i.e., the right side) of the last layer.

θ is the control vector.

The set of neural-network functions for this purpose is actually a subset of the set of all surface fit functions. The use of neural-network functions to model the operating rotorcraft within a closed-loop optimal controller being used to optimise rotorcraft aeromechanical behaviour is the subject of this study.

2.1.2 Primary Controller Function

As the title of this report indicates, the primary function of the closed-loop controller described in this document is to optimise specified rotorcraft aeromechanical behaviour by appropriate selection of the elements of the control vector.

2.2 An Optimal Closed-Loop Neural-Network Controller

The optimal closed-loop neural-network controller which was designed as part of this study and which is described herein, is an extension of the general controller scheme described in Section 2.1. As in the case of the general controller, the optimal closed-loop neural-network controller assumes that the measured behaviour (i.e., the measurement state vector, the measurement vector, the Z-vector, etc.) of a physical system (i.e., the rotorcraft, the plant, etc.) can be completely controlled by means of an appropriate system control vector (i.e., the control vector, the θ-vector, etc.). A schematic representation of this fundamental relationship is presented in Figure 1.

The optimal closed-loop neural-network controller (see Figure 3) differs from the general controller in that the mathematical model of the operating rotorcraft is specified to be a neural-network function whose parameters can be identified and updated during both a learning trajectory phase and a controlled trajectory phase. The learning trajectory phase is that part of the trajectory during which only the model parameters are identified and updated. The control vector is neither optimised nor updated during this phase. The controlled trajectory phase is that part of the trajectory during which either or both the control vector can be optimised and updated, and the neural-network model parameters can be identified and updated.

2.2.1 The Neural-Network Function and Its Geometry

The neural-network function $f_{N^2}(\bullet)$ as used in this study, is comprised of a connected set of nodes arranged in layers between the input control vector (i.e., the θ-vector) and the output measurement vector (i.e., the Z-vector). The convention adopted during this study for pictorial representations (see Figure 4) is that the signal flow and layer indexing goes from left to right, that is the θ-vector is input to the left of $f_{N^2}(\bullet)$ with resulting neural-network internal signal flow proceeding from left to right until the signals exit as the Z-vector at the right extremity of $f_{N^2}(\bullet)$. The lower case letter k denotes the layer index number in ascending order from left to right, that is k increases monotonically from 1 to K when proceeding from the θ-vector to the Z-vector where the upper case letter K denotes the index number of the last layer. As mentioned previously in Section 2.0, Figure 4 illustrates the geometry of the 3-5-3-2 neural-network function $f_{N^2}^{3-5-3-2}(\bullet)$.

The general form of the neural-network function $f_{N^2}(\bullet)$ is $f_{N^2}^{I_1 - I_2 - I_3 - \cdots\cdots - I_K - J_K}(\bullet)$, where the value of I_k for $k = 1, 2, 3, \cdots\cdots K$ specifies the number of available node positions at the origin side (i.e., the left side) of the k-th layer and the value of J_K specifies the number of available node positions at the destination side (i.e., the right side) of the last layer (i.e., the K-th layer). It is emphasised that the actual number of nodes that are used for $f_{N^2}(\bullet)$ in a specific application need not necessarily be maximum number that are available as specified by the values in the superscript chain. If

i is the origin index, that is the node number on the origin side (i.e., the left side), of the k-th layer. The convention adopted during this study for pictorial representations (see Figure 4) is that the node number increases with descending position. $i \in I_k$ where I_k is the set of all active origin nodes for the k-th layer.

j is the destination index, that is the node number on the destination side (i.e., the right side), of the k-th layer. The convention adopted during this study for pictorial representations (see Figure 4) is that the node number increases with descending position. $j \in J_k$ where J_k is the set of all active destination nodes for the k-th layer.

k is the layer index number in ascending order from left to right, that is k increases monotonically from 1 to K when proceeding from the θ-vector to the Z-vector where the upper case letter K denotes the index number of the last layer.

then the signal path between specific nodes is uniquely defined by the indices i, j, k. If

$C_{i,j,k}$ is the attenuation coefficient for the signal directed from the i-th origin node of the k-th layer toward the j-th destination node of the k-th layer, where $C_{i,j,k}$ is constrained according to:

$$C_{i,j,k} \in \left[C_{MIN_{i,j,k}}, C_{MAX_{i,j,k}} \right]$$

for

$$C_{MIN_{i,j,k}} \in (-\infty, +\infty)$$

$$C_{MAX_{i,j,k}} \in (-\infty, +\infty)$$

$$C_{MIN_{i,j,k}} \leq C_{MAX_{i,j,k}}$$

$f_F(u_{j,k})$ is the filter function (i.e., pass-through function) which is applied just prior to (i.e., immediately to the left of) the j-th destination node of the k-th layer.

$u_{j,k}$ is the summation of all the attenuated signals directed from the active origin nodes of the k-th layer toward the j-th destination node of the k-th layer.

$x_{i,j,k}$ is the exit signal from the i-th origin node of the k-th layer which is directed toward the j-th destination node of the k-th layer.

$y_{j,k}$ is the arriving signal at the j-th destination node of the k-th layer.

then

$$y_{j,k} = f_F(u_{j,k}) \qquad \begin{cases} \text{For } k = 1, 2, \cdots K \\ \forall \;\; j \in J_k \end{cases}$$

where

$$u_{j,k} = \sum_{i \in I_k} C_{i,j,k}\, x_{i,j,k} \qquad \begin{cases} \text{For } k = 1, 2, \cdots K \\ \forall \;\; j \in J_k \end{cases}$$

It is noted that the above expression for $u_{j,k}$ can be generalised in terms of Kolmogorov-Gabor (KG) multinomials (Reference 10) of the form

$$u_{j,k} = C_{0,j,k} + \sum_{p \in I_k} C_{p,j,k}\, x_{p,j,k} +$$

$$\sum_{p \in I_k} \sum_{q \in I_k} C_{p,q,j,k}\, x_{p,j,k}\, x_{q,j,k} +$$

$$\sum_{p \in I_k} \sum_{q \in I_k} \sum_{r \in I_k} C_{p,q,r,j,k}\, x_{p,j,k}\, x_{q,j,k}\, x_{r,j,k} +$$

$$\bullet\bullet\bullet\bullet\bullet \qquad \begin{cases} \text{For } k = 1, 2, \cdots K \\ \forall \;\; j \in J_k \end{cases}$$

The input and node compatibility/interface constraints are applied at each layer boundary. Specifically

$$x_{i,j,k} \equiv \theta_i \qquad \begin{cases} \text{For } k = 1 \\ \forall \;\; i \in I_k \\ \forall \;\; j \in J_k \end{cases}$$

$$x_{i,j,k} \equiv y_{i,k-1} \qquad \begin{cases} \text{For } k = 2, 3, \cdots K \\ \forall \;\; i \in J_{k-1} \equiv I_k \\ \forall \;\; j \in J_k \end{cases}$$

where it is assumed that the common signal source constraints apply, that is all the signals exiting from a specific node are the same. Specifically

$$x_{i,j_1,k} \equiv x_{i,j_2,k} \qquad \begin{cases} \text{For } k = 1, 2, \cdots K \\ \forall \;\; i \in I_k \\ \forall \;\; j_1 \in J_k \\ \forall \;\; j_2 \in J_k \end{cases}$$

The output measurement vector (i.e., the Z-vector) is then defined

$$Z_j \equiv y_{j,k} \qquad \begin{cases} \text{For } k = K \\ \forall \;\; j \in J_k \end{cases}$$

The neural-network function $f_{N^2}(\bullet)$ for a specific control vector (i.e., the θ-vector) and attenuation coefficient matrix (i.e., the C-matrix) is defined

$$f_{N^2}(\theta,C) = f_{N^2}^{\,I_1 - I_2 - I_3 - \cdots\cdots - I_K - J_K}(\theta,C) = Z$$

2.2.2 Neural-Network Filter Functions/Pass-Through Functions

The filter/pass-through function $f_{\mathrm{F}}(u_{j,k})$ of the j-th-k-th argument $u_{j,k}$ is applied just prior to (i.e., immediately to the left of) the j-th destination node of the k-th layer and consequently defines the arriving signal $y_{j,k}$ at the j-th destination node of the k-th layer. Specifically

$$y_{j,k} = f_{\mathrm{F}}(u_{j,k}) \qquad \begin{cases} \text{For } k = 1, 2, \cdots K \\ \forall \;\; j \in J_k \end{cases}$$

where the argument $u_{j,k}$ is the summation of all the attenuated signals directed from the active origin nodes of the k-th layer toward the j-th destination node of the k-th layer, that is

$$u_{j,k} = \sum_{i \in I_k} \mathrm{C}_{i,j,k} \, x_{i,j,k} \qquad \begin{cases} \text{For } k = 1, 2, \cdots K \\ \forall \;\; j \in J_k \end{cases}$$

The filter/pass-through function attenuates the $u_{j,k}$ argument in accordance with a mathematical rule/function which is specified for each *(j, k)* tuple. In addition to the No-Pass Function (i.e., the Constant Function) and the Direct-Pass Function (i.e., the Linear Function), the commonly selected filter/pass-through functions are either of the signoid type or of the pulse type (e.g., a radial function, a bell shaped function, et cetera). If these functions are continuous and smooth, that is if they are connected with continuous derivatives, they have the forms which are illustrated in Figure 5. For this study, the Hyperbolic Tangent Function was selected to be the signoid type function, whilst its first derivative was selected to be the pulse type function. The motivation for this selection was to facilitate the analytic evaluation of the partial derivatives required during the optimisation iteration process which is used to update the neural-network parameters, and to provide function compatibility between the signoid and radial type functions. In addition, this selection appears to be suitable for the use of a regularisation method which uses partial derivatives of the error metric to define the regularisation functional that is added to the performance index during the neural-network parameter update process (References 11 through 18). These four types of filter/pass-through functions are described in the following sub-sections.

2.2.2.1 Constant Function: the No-Pass Function

The Constant Function (see Figure 6) is also referred to as the No-Pass Function because the output signal $y_{j,k}$ is specified by the function constant $C_{0_{j,k}}$ and is completely independent of the input signal $u_{j,k}$. For a specific *(j, k)* tuple, the Constant Function is

$$y_{j,k} - y_{0_{j,k}} = C_{0_{j,k}}$$

where

$C_{0_{j,k}}$ is the specified constant.

$y_{0_{j,k}}$ is the vertical translation constant.

It is noted that the node defined by the *(j, k)* tuple can be effectively eliminated by setting $C_{0_{j,k}}$ and $y_{0_{j,k}}$ equal to zero. $C_{0_{j,k}}$ can be thought of as a bias signal in the neural-network system.

2.2.2.2 Linear Function: the Direct-Pass Function

The Linear Function (see Figure 6) is also referred to as the Direct-Pass Function because the output signal $y_{j,k}$ can be made to be identically equal to the input signal $u_{j,k}$ by appropriately specifying the values of $A_{0_{j,k}}$, $C_{0_{j,k}}$, $u_{0_{j,k}}$, and $y_{0_{j,k}}$; specifically by setting $A_{0_{j,k}} = 1$, $C_{0_{j,k}} = 0$, $u_{0_{j,k}} = 0$, and $y_{0_{j,k}} = 0$. For a specific *(j, k)* tuple, the Linear Function is

$$y_{j,k} - y_{0_{j,k}} = A_{0_{j,k}}\left(u_{j,k} - u_{0_{j,k}}\right) + C_{0_{j,k}}$$

where

$A_{0_{j,k}}$	is the specified attenuation constant.
$C_{0_{j,k}}$	is a specified constant.
$u_{0_{j,k}}$	is the horizontal translation constant.
$y_{0_{j,k}}$	is the vertical translation constant.

It is noted that the node defined by this Linear Function can be made to degenerate to the Constant Function be setting $A_{0_{j,k}} = 0$.

If two points $P_1\left(u_{1_{j,k}}, y_{1_{j,k}}\right)$ and $P_2\left(u_{2_{j,k}}, y_{2_{j,k}}\right)$ are known to be contained in the mapping of the desired Linear Function, the constants $A_{0_{j,k}}$ and $C_{0_{j,k}}$ can be readily obtained from

$$A_{0_{j,k}} = \frac{y_{2_{j,k}} - y_{1_{j,k}}}{u_{2_{j,k}} - u_{1_{j,k}}}$$

and

$$C_{0_{j,k}} = y_{2_{j,k}} - y_{0_{j,k}} - A_{0_{j,k}}\left(u_{2_{j,k}} - u_{0_{j,k}}\right)$$

2.2.2.3 Hyperbolic Tangent: the Threshold Function

The Hyperbolic Tangent Function (see Figure 7) is also referred to as the Threshold Function because the output signal $y_{j,k}$ has a constant value (e.g., zero) or is as close as required to a horizontal asymptote for values of the input signal $u_{j,k}$ below a threshold limit. For values of the input signal $u_{j,k}$ above this threshold limit, the output signal $y_{j,k}$ "ramps" to another constant value or as close as required to another horizontal asymptote. For a specific *(j, k)* tuple, the Hyperbolic Tangent Function is

$$y_{j,k} - y_{0_{j,k}} = C_{0_{j,k}} Tanh\left[A_{0_{j,k}}\left(u_{j,k} - u_{0_{j,k}}\right)\right]$$

where

$A_{0_{j,k}}$	is the specified horizontal scaling constant.
$C_{0_{j,k}}$	is the specified attenuation constant.
$u_{0_{j,k}}$	is the horizontal translation constant.
$y_{0_{j,k}}$	is the vertical translation constant.

The horizontal scaling constant $A_{0_{j,k}}$ can be readily determined from geometrical considerations (see Figure 7). If it is desired to have the function pass through a specific point $P\left(b + u_{0_{j,k}},\ \alpha C_{0_{j,k}} + y_{0_{j,k}}\right)$ where $C_{0_{j,k}},\ b \in (0, +\infty)$. and

$\alpha \in (0, 1)$ are specified, and noting that the function passes through point $P_0\left(u_{0_{j,k}}, y_{0_{j,k}}\right)$, then

$$A_{0_{j,k}} = \frac{1}{2b} ln\left(\frac{1 + \alpha}{1 - \alpha}\right) \qquad A_{0_{j,k}} \in (0, +\infty)$$

2.2.2.4 First Derivative of the Hyperbolic Tangent: the Pulse Function

The First Derivative of the Hyperbolic Tangent Function (see Figure 8) is also referred to as the Pulse Function because its width can be made to be as narrow as required by the appropriate selection of the horizontal scaling constant $A_{0_{j,k}}$. For a specific *(j, k)* tuple, the First Derivative of the Hyperbolic Tangent Function defined in the previous subsection is

$$y_{j,k} - y_{0_{j,k}} = \frac{d}{du_{j,k}}\left\{C_{0_{j,k}} Tanh\left[A_{0_{j,k}}\left(u_{j,k} - u_{0_{j,k}}\right)\right]\right\}$$

$$y_{j,k} - y_{0_{j,k}} = A_{0_{j,k}} C_{0_{j,k}} Sech^2\left[A_{0_{j,k}}\left(u_{j,k} - u_{0_{j,k}}\right)\right]$$

where

$A_{0_{j,k}}$ is the specified horizontal scaling constant.

$C_{0_{j,k}}$ is the specified attenuation constant.

$u_{0_{j,k}}$ is the horizontal translation constant.

$y_{0_{j,k}}$ is the vertical translation constant.

The horizontal scaling constant $A_{0_{j,k}}$ can be readily determined from geometrical considerations (see Figure 8). If it is desired to have the function pass through a specific point $P\left(b + u_{0_{j,k}},\ \alpha A_{0_{j,k}} C_{0_{j,k}} + y_{0_{j,k}}\right)$ where $C_{0_{j,k}},\ b \in (0, +\infty)$. and $\alpha \in (0, 1)$ are specified, and noting that the function passes through point $P_0\left(u_{0_{j,k}},\ A_{0_{j,k}} C_{0_{j,k}} + y_{0_{j,k}}\right)$, then

$$A_{0_{j,k}} = \frac{1}{2b} \ln\left(\frac{2}{\sqrt{\alpha}} - 1\right) \qquad\qquad A_{0_{j,k}} \in (0, +\infty)$$

2.2.3 The Sliding Window of Data Acquisition

The purpose of the Closed-Loop Optimal Neural-Network Controller described herein is to optimally control the aeromechanical behaviour of a rotorcraft over a period of time. This behaviour history is the time process which is the "**trajectory of interest**". For convenience and efficiency, each trajectory segment (i.e., either the learning trajectory or the controlled trajectory) is compartmentalised into contiguous time intervals referred to as "**duty cycles**". These duty cycles are sequentially processed until the completion and termination of the current trajectory segment. The various tasks that are required to be processed during the current duty cycle are placed in a priority queue, initiated as appropriate, executed, and completed as time permits. The duration of the duty cycles is typically defined by a recurring physical event such as the start of a rotor revolution (i.e., after n rotor revolutions) or after a fixed time interval (i.e., after Δt_{DC} seconds). The acquisition and processing of the pertinent data required by the controller (i.e., the current measurement $Z-$ vector and the current control $\theta-$ vector) to determine/update the constants of the neural-network plant model and/or to determine the optimal control θ-vector are essential duty cycle tasks.

The "**sliding window of data acquisition**" as illustrated in Figure 9 is a convenient means to describe the initiation and accomplishment of the data acquisition and data processing tasks for the sequential duty cycles. The purpose of sliding window is to provide a means to include previously acquired data with the latest acquired data when determining/updating the constants of the neural-network plant model whilst culling out the older data. Data acquisition (i.e., transmission of the current measurement $Z-$ vector and the current control $\theta-$ vector to the first location in the sliding window) is tasked during the first duty cycle after a specified delay count (i.e., after a specified number of duty cycles) from the beginning of each trajectory segment . This specified delay count is referred to as the "**data acquisition delay**" for the current trajectory segment. Subsequent data acquisition is tasked at a specified duty cycle frequency (i.e., after a specified integral number of duty cycles). This specified duty cycle frequency is referred to as the "**data acquisition frequency**". It is consequently not necessary for data acquisition to occur during each duty cycle although this is possible and is indeed frequently the case.

The sliding window of data acquisition is comprised of the sequentially acquired data sets $\{Z_l - \text{vector and } \theta_l - \text{vector for } l = 1, 2, 3, \bullet\bullet\bullet\bullet \text{ LMAX}\}$ where LMAX is the current number of data sets in the sliding window. Whenever a new set of data is acquired, the positions of the previously acquired data sets in the sliding window are advanced by one (e.g., the Z_1- vector and θ_1- vector become the Z_2- vector and θ_2- vector, respectively; the Z_2- vector and θ_2- vector become the Z_3- vector and θ_3- vector, respectively; and so on until the positions of all the data sets in the sliding window have been advanced by one). The newly acquired data set becomes the new Z_1- vector and θ_1- vector. If the earliest data set in the window (i.e., the $Z_{LMAX}-$ vector and $\theta_{LMAX}-$ vector) is advanced to a position beyond the specified maximum sliding window size, it is eliminated from the sliding window. This specified maximum sliding window size is referred to as the "**window length**".

2.2.4 Optimal Update of the Neural-Network Model

There are two principal categories of optimisation procedures employed to optimally determine/update the neural-network plant model. The first category deals with the task to optimally select the constants of the neural-network plant model (i.e., the "**optimal constants selection process**") and to eliminate and/or add neural-network paths and/or nodes in this plant model. The second category deals with the tasking of data acquisition, the retention and weighting of this data for the optimal constants selection process, and the operation of the optimisation algorithm employed during this optimal constants selection process.

The determination/update of the constants of the neural-network plant model is accomplished using the modern constrained optimisation method described in References 2 through 8. This task is posed as a non-linear programming problem for which a performance index is minimised subject to constraints. In this case, the control vector is comprised of the attenuation coefficient elements $C_{i,j,k}$ of the attenuation coefficient matrix (i.e., the C-matrix) which are defined in Section 2.2.1. The optimisation process selects the values of $C_{i,j,k}$ which minimise a performance index based on the closeness of **predicted** measurement $Z-$vectors (i.e., the $Z-$vectors obtained using the neural-network plant model with the current values of the attenuation coefficient elements $C_{i,j,k}$) to the "**actual**" measurement $Z-$vectors (i.e., the $Z-$vectors obtained from the data sets in the sliding window). Provision has been made to weight the data sets in the sliding window according to position in the window as defined by the index l, for $l = 1, 2, 3, \bullet\bullet\bullet\bullet$ LMAX. The optimal constants selection process is the solution to the following optimisation problem.

$$\underset{C_{i,j,k} \in C}{\mathit{Minimise}} \quad J_{N^2} = \sum_{l=1}^{LMAX} W_{SW_l} \left[Z_{N^2_l} - Z_{A_l}\right]^T W_{N^2} \left[Z_{N^2_l} - Z_{A_l}\right]$$

Subject to:

$$C_{i,j,k} \in \left[C_{MIN_{i,j,k}}, C_{MAX_{i,j,k}}\right]$$

$$C_{MIN_{i,j,k}} \in (-\infty, +\infty)$$

$$C_{MAX_{i,j,k}} \in (-\infty, +\infty)$$

$$C_{MIN_{i,j,k}} \leq C_{MAX_{i,j,k}}$$

where

W_{N^2} is the diagonal weighting coefficient matrix for the quadratic difference term (i.e., the "square" of the difference between the predicted and the actual measurement Z - vectors) which is an element in the performance index J_{N^2} .

W_{SW_l} is the weighting coefficient for the l-th data set of the sliding window.

Z_{A_l} is the actual measurement Z - vector from the l-th data set of the sliding window.

$Z_{N^2_l}$ is the predicted measurement Z - vector from the l-th data set of the sliding window; $= f_{N^2_l}(\theta, C)$.

Although no automatic scheme for the elimination and/or addition of neural-network paths and/or nodes have been implemented as of this time, a general plan for such an automatic scheme has been identified; specifically :

$$\textit{If} \qquad \left\| \mathrm{C}_{i,j,k} \right\| < \varepsilon_{\mathrm{C}} \qquad \textit{or} \qquad \left\| \mathrm{C}_{i,j,k} \right\| \ll \left\| \mathrm{C} \right\|$$

where ε_{C} is a suitably selected small positive real number.

for a specific (i, j, k) tuple (i.e., for a specific i-th origin and j-th destination in a specific k-th layer), close the associated i, j, k path by setting $\mathrm{C}_{i,j,k} = 0$ and removing it from the optimisation control vector. This action has the advantage of reducing the dimension (i.e., the degrees-of-freedom) of the optimisation problem by one for each specific (i, j, k) tuple for which one of these conditions occurs. The reduction of dimension will hopefully enhance the efficiency of the optimisation process.

$$\textit{If} \qquad \left\| \mathrm{C}_{i,j,k} \right\| < \varepsilon_{\mathrm{C}} \qquad \textit{or} \qquad \left\| \mathrm{C}_{i,j,k} \right\| \ll \left\| \mathrm{C} \right\|$$

$\forall \; i \in I_k$ with the specific j-th destination in the specific k-th layer, eliminate the associated node defined by the (j, k) tuple. This is accomplished by closing the associated i, j, k paths to this node and all paths from this node as defined by the $(j, p, k+1)$ tuple $\forall \; p \in J_{k+1}$. Set the associated $\mathrm{C}_{i,j,k}$ and $\mathrm{C}_{j,p,k+1}$ values to zero and remove them from the optimisation control vector. Removal of a node reduces the dimension of the optimisation problem by the sum of the number of $i \in I_k$ and the number of $p \in J_{k+1}$.

$$\textit{If} \qquad \left\| \mathrm{C}_{i,j,k} \right\| > \zeta_{\mathrm{C}} \qquad \textit{or} \qquad \left\| \mathrm{C}_{i,j,k} \right\| \gg \left\| \mathrm{C} \right\|$$

where ζ_C is a suitably selected large positive real number.

for a significant number of $i \in I_k$ with the specific q-th destination for $q \in J_k$ in the specific k-th layer, the possibility exists that neural-network modelling performance can be enhanced by the addition of one or two nodes adjacent to the node defined by the (q, k) tuple.

Specifically, let

$\mathbf{N}_{I_k}$ = the number of paths from the origin nodes to the destination node which is defined by the (q, k) tuple (i.e., the number of $i \in I_k$) for $q \in J_k$ in the specific k-th layer.

$I_k^- = \left\{ i \,\middle|\, i \in \text{lower half of } i \in I_k \right\}$

where the median $i \in I_k$ is ignored when $\mathbf{N}_{I_k}$ is odd

$I_k^+ = \left\{ i \,\middle|\, i \in \text{upper half of } i \in I_k \right\}$

where the median $i \in I_k$ is ignored when $\mathbf{N}_{I_k}$ is odd

$\mathbf{N}_{I_k}^-$ = the number of $i \in I_k^-$ for which

$$\left\| \mathrm{C}_{i,q,k} \right\| > \zeta_C \qquad \text{or} \qquad \left\| \mathrm{C}_{i,q,k} \right\| \gg \left\| \mathrm{C} \right\|$$

$\mathbf{N}_{I_k}^+$ = the number of $i \in I_k^+$ for which

$$\left\| \mathrm{C}_{i,q,k} \right\| > \zeta_C \qquad \text{or} \qquad \left\| \mathrm{C}_{i,q,k} \right\| \gg \left\| \mathrm{C} \right\|$$

then

If $$\mathbf{N}_{I_k}^{-} \geq \text{Trunc}\left(\alpha \mathbf{N}_{I_k}\right)$$

where the $\text{Trunc}(\bullet)$ is the truncation function and α is a suitably selected positive real number $\in [0.0,\ 0.5]$,

then *add* a node adjacent to and "above" (i.e., before) the destination node which is defined by the *(q, k)* tuple by advancing the $j \in J_k$ indices by one for $j \geq q$ and adding the new node to the vacated *(q, k)* position. Paths to and from this new node must be appropriately added by defining the associated $\mathbf{C}_{i,q,k}$ and $\mathbf{C}_{q,p,k+1}$ values for $p \in J_{k+1}$ and $q \in J_k$. This has the effect of increasing the dimension of the optimisation problem by the sum of the numbers of $i \in I_k$ and $j \in J_k$ for each node added.

If $$\mathbf{N}_{I_k}^{+} \geq \text{Trunc}\left(\beta \mathbf{N}_{I_k}\right)$$

where β is a suitably selected positive real number $\in [0.0,\ 0.5]$,

then *add* a node adjacent to and "below" (i.e., after) the destination node which is defined by the *(q, k)* tuple by advancing the $j \in J_k$ indices by one for $j \geq q+1$ and adding the new node to the vacated $(q+1, k)$ position. Paths to and from this new node must be appropriately added by defining the associated $\mathbf{C}_{i,q+1,k}$ and $\mathbf{C}_{q+1,p,k+1}$ values for $p \in J_{k+1}$ and $q \in J_k$. This has the effect of increasing the dimension of the optimisation problem by the sum of the numbers of $i \in I_k$ and $j \in J_k$ for each node added.

It is felt that more experience using this controller should be obtained before attempting to define the details required for implementation of an automatic procedure such as the one described above, to modify the initial "geometry" of the neural-network plant model.

The tasking of data acquisition (i.e., the definition of "data acquisition delay" and "data acquisition frequency"), the retention and weighting of this data for the optimal constants selection process (i.e., the definition of "data window length" and the values of W_{SW_l} the weighting coefficients for the l-th data sets of the sliding window), and the operation of the optimisation algorithm employed during this optimal constants selection process (e.g., the selection of the convergence tolerance values and the maximum number of iterations in each optimisation solution process) is not amenable to the use of automated optimisation methods such as those employed during the optimal constants selection process. Although this problem can be posed as an integer programming problem, attempts at its solution at this time are accomplished by manually selecting the governing parameters based on the experience of operating the controller.

2.2.5 Control Optimisation

One of the important tasks which can be requested during a duty cycle is the optimal selection of the control θ – vector (i.e., the "**optimal control selection process**") to be used during the next duty cycle. There are two principal categories of optimisation procedures employed for this optimal control selection process. The first category deals with the optimisation of the elements of the control θ – vector, subject to constraints, which minimises a metric of selected elements of the measurement Z – vector. Although this optimal control selection process utilises the most recently determined neural-network plant model (i.e., neural-network plant model defined by the most recently determined neural-network plant model geometry and the associated attenuation coefficient elements $C_{i,j,k}$ as described in Section 2.2.4) to define the required elements of the measurement Z – vector, the sliding window of data acquisition (see Section 2.2.3) is not employed directly in this process; it is assumed that the plant model is already defined. The second category deals with the operation of the optimisation algorithm employed during this optimal control selection process.

As in the case of the optimal constants selection process described in Section 2.2.4, the selection of the optimal control θ – vector is accomplished using the modern constrained optimisation method described in References 2 through 8. This task is posed as a non-linear programming problem for which a performance index is minimised subject to constraints. In this case, the control vector is comprised of selected elements of the control θ – vector (see Section 2.2.1). The optimisation process selects the values of these elements of the control θ – vector which minimise a performance index defined as a metric of selected elements of the measurement $Z-$ vector. The optimal control selection process is the solution to the following optimisation problem.

$$\underset{\theta_p \in \theta}{Minimise} \quad J_{CV} = Z_{CV}^{T} \, W_{CV} \, Z_{CV} \qquad \text{for } p \in I_\theta$$

Subject to:

$$\theta_p \in \left[\theta_{MIN_p}, \theta_{MAX_p}\right] \qquad \text{for } p \in I_\theta$$

$$\theta_{MIN_p} \in (-\infty, +\infty) \qquad \text{for } p \in I_\theta$$

$$\theta_{MAX_p} \in (-\infty, +\infty) \qquad \text{for } p \in I_\theta$$

$$\theta_{MIN_p} \leq \theta_{MAX_p} \qquad \text{for } p \in I_\theta$$

$$\theta_p^2 + \theta_q^2 \leq \theta_{MAG_p}^2 \qquad \text{for } p, q \in I_\theta$$

where

I_θ is the set of all $p \ni \theta_p \in \theta$.

W_{CV} is the diagonal weighting coefficient matrix for the quadratic term (i.e., the "square" of the predicted $Z-$ vector) which is an element in the performance index J_{CV}.

Z_{CV} is the predicted measurement $Z-$ vector evaluated during the control $\theta-$ vector optimisation/update process. $Z_{CV} = f_{N^2}(\theta, C)$.

As in the case of the optimal constants selection process described in Section 2.2.4, the operation of the optimisation algorithm employed during this optimal control selection process (e.g., the selection of the convergence tolerance values and the maximum number of iterations in each optimisation solution process) and the frequency of tasking this process can be optimised. It is emphasised that in the real time trajectory environment, tasking the optimal control selection process during each duty cycle and/or requiring convergence of the optimisation process to within a small tolerance is not necessarily the "optimal" or "best" way to operate the optimisation algorithm. The frequency of tasking this optimal control selection process and the associated required amount of computation and processing (e.g., requiring convergence to within a small tolerance) within the duty cycles in which this process is tasked is indeed relevant to the overall trajectory optimisation and is amenable to optimisation. Although this problem can be posed as an integer programming problem, attempts at its solution at this time are accomplished by manually selecting the governing parameters based on the experience of operating the controller.

2.3 The Optimal Constants and Optimal Control Selection Processes as Non-linear Programming Problems

The problems which are addressed in both the **optimal constants selection process** described in Section 2.2.4 and the **optimal control selection process** described in Section 2.2.5 are special cases of the general Non-linear Programming (NLP) Problem. The selection processes for both of these cases seek the optimal control vector which minimises a performance index subject to constraints on the control vector. The performance index is in general non-linear. Although the constraints on the control vector are constant limiting values for the optimal constants selection process as of the date of this report, they can also be non-linear if required. Provision has been made for quadratic constraints (e.g., harmonic magnitude constraints) on the control vector for the optimal control selection process to be applied as required. These selection processes thus require an optimisation technique which treats a more difficult non-linear problem than the relatively simple quadratic programming problem.

The general non-linear programming (NLP) problem is defined in Section 2.3.1, and the method of its solution which is employed in this research is described in Section 2.3.2.

2.3.1 The General Non-linear Programming Problem

The general non-linear programming (NLP) problem can be expressed in the form

$$\underset{\theta_p \in \theta}{Minimise} \qquad J = g[Z(\theta)] \qquad \text{for } p \in I_\theta$$

Subject to:

$$\phi(\theta) = 0$$

$$\psi(\theta) \geq 0$$

where

$g[Z(\theta)]$ is the scalar performance index which is a function of the plant output measurement vector (i.e., the $Z-$vector). In general, this function can be non-linear.

I_θ is the set of all $p \ni \theta_p \in \theta$.

$Z(\theta)$ is the predicted measurement $Z-$vector evaluated during the optimisation process. $Z = f_{N^2}(\theta, C)$.

θ is the control vector $\theta-$vector.

$\phi(\theta)$ is the equality constraint vector function which in general can be dependent on the $\theta-$vector.

$\psi(\theta)$ is the inequality constraint vector function which in general can be dependent on the $\theta-$vector.

2.3.2 A Solution to the General Non-linear Programming Problem

Investigation of various methods to solve the General Non-linear Programming (NLP) problem led to the selection (Reference 1) of the highly successful modern methods of Schittkowski, Powell, Stoer, and Gill et al (References 2 through 8). These general NLP solution methods were coded in FORTRAN and are readily available as IMSL library routines (specifically, IMSL main driver routines DNCONF and DNCONG described in Reference 9). These methods solve the general NLP problem by solving a sequence of related quadratic programming sub-problems (QPSs) until either convergence is obtained or the specified maximum number of iterations (i.e., the specified maximum number of quadratic programming problems to be solved) is reached. One important advantage of this technique is that quadratic programming problems can be solved efficiently. A very important property of quadratic programming formulations is that if the quadratic coefficient matrix in the performance index is positive definite, the problem has a unique solution which is, of course, the global solution. These methods worked quite well in the research described in Reference 1, and have proven to be quite robust and efficient in the research described herein.

The general quadratic programming problem (QPP) can be expressed in the form

$$\underset{\theta_p \in \theta}{Minimise} \qquad J = g(\theta) = \theta^{\mathrm{T}} C_{\mathrm{Q}} \theta + C_{\mathrm{L}} \theta \qquad \text{for } p \in I_\theta$$

Subject to:

$$\phi(\theta) = A_\phi \theta + B_\phi = 0_\phi$$

$$\psi(\theta) = A_\psi \theta + B_\psi \geq 0_\psi$$

where

A_ϕ is the coefficient matrix in the linear term of the linear equality constraint function $\phi(\theta)$.

A_ψ is the coefficient matrix in the linear term of the linear inequality constraint function $\psi(\theta)$.

B_ϕ is the constant vector term of the linear equality constraint function $\phi(\theta)$.

B_ψ is the constant vector term of the linear inequality constraint function $\psi(\theta)$.

C_L is the coefficient matrix in the linear term of the quadratic performance index function $g(\theta)$.

C_Q is the coefficient matrix in the quadratic term of the quadratic performance index function $g(\theta)$.

$g(\theta)$ is the scalar performance index which in this case is a quadratic function of the control vector θ – vector.

I_θ is the set of all $p \ni \theta_p \in \theta$.

O_ϕ is the right hand side null or zero vector of the linear equality constraint function $\phi(\theta)$.

O_ψ is the right hand side null or zero vector of the linear inequality constraint function $\psi(\theta)$.

θ is the control vector θ – vector.

$\phi(\theta)$ is the equality constraint vector function which in this case is a linear function of the control vector θ – vector.

$\psi(\theta)$ is the inequality constraint vector function which in this case is a linear function of the control vector θ – vector.

The successive quadratic programming sub-problems (QPSs) used to solve the general non-linear programming (NLP) problem are formulated by using a quadratic approximation of the general NLP performance index function $g(\theta)$ and linear approximations of the general NLP equality and inequality constraint functions $\phi(\theta)$ and $\psi(\theta)$. These approximations are obtained by simple replacement of the $g(\theta)$, $\phi(\theta)$, and $\psi(\theta)$ functions with their appropriately truncated matrix Taylor Series expansions, where if the Hessian of $g(\theta)$ $\left(\text{i.e., } \frac{\partial^2 g(\theta)}{\partial \theta^2}\right)$ is not positive definite, the algorithm adjusts it so that it is so that global optimality of the QPS is assured. Specifically, at each iteration step the quadratic programming sub-problem (QPS) to be solved is:

$$\underset{\theta_p \in \theta}{Minimise} \quad J = \frac{1}{2}\left[\theta - \theta_0\right]^{\mathrm{T}} C_{\mathrm{Q}} \left[\theta - \theta_0\right] + C_{\mathrm{L}}\left[\theta - \theta_0\right] \qquad \text{for } p \in I_\theta$$

Subject to:

$$\phi(\theta) \approx A_\phi\left[\theta - \theta_0\right] + B_\phi = O_\phi$$

$$\psi(\theta) \approx A_\psi\left[\theta - \theta_0\right] + B_\psi \geq O_\psi$$

where

$$A_\phi = \left.\frac{\partial\phi(\theta)}{\partial\theta}\right|_{\theta=\theta_0} \quad \text{and} \quad B_\phi = \phi(\theta)\big|_{\theta=\theta_0}$$

$$A_\psi = \left.\frac{\partial\psi(\theta)}{\partial\theta}\right|_{\theta=\theta_0} \quad \text{and} \quad B_\psi = \psi(\theta)\big|_{\theta=\theta_0}$$

$$C_Q = \left.\frac{\partial^2 g(\theta)}{\partial\theta^2}\right|_{\theta=\theta_0} \quad \text{and} \quad C_L = \left.\frac{\partial g(\theta)}{\partial\theta}\right|_{\theta=\theta_0}$$

$$g(\theta) \approx \frac{1}{2}\left[\theta-\theta_0\right]^T C_Q \left[\theta-\theta_0\right] + C_L\left[\theta-\theta_0\right]$$

I_θ is the set of all $p \ni \theta_p \in \theta$.

0_ϕ is the right hand side null or zero vector of the linear equality constraint function $\phi(\theta)$.

0_ψ is the right hand side null or zero vector of the linear inequality constraint function $\psi(\theta)$.

θ is the control vector $\theta-$vector.

θ_0 is the value of the control vector $\theta-$vector at the start of each quadratic programming sub-problem (QPS).

If optimality as measured by satisfying the Kuhn-Tucker optimality criterion at the completion of an iteration step and if the specified maximum number of iterations has not been reached, the Hessian is updated (References 3 and 4), θ_0 is set equal to the last value of θ, and a new iteration is attempted.

2.4 Trajectory Data

The time history of the rotorcraft behaviour of interest, that is the **"trajectory of interest"** (see Section 2.2.3), is the source of the data that is acquired and/or defined during the specified duty cycles. Provisions in the Optimal Neural-Network Controller (ONNC) System (i.e., the code which was developed to implement the Closed-Loop Neural-Network Controller described herein) were made to optionally accept one of four forms of this data. These optional data forms are: 1) On-Line Trajectory Test Data (described in Section 2.4.1), 2) Off-Line Trajectory Data Tables (described in Section 2.4.2), 3) Analytic Trajectory Synthesis (described is Section 2.4.3), and 4) User Supplied Trajectory Model (described in Section 2.4.4).

2.4.1 On-Line Trajectory Data

A position/slot in the Optimal Neural-Network Controller (ONNC) System was provided to accept data sets in real time from an ongoing test. To activate this option, the DSTATE subroutine must be specifically designed and then coded to satisfy the requirements of test at hand. In general, this DSTATE subroutine will include the basic features of the Off-Line Trajectory Data Tables TSTATE subroutine described in Section 2.4.2, however it will additionally need to be formatted to accept the data sets transmitted from the ongoing test. This DSTATE routine will also need to be compatible with the ONNC System which reads one data set at a time commensurate with real time duty cycle methodology.

2.4.2 Off-Line Trajectory Data Tables

The TSTATE subroutine in the Optimal Neural-Network Controller (ONNC) System was provided to read off-line trajectory data sets from input tables one data set at a time commensurate with real time duty cycle methodology.

2.4.3 Analytic Trajectory Synthesis

The ASTATE subroutine in the Optimal Neural-Network Controller (ONNC) System was provided to analytically synthesise off-line trajectory data sets one data set at a time commensurate with real time duty cycle methodology. Whenever trajectory data is to be acquired, the analytic vector synthesis function $\Xi(t)$ is evaluated. Specifically:

let
$$\xi(t) = \begin{bmatrix} \theta_s(t) \\ Z_s(t) \end{bmatrix}$$

then
$$\Xi(t) \equiv \xi(t) = \begin{bmatrix} \theta_s(t) \\ Z_s(t) \end{bmatrix}$$

where t is the current time.

$Z_s(t)$ is the **synthesised** control Z-vector at time t with dimension $(N \times 1)$.

$\theta_s(t)$ is the **synthesised** control θ-vector at time t with dimension $(M \times 1)$.

$\xi(t)$ is the **synthesised** combined trajectory data vector with dimension $([\mathrm{M}+\mathrm{N}] \times 1)$.

Each element $\xi_i(t)$ of $\xi(t)$ is defined by

$$\xi_i(t) = \left[A_{3_i} + B_{3_i}\,\mathrm{Uran}\left(\mathrm{ISEED}_{3_i}\right)\right] + \left[C_{3_i} + D_{3_i}\,\mathrm{Uran}\left(\mathrm{JSEED}_{3_i}\right)\right]\mathrm{H}_i(t)$$

$$\forall \quad i \in [1,\ (\mathrm{M}+\mathrm{N})]$$

where A_{3_i}, B_{3_i}, C_{3_i}, and D_{3_i} are input constants..

$\mathrm{H}_i(t)$ is the composite synthesis function for the i-th element of $\xi(t)$.

ISEED_{3_i} and JSEED_{3_i} are input seeds for the VAX FORTRAN uniformly distributed random number generator function $\mathrm{RAN}(\bullet)$ for the i-th element of $\xi(t)$. Although there are no restrictions on the value of this seed other than it is an INTEGER*4 variable, the best results are obtained when it is initially input as a large odd integer.

$\mathrm{RAN}(\bullet)$ is the VAX FORTRAN uniformly distributed random number generator function described in Appendix D of Reference 19. $\mathrm{RAN}(\bullet) \in [0.0,\ 1.0]$

$\mathrm{Uran}(\bullet)$ is the uniformly distributed random number generator function $\ni$ $\mathrm{Uran}(\bullet) \in [-1.0,\ 1.0]$. $\mathrm{Uran}(\bullet)$ is defined by

$$\mathrm{Uran}(\bullet) \equiv 2 * \mathrm{RAN}(\bullet) - 1.0$$

It is noted that the first term $\left[A_{3_i} + B_{3_i}\,\text{Uran}\left(\text{ISEED}_{3_i}\right)\right]$ in the equation defining the synthesised combined trajectory data vector $\xi(t)$ represents a bias in the composite synthesis function $\text{H}_i(t)$ whilst the second term $\left[C_{3_i} + D_{3_i}\,\text{Uran}\left(\text{JSEED}_{3_i}\right)\right]\text{H}_i(t)$ in this equation represents the statistical uncertainty in this function.

The composite synthesis function $\text{H}_i(t)$ for the i-th element of $\xi(t)$ is the summation of up to seven individual modelling functions $h_m\left(\tau_m\right)$ where $m = 1, 2, 3, \bullet\bullet\bullet\bullet$ MMAX. Specifically:

$$\text{H}_i(t) = \sum_{m=1}^{\text{MMAX}} h_m\left(\tau_m\right) \qquad \text{for MMAX} \in [1, 7]$$

Eight different individual modelling functions $h_m\left(\tau_m\right)$ are currently provided in the ONNC System. These functions are described in the following sub-sections (i.e., Sections 2.4.3.1 through 2.4.3.8). With the exception of the Uniformly Distributed Random Function described in Section 2.4.3.8, the individual modelling functions $h_m\left(\tau_m\right)$ can include a random bias and/or a statistical uncertainty. Specifically:

$$\begin{aligned} h_m\left(\tau_m\right) = {} & \left[A_{2_m} + B_{2_m}\,\text{Uran}\left(\text{ISEED}_{2_m}\right)\right] \\ & + \left[C_{2_m} + D_{2_m}\,\text{Uran}\left(\text{JSEED}_{2_m}\right)\right] g_m\left(\tau_m\right) \end{aligned}$$

$$\forall \; m \in [1, \text{MMAX}]$$

where A_{2_m}, B_{2_m}, C_{2_m}, and D_{2_m} are input constants..

$g_m\left(\tau_m\right)$ is the core deterministic modelling function of the m-th specified individual modelling function $h_m\left(\tau_m\right)$.

$\mathrm{ISEED2}_m$ and $\mathrm{JSEED2}_m$ are input seeds for the VAX FORTRAN uniformly distributed random number generator function $\mathrm{RAN}(\bullet)$ for the m-th specified individual modelling function $h_m(\tau_m)$. Although there are no restrictions on the value of this seed other than it is an INTEGER*4 variable, the best results are obtained when it is initially input as a large odd integer.

τ_m is the periodic time argument for the m-th specified individual modelling function $h_m(\tau_m)$.

It is noted that the first term $\left[A_{2_m} + B_{2_m}\mathrm{Uran}\left(\mathrm{ISEED2}_m\right)\right]$ in the equation defining the m-th specified individual modelling function $h_m(\tau_m)$ represents a bias in this function whilst the second term $\left[C_{2_m} + D_{2_m}\mathrm{Uran}\left(\mathrm{JSEED2}_m\right)\right] g_m(\tau_m)$ in this equation represents the statistical uncertainty in this function.

Periodicity with phase shift relative to an epoch time for the core deterministic modelling function $g_m(\tau_m)$ of the m-th specified individual modelling function $h_m(\tau_m)$ is accomplished by specifying the period T_m time, phase shift t_{ϕ_m} time, and the epoch t_{ϕ_m} time (see Figures 10 and 11). Specifically, the periodic time argument τ_m for $g_m(\tau_m)$ and $h_m(\tau_m)$ is

$$\tau_m \equiv \mathrm{DMOD}\left(\left[t - t_{0_m} - t_{\phi_m}\right], \mathrm{T}_m\right)$$

where $\mathrm{DMOD}(\bullet)$ is the VAX/VMS FORTRAN Intrinsic **Remainder** Function described in Group 3 of Appendix B of this document and in Appendix D of Reference 19.

t is the current time.

t_{ϕ_m} is the phase shift time for the m-th specified individual modelling function $h_m(\tau_m)$.

t_{0_m} is the reference/epoch time for the m-th specified individual modelling function $h_m(\tau_m)$.

T_m is the period time for the m-th specified individual modelling function $h_m(\tau_m)$.

2.4.3.1 Linear/Ramp Function

The Linear/Ramp Function (see Figure 12) is expressed by

$$g_m(\tau_m) \equiv y - y_0 = a\,\tau_m + c$$

where

a is the specified attenuation constant (i.e., the slope).

c is a specified constant (i.e., the intercept).

y is the value of the m-th core deterministic modelling function $g_m(\tau_m)$ plus y_0.

y_0 is the vertical translation constant.

τ_m is the periodic time argument for the m-th core deterministic modelling function $g_m(\tau_m)$.

If two points $P_1\left(\tau_{m_1},\ y_1\right)$ and $P_2\left(\tau_{m_2},\ y_2\right)$ are known to be contained in the mapping of the desired Linear/Ramp Function, the constants a and c can be readily obtained from

$$a = \frac{y_2 - y_1}{\tau_{m_2} - \tau_{m_1}}$$

and

$$c = y_2 - y_0 - a\,\tau_{m_2}$$

2.4.3.2 Serpentine Curve Function

The Serpentine Curve Function (see Figure 13) is expressed by

$$g_m\left(\tau_m\right) \equiv y - y_0 = \frac{ab\,\tau_m}{a^2 + \tau_m^2}$$

where

a is the specified horizontal scaling constant.

b is the specified amplitude constant.

y is the value of the m-th core deterministic modelling function $g_m\left(\tau_m\right)$ plus y_0.

y_0 is the vertical translation constant.

τ_m is the periodic time argument for the m-th core deterministic modelling function $g_m\left(\tau_m\right)$.

Scaling of this Serpentine Curve Function is readily accomplished by noting the geometrical property that $g_m(\tau_m)$ obtains its maximum and minimum values $\pm b/2$ when

$$\frac{d}{d\tau_m}\left[g_m(\tau_m)\right] = 0 \qquad \text{which occurs when} \qquad \tau_m = \pm a$$

2.4.3.3 Witch of Agnesi Function

The Witch of Agnesi Function (see Figure 14) is expressed by

$$g_m(\tau_m) \equiv y - y_0 = \frac{a^3}{a^2 + b^2\tau_m^2}$$

where

a	is the specified amplitude constant.
b	is a derived horizontal scaling constant.
y	is the value of the m-th core deterministic modelling function $g_m(\tau_m)$ plus y_0.
y_0	is the vertical translation constant.
τ_m	is the periodic time argument for the m-th core deterministic modelling function $g_m(\tau_m)$.

Scaling of this Witch of Agnesi Function is readily accomplished by noting the geometrical properties that $g_m(\tau_m)$ obtains its maximum value a when

$$\frac{d}{d\tau_m}\left[g_m\left(\tau_m\right)\right] = 0 \qquad \text{which occurs when} \qquad \tau_m = 0$$

and that $$g_m\left(\tau_m\right) \rightarrow 0 \quad \text{as} \quad \tau_m \rightarrow \pm\infty$$

and by appropriately specifying a scaling coefficient $c \ni c \in (0,\ 1.0)$ so that

$$y - y_0 = ca \qquad \text{when} \qquad \tau_m = \pm a$$

The derived horizontal scaling constant b then becomes

$$b = \pm\sqrt{\frac{1 - c}{c}}$$

2.4.3.4 Inverted Witch of Agnesi Function

The Inverted Witch of Agnesi Function (see Figure 15) is expressed by

$$g_m\left(\tau_m\right) \equiv y - y_0 = a - \frac{a^3}{a^2 + b^2\tau_m^2}$$

where

- a is the specified amplitude constant.
- b is a derived horizontal scaling constant.
- y is the value of the m-th core deterministic modelling function $g_m\left(\tau_m\right)$ plus y_0.
- y_0 is the vertical translation constant.
- τ_m is the periodic time argument for the m-th core deterministic modelling function $g_m\left(\tau_m\right)$.

Scaling of this Inverted Witch of Agnesi Function is readily accomplished by noting the geometrical properties that $g_m\left(\tau_m\right)$ obtains its minimum value 0 when

$$\frac{d}{d\tau_m}\left[g_m\left(\tau_m\right)\right] = 0 \qquad \text{which occurs when} \qquad \tau_m = 0$$

and that

$$g_m\left(\tau_m\right) \rightarrow a \qquad \text{as} \quad \tau_m \rightarrow \pm\infty$$

and by appropriately specifying a scaling coefficient $c \ni c \in (0,\ 1.0)$ so that

$$y - y_0 = ca \qquad \text{when} \qquad \tau_m = \pm a$$

The derived horizontal scaling constant b then becomes

$$b = \pm\sqrt{\frac{c}{1-c}}$$

2.4.3.5 Enveloped Sinusoidal Function

The Enveloped Sinusoidal Function (see Figure 16) is expressed by

$$g_m\left(\tau_m\right) \equiv y - y_0 = C\,Exp_e\left[\alpha\left(\tau_m - \psi\right)\right]Cos\left[n\,\omega\left(\tau_m - \phi\right)\right]$$

where

C is the specified attenuation constant of the above equation.

f is the fundamental of primary frequency of the sinusoidal factor of the above equation.

n is harmonic frequency number of the sinusoidal factor of the above equation.

nf is the net frequency (i.e., the harmonic frequency number n times the fundamental of primary frequency f) of the sinusoidal factor of the above equation.

$n\omega$ is 2π times the net frequency nf of the sinusoidal factor of the above equation.

α is a derived horizontal scaling constant for the exponential factor of the above equation.

ϕ is a phase time constant of the sinusoidal factor of the above equation.

ψ is a derived horizontal shift constant for the exponential factor of the above equation.

ω is 2π times the fundamental of primary frequency f of the sinusoidal factor of the above equation.

y is the value of the m-th core deterministic modelling function $g_m(\tau_m)$ plus y_0.

y_0 is the vertical translation constant.

τ_m is the periodic time argument for the m-th core deterministic modelling function $g_m(\tau_m)$.

Scaling of this Enveloped Sinusoidal Function can be accomplished by directly specifying values of C, n, α, ϕ, ψ, and ω or from consideration of geometrical properties. The exponential envelope factor $C\,Exp_e\left[\alpha\left(\tau_m - \psi\right)\right]$ of the above equation can be thought of as the coefficient of the oscillatory factor $Cos\left[n\omega\left(\tau_m - \phi\right)\right]$ of this equation. The overall rate of convergence or divergence can readily be defined by specifying a required value of the exponential envelope factor at a selected value τ_m^* of τ_m; specifically

specify $\quad B \equiv C\,Exp_e\left[\alpha\left(\tau_m^* - \psi\right)\right] \quad$ for the selected value τ_m^*

Noting that $\quad C\,Exp_e\left[\alpha\left(\tau_m - \psi\right)\right] \equiv C \quad$ at $\quad \tau_m = \psi$

let $$A = \tau_m^* - \psi$$

then $$\alpha = \frac{1}{A} ln\left(\left|\frac{B}{C}\right|\right)$$

Note that divergence of the exponential envelope factor occurs when $B > C$, the exponential envelope factor is invariant when $B \equiv C$, and convergence of the exponential envelope factor occurs when $B < C$.

Either harmonic frequency number n together with 2π times the fundamental frequency f (i.e., ω) of the of the sinusoidal factor can be directly specified, or the required value of $n\omega$ can be derived from a specified net period P; specifically

$$n\omega = \frac{2\pi}{\mathrm{P}}$$

2.4.3.6 Hyperbolic Tangent: the Threshold Function

The Hyperbolic Tangent Threshold Function (see Figure 17) is expressed by

$$g_m\left(\tau_m\right) \equiv y - y_0 = C\,Tanh\left(A\tau_m\right)$$

where

A is a derived horizontal scaling constant. $A \in (0.0, +\infty)$

C is the specified attenuation constant. $C \in (0.0, +\infty)$

y is the value of the m-th core deterministic modelling function $g_m(\tau_m)$ plus y_0.

y_0 is the vertical translation constant.

τ_m is the periodic time argument for the m-th core deterministic modelling function $g_m(\tau_m)$.

Scaling of this Hyperbolic Tangent Threshold Function $g_m(\tau_m)$ is readily accomplished by noting the geometrical property that

$$g_m(\tau_m) \rightarrow \pm C \qquad \text{as} \qquad \tau_m \rightarrow \pm\infty$$

and by defining a required value αC of this function for a specified value $b \in (0.0, +\infty)$ of τ_m. Specifically

$$y - y_0 = g_m(\tau_m)\Big|_{\tau_m = b} \equiv \alpha C \qquad \text{for} \quad \alpha \in (0.0, 1.0)$$

then

$$A = \frac{1}{2b} \, ln\left(\frac{1 + \alpha}{1 - \alpha}\right)$$

2.4.3.7 First Derivative of the Hyperbolic Tangent: the Pulse Function

The First Derivative of the Hyperbolic Tangent Pulse Function (see Figure 18) is expressed by

$$g_m(\tau_m) \equiv y - y_0 = \frac{d}{d\tau_m}\{C\,Tanh(A\tau_m)\} = AC\,Sech^2(A\tau_m)$$

where

A is a derived horizontal scaling constant. $A \in (0.0,\ +\infty)$

C is the attenuation constant. $C \in (0.0,\ +\infty)$

y is the value of the m-th core deterministic modelling function $g_m(\tau_m)$ plus y_0.

y_0 is the vertical translation constant.

τ_m is the periodic time argument for the m-th core deterministic modelling function $g_m(\tau_m)$.

Scaling of the First Derivative of the Hyperbolic Tangent Pulse Function $g_m(\tau_m)$ is readily accomplished by noting the geometrical properties that $g_m(\tau_m)$ obtains its maximum value AC when

$$\frac{d}{d\tau_m}\left[g_m(\tau_m)\right] = 0 \qquad \text{which occurs when} \qquad \tau_m = 0$$

and that $g_m(\tau_m) \rightarrow 0$ as $\tau_m \rightarrow \pm\infty$

and by defining a required value αAC of this function for a specified value $b \in (0.0,\ +\infty)$ of τ_m. Specifically

$$y - y_0 = g_m(\tau_m)\Big|_{\tau_m = b} \equiv \alpha AC \qquad \text{for} \quad \alpha \in (0.0,\ 1.0)$$

then

$$A = \frac{1}{2b}\,ln\left(\frac{2}{\sqrt{\alpha}} - 1\right) \qquad \text{and} \qquad C = \frac{AC}{A}$$

2.4.3.8 Uniformly Distributed Random Function

The Uniformly Distributed Random Function (see Figure 19) is expressed by

$$h_m(\tau_m) \equiv y - y_0 = \left[A_{1_m} + B_{1_m} \mathrm{Uran}\left(\mathrm{ISEED}_{1_m}\right)\right]$$

where A_{1_m} and B_{1_m} are input constants.

$h_m(\tau_m)$ is the m-th specified individual modelling function $h_m(\tau_m)$ rather than a core deterministic modelling function $g_m(\tau_m)$ such as those defined in the above sub-paragraphs.

ISEED_{1_m} is the input seed for the VAX FORTRAN uniformly distributed random number generator function $\mathrm{RAN}(\bullet)$ for this m-th specified individual modelling function $h_m(\tau_m)$. Although there are no restrictions on the value of this seed other than it is an INTEGER*4 variable, the best results are obtained when it is initially input as a large odd integer.

τ_m is the periodic time argument for the m-th specified individual modelling function $h_m(\tau_m)$.

2.4.4 User Supplied Trajectory Model

A position/slot in the Optimal Neural-Network Controller (ONNC) System was provided to allow the user to specifically define the trajectory data sets by designing and coding the USTATE subroutine which will satisfy the user's requirements. This USTATE routine will, however, need to be compatible with the ONNC System which reads one data set at a time commensurate with real time duty cycle methodology.

2.5 The Stand-Alone Optimal Neural-Network Controller System

The Optimal Neural-Network Controller (ONNC) System was designed and developed to enable and facilitate accomplishment of the research described herein. The ONNC System is the means to execute the concepts described in the preceding sections. This system, which was originally coded in FORTRAN for a Digital Equipment Corporation (DEC) VAX/VMS system, currently resides/operates on a Compaq-DEC Alpha 4100 Model Processor.

The general hierarchy showing the principal routines of the ONNC System is illustrated in Figure 20. The Input and other important parameters of this system are defined in Appendix A. The principal routines of the ONNC System are described in Appendix B and their listings are presented in Appendix C.

2.6 Results

During the course of development and debug of the Optimal Neural-Network Controller (ONNC) System, several neural-network models which differed in the number of layers, number of nodes per specific layer, and the values of the constants in the associated specific neural-network filter functions were examined. Additionally, trajectory data was defined from both tabular test data and analytic trajectory synthesis. Variations in **data acquisition frequency** and **window length** for the **sliding window of data acquisition** were also considered.

Two principal categories of cases were selected to be used to study this dynamic optimal neural-network controller process in detail. Simplified static test data from a 40 x 80 Foot Wind Tunnel test performed for the BO-105 Individual Blade Control (IBC) test programme (Reference 20) was used to define the neural-network plant model constants in the first category of cases. A dynamic state propagation based on an analytically synthesised trajectory (i.e., a synthesised time history of the control θ-vector and the measurement Z-vector) was used to define the neural-network plant model constants from which the control θ-vector was optimised in the second category of cases.

The data for the first case (Reference 20) consisted of a table of values of the scalar vibration metric at 30 degree increments of the "two-per-rev" phase angle starting at 0 degrees and ending at 360 degrees (i.e., at $0°, 30°, 60°, \bullet\bullet\bullet\bullet, 360°$). A $1-12-4-1$ neural-network function $f_{N^2}^{1-12-4-1}(\bullet)$ was initially selected to define a neural-network plant model which would represent this relationship. Threshold functions (i.e., Hyperbolic Tangents) were selected for the filter functions at the destination nodes of the first and second neural-network layers, whilst a direct-pass function (i.e., a linear function) was selected for the output node (i.e., the destination node of the third neural-network layer). When convergence of the optimal constants selection process (see Section 2.2.4) for this $1-12-4-1$ neural-network model was not obtained, additional destination nodes were added to the first and second neural-network layers. After several nodal schemes were tried, a $1-14-5-1$ neural-network function $f_{N^2}^{1-14-5-1}(\bullet)$ was finally selected (see Figure 21) for the plant model. The motivation for this geometry was to provide filter functions at the ends and between the input tabular "two-per-rev" phase angles (i.e., at -15°, 15°, 45°, $\bullet\bullet\bullet\bullet$, 375°) at the destination nodes of the first layer. An additional destination node for the second layer was provided to handle the additional signals resulting from the increased destination nodes of the first layer. Convergence of the optimal constants selection process for this case was slow and not good. It appears as if the values of the vibration metric computed from the solution neural-network plant model approach either the upper or the lower table values, and that convergence scatters about multiple solutions to this optimal constants selection process. This is due to the fact that the optimal constants selection process as defined in Section 2.2.4 is in actuality an ill-posed problem with multiple solutions. An abbreviated listing of this first case is presented in Appendix D.

The data for the second case was generated using a synthesised trajectory. A (3×1) control θ-vector and a (4×1) measurement Z-vector were assumed. Several neural-network geometrical structures were considered before selecting the $3-8-5-4$ neural-network function $f_{N^2}^{3-8-5-4}(\bullet)$ (see Figure 22) for the plant model. Convergence of the optimal constants selection process for this case was also slow and not too good. It appears as if the values of the measurement Z-vector computed from the solution neural-network plant model approach either the upper or the lower synthesised values, and that convergence scatters about multiple solutions to this optimal constants selection process. As in the first case this is due to the fact that the optimal constants selection process as defined in Section 2.2.4 is in actuality an ill-posed problem with multiple solutions. An abbreviated listing of this first case is presented in Appendix E.

Although the ONNC System operated as planned and designed, and in particular, the optimisation algorithm proved to be quite robust and reliable for this application, convergence of the optimal constants selection process was slow and not too good. This is certainly not catastrophic however. The research of A. J. Meade, Jr. et al (References 11 through 18) points to a solution to this problem; specifically, addition of a regularising functional to the performance index J_{N^2} of the optimal constants selection process. In addition, it is noted and emphasised that even though the optimal constants selection process is not fully converged, the optimal control selection process (see Section 2.2.5) can still provide a control θ-vector solution which is better than that obtained by conventional methods. This will occur when the solution neural-network plant model is a better representation of the actual plant than the conventional model which is usually linear. Indeed as was pointed out at the end of Section 2.2.4, the operation of the optimisation algorithm itself (i.e., the selection of the convergence tolerance values and the maximum number of iterations in each optimisation process) can be optimised in the context of the dynamic data gathering - control optimisation process. It is emphasised that it is not necessarily necessary to converge fully to a solution for the neural-network plant model constants in order to make this procedure attractive.

3.0 CONCLUSIONS and RECOMMENDATIONS

The Optimal Neural-Network Controller (ONNC) System which was developed as part of this research, operated as planned and designed. Although the sliding window of data acquisition and the control θ-vector optimisation worked well, the update of the neural-network plant model by means of the optimal constants selection process was in general, slow to converge and/or converged to multiple solutions for the neural-network constants. This is due to the fact that the optimal constants selection process as defined in Section 2.2.4 is in actuality an ill-posed problem with multiple solutions. Fortunately as noted in Section 2.6, this is not necessarily catastrophic since the primary objective of this process is to determine a nearly optimal control θ-vector regardless of the state of refinement of the plant model which is merely a means to that end.

In addition to the general need to examine a greater diversity of rotorcraft cases and to experiment with the types of the neural-network filter functions and the values of their associated constants, three principal areas of improvement and development of the optimal constants selection process have been identified; these are:

1. Implement a regularisation method in the optimal constants selection process such as that developed by A. J. Meade, Jr. et al (References 11 through 18) which adds a regularisation functional $\Lambda_l\left[f_{N^2}(\theta,C)\right]$ to the performance index J_{N^2} of this process (see Section 2.2.4). This regularised performance index J_R is

$$J_R = J_{N^2} + \alpha \sum_{l=1}^{LMAX} W_{SW_l}\, \Lambda_l\left[f_{N^2}(\theta,C)\right]$$

where

α is a specified weighting/smoothing constant; $\alpha > 0$.

W_{SW_l} is the weighting coefficient for the l-th data set of the sliding window.

A candidate regularisation functional was identified. This functional is a metric of the first partial derivatives with respect to C; specifically

$$\Lambda_l\left[f_{N^2}(\theta,C)\right] = \sum_{k=1}^{K}\sum_{j \in J_k}\sum_{i \in I_k} W_{R_{i,j,k}}\left(\frac{\partial f_{N^2}(\theta,C)}{\partial C_{i,j,k}}\right)^2$$

where the $W_{R_{i,j,k}}$ are specified weighting constants; $W_{R_{i,j,k}} > 0$.

The motivation behind the selection of a first partial derivative metric as the functional to be adjoined to the performance index is simply that the process of driving the first partial derivatives to zero with the optimisation algorithm can act as a powerful smoothing agent for the neural-network optimal constants selection process. This latter property arises from the definition of a limit. Specifically, as the solution C^* to the optimal constants selection process is approached, at some point there will exist a δ-neighbourhood $N_\delta\left(C^*\right)$ about C^* $\ni$ given an $\varepsilon > 0$, whenever $C \in N_\delta\left(C^*\right)$,

$$\left\| \frac{\partial f_{N^2}(\theta,C)}{\partial C} \right\| < \varepsilon$$

which simply means that the tendency of the neural-network plant model to deviate from the actual plant between evaluations of the neural-network constants will be small near the evaluation points. It is noted that the higher partial derivatives of the filter functions defined in Sections 2.2.2.1 through 2.2.2.4 are simple and readily evaluated.

2. Implement an automatic nodal addition/deletion scheme in the optimal constants selection process such as that described in Section 2.2.4.

3. Develop and implement concepts to automatically adjust the constants in the neural-network filter functions to provide better and more compatible scaling of these functions for the input trajectory data.

4.0 REFERENCES

1. J. A. Leyland, "A Higher Harmonic Optimal Controller to Optimise Rotorcraft Aeromechanical Behaviour", NASA Technical Memorandum 110390, March 1996

2. P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, "Model Building and Practical Aspects of Non-Linear Programming", in *Computational Mathematical Programming*, (edited by K. Schittkowski), NATO ASI Series, 15, Springer-Verlag, Berlin, Germany, 1985

3. M. J. D. Powell, "A Fast Algorithm for Non-linearly Constrained Optimisation Calculations", in *Numerical Analysis Proceedings*, Dundee 1977

4. M. J. D. Powell, "A Fast Algorithm for Non-linearly Constrained Optimisation Calculations", in *Lecture Notes in Economics and Mathematics*, 630, Springer-Verlag, Berlin, Germany, pp 144-157, 1978

5. K. Schittkowski, "Non-linear Programming Codes", in *Lecture Notes in Economics and Mathematics*, 183, Springer-Verlag, Berlin, Germany, 1980

6. K. Schittkowski, "On the Convergence of a Sequential Quadratic Programming Method with an Augmented Lagrangian Line Search Function", in *Mathematik Operationsforschung und Statistik, Serie Optimization*, 4, pp 197-216, 1983

7. K.,Schittkowski, "NLPQL: A FORTRAN Subroutine Solving Constrained Non-linear Programming Problems", (edited by Clyde L. Monna), in *Annals of Operations Research*, 5, pp 485-500, 1986

8. J. Stoer, "Principles of Sequential Quadratic Programming Methods for Solving Non-linear Problems", in *Computational Mathematical Programming*, (Edited by K. Schittkowski), NATO ASI Series, 15, Springer-Verlag, Berlin, Germany, 1985

9. "FORTRAN Subroutines for Mathematical Applications, Version 1.1, Volume 3", *IMSL MATH/LIBRARY User's Manual*, MALB-USM-UNBND-EN8901-1.1, January 1989

10. A. J. D. Bateman, D. G. Ward, R. L. Barron, and M. S. Whalley, "Neural Network Limit Avoidance System for Rotorcraft", Barron Associates, Inc., Charlottesville VA, Technical Report 194 FTR, September 1998

11. A. J. Meade, Jr., Seminar entitled "Artificail Neural Networks in Experimental and Computational Mechanics", NASA-Ames Research Centre (Code ARA), 28 June 1999

12. A. J. Meade, Jr., and B. A. Zeldin, "Establishing Criteria to Ensure Successful Feedforward Artificail Neural Network Modelling of Mechanical Systems", in *Mathematical Computer Modelling*, Vol. 27, No. 5, pp. 61-74, 1998

13. A. N. Tikhonov, A. V. Goncharsky, V. V. Stepanov, and A. G. Yagola, "Numerical Methods for the Solution of Ill-Posed Problems", *Mathematics and Its Applications, Volume 328*, Kluwer Academic Publishers, The Netherlands, August 1995

14. B. A. Zeldin, and A. J. Meade, Jr., "Integrating Experimental Data and Mathematical Models in Simulation of Physical Systems", in *AIAA Journal*, Vol. 35, No. 11, pp. 1787-1790, 1997

15. A. J. Meade, Jr., and A. A. Fernandez, "Solution of Non-linear Ordinary Differential Equations by Feedforward Neural Networks", in *Mathematical Computer Modelling*, Vol. 20, No. 9, pp. 19-44, 1994

16. A. J. Meade, Jr., and B. A. Zeldin, "Approximation Properties of Local Bases Assembled from Neural Network Transfer Functions", in *Mathematical Computer Modelling*, Vol. 28, No. 9, pp. 43-62, 1998

17. A. J. Meade, Jr., M. Kokkolaras, and B. A. Zeldin, "Sequential Function Approximation for the Solution of Differential Equations", in *Communications in Numerical Methods in Engineering*, Vol. 13, pp. 977-986, 1997

18. A. J. Meade, Jr., and R. Moreno, "A Recurrent Artificial Neural Network Model of Duffing's Equation without Traing", in *Smart Engineering Systems Design*, Vol. 1, pp. 135-148, 1998

19. "Programming in VAX FORTRAN. Software Version: V4.0", *VAX/VMS Manual No AA-D034D-TE,* Digital Equipment Corporation (DEC), Maynard, Mass, September 1984

20. Sesi Kottapalli, Engineering Notes from the Individual Blade Control (IBC) Test Programme, Nov 1998

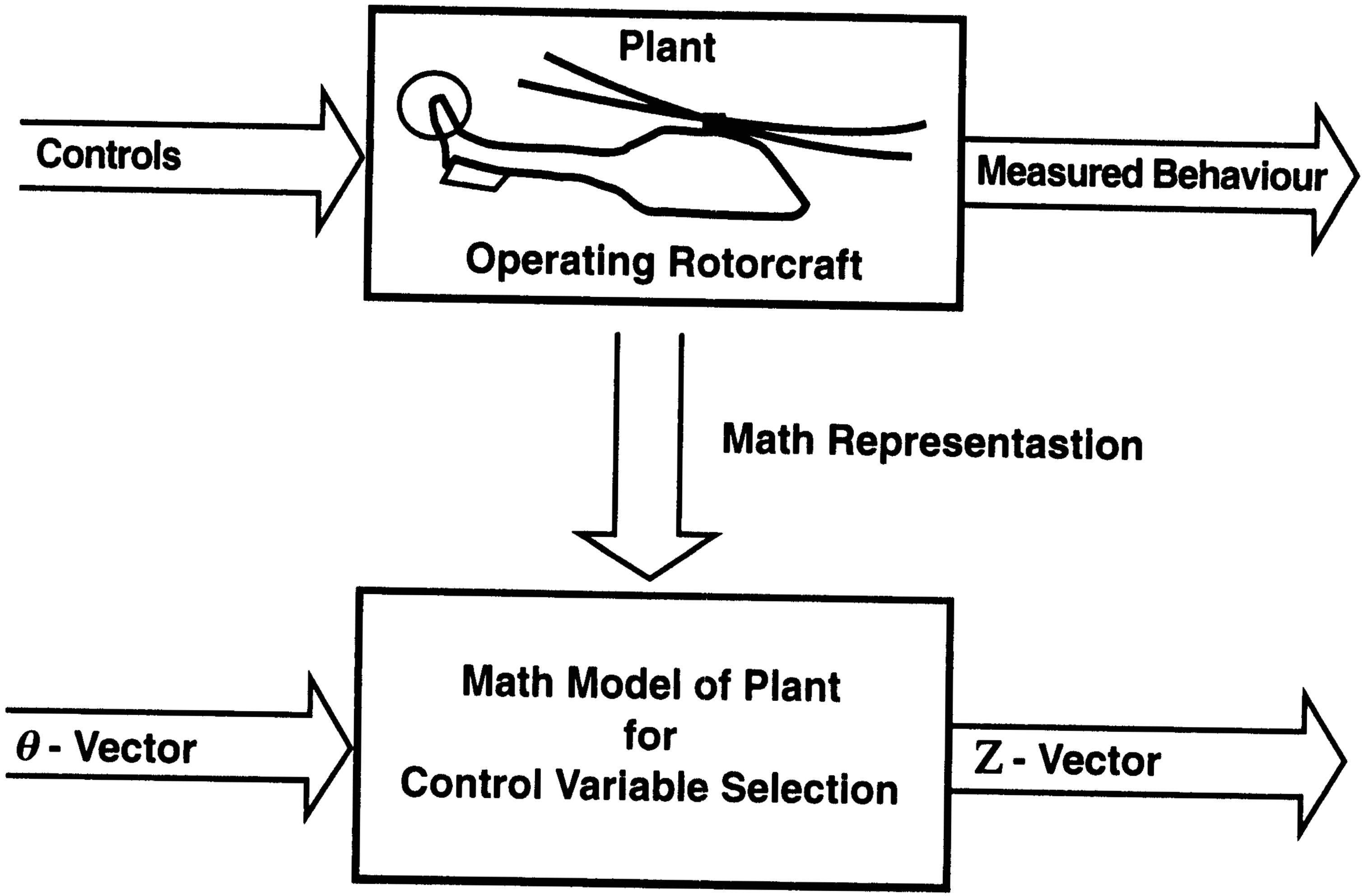

Figure 1. General Controlled Rotorcraft Aeromechanical Behaviour Response

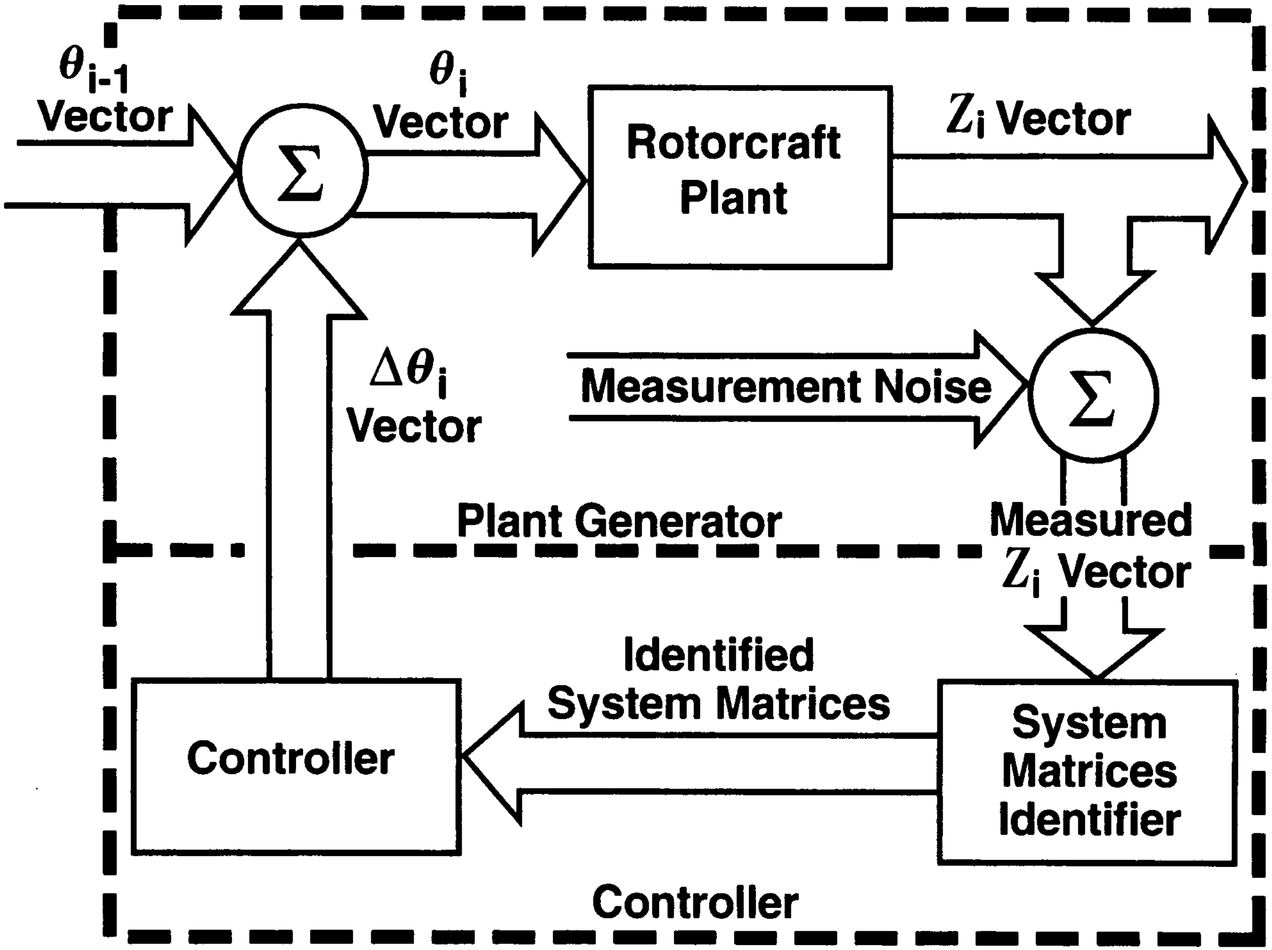

Figure 2. General Closed-Loop Rotorcraft Aeromechanical Behaviour Controller

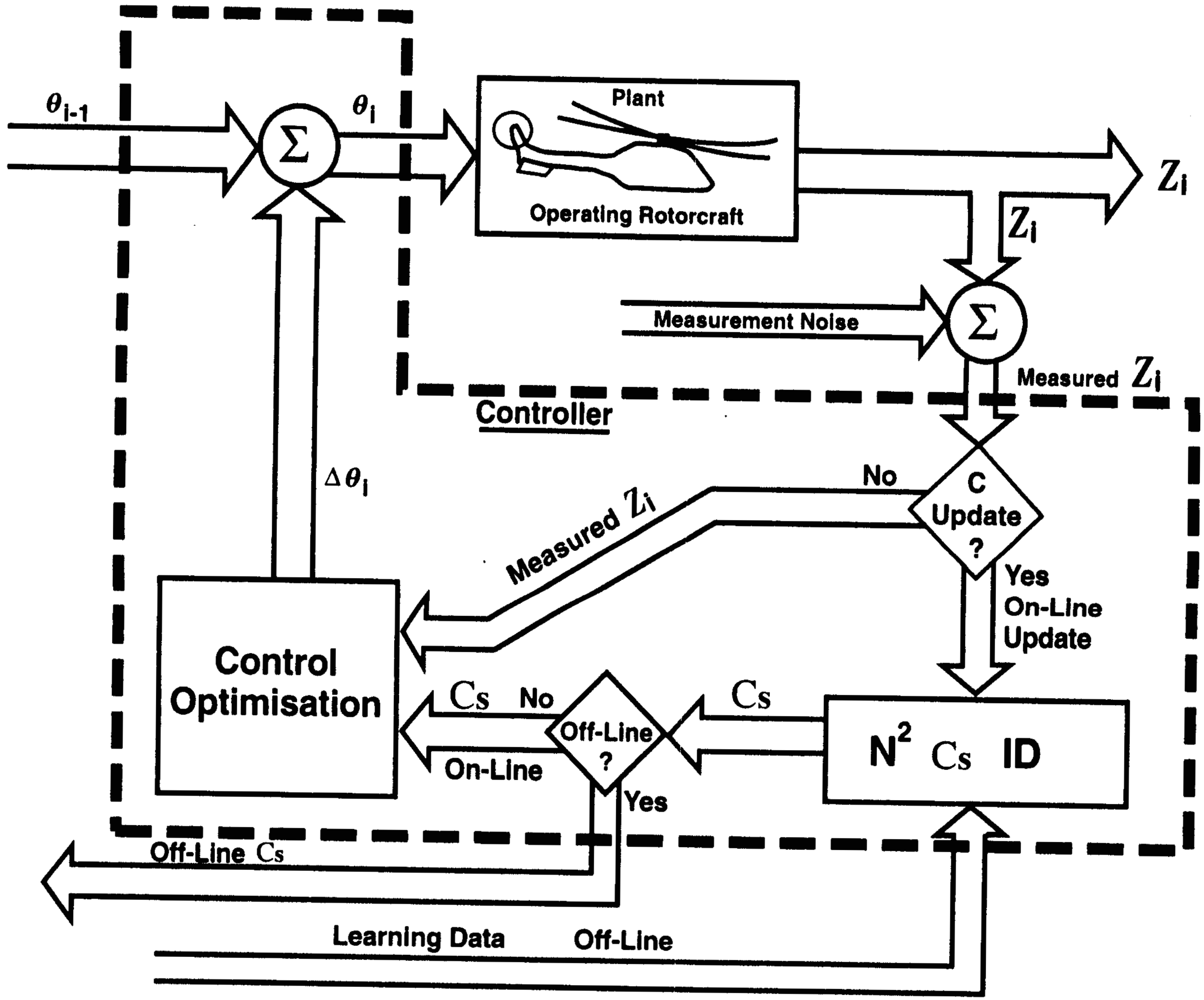

Figure 3. General Closed-Loop Neural-Network (N^2) Controller

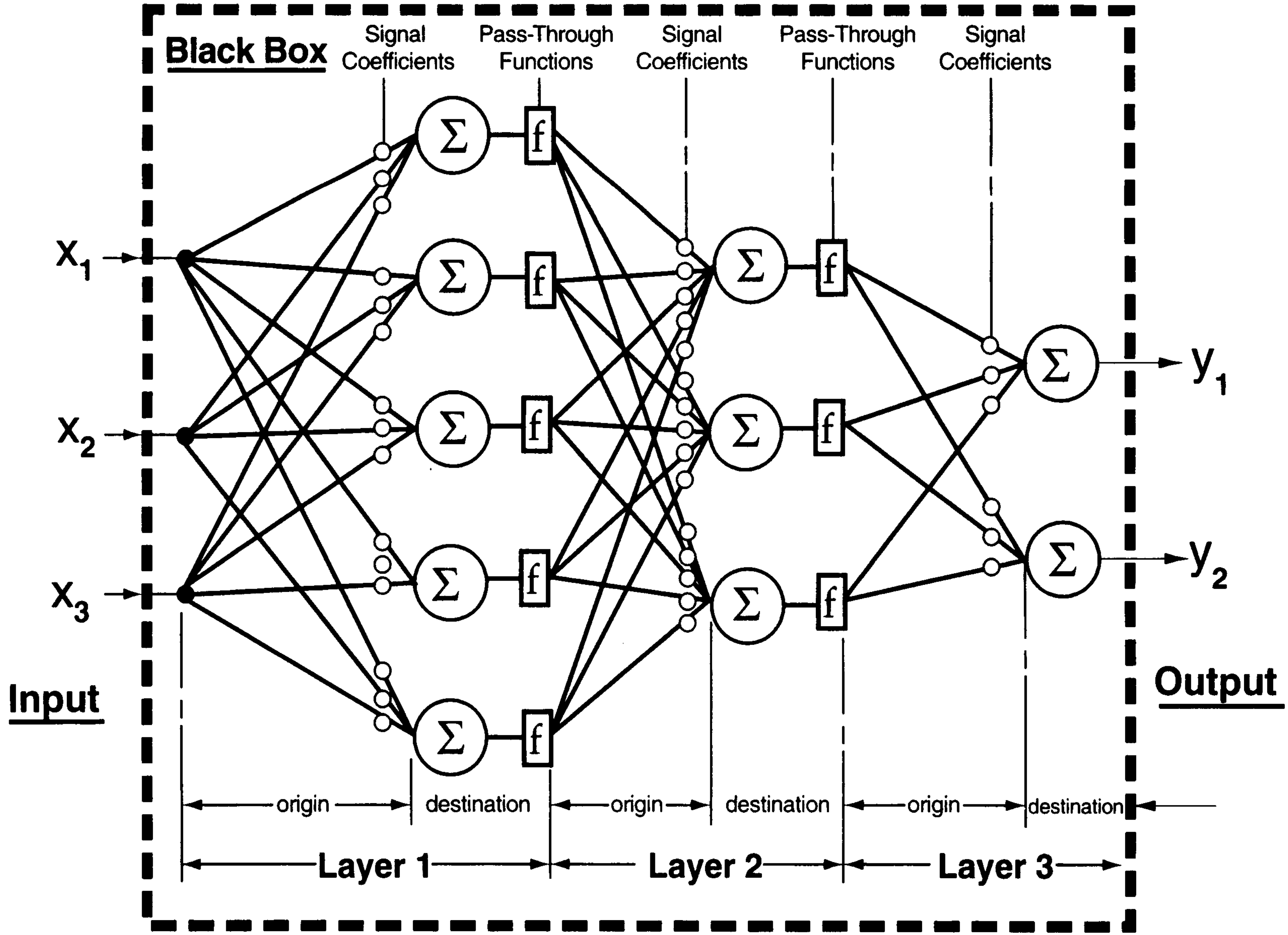

Figure 4. Schematic for the 3-5-3-2 Neural-Network Function $f_{N^2}^{3-5-3-2}(\bullet)$

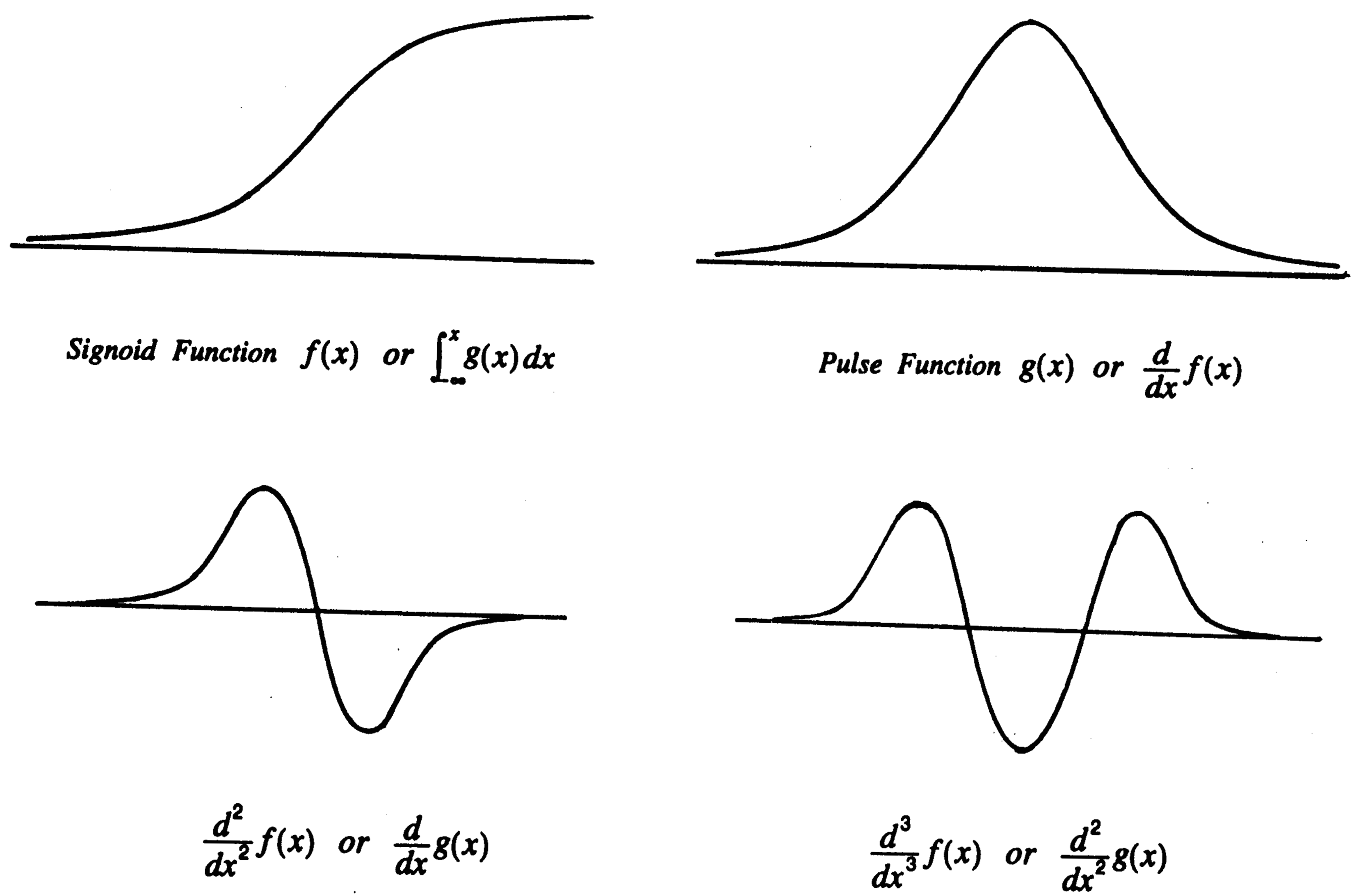

Figure 5. Signoid and Pulse Type Filter Functions and Their Derivatives

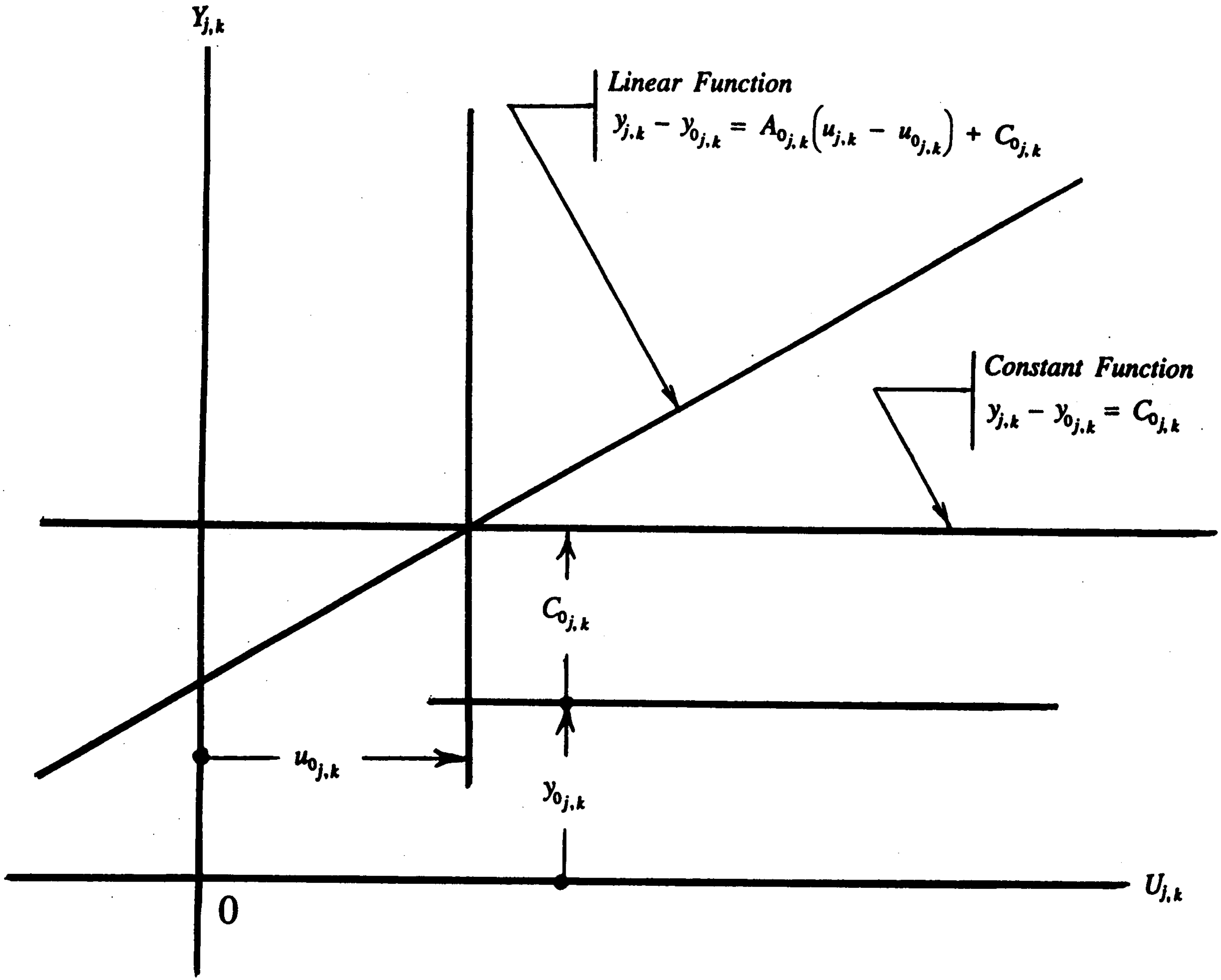

Figure 6. Constant and Linear Neural-Network Filter Functions

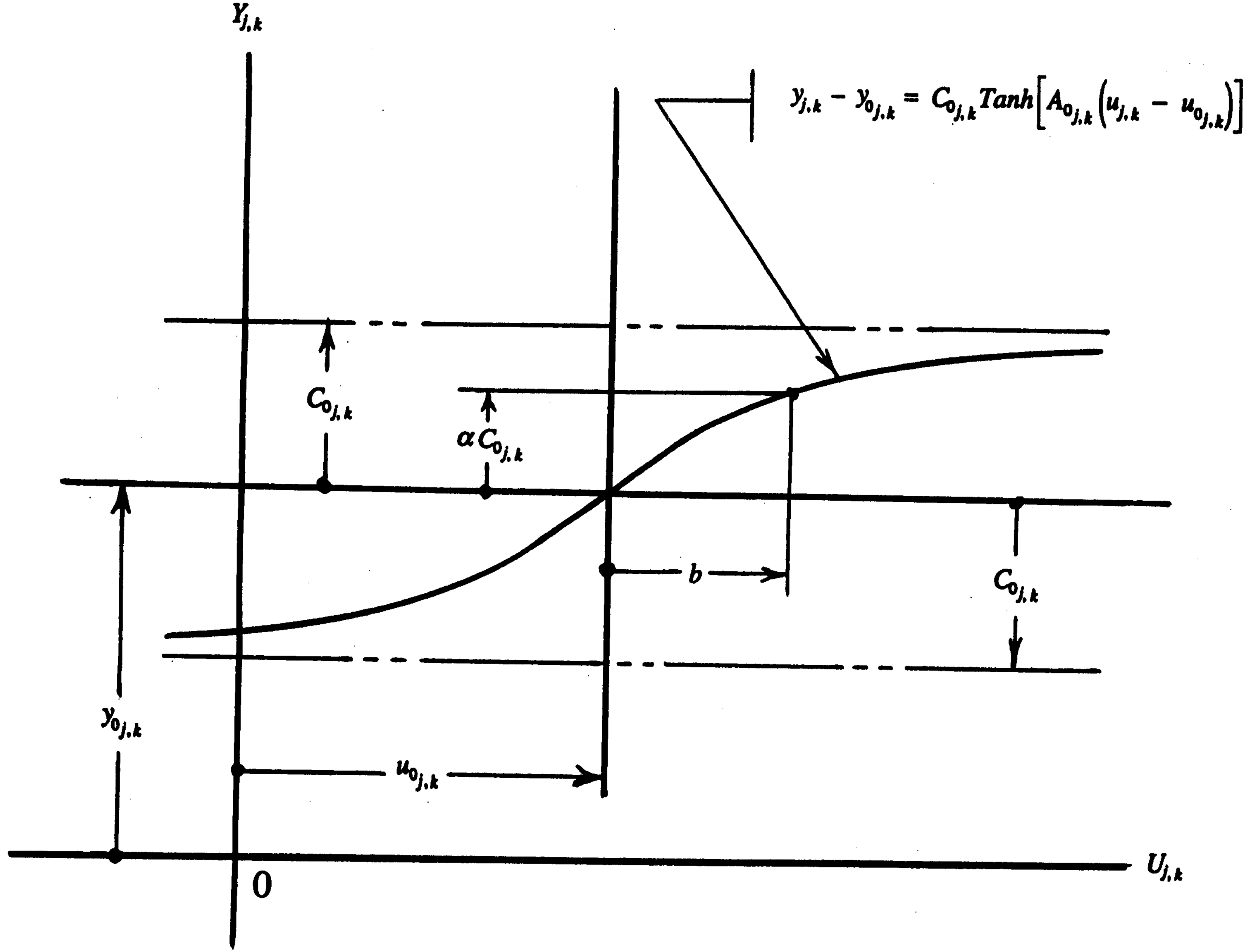

Figure 7. Hyperbolic Tangent Neural-Network Filter Function

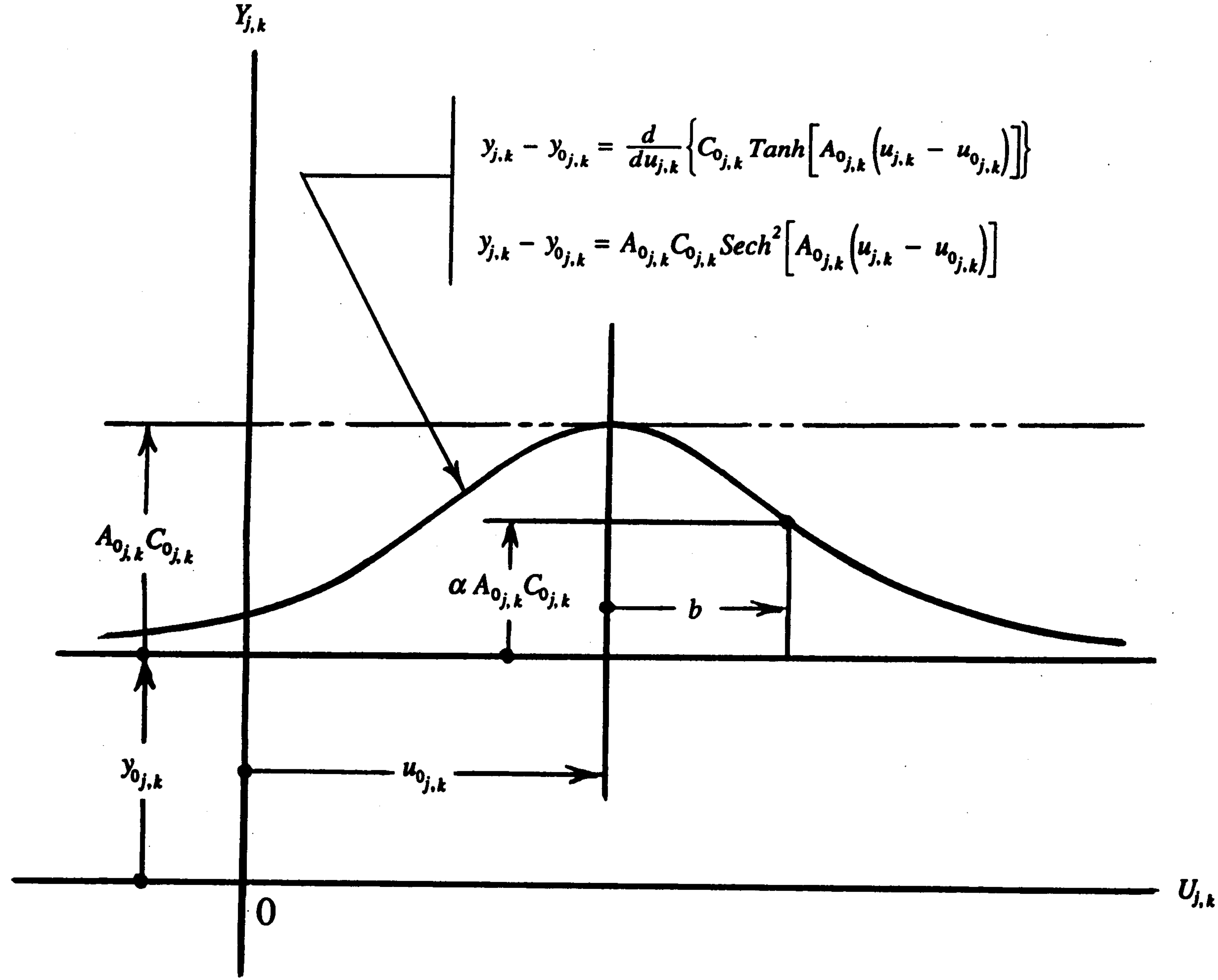

Figure 8. First Derivative of the Hyperbolic Tangent Neural-Network Filter Function

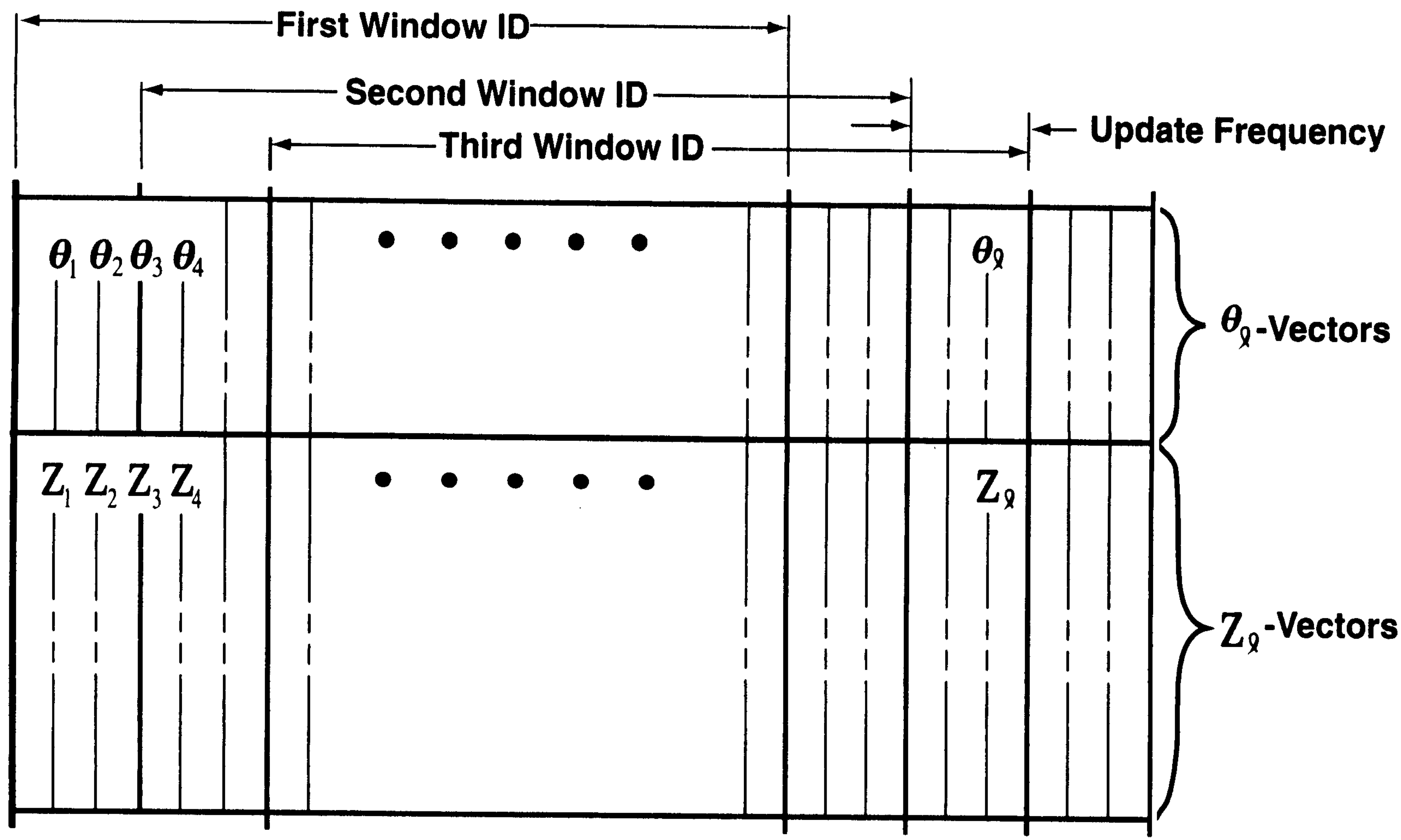

Figure 9. Sliding Window of Data Acquiaition

$$\tau_m \equiv \mathrm{DMOD}\left(\left[t - t_{0_m} - t_{\phi_m}\right], \mathrm{T}_m\right)$$

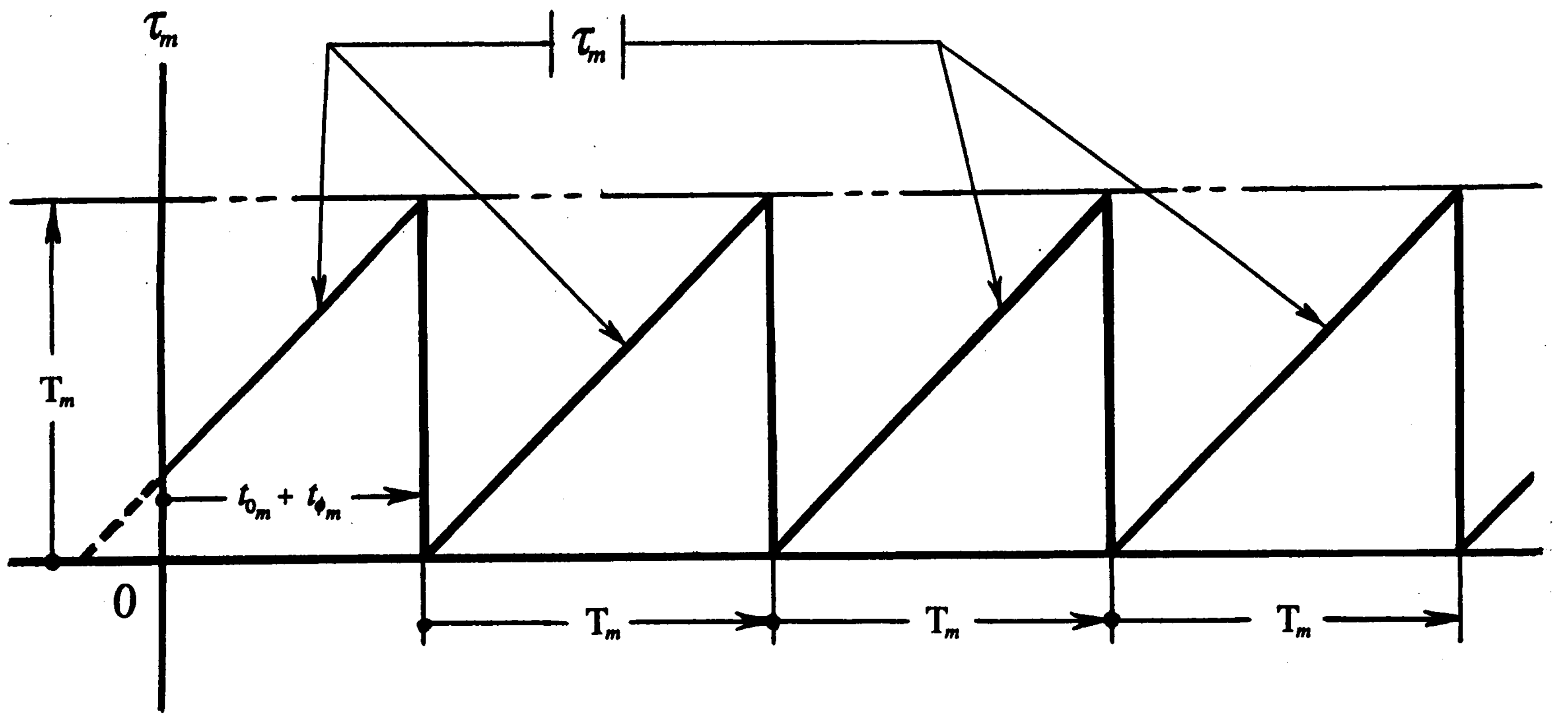

Figure 10. Periodic Time Argument τ_m

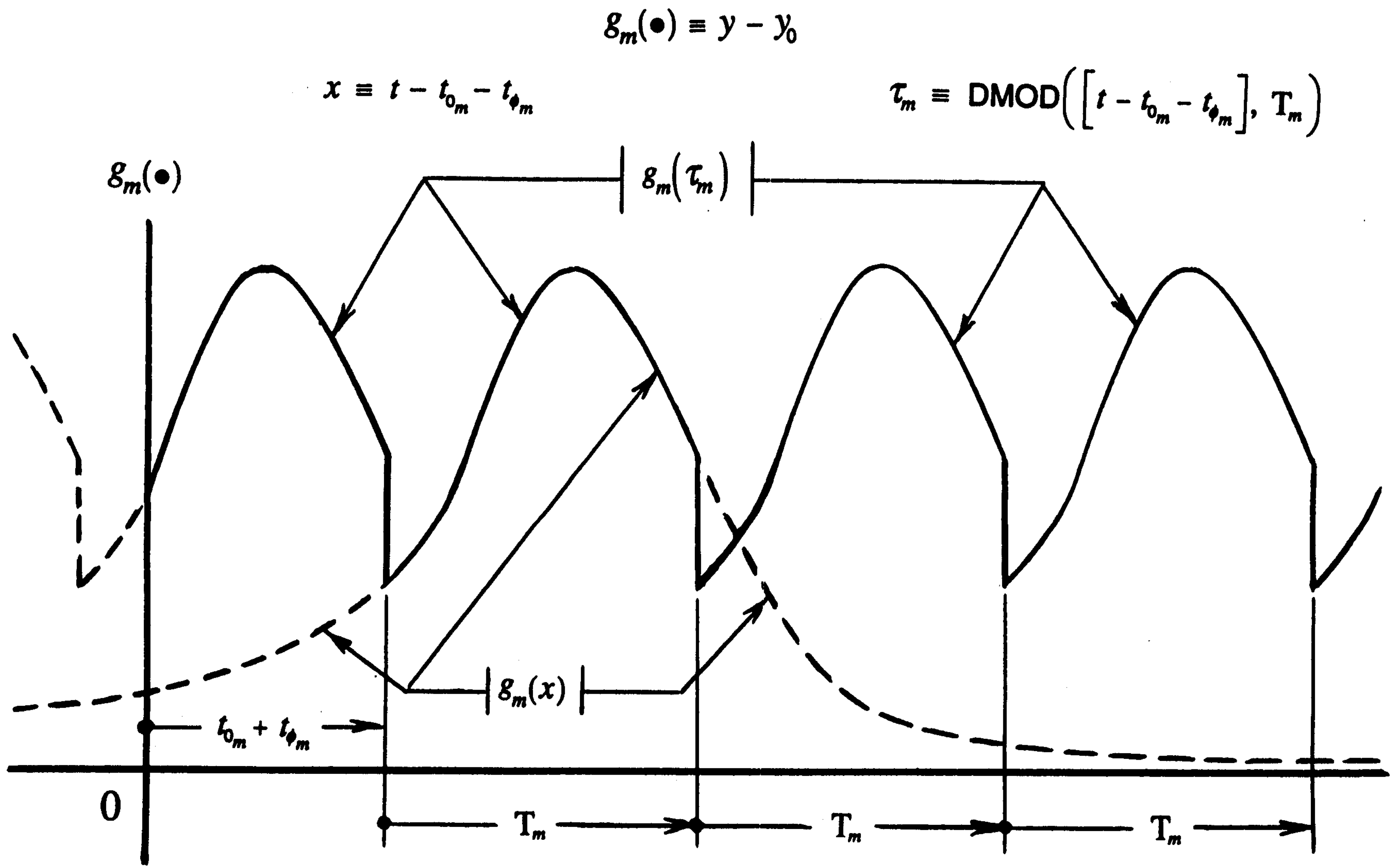

Figure 11. General Periodic Modelling Function $g_m(\bullet)$

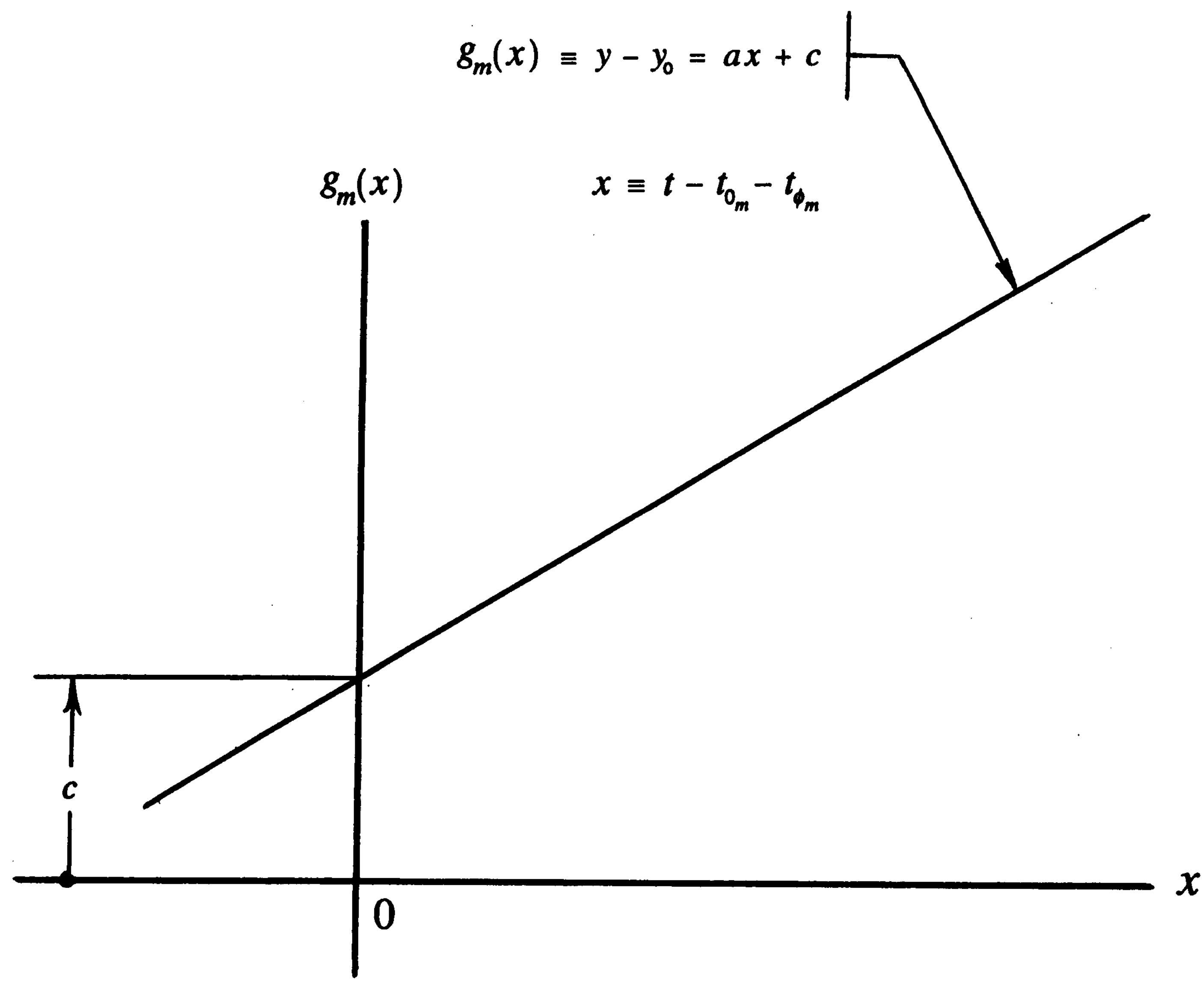

Figure 12. Linear/Ramp Function

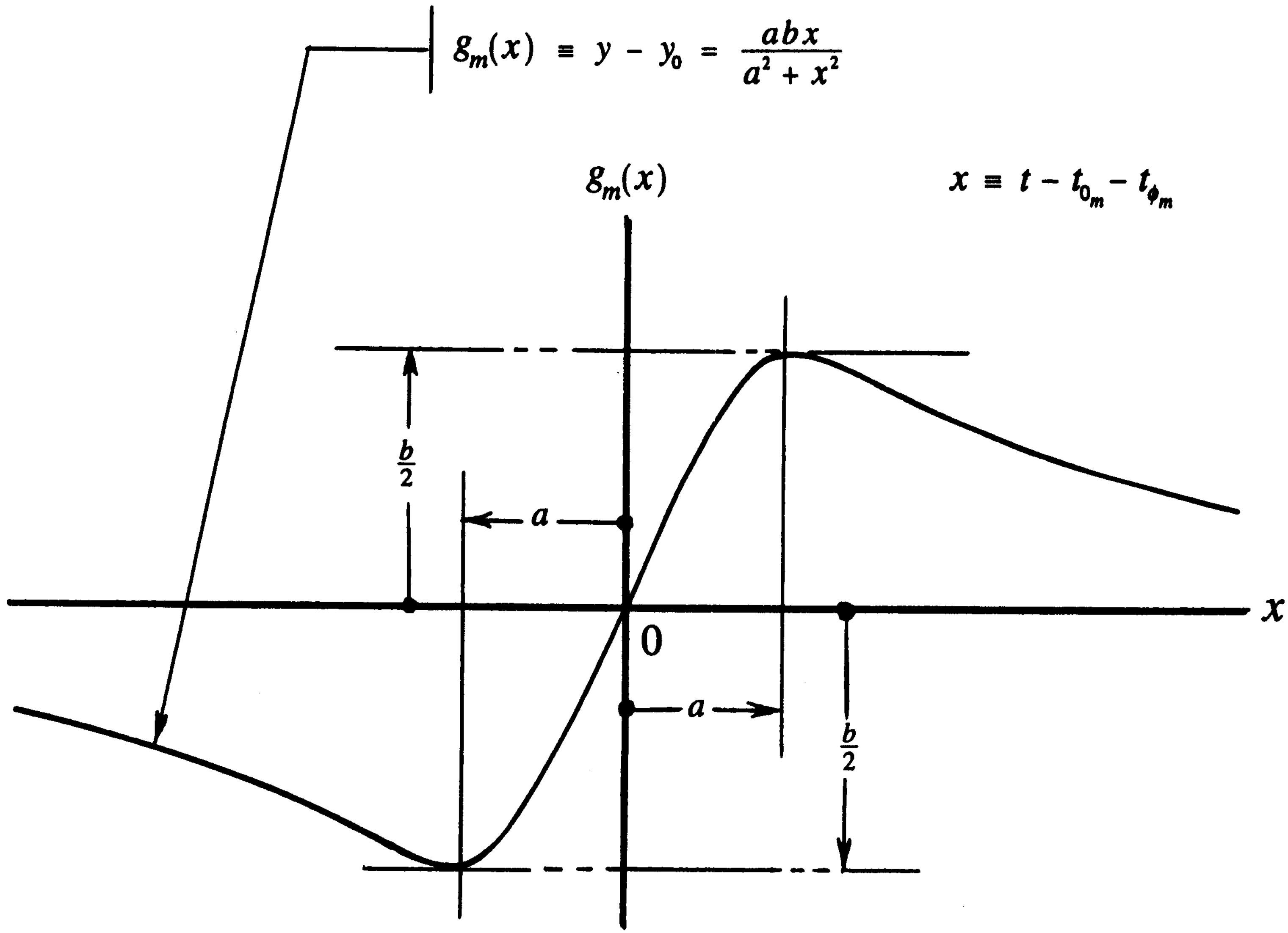

Figure 13. Serpentine Curve Function

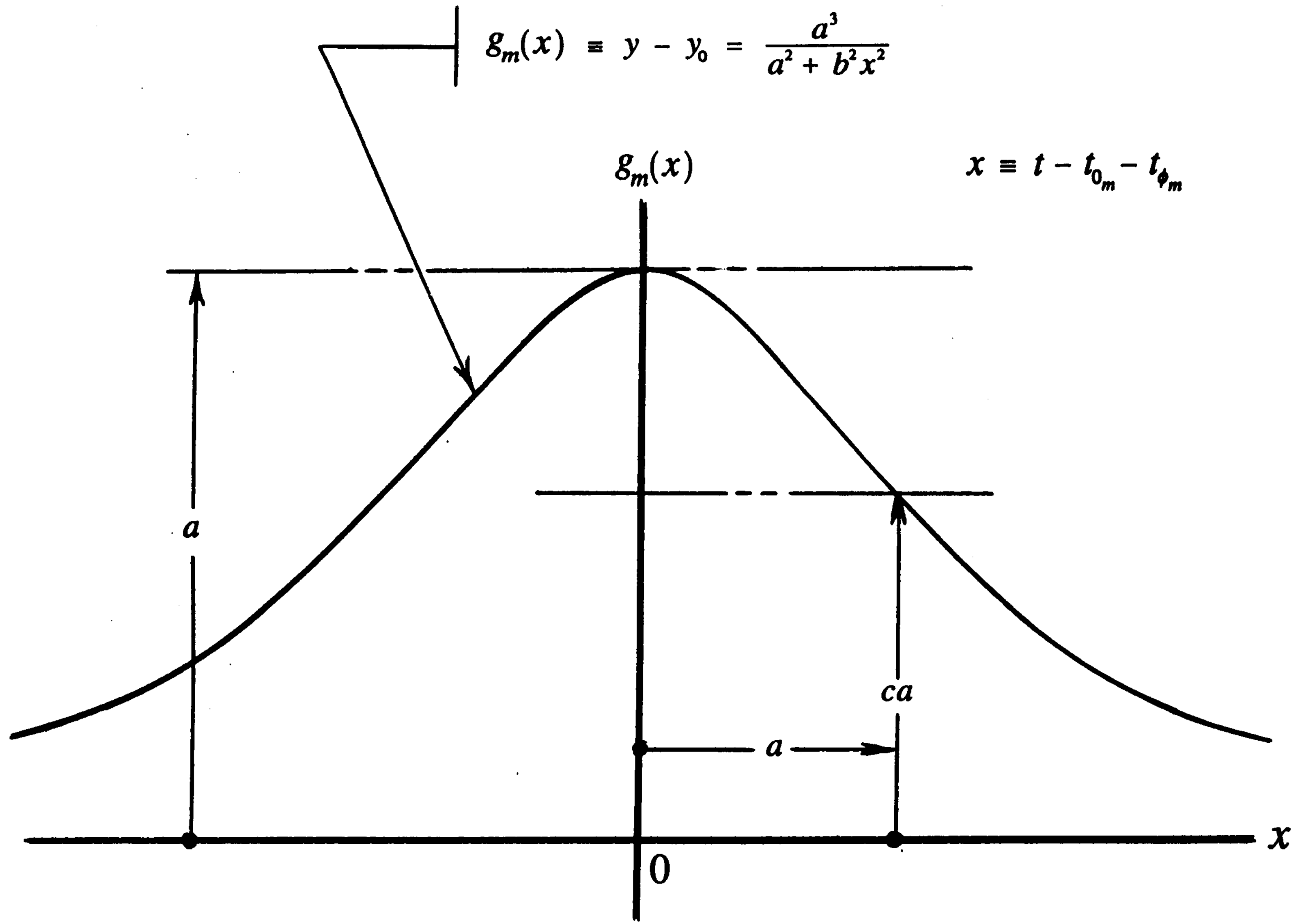

Figure 14. Witch of Agnesi Function

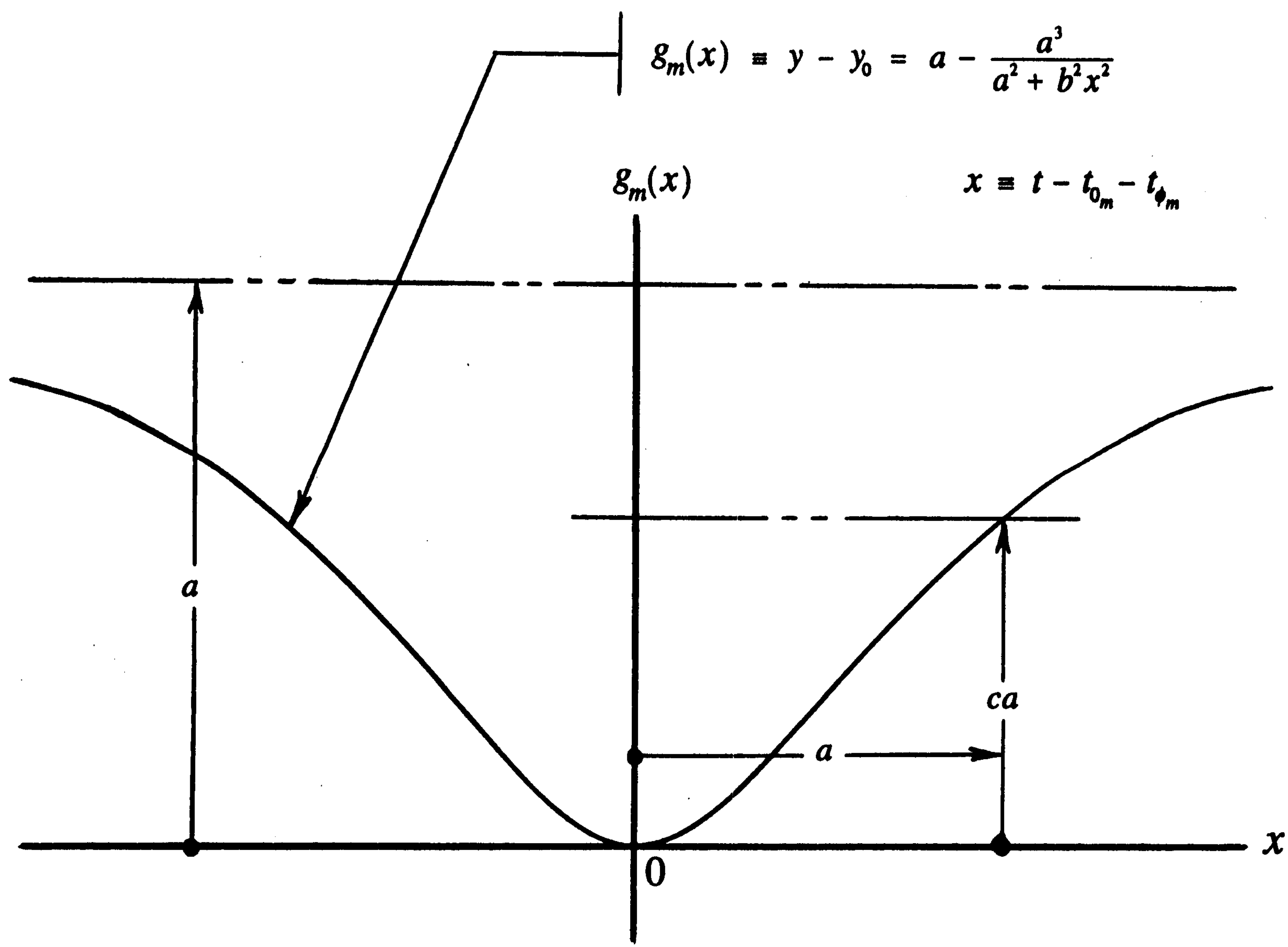

Figure 15. Inverted Witch of Agnesi Function

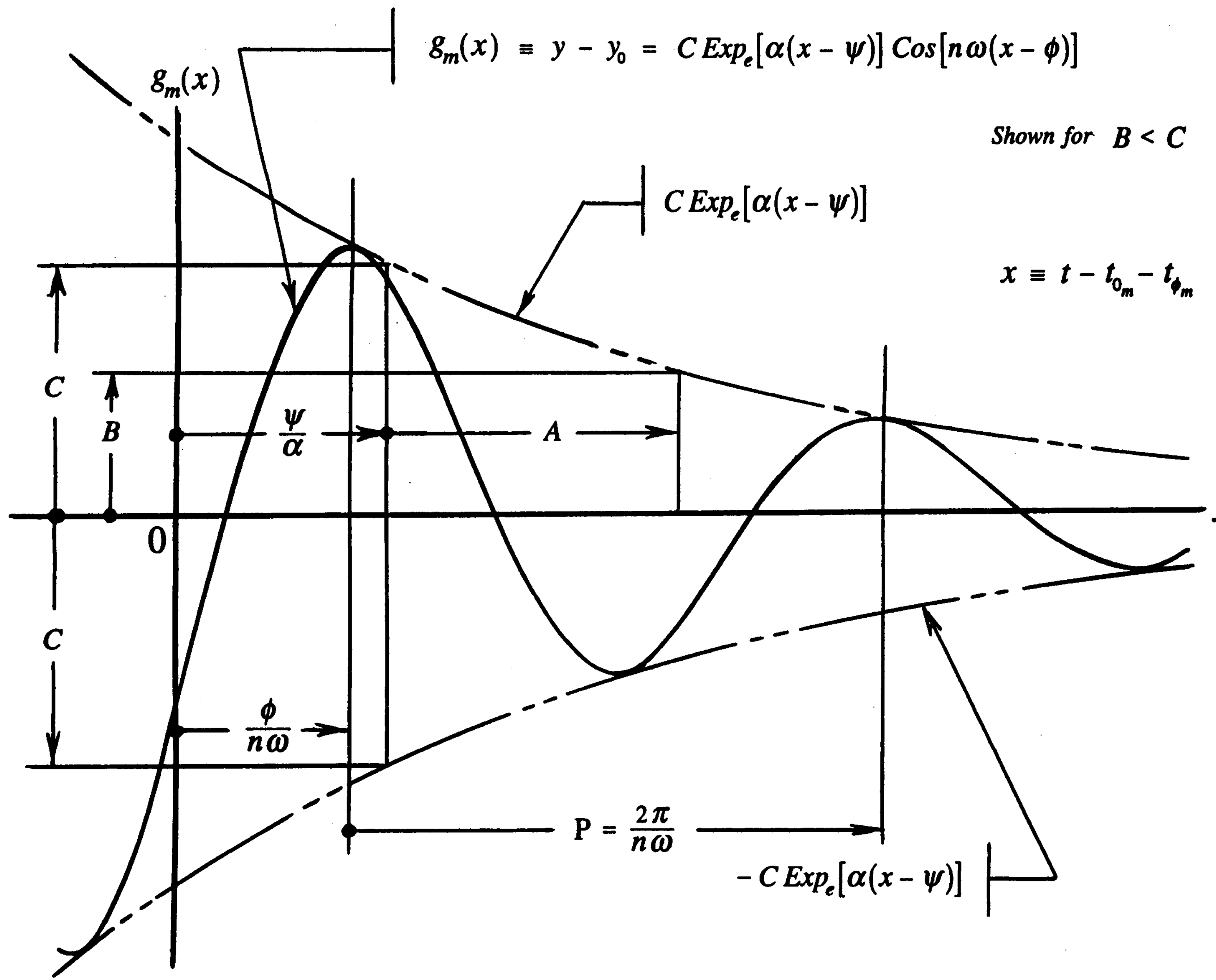

Figure 16. Enveloped Sinusoidal Function

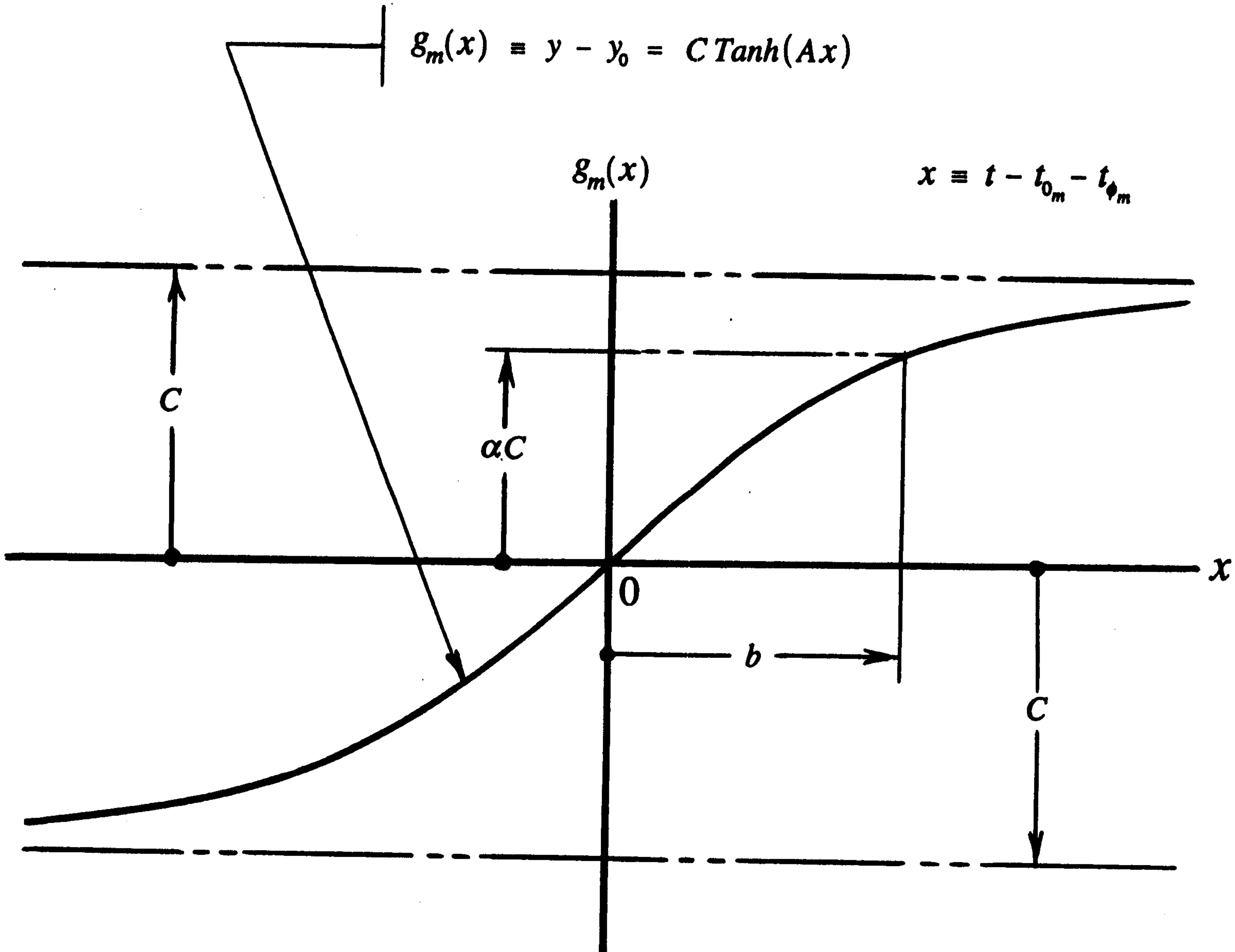

Figure 17. Hyperbolic Tangent Function

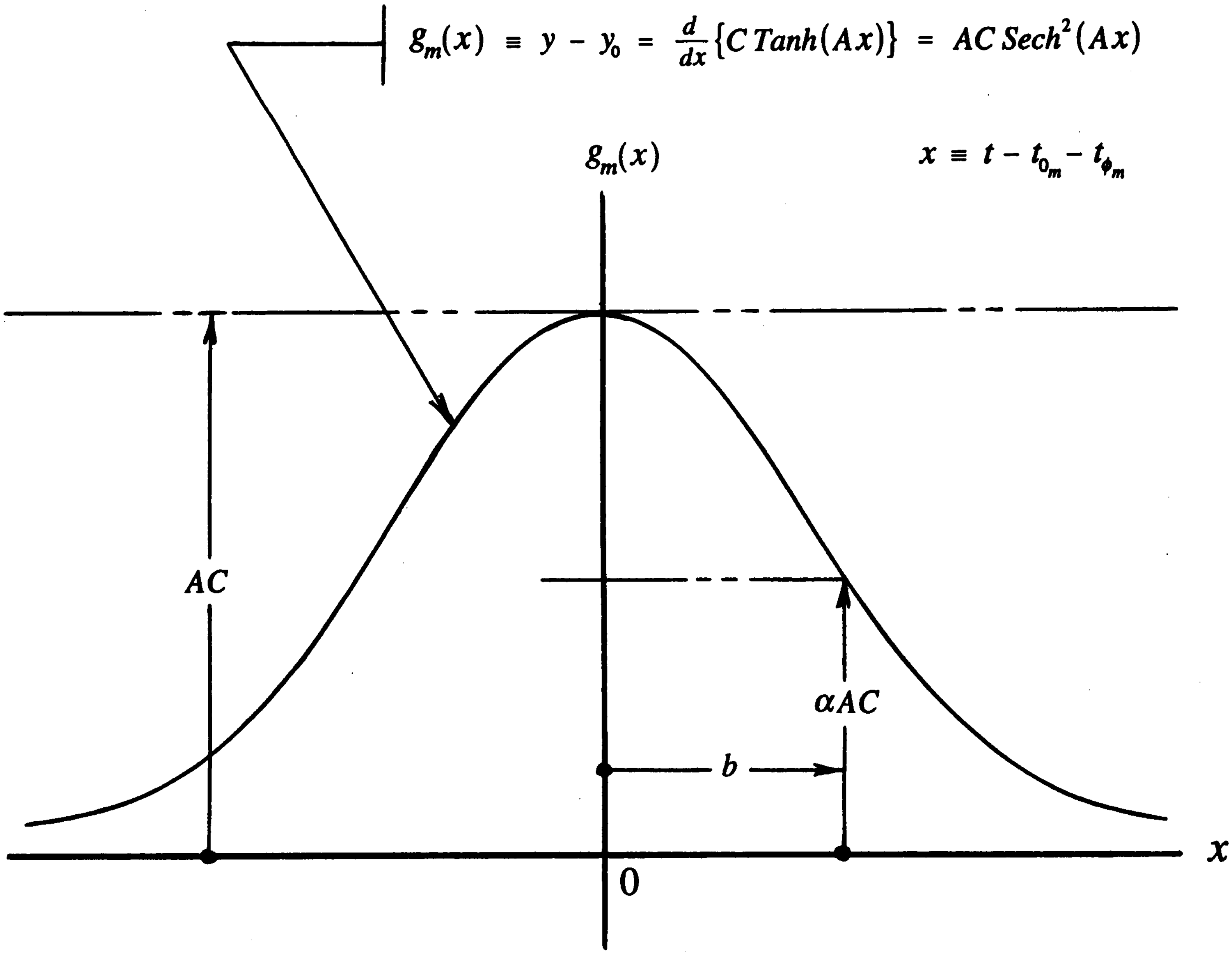

Figure 18. First Derivative of the Hyperbolic Tangent Function

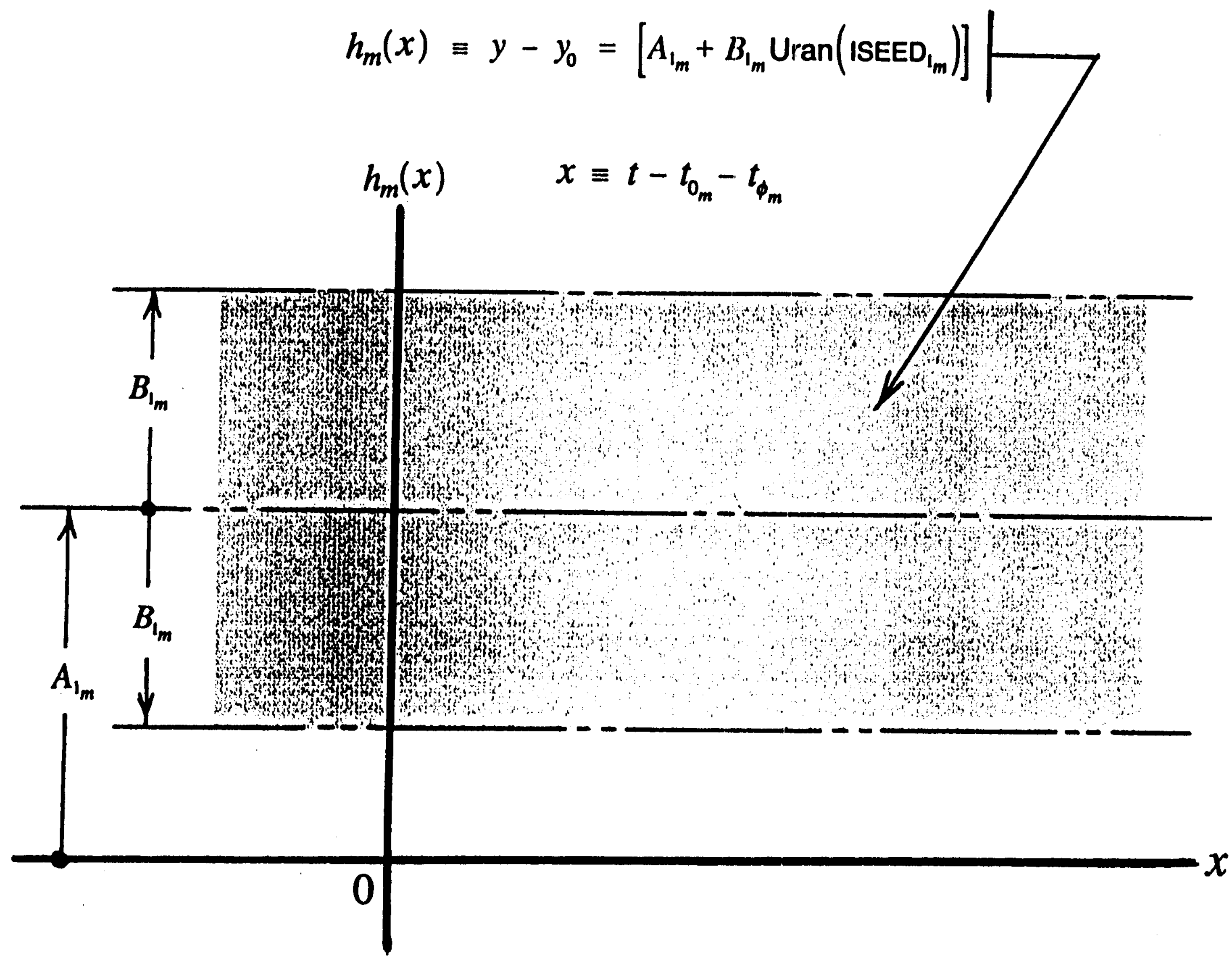

Figure 19. Uniformly Distributed Random Function

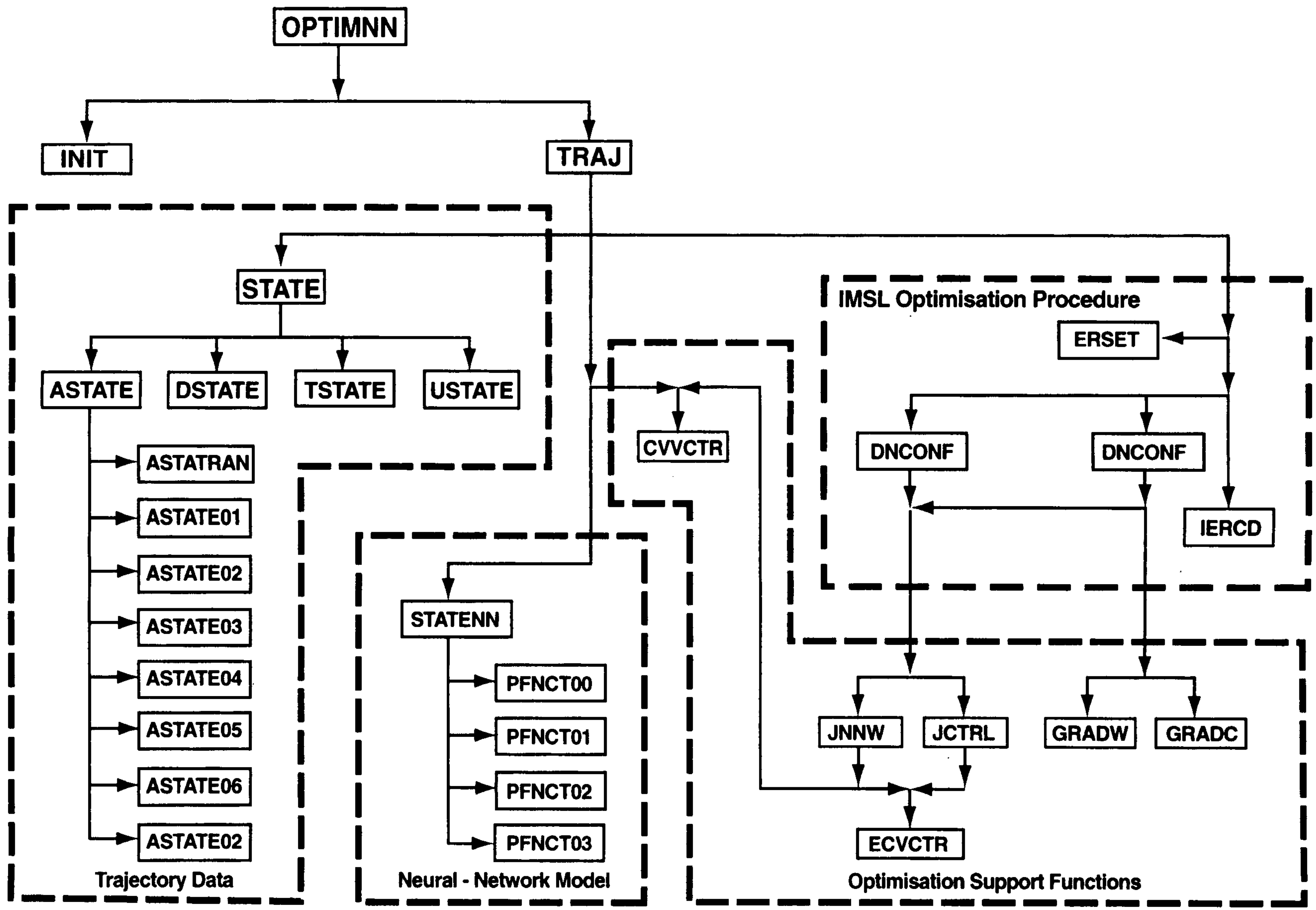

Figure 20. General Organisation of the ONNC System

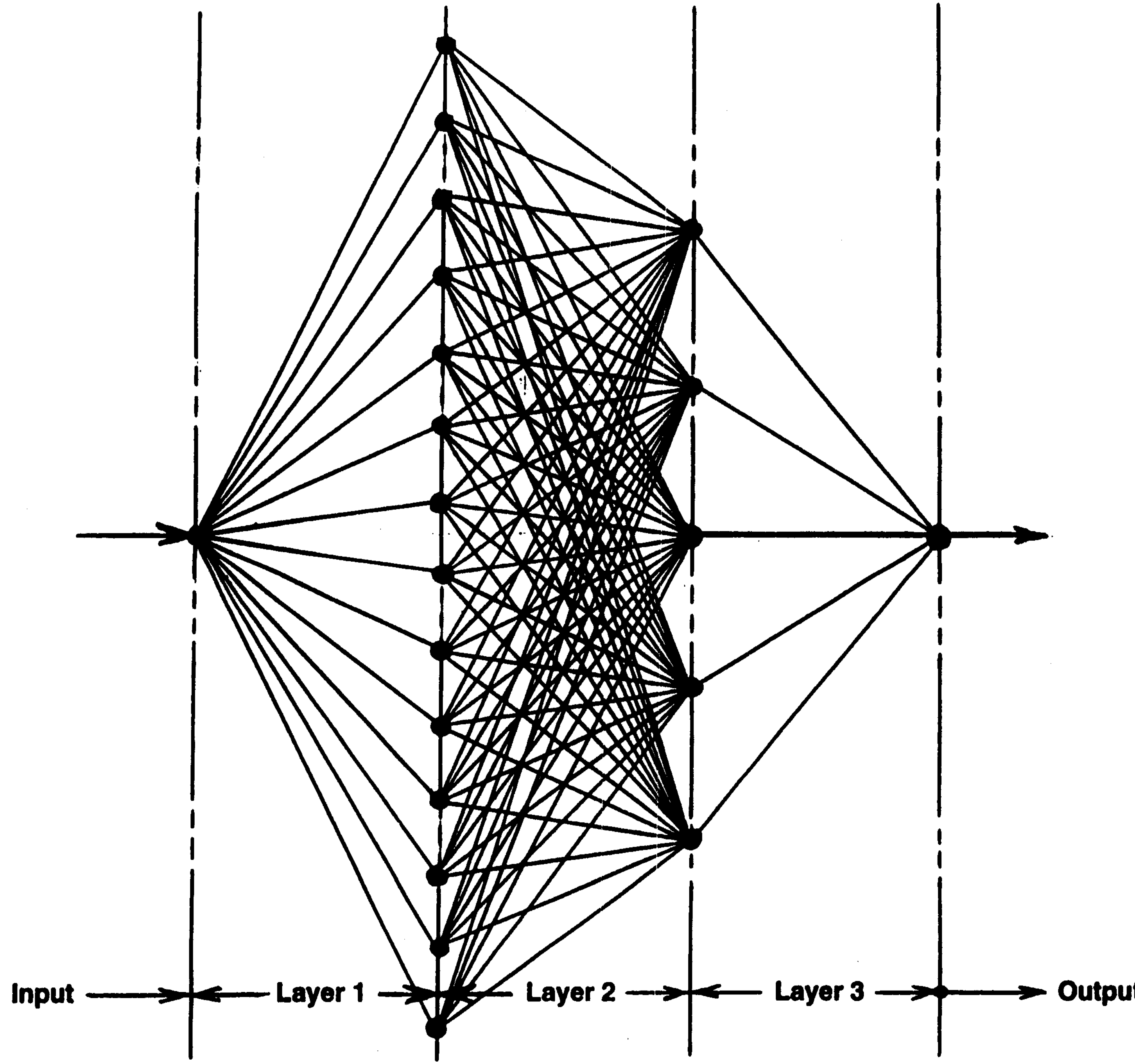

Figure 21. A 1 – 14 – 5 – 1 Neural-Network Model

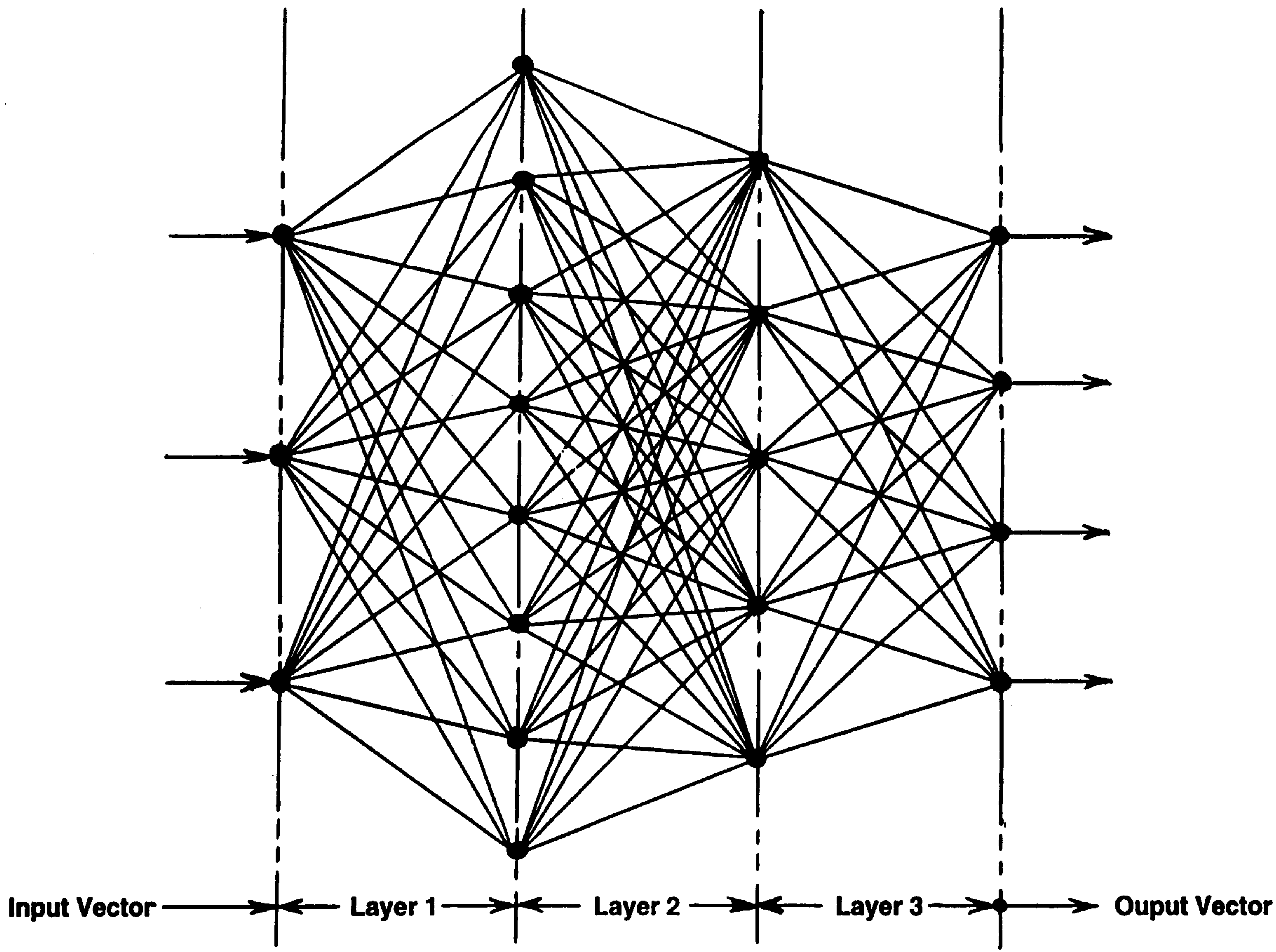

Figure 22. A 3 – 8 – 5 – 4 Neural-Network Model

Appendix A

Principal Parameters

in the

OPTIMNN Code

Appendix A

Table of Contents

Parameter List

Parameter List (Continued)

Parameter	Page	Parameter	Page	Parameter	Page
K	23	• NFUNCT(J,K)	14	• PSI(L_3,L_2,L_1)	20
K	24	• NI(K)	15	PT100	48
K	26	NICV	44	PT200	48
K	38	•• NIDIM	3	PT300	48
K	43	NIJKCVC	44	PT500	48
L	2	NIJKCVL	45	PT800	48
L	6	•• NIJKDIM	3	RTD	48
L	9	• NJ(K)	15	• SCVC(I)	27
L	11	•• NJDIM	3	• SCVNNC(I,J,K)	24
L	33	NJEC	45	• SCVNNL(I,J,K)	21
L	43	NJJECC	45	• SMALL1	7
L_1	2	NJJECL	45	• SMALL2	7
L_1	6	•• NJKDIM	3	• SMALL3	7
L_1	19	• NK	15	• SMALL4	7
L1	33	•• NKDIM	3	• SMAXC(I)	28
L_2	2	•• NL1DIM	4	• STMODC	11
L_2	19	• NL2(L_1)	7	• STMODL	9
L2	33	• NL2(L_1)	19	SUMSQ	46
L_3	2	•• NL21	4	SUMSQW(L)	47
L_3	19	•• NL2DIM	4	T	34
L3	33	• NL3(L_2,L_1)	19	TABS	34
• LARGE1	6	•• NL321	4	••• TBLMAX	7
• LARGE2	6	•• NL3DIM	4	••• TBLMAX	35
• LARGE3	6	•• NLDIM	4	• TCINIT	12
• LARGE4	6	•• NLTBL	5	• TCFINL	12
• LDELAY	9	• NN(L_3,L_2,L_1)	19	• TCSTEP	12
LMAX	33	• NNCID	11	• TCTYPE	12
LMAX	44	NNID	34	TCUT	35
LSTEP	33	• NNLID	9	••• TD(L)	7
LTBL	6	NNUP	34	••• TD(L)	35
LTBL	34	NNUP0	34	TEN	48
• MITNC	29	• OMEGA(L_3,L_2,L_1)	20	TENM2	48
• MITNNNC	26	ONE	48	TENM3	48
• MITNNNL	23	• OUTC	29	TENM6	48
• MULT	7	• OUTNNC	26	TENM8	48
•• NCON	2	• OUTNNL	23	TENP2	48
NCONC	44	• PERIOD(L_3,L_2,L_1)	20	TENP3	48
NCONNNC	44	• PHASE(L_3,L_2,L_1)	20	TENP6	48
NCONNNL	44	• PHI(L_3,L_2,L_1)	20	TENP8	48
•• NCV	2	PI	48	• TINIT	7
•• NEC	3	PINDX	45	• TFINL	7

Parameter List (Continued)

• Denotes Data Defined by **NAMELIST CDATA Input Data** read from Subroutine **INIT**. The **CDATA** NAMELIST input is used to define the **trajectory, optimisation, and neural-net models** and **options** required to operate the **OPTIMNN** System.

•• Denotes Data Defined by **PARAMETER Statements**. This Data defines the **Dimensions** of the **Principal Arrays** of the **OPTIMNN** System.

••• Denotes Data Defined by **NAMELIST CDATA Input Data** read from Subroutine **INIT**, or by **Directly Read On-Line Test Data**, or by **Internally Computed Data**. This Data defines the "**Actual**" (Reference) **Plant Input Vector** and/or **Output Vector** at Specified Trajectory Time Points.

INPUT Parameters

to the

OPTIMNN Code

• via **Namelist CDATA**

•• via **PARAMETER Statements**

••• via **Namelist DDATA,**
or **On-Line Test Data,**
or **Internally Computed Data**

• Denotes Data Defined by **NAMELIST CDATA Input Data** read from Subroutine **INIT**. The **CDATA** NAMELIST input is used to define the **trajectory, optimisation, and neural-net models** and **options** required to operate the **OPTIMNN** System.

•• Denotes Data Defined by **PARAMETER Statements**. This Data defines the **Dimensions** of the **Principal Arrays** of the **OPTIMNN** System.

••• Denotes Data Defined by **NAMELIST CDATA Input Data** read from Subroutine **INIT**, or by **Directly Read On-Line Test Data**, or by **Internally Computed Data**. This Data defines the **"Actual"** (Reference) **Plant Input Vector** and/or **Output Vector** at Specified Trajectory Time Points.

Input Group 1

Dimensions of the Principal Arrays of the OPTIMNN System

Parameter	Default or Initial Value	Definition
I	1	**Index** which Specifies the **I-th Element Position** in the **Origin Vector** for the **Specific Neural-Network Layer** specified by the index "K" (INTEGER*4).
J	1	**Index** which Specifies the **J-th Element Position** in the **Destination Vector** for the **Specific Neural-Network Layer** specified by the index "K" (INTEGER*4).
K	1	**Index** which Specifies the **K-th Specific Layer** in the **Neural-Network** (INTEGER*4).
L	1	**Index** which Specifies the **L-th Data Set** in the **Data Sliding Window** (INTEGER*4).
L_1	1	**Index** which **Assigns** the **Analytic Modelling Function** for a specific (L_3,L_2,L_1) **to** either the **Plant Input Vector** (i.e., the Plant Control Vector) **or** the **Plant Output Vector** (i.e., Plant Measurement/State Vector) (INTEGER*4). = 1 Specifies that the Model applies to an element of the **Plant Input Vector** (i.e., the Plant Control Vector). = 2 Specifies that the Model applies to an element of the **Plant Output Vector** (i.e., Plant Measurement/State Vector).
L_2	1	**Index** which Specifies the **Element Number** for the **Plant Input Vector** (i.e., the Plant Control Vector) if L_1 = 1, **or** the **Plant Output Vector** (i.e., Plant Measurement/State Vector) if L_1 = 2 (INTEGER*4).
L_3	1	**Index** which Specifies the **Element Number** of the **Analytic Modelling Function** for a specific (L_2,L_1) (INTEGER*4).
** NCON	See Def	**Dimension** of the **Constraint Function Vector Arrays** such as the CON(IIJK) and CON(III) Vectors (INTEGER*4). NCON = NL2DIM
** NCV	NL2DIM	**Dimension** of the **Optimisation Control Vector Arrays** such as the CV(IJK) and CV(II) Vectors (INTEGER*4). NCV = JMAX0(NL2DIM, NIDIM*NJDIM*NKDIM)

Input Group 1 (Continued)

Dimensions of the Principal Arrays of the OPTIMNN System

Parameter	Default or Initial Value	Definition
•• NEC	NL2DIM	**Dimension** of the **Optimisation End Conditions Vector Arrays** such as the EC(JJJ) and EC(JJ) Vectors (INTEGER*4). NEC = NL2DIM
•• NIDIM	16	Dimension (i.e.,., the l.u.b) of the **I-th Subscript** of the **Neural-Network Arrays** (i.e., the subscript which defines the element position in the **Origin Vector** for the Neural-Network Layers) such as those defined in Groups 5 and B (e.g., the CW(I,J,K), and XNN(I,J,K) Arrays) but not necessarily limited to arrays defined in these groups (INTEGER*4).
•• NIJKDIM	See Def	Equivalent Single Dimension of the **Three-Dimensional Neural-Network Arrays** such as those defined in Groups 5 and B (e.g., the CW(I,J,K), and XNN(I,J,K) Arrays) but not necessarily limited to arrays defined in these groups (INTEGER*4). NIJKDIM = NIDIM*NJDIM*NKDIM
•• NJDIM	16	Dimension (e.g., the l.u.b) of the **J-th Subscript** of the **Neural-Network Arrays** (i.e., the subscript which defines the element position in the **Destination Vector** for the Neural-Network Layers) such as those defined in Groups 5 and B (e.g., the AN(J,K), CW(I,J,K), UNN(J,K), and XNN(I,J,K) Arrays) but not necessarily limited to arrays defined in these groups (INTEGER*4).
•• NJKDIM	See Def	Equivalent Single Dimension of the **Two-Dimensional Neural-Network Arrays** such as those defined in Groups 5 and B (e.g., the AN(J,K), and UNN(J,K) Arrays) but not necessarily limited to arrays defined in these groups (INTEGER*4). NJKDIM = NJDIM*NKDIM
•• NKDIM	4	Dimension (i.e., the l.u.b) of the **K-th Subscript** of the **Neural-Network Arrays** (i.e., the subscript which defines the **Specific Neural-Network Layer**) such as those defined in Groups 5 and B (e.g., the AN(J,K), CW(I,J,K), NI(K), NJ(K), UNN(J,K), and XNN(I,J,K) Arrays) but not necessarily limited to arrays defined in these groups (INTEGER*4).

Input Group 1 (Continued)

Dimensions of the Principal Arrays of the OPTIMNN System

Parameter	Default or Initial Value	Definition
•• NL1DIM	2	Dimension (e.g., the l.u.b) of the **L_1-th Subscript** of the **Analytic Trajectory Synthesis Arrays** (i.e., the Subscript which Assigns the **Analytic Modelling Function** for a specific (L_3,L_2,L_1) to either the **Plant Input Vector** if $L_1 = 1$, or the **Plant Output Vector** if $L_1 = 2$) such as those defined in Group 6 (e.g., the A3(L_2,L_1), IFUNCT(L_3,L_2,L_1), NL2(L_1), NL3(L_2,L_1), and NN(L_3,L_2,L_1) Arrays) but not necessarily limited to arrays defined in that group (INTEGER*4).
•• NL21	See Def	Equivalent Single Dimension of the **Two-Dimensional Neural-Network Arrays** such as those defined in Groups 6 (e.g., the A3(L_2,L_1) and NL3(L_2,L_1) Arrays) but not necessarily limited to arrays defined in that group (INTEGER*4). NL21 = NL2DIM*NL1DIM
•• NL2DIM	12	Dimension (e.g., the l.u.b) of the **L_2-th Subscript** of the **Analytic Trajectory Synthesis Arrays** (i.e., the Subscript which the Element Number for the **Plant Input Vector** if $L_1 = 1$, or the **Plant Output Vector**) such as those defined in Group 6 (e.g., the A3(L_2,L_1), IFUNCT(L_3,L_2,L_1), NL3(L_2,L_1), and NN(L_3,L_2,L_1) Arrays) but not necessarily limited to arrays defined in that group (INTEGER*4).
•• NL321	See Def	Equivalent Single Dimension of the **Three-Dimensional Analytic Trajectory Synthesis Arrays** such as those defined in Group 6 (e.g., the IFUNCT(L_3,L_2,L_1) and NN(L_3,L_2,L_1) Arrays) but not necessarily limited to arrays defined in that group (INTEGER*4). NL321 = NL3DIM*NL2DIM*NL1DIM
•• NL3DIM	7	Dimension (e.g., the l.u.b) of the **L_3-th Subscript** of the **Analytic Trajectory Synthesis Arrays** (i.e., the Subscript which Specifies the Element Number of the **Analytic Modelling Function** for a specific (L_2,L_1)) such as those defined in Group 6 (e.g., the IFUNCT(L_3,L_2,L_1) and NN(L_3,L_2,L_1) Arrays) but not necessarily limited to arrays defined in that group (INTEGER*4).
•• NLDIM	300	Dimension (i.e., the l.u.b) of the **L-th Subscript** of the **Data Set Arrays** (i.e., the subscript which defines the **Specific Data Set**) in the **Data Sliding Window** such as those defined in Group A (e.g., the TD(L), XD(I,L), and YD(J,L) Arrays) but not necessarily limited to arrays defined in this group (INTEGER*4).

Input Group 1 (Continued)

Dimensions of the Principal Arrays of the OPTIMNN System

Parameter	Default or Initial Value	Definition
** NLTBL	600	Dimension (i.e., the l.u.b) of the **LTBL-th Subscript** of the **Data Set Arrays** (i.e., the subscript which defines the **Specific Data Set**) such as those defined in Group 2 (e.g., the TTBL(LTBL), XTBL(I,LTBL), and YTBL(J,LTBL) Arrays), but not necessarily limited to arrays defined in this group, in the **Plant Model Data Table** used when the "Actual Plant" is modelled using Routine **TSTATE** (INTEGER*4).

Input Group 2

Overall Trajectory (Excluding Learning and Control Trajectories) Propagation Parameters

Parameter	Default or Initial Value	Definition
• CONST1	0.200	Input Constant (REAL*8).
• CONST2	0.500	Input Constant (REAL*8).
• CONST3	0.800	Input Constant (REAL*8).
• CONST4	1.200	Input Constant (REAL*8).
• CONST5	1.500	Input Constant (REAL*8).
I	1	**Index** which Specifies the **I-th Element Position** in the **Plant Input Vectors** (i.e., the **Plant Control Vector)** XA(I), XD(I,L), and XN(I) (INTEGER*4).
J	1	**Index** which Specifies the **J-th Element Position** in the **Plant Output Vectors** (i.e., the **Plant Measurement/State Vector**) YA(J), YD(J,L), and YN(J) (INTEGER*4).
L	1	**Index** which Specifies the **L-th Data Set** in the **Data Sliding Window** (INTEGER*4).
L_1	1	**Index** which **Assigns** the **Analytic Modelling Function** for a specific (L_3,L_2,L_1) **to** either the **Plant Input Vector** (i.e., the Plant Control Vector) **or** the **Plant Output Vector** (i.e., Plant Measurement/State Vector) (INTEGER*4). = 1 Specifies that the Model applies to an element of the **Plant Input Vector** (i.e., the Plant Control Vector). = 2 Specifies that the Model applies to an element of the **Plant Output Vector** (i.e., Plant Measurement/State Vector).
• LARGE1	1.0 D+03	Input Constant with a large value (REAL*8).
• LARGE2	1.0 D+06	Input Constant with a large value (REAL*8).
• LARGE3	1.0 D+09	Input Constant with a large value (REAL*8).
• LARGE4	1.0 D+12	Input Constant with a large value (REAL*8).
LTBL	1	**Index** which Specifies the **LTBL-th Data Set** in the **Plant Model Data Table** when the "Actual Plant" is modelled using Routine **TSTATE** (INTEGER*4).

Input Group 2 (Continued)

Overall Trajectory (Excluding Learning and Control Trajectories) Propagation Parameters

Parameter	Default or Initial Value	Definition
• MULT	0	**Subsequent Case Flag**. MULT is automatically reset to zero after each case is completed. It is necessary to input MULT equal to a positive integer value if it is desired to run a subsequent case with new NAMELIST **CDATA** values. (INTEGER*4).
• NL2(L_1)	1	**Total Number** of **Elements** in the **Plant Input Vector** (i.e., the Plant Control Vector) if $L_1 = 1$, or the **Plant Output Vector** (i.e., Plant Measurement/State Vector) if $L_1 = 2$; NOT to be confused with NL2DIM, the Dimension of the L_2-th Subscript of the Analytic Trajectory Synthesis Arrays (INTEGER*4).
• SMALL1	1.0 D-03	Input Constant with a small value (REAL*8).
• SMALL2	1.0 D-06	Input Constant with a small value (REAL*8).
• SMALL3	1.0 D-09	Input Constant with a small value (REAL*8).
• SMALL4	1.0 D-12	Input Constant with a small value (REAL*8).
••• TBLMAX	1	The **Number of Data Sets** (i.e., the Maximum Value that the index **LTBL** can have) in the **Plant Model Data Table** (INTEGER*4). $1 \leq$ **LTBL** $\leq$ **TBLMAX**
••• TD(L)		Either the **Absolute Time** (TABS) **or** the **Relative Time** (TREL) as appropriately defined by TLTYPE or TCTYPE corresponding to the **"Actual" (Reference) Plant** defined in the **L-th Data Set** in the **Data Sliding Window** (REAL*8).
• TINIT	0.000	**Initial Absolute Time** for the Entire Process (REAL*8).
• TFINL	0.000	**Final Absolute Time** for the Entire Process (REAL*8).
••• TTBL(LTBL)		Either the **Absolute Time** (TABS) **or** the **Relative Time** (TREL) as appropriately defined by TLTYPE or TCTYPE corresponding to the **"Actual" (Reference) Plant** defined in the **LTBL-th Data Set** of the **Plant Model Data Table** used when the "Actual Plant" is modelled using Routine **TSTATE** (REAL*8).

Input Group 2 (Continued)

Overall Trajectory (Excluding Learning and Control Trajectories) Propagation Parameters

Parameter	Default or Initial Value	Definition
••• XD(I,L)		The **I-th Element** of the **Input Vector** (i.e., the **Control Vector**) to the **"Actual" (Reference) Plant** defined in the **L-th Data Set** in the **Data Sliding Window** (REAL*8).
••• XTBL(I,LTBL)		The **I-th Element** of the **Input Vector** (i.e., the **Control Vector**) to the **"Actual" (Reference) Plant** defined in the **LTBL-th Data Set** of the **Plant Model Data Table** used when the "Actual Plant" is modelled using Routine **TSTATE** (REAL*8).
••• YD(J,L)		The **J-th Element** of the **Output Vector** (i.e., the **Measurement/State Vector**) from the **"Actual" (Reference) Plant** defined in the **L-th Data Set** in the **Data Sliding Window** (REAL*8).
••• YTBL(J,LTBL)		The **J-th Element** of the **Output Vector** (i.e., the **Measurement/State Vector**) from the **"Actual" (Reference) Plant** defined in the **LTBL-th Data Set** of the **Plant Model Data Table** used when the "Actual Plant" is modelled using Routine **TSTATE** (REAL*8).

Input Group 3

Learning Trajectory Propagation Parameters

Parameter	Default or Initial Value	Definition
• DLFREQ	1	**Data Set Read Frequency** After the First Neural-Net (NN) CW(I,J,K)s Update During the Learning Trajectory (INTEGER*4).
• DLLGTH	10	**Window Length/Size** During the Learned Trajectory; Number of Read Data Sets Contained in a Window During the Learned Trajectory (INTEGER*4).
L	1	**Index** which Specifies the **L-th Data Set** in the **Data Sliding Window** (INTEGER*4).
• LDELAY	0	**Data Set Read Delay Counter Limit** During the Learning Trajectory (INTEGER*4).
• NNLID	1	Neural-Net (NN) **CW(I,J,K)s Update Frequency/Inhibit Flag** for the Learning Trajectory (INTEGER*4). ≤ 0 Do <u>Not</u> Update NN CW(I,J,K)s during the Learning Trajectory. > 0 Update NN CW(I,J,K)s every NNLID times during the Learning Trajectory.
• STMODL	1	Specifies the **"Actual" (Reference) Plant Model Option** during the Learning Trajectory (INTEGER*4). = 1 **Synthesise** the "Actual" (Reference) Plant Model by **Combining Selected Individual Analytic Models**, that is Model the "Actual" Plant by using Routine **ASTATE**. = 2 **Define** the "Actual" (Reference) Plant Model directly from **On-Line Test Data**, that is Model the "Actual" Plant by using Routine **DSTATE**. = 3 **Define** the "Actual" (Reference) Plant Model from **Stored Data Tables**, that is Model the "Actual" Plant by using Routine **TSTATE**. = 4 **Define** the Actual (Reference) Plant Model from a **User Supplied Model**, that is Model the "Actual" Plant by using Routine **USTATE**.
• TLINIT	0.000	Either the **Initial** value of the **Absolute Time** (i.e., **TABS**) or the **Initial** value of the **Relative Time** (i.e., **TREL**) as appropriately defined by TLTYPE for the Learning Trajectory (REAL*8).
• TLFINL	0.000	Either the **Final** value of the **Absolute Time** (i.e., **TABS**) or the **Final** value of the **Relative Time** (i.e., **TREL**) as appropriately defined by TLTYPE for the Learning Trajectory (REAL*8).

Input Group 3 (Continued)

Learning Trajectory Propagation Parameters

Parameter	Default or Initial Value	Definition
• TLSTEP	1.000	**Time Step** for the Learning Trajectory (REAL*8).
• TLTYPE	0	**Time Type** Definition Flag for the Learning Trajectory (INTEGER*4). ≤ 0 **T** = the Current **Absolute Time** (i.e., **T = TABS**), that is time is measured from the Start of the Entire Process (i.e., time is measured from that defined by TLINIT) > 0 **T** = the Current **Relative Time** (i.e., **T = TREL**), that is time is measured from the Start of the Learning Trajectory (i.e., time is measured from that defined by TLINIT).
• WTSNNL(L)	1.000	**Weighting Coefficient** of the **L-th Data Set** in the **Data Sliding Window** during the Learning Trajectory (REAL*8).

<u>**Input Group 4**</u>

<u>**Controlled Trajectory Propagation Parameters**</u>

Parameter	Default or Initial Value	Definition
• CDELAY	0	**Data Set Read Delay Counter Limit** During the Controlled Trajectory (INTEGER*4).
• CVTID	1	**Plant Input Vector** (i.e., the Plant Control Vector) **XA(•) Update Inhibit Flag** for the Controlled Trajectory (INTEGER*4). ≤ 0 Do <u>Not</u> Update XA(•) during the Controlled Trajectory. > 0 Update XA(•) every CVTID times during the Controlled Trajectory.
• DCFREQ	1	**Data Set Read Frequency** After the First Neural-Net (NN) CW(I,J,K)s Update During the Controlled Trajectory (INTEGER*4).
• DCLGTH	10	**Window Length/Size** During the Controlled Trajectory; Number of Read Data Sets Contained in a Window During the Controlled Trajectory. If DCLGTH is Input Less Than or Equal to Zero, DCLGTH is Reset Equal To DLLGTH (INTEGER*4).
• ISTEP0	1	**Step Reset Inhibit Flag** for the Controlled Trajectory (INTEGER*4). = 0 **and only if** STMODC ≠ 2 or 3, Reset ISTEP to ISTEP = 1 at the start of the Controlled Trajectory. ≠ 0 **or if** STMODC = 2 or 3, Reset ISTEP to STEP = ISTEP + ISTEP0 at the start of the Controlled Trajectory.
L	1	**Index** which Specifies the **L-th Data Set** in the **Data Sliding Window** (INTEGER*4).
• NNCID	1	Neural-Net (NN) **CW(I,J,K)s Update Frequency/Inhibit Flag** for the Controlled Trajectory (INTEGER*4). ≤ 0 Do <u>Not</u> Update NN CW(I,J,K)s during the Controlled Trajectory. > 0 Update NN CW(I,J,K)s every NNLID times during the Controlled Trajectory.
• STMODC	1	Specifies the **"Actual" (Reference) Plant Model Option** during the Controlled Trajectory (INTEGER*4). = 1 **Synthesise** the "Actual" (Reference) Plant Model by **Combining Selected Individual Analytic Models**, that is Model the "Actual" Plant by using Routine **ASTATE**.

Input Group 4 (Continued)

Controlled Trajectory Propagation Parameters

Parameter	Default or Initial Value	Definition
• STMODC	1	(Continued) = 2 **Define** the "Actual" (Reference) Plant Model directly from **On-Line Test Data**, that is Model the "Actual" Plant by using Routine **DSTATE**. = 3 **Define** the "Actual" (Reference) Plant Model from **Stored Data Tables**, that is Model the "Actual" Plant by using Routine **TSTATE**. = 4 **Define** the Actual (Reference) Plant Model from a **User Supplied Model**, that is Model the "Actual" Plant by using Routine **USTATE**.
• TCINIT	0.000	Either the **Initial** value of the **Absolute Time** (i.e., **TABS**) or the **Initial** value of the **Relative Time** (i.e., **TREL**) as appropriately defined by TCTYPE for the Controlled Trajectory (REAL*8).
• TCFINL	0.000	Either the **Final** value of the **Absolute Time** (i.e., **TABS**) or the **Final** value of the **Relative Time** (i.e., **TREL**) as appropriately defined by TCTYPE for the Controlled Trajectory (REAL*8).
• TCSTEP	1.000	**Time Step** for the Controlled Trajectory (REAL*8).
• TCTYPE	0	**Time Type** Definition Flag for the Learning Trajectory (INTEGER*4). ≤ 0 **T** = the Current **Absolute Time** (i.e., **T** = **TABS**), that is time is measured from the Start of the Entire Process (i.e., time is measured from that defined by TLINIT) > 0 **T** = the Current **Relative Time** (i.e., **T** = **TREL**), that is time is measured from the Start of the Controlled Trajectory (i.e., time is measured from that defined by TCINIT).
• UPDATE	1	**Sliding Window First Data Set Update Flag** for the Controlled Trajectory (INTEGER*4). ≤ 0 Do Not Update the First Data Set (L = 1) of the Sliding Window Table (i.e., XD(I,1) and YD(J,1)) to those determined by the Current Control Optimisation (i.e., XN(I) and YN(J)). > 0 Update the First Data Set (L = 1) of the Sliding Window Table (i.e., XD(I,1) and YD(J,1)) to those determined by the Current Control Optimisation (i.e., XN(I) and YN(J)).

Input Group 4 (Continued)

Controlled Trajectory Propagation Parameters

Parameter	Default or Initial Value	Definition
• WTSNNC(L)	1.000	**Weighting Coefficient** of the **L-th Data Set** in the **Data Sliding Window** during the Controlled Trajectory (REAL*8).

Input Group 5

Neural Network Parameters

Parameter	Default or Initial Value	Definition
• AN(J,K)	0.500	A **Constant** of the **Pass-Through Function** (Node Filter) Model at the **J-th Exit (Destination) Position** for the **K-th Neural-Net Layer** (REAL*8).
• BN(J,K)	0.500	A **Constant** of the **Pass-Through Function** (Node Filter) Model at the **J-th Exit (Destination) Position** for the **K-th Neural-Net Layer** (REAL*8).
• CN(J,K)	1.000	A **Constant** of the **Pass-Through Function** (Node Filter) Model at the **J-th Exit (Destination) Position** for the **K-th Neural-Net Layer** (REAL*8).
• CW(I,J,K)	1.000	**Neural-Signal Weighting Coefficient** associated with the **Entry Signal XNN(I,J,K)** to the **K-th Neural-Net Layer** from the **I-th Entry (Origin) Position** directed to the **J-th Exit (Destination) Position** (REAL*8).
• DN(J,K)	-1.0D+06	A **Constant** of the **Pass-Through Function** (Node Filter) Model at the **J-th Exit (Destination) Position** for the **K-th Neural-Net Layer** (REAL*8).
I	1	**Index** which Specifies the **I-th Element Position** in the **Origin Vector** for the **Specific Neural-Network Layer** specified by the index **K** (INTEGER*4).
J	1	**Index** which Specifies the **J-th Element Position** in the **Destination Vector** for the **Specific Neural-Network Layer** specified by the index **K** (INTEGER*4).
K	1	**Index** which Specifies the **K-th Specific Layer** in the **Neural-Network** (INTEGER*4).
• NFUNCT(J,K)	0	Specifies the **Pass-Through Function (Node Filter) Model** at the **J-th Exit (Destination) Position** for the **K-th Neural-Net Layer** (INTEGER*4). = 0 Specifies the **No-Pass** (i.e., the **Constant** Function) Node Filter Function defined by Routine **PFNCT00**. = 1 Specifies the **Direct-Pass** (i.e., the **Linear** Function) Node Filter Function defined by Routine **PFNCT01**. = 2 Specifies the **Hyperbolic Tangent** (i.e., the **Threshold** Function) Node Filter Function defined by Routine **PFNCT02**.

Input Group 5 (Continued)

Neural Network Parameters

Parameter	Default or Initial Value	Definition
• NFUNCT(J,K)	0	(Continued) = 3 Specifies the **First Derivative of the Hyperbolic Tangent** (i.e., the **Pulse** Function) Node Filter Function defined by Routine **PFNCT03**.
• NI(K)	3	**Total Number of Elements** in the **Origin Vector** (i.e., the Total Number of Origin Positions) for the Specific Neural-Network Layer specified by the index **K** (INTEGER*4).
• NJ(K)	1	**Total Number of Elements** in the **Destination Vector** (i.e., the Total Number of Destination Positions) for the Specific Neural-Network Layer specified by the index **K** (INTEGER*4).
• NK	2	**Total Number of Layers** (i.e., the l.u.b. of the index **K**) in the Neural-Network (INTEGER*4).
• XN0(J,K)	0.000	The Input Signal UNN(J,K) **Horizontal Translation Constant** for the **Origin** of the **Pass-Through Function** (Node Filter) at the **J-th Exit (Destination) Position** for the **K-th Neural-Net Layer** (REAL*8).
• YN0(J,K)	0.000	The Exit Signal YNN(J,K) **Vertical Translation Constant** for the **Origin** of the **Pass-Through Function** (Node Filter) at the **J-th Exit (Destination) Position** for the **K-th Neural-Net Layer** (REAL*8).

Input Group 6

Analytic Trajectory Synthesis Parameters

Parameter	Default or Initial Value	Definition
• $A(L_3,L_2,L_1)$	0.500	A **Constant** of the **Analytic Trajectory Modelling Functions**. (REAL*8).
• $A1(L_3,L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called as a **Primary Analytic Model**. (REAL*8).
• $A2(L_3,L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called to **Randomise an Individual Primary Analytic Model**. (REAL*8).
• $A3(L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called to **Randomise the Combined Primary Analytic Models**. (REAL*8).
• $ALPHA(L_3,L_2,L_1)$	1.000	The **Convergence/Divergence Constant** of the **Exponential Part** of the **Enveloped Sinusoidal** Analytic Trajectory Modelling Function (REAL*8).
• $B(L_3,L_2,L_1)$	0.500	A **Constant** of the **Analytic Trajectory Modelling Functions**. (REAL*8).
• $B1(L_3,L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called as a **Primary Analytic Model**. (REAL*8).
• $B2(L_3,L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called to **Randomise an Individual Primary Analytic Model**. (REAL*8).
• $B3(L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called to **Randomise the Combined Primary Analytic Models**. (REAL*8).
• $C(L_3,L_2,L_1)$	0.250	A **Constant** of the **Analytic Trajectory Modelling Functions**. (REAL*8).
• $C1(L_3,L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called as a **Primary Analytic Model**. (REAL*8).

Input Group 6 (Continued)

Analytic Trajectory Synthesis Parameters

Parameter	Default or Initial Value	Definition
• $C2(L_3,L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called to **Randomise an Individual Primary Analytic Model**. (REAL*8).
• $C3(L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called to **Randomise the Combined Primary Analytic Models**. (REAL*8).
• $D(L_3,L_2,L_1)$	-1.0D+06	A **Constant** of the **Analytic Trajectory Modelling Functions**. (REAL*8).
• $D1(L_3,L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called as a **Primary Analytic Model**. (REAL*8).
• $D2(L_3,L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called to **Randomise an Individual Primary Analytic Model**. (REAL*8).
• $D3(L_2,L_1)$	0.000	A **Constant** of the **Random Uniform Distribution** Analytic Trajectory Modelling Function when this function is called to **Randomise the Combined Primary Analytic Models**. (REAL*8).
• $IFUNCT(L_3,L_2,L_1)$	0	Specifies the **Analytic Function Model** for a Specific (L_3,L_2,L_1) (INTEGER*4)..

= 0 Specifies the **Random Uniform Distribution** Function defined by Routine **ASTATRAN**.

= 1 Specifies the **Linear** Function (i.e., the **Ramp** Function) defined by Routine **ASTATE01**.

= 2 Specifies the **Serpentine Curve** Function defined by Routine **ASTATE02**.

= 3 Specifies the **Witch of Agnesi** Function defined by Routine **ASTATE03**.

= 4 Specifies the **Inverted Witch of Agnesi** Function defined by Routine **ASTATE04**.

= 5 Specifies the **Enveloped Sinusoidal** Function defined by Routine **ASTATE05**.

= 6 Specifies the **Hyperbolic Tangent** Function (i.e., the **Threshold** Function) defined by Routine **ASTATE06**.

= 7 Specifies the **First Derivative of the Hyperbolic Tangent Function** (i.e., the **Pulse** Function) defined by Routine **ASTATE07**.

Input Group 6 (Continued)

Analytic Trajectory Synthesis Parameters

Parameter	Default or Initial Value	Definition
• $ISEED1(L_3,L_2,L_1)$	78985723	The **Seed** required for the **First Call** to the VAX/VMS System Subroutine RAN from Subroutine ASTATRAN when ASTATRAN is called as a Primary Analytic Model. This seed is updated automatically upon exit from RAN. Although there are no restrictions on the value of this seed other than that it is an INTEGER*4 variable, the best results are obtained when it is initially input as a large odd integer (INTEGER*4).
• $ISEED2(L_3,L_2,L_1)$	81692875	The **Seed** required for the **First Call** to the VAX/VMS System Subroutine RAN from Subroutine ASTATRAN when ASTATRAN is called to Randomise an Individual Primary Analytic Model. This seed is updated automatically upon exit from RAN. Although there are no restrictions on the value of this seed other than that it is an INTEGER*4 variable, the best results are obtained when it is initially input as a large odd integer (INTEGER*4).
• $ISEED3(L_2,L_1)$	72919329	The **Seed** required for the **First Call** to the VAX/VMS System Subroutine RAN from Subroutine ASTATRAN when ASTATRAN is called to Randomise the Combined Primary Analytic Models. This seed is updated automatically upon exit from RAN. Although there are no restrictions on the value of this seed other than that it is an INTEGER*4 variable, the best results are obtained when it is initially input as a large odd integer (INTEGER*4).
• $JSEED1(L_3,L_2,L_1)$	95428381	The **Seed** required for the **Second Call** to the VAX/VMS System Subroutine RAN from Subroutine ASTATRAN when ASTATRAN is called as a Primary Analytic Model. This seed is updated automatically upon exit from RAN. Although there are no restrictions on the value of this seed other than that it is an INTEGER*4 variable, the best results are obtained when it is initially input as a large odd integer (INTEGER*4).
• $JSEED2(L_3,L_2,L_1)$	68377297	The **Seed** required for the **Second Call** to the VAX/VMS System Subroutine RAN from Subroutine ASTATRAN when ASTATRAN is called to Randomise an Individual Primary Analytic Model. This seed is updated automatically upon exit from RAN. Although there are no restrictions on the value of this seed other than that it is an INTEGER*4 variable, the best results are obtained when it is initially input as a large odd integer (INTEGER*4).

Input Group 6 (Continued)

Analytic Trajectory Synthesis Parameters

Parameter	Default or Initial Value	Definition
• JSEED3(L_2,L_1)	89672847	The **Seed** required for the **Second Call** to the VAX/VMS System Subroutine RAN from Subroutine ASTATRAN when ASTATRAN is called to Randomise the Combined Primary Analytic Models. This seed is updated automatically upon exit from RAN. Although there are no restrictions on the value of this seed other than that it is an INTEGER*4 variable, the best results are obtained when it is initially input as a large odd integer (INTEGER*4).
L_1	1	**Index** which **Assigns** the **Analytic Modelling Function** for a specific (L_3,L_2,L_1) **to** either the **Plant Input Vector** (i.e., the Plant Control Vector) **or** the **Plant Output Vector** (i.e., Plant Measurement/State Vector) (INTEGER*4). = 1 Specifies that the Model applies to an element of the **Plant Input Vector** (i.e., the Plant Control Vector). = 2 Specifies that the Model applies to an element of the **Plant Output Vector** (i.e., Plant Measurement/State Vector).
L_2	1	**Index** which Specifies the **Element Number** for the **Plant Input Vector** (i.e., the Plant Control Vector) if $L_1 = 1$, **or** the **Plant Output Vector** (i.e., Plant Measurement/State Vector) if $L_1 = 2$ (INTEGER*4).
L_3	1	**Index** which Specifies the **Element Number** of the **Analytic Modelling Function** for a specific (L_2,L_1) (INTEGER*4).
• NL2(L_1)	1	**Total Number** of **Elements** in the **Plant Input Vector** (i.e., the Plant Control Vector) if $L_1 = 1$, **or** the **Plant Output Vector** (i.e., Plant Measurement/State Vector) if $L_1 = 2$; NOT to be confused with NL2DIM, the Dimension of the L_2-th Subscript of the Analytic Trajectory Synthesis Arrays (INTEGER*4).
• NL3(L_2,L_1)	1	**Total Number** of **Analytic Modelling Functions** for a specific (L_2,L_1); NOT to be confused with NL3DIM, the Dimension of the L_3-th Subscript of the Analytic Trajectory Synthesis Arrays (INTEGER*4).
• NN(L_3,L_2,L_1)	1.000	**Harmonic Number** of the **Primary Frequency** (i.e., the Coefficient of the Primary Frequency) of the **Sinusoidal Part** of the **Enveloped Sinusoidal** Analytic Trajectory Modelling Function (REAL*8).

Input Group 6 (Continued)

Analytic Trajectory Synthesis Parameters

Parameter	Default or Initial Value	Definition
• OMEGA(L_3,L_2,L_1)	2π	**2π times** the **Primary Frequency** (or the **Period** if NN(L_3,L_2,L_1) $\geq 10^8$) of the **Sinusoidal Part** of the **Enveloped Sinusoidal** Analytic Trajectory Modelling Function (REAL*8).
• PERIOD(L_3,L_2,L_1)	$1.0\times10^{+10}$	**Period** of the **Analytic Trajectory Modelling Function** defined for a specific (L_3,L_2,L_1) (REAL*8).
• PHASE(L_3,L_2,L_1)	0.000	Either the **Absolute Time** (TABS) or the **Relative Time** (TREL) as appropriately defined by TLTYPE or TCTYPE of the **Start of a Cycle** of the **Analytic Modelling Function** defined for a specific (L_3,L_2,L_1) (REAL*8).
• PHI(L_3,L_2,L_1)	0.000	The **Phase Angle** of the **Sinusoidal Part** of the **Enveloped Sinusoidal** Analytic Trajectory Modelling Function (REAL*8).
• PSI(L_3,L_2,L_1)	0.000	The **Phase Angle** of the **Exponential Part** of the **Enveloped Sinusoidal** Analytic Trajectory Modelling Function (REAL*8).
• X0(L_3,L_2,L_1)	0.000	Either the **Absolute Time** (TABS) or the **Relative Time** (TREL) as appropriately defined by TLTYPE or TCTYPE of the **Origin** of the **Analytic Trajectory Modelling Function** defined for a specific (L_3,L_2,L_1) (REAL*8).
• Y0(L_3,L_2,L_1)	0.000	A **Constant** added to the **Analytic Trajectory Modelling Function** defined for a specific (L_3,L_2,L_1) (REAL*8).
• YR1(L_3,L_2,L_1)	0.000	A **Constant** added to the **Random Uniform Distribution** Analytic Trajectory Modelling Function defined for a specific (L_3,L_2,L_1) when this function is called as a **Primary Analytic Model**. (REAL*8).
• YR2(L_3,L_2,L_1)	0.000	A **Constant** added to the **Random Uniform Distribution** Analytic Trajectory Modelling Function defined for a specific (L_3,L_2,L_1) when this function is called to **Randomise an Individual Primary Analytic Model**. (REAL*8).
• YR3(L_2,L_1)	0.000	A **Constant** added to the **Random Uniform Distribution** Analytic Trajectory Modelling Function defined for a specific (L_3,L_2,L_1) when this function is called to **Randomise the Combined Primary Analytic Models**. (REAL*8).

Input Group 7

Neural-Net Optimisation Parameters During the Learning Trajectory

Parameter	Default or Initial Value	Definition
Control Vector Sub-Group		
I	1	**Index** which Specifies the **I-th Element Position** in the **Origin Vector** for the **Specific Neural-Network Layer** specified by the index "K" (INTEGER*4).
• IJKCVL(I,J,K)	0	**Control Vector Identification Flag**. This flag vector specifies whether or not the **Neural-Signal Weighting Coefficient** (i.e., **CW(I,J,K)**) associated with the Entry Signal XNN(I,J,K) to the **K-th Neural-Net Layer** from the **I-th Entry (Origin) Position** directed to the **J-th Exit (Destination) Position** will be an element in the Optimisation Control Vector **CV(•)** (INTEGER*4). ≤ 0 CW(I,J,K) IS NOT an Element of CV(•). > 0 CW(I,J,K) IS an Element of CV(•).
J	1	**Index** which Specifies the **J-th Element Position** in the **Destination Vector** for the **Specific Neural-Network Layer** specified by the index "K" (INTEGER*4).
K	1	**Index** which Specifies the **K-th Specific Layer** in the **Neural-Network** (INTEGER*4).
• SCVNNL(I,J,K)	1.000	The **Vector of Scaling Coefficients** for the elements of the **Neural-Signal Weighting Coefficient Matrix** (i.e., **CW(I,J,K)**) required by the Optimisation Process (REAL* 8).
End Conditions Vector Sub-Group		
J	1	**Index** which Specifies the **J-th Element Position** in the **Plant Output Vectors** (i.e., the **Plant Measurement/State Vector**) YA(J), YD(J,L), and YN(J) (INTEGER*4).
• JJECL(J)	0	**End Conditions Identification Flag.** This flag vector specifies whether or not **[YN(J) - YA(J)]** (i.e., the **difference** between the **J-th Elements** of the **Measurement/State Vectors**) will be an element in the Optimisation End Conditions Vector **EC(•)** and if **WTNNL(J)*[YN(J) - YA(J)]^2** will be a term in the Performance Index **PINDX** (INTEGER*4).

Input Group 7 (Continued)

Neural-Net Optimisation Parameters During the Learning Trajectory

Parameter	Default or Initial Value	Definition

End Conditions Vector Sub-Group (Continued)

• JJECL(J)

(Continued)

≤ 0 [YN(J) - YA(J)] IS NOT an Element of EC(•) and WTNNL*[YN(J) - YA(J)]^2 IS NOT a term in PINDX.

> 0 [YN(J) - YA(J)] IS an Element of EC(•) and WTNNL*[YN(J) - YA(J)]^2 IS a term in PINDX.

• WTNNL(J) 1.000

Weighting Coefficient element in the **WTNNL(J)*[YN(J) - YA(J)]^2** term in the Performance Index **PINDX** (REAL* 8).

Constraint Vector Sub-Group

• AMAXNNL(I,J,K) 100.00

The **Least Upper Bounds** (l.u.b.) of the **Control Vector Elements** (REAL* 8).

CW(I,J,K) ≤ AMAXNNL(I,J,K)

• AMINNNL(I,J,K) -100.00

The **Greatest Least Bounds** (g.l.b.) of the **Control Vector Elements** (REAL* 8).

AMINNNL(I,J,K) ≤ CW(I,J,K)

I 1

Index which Specifies the **I-th Element Position** in the **Origin Vector** for the **Specific Neural-Network Layer** specified by the index "K" (INTEGER*4).

• ICONNNL(I,J,K) 0

Constraint Function Vector Identification Flag. This flag vector specifies whether or not the **Neural-Signal Weighting Coefficient** (i.e., **CW(I,J,K)**) associated with the Entry Signal XNN(I,J,K) to the **K-th Neural-Net Layer** from the **I-th Entry (Origin) Position** directed to the **J-th Exit (Destination) Position** will be constrained by an element of the Constraint Function Vector CON(•) (INTEGER*4). **Currently NOT an option**

≤ 0 CW(I,J,K) IS NOT Constrained by an element of CON(•).

> 0 CW(I,J,K) IS Constrained by an element of CON(•).

Input Group 7 (Continued)

Neural-Net Optimisation Parameters During the Learning Trajectory

Parameter	Default or Initial Value	Definition

Constraint Vector Sub-Group (Continued)

Parameter	Default or Initial Value	Definition
J	1	**Index** which Specifies the **J-th Element Position** in the **Destination Vector** for the **Specific Neural-Network Layer** specified by the index "K" (INTEGER*4).
K	1	**Index** which Specifies the **K-th Specific Layer** in the **Neural-Network** (INTEGER*4).

Optimisation Parameters Sub-Group

Parameter	Default or Initial Value	Definition
• IOPTNNL	0	**Gradient Evaluation Option** Specification Flag (INTEGER*4). ≤ 0 **No** Neural-Net Update/Optimisation. = 1 The Gradients required during the Neural-Net Update/Optimisation Process are evaluated using a **Finite Differences Method**. = 2 The Gradients required during the Neural-Net Update/Optimisation Process are evaluated using an **Analytic Method**. **Currently NOT an option**
• MITNNNL	200	The **Maximum Number of Optimisation Iterations** allowed before the Optimisation Process is terminated. (INTEGER*4).
• OUTNNL	0	The **Optimisation Iteration Output Level Specification Flag**. (INTEGER*4). = 0 **No** Optimisation Iteration Information is written. = 1 Only the **Final** Optimisation Iteration **Convergence Information** is written. = 2 **One Line of Intermediate** Optimisation Iteration Information is written for **Each Iteration**. = 3 **Detailed Intermediate** Optimisation Iteration Information is written for **Each Iteration**.

Input Group 8

Neural-Net Optimisation Parameters During the Controlled Trajectory

Parameter	Default or Initial Value	Definition
Control Vector Sub-Group		
I	1	**Index** which Specifies the **I-th Element Position** in the **Origin Vector** for the **Specific Neural-Network Layer** specified by the index "K" (INTEGER*4).
• IJKCVC(I,J,K)	0	**Control Vector Identification Flag**. This flag vector specifies whether or not the **Neural-Signal Weighting Coefficient** (i.e., **CW(I,J,K)**) associated with the Entry Signal XNN(I,J,K) to the **K-th Neural-Net Layer** from the **I-th Entry (Origin) Position** directed to the **J-th Exit (Destination) Position** will be an element in the Optimisation Control Vector **CV(•)** (INTEGER*4). ≤ 0 CW(I,J,K) IS NOT an Element of CV(•). > 0 CW(I,J,K) IS an Element of CV(•).
J	1	**Index** which Specifies the **J-th Element Position** in the **Destination Vector** for the **Specific Neural-Network Layer** specified by the index "K" (INTEGER*4).
K	1	**Index** which Specifies the **K-th Specific Layer** in the **Neural-Network** (INTEGER*4).
• SCVNNC(I,J,K)	1.000	The **Vector** of **Scaling Coefficients** for the elements of the **Neural-Signal Weighting Coefficient Matrix** (i.e., **CW(I,J,K)**) required by the Optimisation Process (REAL* 8).
End Conditions Vector Sub-Group		
J	1	**Index** which Specifies the **J-th Element Position** in the **Plant Output Vectors** (i.e., the **Plant Measurement/State Vector**) YA(J), YD(J,L), and YN(J) (INTEGER*4).
• JJECC(J)	0	**End Conditions Identification Flag.** This flag vector specifies whether or not **[YN(J) - YA(J)]** (i.e., the **difference** between the **J-th Elements** of the **Measurement/State Vectors**) will be an element in the Optimisation End Conditions Vector **EC(•)** and if **WTNNC(J)*[YN(J) - YA(J)]^2** will be a term in the Performance Index **PINDX** (INTEGER*4).

Input Group 8 (Continued)

Neural-Net Optimisation Parameters During the Controlled Trajectory

Parameter	Default or Initial Value	Definition

End Conditions Vector Sub-Group (Continued)

• JJECC(J) — (Continued)

- ≤ 0 — [YN(J) - YA(J)] IS NOT an Element of EC(•) and WTNNC*$[YN(J) - YA(J)]^2$ IS NOT a term in PINDX.
- > 0 — [YN(J) - YA(J)] IS an Element of EC(•) and WTNNC*$[YN(J) - YA(J)]^2$ IS a term in PINDX.

• WTNNC(J) — 1.000 — **Weighting Coefficient** element in the **WTNNC(J)*$[YN(J) - YA(J)]^2$** term in the Performance Index **PINDX** (REAL* 8).

Constraint Vector Sub-Group

• AMAXNNC(I,J,K) — 100.00 — The **Least Upper Bounds** (l.u.b.) of the **Control Vector Elements** (REAL* 8).

CW(I,J,K) ≤ AMAXNNC(I,J,K)

• AMINNNC(I,J,K) — -100.00 — The **Greatest Least Bounds** (g.l.b.) of the **Control Vector Elements** (REAL* 8).

AMINNNC(I,J,K) ≤ CW(I,J,K)

I — 1 — **Index** which Specifies the **I-th Element Position** in the **Origin Vector** for the **Specific Neural-Network Layer** specified by the index "K" (INTEGER*4).

• ICONNNC(I,J,K) — 0 — **Constraint Function Vector Identification Flag**. This flag vector specifies whether or not the **Neural-Signal Weighting Coefficient** (i.e., **CW(I,J,K)**) associated with the Entry Signal XNN(I,J,K) to the **K-th Neural-Net Layer** from the **I-th Entry (Origin) Position** directed to the **J-th Exit (Destination) Position** will be constrained by an element of the Constraint Function Vector CON(•) (INTEGER*4). Currently NOT an option

- ≤ 0 — CW(I,J,K) IS NOT Constrained by an element of CON(•).
- > 0 — CW(I,J,K) IS Constrained by an element of CON(•).

Input Group 8 (Continued)

Neural-Net Optimisation Parameters During the Controlled Trajectory

Parameter	Default or Initial Value	Definition

Constraint Vector Sub-Group (Continued)

Parameter	Default or Initial Value	Definition
J	1	**Index** which Specifies the **J-th Element Position** in the **Destination Vector** for the **Specific Neural-Network Layer** specified by the index "K" (INTEGER*4).
K	1	**Index** which Specifies the **K-th Specific Layer** in the **Neural-Network** (INTEGER*4).

Optimisation Parameters Sub-Group

Parameter	Default or Initial Value	Definition
• IOPTNNC	0	**Gradient Evaluation Option** Specification Flag (INTEGER*4). ≤ 0 **No** Neural-Net Update/Optimisation. = 1 The Gradients required during the Neural-Net Update/Optimisation Process are evaluated using a **Finite Differences Method.** = 2 The Gradients required during the Neural-Net Update/Optimisation Process are evaluated using an **Analytic Method.** <u>Currently NOT an option</u>
• MITNNNC	200	The **Maximum Number of Optimisation Iterations** allowed before the Optimisation Process is terminated. (INTEGER*4).
• OUTNNC	0	The **Optimisation Iteration Output Level Specification Flag.** (INTEGER*4). = 0 **No** Optimisation Iteration Information is written. = 1 Only the **Final** Optimisation Iteration **Convergence Information** is written. = 2 **One Line** of **Intermediate** Optimisation Iteration Information is written for **Each Iteration**. = 3 **Detailed Intermediate** Optimisation Iteration Information is written for **Each Iteration**.

Input Group 9

Control Optimisation Parameters During the Controlled Trajectory

Parameter	Default or Initial Value	Definition
Control Vector Sub-Group		
I	1	**Index** which Specifies the **I-th Element Position** in the **Plant Input Vectors** (i.e., the **Plant Control Vector**) XA(I), XD(I,L), and XN(I) (INTEGER*4).
• ICV(I)	0	**Control Vector Identification Flag**. This flag vector specifies whether or not the **I-th Element** of the **Actual/Working Control Vector** (e.g., **XA(I)**) will be an element in the Optimisation Control Vector **CV(•)** (INTEGER*4). ≤ 0 XA(I) IS NOT an Element of CV(•). > 0 XA(I) IS an Element of CV(•).
• SCVC(I)	1.000	The **Vector** of **Scaling Coefficients** for the elements of the **Actual/Working Control Vector** (e.g., **XA(I)**) required by the Optimisation Process (REAL* 8).
End Conditions Vector Sub-Group		
J	1	**Index** which Specifies the **J-th Element Position** in the **Plant Output Vectors** (i.e., the **Plant Measurement/State Vector**) YA(J), YD(J,L), and YN(J) (INTEGER*4).
• JEC(J)	0	**End Conditions Identification Flag**. This flag vector specifies whether or not the **J-th Element** of the **Actual/Working Measurement/State Vector** (i.e., **YA(J)**) will be an element in the Optimisation End Conditions Vector **EC(•)** and if **WTC(J)*[YA(J)]**^{2} will be a term in the Performance Index **PINDX** (INTEGER*4). ≤ 0 YA(J) IS NOT an Element of EC(•) and WTC*[YA(J)]^2 IS NOT a term in PINDX. > 0 YA(J) IS an Element of EC(•) and WTC*[YA(J)]^2 IS a term in PINDX.
• WTC(J)	1.000	**Weighting Coefficient** element in the **WTC(J)*[YA(J)]**^{2} term in the Performance Index **PINDX** (REAL* 8).

Input Group 9 (Continued)

Control Optimisation Parameters During the Controlled Trajectory

Parameter	Default or Initial Value	Definition
Constraint Vector Sub-Group		
• AMAXC(I)	10.00	The **Least Upper Bounds** (l.u.b.) of the **Control Vector Elements** (REAL* 8). $XA(I) \leq AMAXC(I)$
• AMINC(I)	-10.00	The **Greatest Least Bounds** (g.l.b.) of the **Control Vector Elements** (REAL* 8). $AMINC(I) \leq XA(I)$
I	1	**Index** which Specifies the **I-th Element Position** in the **Plant Input Vectors** (i.e., the **Plant Control Vector)** XA(I), XD(I,L), and XN(I) (INTEGER*4).
• ICONC(I)	0	**Constraint Function Vector Identification Flag.** This flag vector specifies whether or not the **I-th Element** of the **Actual/Working Control Vector** (e.g., **XA(I))** will be constrained by an element of the Constraint Function Vector CON(•) INTEGER*4). < 0 XA(I) IS Constrained in an element of CON(•). <u>**Currently NOT an option**</u> ≤ 0 XA(I) IS NOT Constrained in an element of CON(•). > 0 XA(I) and XA(IARG) ARE Constrained in an element of CON (•) according to:. $[XA(I)]^2 + [XA(IARG)]^2 \leq [SMAXC(I)]^2$ where: IARG = ICONC(I)
• SMAXC(I)	10.00	The **Least Upper Bound** (l.u.b.) **Constraint Vector** for the **sum of the squares** of **two elements** of the **Actual/Working Control Vector** (see ICONC(I)) (REAL* 8). $[XA(I)]^2 + [XA(IARG)]^2 \leq [SMAXC(I)]^2$ where: IARG = ICONC(I)

Input Group 9 (Continued)

Control Optimisation Parameters During the Learning Trajectory

Parameter	Default or Initial Value	Definition

Optimisation Parameters Sub-Group

- IOPTC — 0 — **Gradient Evaluation Option** Specification Flag (INTEGER*4).
  - ≤ 0 **No** Control Optimisation.
  - = 1 The Gradients required during the Control Optimisation Process are evaluated using a **Finite Differences Method**.
  - = 2 The Gradients required during the Control Optimisation Process are evaluated using an **Analytic Method**. Currently NOT an option

- MITNC — 200 — The **Maximum Number of Optimisation Iterations** allowed before the Optimisation Process is terminated. (INTEGER*4).

- OUTC — 0 — The **Optimisation Iteration Output Level Specification Flag**. (INTEGER*4).
  - = 0 **No** Optimisation Iteration Information is written.
  - = 1 Only the **Final** Optimisation Iteration **Convergence Information** is written.
  - = 2 **One Line** of **Intermediate** Optimisation Iteration Information is written for **Each Iteration**.
  - = 3 **Detailed Intermediate** Optimisation Iteration Information is written for **Each Iteration**.

Internally Set Parameters

in the

OPTIMNN Code

•••Denotes Data Defined by **NAMELIST CDATA Input Data** read from Subroutine **INIT,** or by **Directly Read On-Line Test Data,** or by **Internally Computed Data**. This Data defines the **"Actual"** (Reference) **Plant Input Vector** and/or **Output Vector** at Specified Trajectory Time Points.

Internally Set Parameters Group A

Internally Set Parameters for Trajectory Propagation

Parameter	Default or Initial Value	Definition
CVUP		**Plant Input Vector** (i.e., the **Plant Control Vector) XA(I) Update Frequency Flag** for the Controlled Trajectory (INTEGER*4). CVUP = JMOD(ISTEP-1, CVTID) = 0 Update the Plant Input Vector XA(I). ≠ 0 Do Not Update the Plant Input Vector XA(I).
DATAR		**Data Set Read Flag** (INTEGER*4). DATAR = JMOD(ISTEP-1, DFREQ) = 0 Read Data Set if and only if DELAY < LSTEP. ≠ 0 Do Not Read Data Set.
DELAY		**Delay Count for Data Set Read** (INTEGER*4). = LDELAY During the Learning Trajectory. = CDELAY During the Controlled Trajectory. ≥ LSTEP Do Not a Read Data Set. < LSTEP Read a Data Set if and only if DATAR = 0.
DFREQ		**Data Set Read Frequency After** the **First Neural-Net** (NN) CW(I,J,K)s **Update** (INTEGER*4). = DLFREQ During the Learning Trajectory. = DCFREQ During the Controlled Trajectory.
DFREQ0		**Data Set Read Inhibit Flag** for the **First Read Attempt** during the Learning Phase at the Start of the Controlled Trajectory (IPHASE = 5) (INTEGER*4). **Currently NOT an option** = 0 Do Not Inhibit Data Set Read. ≠ 0 Inhibit Data Set Read.
DLGTH		**Data Sliding Window Length/Size** (i.e., the Maximum Number of Data Sets Contained in a Data Sliding Window (INTEGER*4). = DLLGTH During the Learning Trajectory. = DCLGTH During the Controlled Trajectory.
I	1	**Index** which Specifies the **I-th Element Position** in the **Plant Input Vectors** (i.e., the **Plant Control Vector)** XA(I), XD(I,L), and XN(I) (INTEGER*4).

Internally Set Parameters Group A (Continued)

Internally Set Parameters for Trajectory Propagation

Parameter	Default or Initial Value	Definition
ICUT		**Trajectory Phase Cut (Termination)** Flag (INTEGER*4). = 0 No Cut upon completion of current step. ≠ 0 Cut Phase (Terminate Phase) upon completion of current step.
IPHASE		**Trajectory Phase** Identification Pointer (INTEGER*4). **= 0** Prior to Start of the Learning Trajectory. **t < TLINIT** **= 1** At the Start of the Learning Trajectory. **t = TLINIT** **= 2** During the Learning Trajectory. **TLINIT < t < TLFINL** **= 3** At Termination of the Learning Trajectory. **t = TLFINL** **= 4** Between the Learning & Controlled Trajectories. **TLFINL < t < TCINIT** **= 5** At Start of the Controlled Trajectory. **t = TCINIT** **= 6** During the Controlled Trajectory. **TCINIT < t < TCFINL** **= 7** At Termination of the Controlled Trajectory. **t = TCFINL** **= 8** After Termination of the Controlled Trajectory. **t > TCFINL**
ISTEP	0	The **Step Number** of the Current Trajectory Integration Step. Note that **if**: (INTEGER*4). **ISTEP0 = 0 and STMODC ≠ 2 or 3**, ISTEP is reset to **ISTEP = 1** at the start of the Controlled Trajectory. **ISTEP0 ≠ 0 or STMODC = 2 or 3**, ISTEP is reset to **ISTEP = ISTEP + ISTEP0** at the start of the Controlled Trajectory.

Internally Set Parameters Group A (Continued)

Internally Set Parameters for Trajectory Propagation

Parameter	Default or Initial Value	Definition
J	1	**Index** which Specifies the **J-th Element Position** in the **Plant Output Vectors** (i.e., the **Plant Measurement/State Vector**) YA(J), YD(J,L), and YN(J) (INTEGER*4).
L	1	**Index** which Specifies the **L-th Data Set** in the **Data Sliding Window** (INTEGER*4). $1 \leq L \leq LMAX$
L1	1	**Index** which **Assigns** the **Analytic Modelling Function** for a specific (L_3,L_2,L_1) **to** either the **Plant Input Vector** (i.e., the Plant Control Vector) **or** the **Plant Output Vector** (i.e., Plant Measurement/State Vector). **L1** is the **name** for the **subscript** L_1 in the computer code (INTEGER*4). = 1 Specifies that the Model applies to an element of the **Plant Input Vector** (i.e., the Plant Control Vector). = 2 Specifies that the Model applies to an element of the **Plant Output Vector** (i.e., Plant Measurement/State Vector).
L2	1	**Index** which Specifies the **Element Number** for the **Plant Input Vector** (i.e., the Plant Control Vector) if L1 = 1, **or** the **Plant Output Vector** (i.e., Plant Measurement/State Vector) if L1 = 2. **L2** is the **name** for the **subscript** L_2 in the computer code (INTEGER*4).
L3	1	**Index** which Specifies the **Element Number** of the **Analytic Modelling Function** for a specific (L_2,L_1). **L3** is the **name** for the **subscript** L_3 in the computer code (INTEGER*4).
LMAX	1	The Current **Number of Data Sets** in the **Data Sliding Window** (i.e., the Maximum Value that the index **L** can have) (INTEGER*4). $1 \leq L \leq LMAX \leq DLGTH$
LSTEP		**Data Read Counter Number** during the Current Trajectory (INTEGER*4). $LSTEP = 1 + (ISTEP - 1)/DFREQ$

Internally Set Parameters Group A (Continued)

Internally Set Parameters for Trajectory Propagation

Parameter	Default or Initial Value	Definition
LTBL	1	**Index** which Specifies the **LTBL-th Data Set** of the **Plant Model Data Table** (INTEGER*4). $1 \leq$ **LTBL** $\leq$ **TBLMAX**
NNID		Neural-Net (NN) **CW(I,J,K)s Update Frequency/Inhibit Flag** (INTEGER*4). = NNLID During the Learning Trajectory. = NNCID During the Controlled Trajectory. ≤ 0 Do Not Update NN CW(I,J,K)s. > 0 Update NN CW(I,J,K)s every NNID times.
NNUP		Neural-Net (NN) **CW(I,J,K)s Update Frequency Flag** (INTEGER*4). NNUP = JMOD(ISTEP-1, NNID) = 0 Update NN CW(I,J,K)s. ≠ 0 Do Not Update NN CW(I,J,K)s.
NNUP0		Neural-Net (NN) **CW(I,J,K)s Update Inhibit Flag** (INTEGER*4). **Currently NOT an option** = 0 Do Not Inhibit NN CW(I,J,K)s Update at Start of the Controlled Trajectory. = 1 Inhibit NN CW(I,J,K)s Update at Start of the Controlled Trajectory iff a NN CW(I,J,K)s Update was done at the End of the Learning Trajectory.
T		Either the Current **Absolute Time** (TABS) **or** the Current **Relative Time** (TREL) as appropriately specified by TLTYPE or TCTYPE. **If**: (REAL*8). **TLTYPE ≤ 0** during the **Learning** Trajectory, then: **T = TABS** . **TLTYPE > 0** during the **Learning** Trajectory, then: **T = TREL**. **TCTYPE ≤ 0** during the **Controlled** Trajectory, then: **T = TABS** . **TCTYPE > 0** during the **Controlled** Trajectory, then: **T = TREL**.
TABS		Current **Absolute Time**. Note that **if**: (REAL*8). **TLTYPE ≤ 0** during the **Learning** Trajectory, then: TABS is measured from **TLINIT**. **TLTYPE > 0** during the **Learning** Trajectory, then: TABS is measured from **TINIT + TLINIT**.

Internally Set Parameters Group A (Continued)

Internally Set Parameters for Trajectory Propagation

Parameter	Default or Initial Value	Definition
TABS		(Continued) **TCTYPE ≤ 0** during the **Controlled** Trajectory, then: TABS is measured from **TCINIT**. TCTYPE > **0** during the **Controlled** Trajectory, then: TABS is measured from **TLFINL + TCINIT**.
••• TBLMAX	1	The **Number of Data Sets** (i.e., the Maximum Value that the index **LTBL** can have) in the **Plant Model Data Table** (INTEGER*4). **1 ≤ LTBL ≤ TBLMAX**
TCUT		**Trajectory Cut (Termination) Time** (REAL*8). **TLTYPE ≤ 0** during the **Learning** Trajectory, then: TCUT is the value of the **Absolute Time** (i.e., the value of **TABS)** specified by **TFINL**. **TLTYPE > 0** during the **Learning** Trajectory, then: TCUT is the value of the **Relative Time** (i.e., the value of **TREL**) specified by **TFINL**. **TCTYPE ≤ 0** during the **Controlled** Trajectory, then: TCUT is the value of the **Absolute Time** (i.e., the value of **TABS**) specified by **TFINL**. **TCTYPE > 0** during the **Controlled** Trajectory, then: TCUT is the value of the **Relative Time** (i.e., the value of **TREL**) specified by **TFINL**.
••• TD(L)		Either the **Absolute Time** (TABS) **or** the **Relative Time** (TREL) as appropriately defined by TLTYPE or TCTYPE corresponding to the **"Actual" (Reference) Plant Model** defined in the **L-th Data Set** in the **Data Sliding Window** (REAL*8).
TREL		Current **Relative Time**. Note that **if**: (REAL*8). **TLTYPE ≤ 0** during the **Learning** Trajectory, then: TREL is measured from **ZERO**. **TLTYPE > 0** during the **Learning** Trajectory, then: TREL is measured from **TLINIT**. **TCTYPE ≤ 0** during the **Controlled** Trajectory, then: TREL is measured from **ZERO**. **TCTYPE > 0** during the **Controlled** Trajectory, then: TREL is measured from **TCINIT**.

Internally Set Parameters Group A (Continued)

Internally Set Parameters for Trajectory Propagation

Parameter	Default or Initial Value	Definition
TSTEP		Trajectory Integration **Time Step** (REAL*8). = TLSTEP During the **Learning** Trajectory. = TCSTEP During the **Controlled** Trajectory.
••• TTBL(LTBL)		Either the **Absolute Time** (TABS) **or** the **Relative Time** (TREL) as appropriately defined by TLTYPE or TCTYPE corresponding to the **"Actual" (Reference) Plant Model** defined in the **LTBL-th Data Set** of the **Plant Model Data Table** used when the "Actual Plant" is modelled using Routine **TSTATE** (REAL*8).
XA(I)		The **I-th Element** of the **Input Vector** (i.e., the **Control Vector**) to the **"Actual" (Reference) Plant Model** which is modelled by **One** of: Routine **ASTATE** (i.e., Analytic Trajectory State Synthesis), **or** Routine **DSTATE** (i.e., Trajectory State from On-Line Test Data), **or** Routine **TSTATE** (i.e., Trajectory State from Stored Data Tables), **or** Routine **USTATE** (i.e., Trajectory State from a User Supplied Model) (REAL*8).
••• XD(I,L)		The **I-th Element** of the **Input Vector** (i.e., the **Control Vector**) to the **"Actual" (Reference) Plant Model** defined in the **L-th Data Set** in the **Data Sliding Window** (REAL*8).
XN(I)		The **I-th Element** of the **Input Vector** (i.e., the **Control Vector**) to the **Neural Network Plant Model** which corresponds to XA(I) (REAL*8).
••• XTBL(I,LTBL)		The **I-th Element** of the **Input Vector** (i.e., the **Control Vector**) to the **"Actual" (Reference) Plant Model** defined in the **LTBL-th Data Set** of the **Plant Model Data Table** used when the "Actual Plant" is modelled using Routine **TSTATE** (REAL*8).
YA(J)		The **J-th Element** of the **Output Vector** (i.e., the **Measurement/State Vector**) from the **"Actual" (Reference) Plant Model** which is modelled by **One** of: Routine **ASTATE** (i.e., Analytic Trajectory State Synthesis), **or** Routine **DSTATE** (i.e., Trajectory State from On-Line Test Data), **or** Routine **TSTATE** (i.e., Trajectory State from Stored Data Tables), **or** Routine **USTATE** (i.e., Trajectory State from a User Supplied Model) (REAL*8).

Internally Set Parameters Group A (Continued)

Internally Set Parameters for Trajectory Propagation

Parameter	Default or Initial Value	Definition
*** YD(J,L)		The **J-th Element** of the **Output Vector** (i.e., the **Measurement/State Vector**) from the **"Actual" (Reference) Plant Model** defined in the **L-th Data Set** in the **Data Sliding Window** (REAL*8).
YN(J)		The **J-th Element** of the **Output Vector** (i.e., the **Measurement/State Vector**) from the **Neural Network Plant Model** which corresponds to YA(J) (REAL*8).
*** YTBL(J,LTBL)		The **J-th Element** of the **Output Vector** (i.e., the **Measurement/State Vector**) from the **"Actual" (Reference) Plant Model** defined in the **LTBL-th Data Set** of the **Plant Model Data Table** used when the "Actual Plant" is modelled using Routine **TSTATE** (REAL*8).

Internally Set Parameters Group B

Internally Set Parameters for Neural Network Operation

Parameter	Default or Initial Value	Definition
I	1	**Index** which Specifies the **I-th Element Position** in the **Origin Vector** for the **Specific Neural-Network Layer** specified by the index "**K**" (INTEGER*4).
J	1	**Index** which Specifies the **J-th Element Position** in the **Destination Vector** for the **Specific Neural-Network Layer** specified by the index "**K**" (INTEGER*4).
K	1	**Index** which Specifies the **K-th Specific Layer** in the **Neural-Network** (INTEGER*4).
UNN(J,K)		**Input Signal to** the **Pass-Through Function** (Node Filter) **at** the **J-th Exit (Destination) Position** of the **K-th** Neural-Net Layer (REAL*8). $$\mathrm{UNN(J,K)} = \sum_{\mathrm{I}} \mathrm{CW(I,J,K)} * \mathrm{XNN(I,J,K)}$$
XNN(I,J,K)		**Entry Signal to** the **K-th Neural-Net Layer from** the **I-th Entry (Origin) Position** directed to the **J-th** Exit (Destination) Position. (REAL*8).
YNN(J,K)		**Exit Signal from** the **J-th Exit (Destination) Position** of the **K-th** Neural-Net Layer (REAL*8).

Internally Set Parameters Group C

Internally Set Parameters for the Optimisation Processes

Parameter	Default or Initial Value	Definition
CMAXC(II)		The **Vector** of **l.u.b. Values** corresponding to the elements of the **Optimisation Control Vector** CV(•) set to the value of the appropriate element of AMAXC(I) for **Control Optimisation** during the **Controlled Trajectory Phase**. (REAL* 8).
CMAXNNC(IJK)		The **Vector** of **l.u.b. Values** corresponding to the elements of the **Optimisation Control Vector** CV(•) set to the value of the appropriate element of AMAXNNC(I,J,K) for **Neural-Net Optimisation** during the **Controlled Trajectory Phase**. (REAL* 8).
CMAXNNL(IJK)		The **Vector** of **l.u.b. Values** corresponding to the elements of the **Optimisation Control Vector** CV(•) set to the value of the appropriate element of AMAXNNL(I,J,K) for **Neural-Net Optimisation** during the **Learning Trajectory Phase**. (REAL* 8).
CMINC(II)		The **Vector** of **g.l.b. Values** corresponding to the elements of the **Optimisation Control Vector** CV(•) set to the value of the appropriate element of AMINC(I) for **Control Optimisation** during the **Controlled Trajectory Phase**. (REAL* 8).
CMINNNC(IJK)		The **Vector** of **g.l.b. Values** corresponding to the elements of the **Optimisation Control Vector** CV(•) set to the value of the appropriate element of AMINNNC(I,J,K) for **Neural-Net Optimisation** during the **Controlled Trajectory Phase**. (REAL* 8).
CMINNNL(IJK)		The **Vector** of **g.l.b. Values** corresponding to the elements of the **Optimisation Control Vector** CV(•) set to the value of the appropriate element of AMINNNL(I,J,K) for **Neural-Net Optimisation** during the **Learning Trajectory Phase**. (REAL* 8).
CON(•)		The **Actual/Working Constraint Function Vector** (REAL*8). where • denotes **IIJK** during Neural-Net Update/Optimisation and **III** during Control Update/Optimisation.
CV(•)		The **Actual/Working Optimisation Control Vector** (REAL*8). where • denotes **IJK** during Neural-Net Update/Optimisation and **II** during Control Update/Optimisation.

Internally Set Parameters for the Optimisation Processes

Parameter	Default or Initial Value	Definition
CV0(•)		The **Initial Estimate** of the **Optimisation Control Vector** CV(•) (REAL* 8). where • denotes **IJK** during Neural-Net Update/Optimisation and **II** during Control Update/Optimisation.
CVBDC	0	The **Control Variable Bounds Specification Flag** for **Control Optimisation** during the **Controlled Trajectory Phase**. (INTEGER*4). where • denotes II, and if = 0 **Both Lower** and **Upper Bounds** (i.e., the **CMINC**(•) and **CMAXC**(•) Vectors) are specified for **All** Elements of the Optimisation Control Vector CV(•). = 1 **All Elements** of the Optimisation Control Vector CV(•) are constrained to be ≥ **zero**. = 2 **All Elements** of the Optimisation Control Vector CV(•) are constrained to be ≤ **zero**. = 3 **Both Lower** and **Upper Bounds** (i.e., the **CMINC**(•) and **CMAXC**(•) Vectors) are specified for **All** Elements of the Optimisation Control Vector CV(•) by specifying **Only** the First Element of the **CMINC**(•) and **CMAXC**(•) Vectors (i.e., **CMINC(1)** and **CMAXC(1)**. In this case, all other Elements of the **CMINC**(•) and **CMAXC**(•) Vectors are internally set Equal to the values of **CMINC(1)** and **CMAXC(1)**, respectively.
CVBDNNC	0	The **Control Variable Bounds Specification Flag** for **Neural-Net Optimisation** during the **Controlled Trajectory Phase**. (INTEGER*4). = 0 **Both Lower** and **Upper Bounds** (i.e., the **CMINNNC**(•) and **CMAXNNC**(•) Vectors) are specified for **All** Elements of the Optimisation Control Vector CV(•). = 1 **All Elements** of the Optimisation Control Vector CV(•) are constrained to be ≥ **zero**. = 2 **All Elements** of the Optimisation Control Vector CV(•) are constrained to be ≤ **zero**.

Internally Set Parameters for the Optimisation Processes

Parameter	Default or Initial Value	Definition
CVBDNNC		(Continued) = 3 **Both Lower** and **Upper Bounds** (i.e., the **CMINNNC(•)** and **CMAXNNC(•)** Vectors) are specified for **All** Elements of the Optimisation Control Vector CV(•) by specifying **Only** the First Element of the **CMINNNC(•)** and **CMAXNNC(•)** Vectors (i.e., **CMINNNC(1)** and **CMAXNNC(1)**. In this case, all other Elements of the **CMINNNC(•)** and **CMAXNNC(•)** Vectors are internally set Equal to the values of **CMINNNC(1)** and **CMAXNNC(1)**, respectively.
CVBDNNL	0	The **Control Variable Bounds Specification Flag** for **Neural-Net Optimisation** during the **Learning Trajectory Phase**. (INTEGER*4). = 0 **Both Lower** and **Upper Bounds** (i.e., the **CMINNNL(•)** and **CMAXNNL(•)** Vectors) are specified for **All** Elements of the Optimisation Control Vector CV(•). = 1 **All Elements** of the Optimisation Control Vector CV(•) are constrained to be ≥ **zero**. = 2 **All Elements** of the Optimisation Control Vector CV(•) are constrained to be ≤ **zero**. = 3 **Both Lower** and **Upper Bounds** (i.e., the **CMINNNL(•)** and **CMAXNNL(•)** Vectors) are specified for **All** Elements of the Optimisation Control Vector CV(•) by specifying **Only** the First Element of the **CMINNNL(•)** and **CMAXNNL(•)** Vectors (i.e., **CMINNNL(1)** and **CMAXNNL(1)**. In this case, all other Elements of the **CMINNNL(•)** and **CMAXNNL(•)** Vectors are internally set Equal to the values of **CMINNNL(1)** and **CMAXNNL(1)**, respectively.
CVSC(II)		The **Vector** of **Scaling Coefficients** corresponding to the elements of the **Optimisation Control Vector** CV(•) set to the value of the appropriate element of SCVC(I) for **Control Optimisation** during the **Controlled Trajectory Phase**. (REAL* 8).

Internally Set Parameters Group C (Continued)

Internally Set Parameters for the Optimisation Processes

Parameter	Default or Initial Value	Definition
CVSNNC(IJK)		The **Vector** of **Scaling Coefficients** corresponding to the elements of the **Optimisation Control Vector** CV(•) set to the value of the appropriate element of SCVNNC(I,J,K) for **Neural-Net Optimisation** during the **Controlled Trajectory Phase**. (REAL* 8).
CVSNNL(IJK)		The **Vector** of **Scaling Coefficients** corresponding to the elements of the **Optimisation Control Vector** CV(•) set to the value of the appropriate element of SCVNNL(I,J,K) for **Neural-Net Optimisation** during the **Learning Trajectory Phase**. (REAL* 8).
EC(•)		The **Actual/Working Optimisation End Conditions Vector** (REAL*8). where • denotes **JJJ** during Neural-Net Update/Optimisation and **JJ** during Control Update/Optimisation
I	1	**Index** which Specifies the **I-th Element Position** in the **Origin Vector** for the **Specific Neural-Network Layer** specified by the index "K" (INTEGER*4).
ICVDEF		The **Control Vector Disposition** Flag (INTEGER*4). = 1 **Load** CV(•) for **Neural-Net Optimisation** during the **Learning Trajectory Phase**. = 2 **Unload** CV(•) for **Neural-Net Optimisation** during the **Learning Trajectory Phase**. = 3 **Load** CV(•) for **Neural-Net Optimisation** during the **Controlled Trajectory Phase**. = 4 **Unload** CV(•) for **Neural-Net Optimisation** during the **Controlled Trajectory Phase**. = 5 **Load** CV(•) for **Control Optimisation** during the **Controlled Trajectory Phase**. = 6 **Unload** CV(•) for **Control Optimisation** during the **Controlled Trajectory Phase**.
IECDEF		The **End Condition Disposition** Flag (INTEGER*4). = 1 **Load** EC(•) for **Neural-Net Optimisation** during the **Learning Trajectory Phase**. = 2 **Load** EC(•) for **Neural-Net Optimisation** during the **Controlled Trajectory Phase**.

Internally Set Parameters Group C (Continued)

Internally Set Parameters for the Optimisation Processes

Parameter	Default or Initial Value	Definition
IECDEF		(Continued) = 3 **Load** EC(•) for **Control Optimisation** during the **Controlled Trajectory Phase.**
II	0	**Subscript/Index** which defines a particular element of the **Optimisation Control Vector** CV(•) during **Control Update/Optimisation** (INTEGER*4).
III	0	**Subscript/Index** which defines a particular element of the **Constraint Function Vector** CON(•) during **Control Update/Optimisation** (INTEGER*4).
IIJK	0	**Subscript/Index** which defines a particular element of the **Constraint Function Vector** CON(•) during **Neural-Net Update/Optimisation** (INTEGER*4).
IJK	0	**Subscript/Index** which defines a particular element of the **Optimisation Control Vector** CV(•) during **Neural-Net Update/Optimisation** (INTEGER*4).
J	1	**Index** which Specifies the **J-th Element Position** in the **Destination Vector** for the **Specific Neural-Network Layer** specified by the index **K** (INTEGER*4).
JJ	0	**Subscript/Index** which defines a particular element of the **Optimisation End Conditions Vector** EC(•) during **Control Update/Optimisation** (INTEGER*4).
JJJ	0	**Subscript/Index** which defines a particular element of the **Optimisation End Conditions Vector** EC(•) during **Neural-Net Update/Optimisation** (INTEGER*4).
K	1	**Index** which Specifies the **K-th Specific Layer** in the **Neural-Network** (INTEGER*4).
L	1	**Index** which Specifies the **L-th Data Set** in the **Data Sliding Window** (INTEGER*4). $1 \leq L \leq LMAX$

Internally Set Parameters Group C (Continued)

Internally Set Parameters for the Optimisation Processes

Parameter	Default or Initial Value	Definition
LMAX	1	The Current **Number of Data Sets** in the **Data Sliding Window** (i.e., the Maximum Value that the index **L** can have) (INTEGER*4). $1 \leq L \leq LMAX \leq DLGTH$
NCONC	0	**Total Number of Elements** in the **Actual/Working** Optimisation Constraint Function Vector CON(III) (i.e., the Dimension of the Actual/Working Optimisation Constraint Function Vector, NOT to be confused with the Dimension of the CON(III) Array) **for Control Update/Optimisation** during the **Controlled Trajectory Phase** (INTEGER*4).
NCONNNC	0	**Total Number of Elements** in the **Actual/Working** Optimisation Constraint Function Vector CON(IIJK) (i.e., the Dimension of the Actual/Working Optimisation Constraint Function Vector, NOT to be confused with the Dimension of the CON(IIJK) Array) **for Neural-Net Update/Optimisation** during the **Controlled Trajectory Phase** (INTEGER*4).
NCONNNL	0	**Total Number of Elements** in the **Actual/Working** Optimisation Constraint Function Vector CON(IIJK) (i.e., the Dimension of the Actual/Working Optimisation Constraint Function Vector, NOT to be confused with the Dimension of the CON(IIJK) Array) **for Neural-Net Update/Optimisation** during the **Learning Trajectory Phase** (INTEGER*4).
NICV	0	**Total Number of Elements** in the **Actual/Working Optimisation Control Vector** CV(II) (i.e., the Dimension of the Actual/Working Optimisation Control Vector, NOT to be confused with the Dimension of the CV(II) Array) **for Control Update/Optimisation** during the **Controlled Trajectory Phase** (INTEGER*4).
NIJKCVC	0	**Total Number of Elements** in the **Actual/Working Optimisation Control Vector** CV(IJK) (i.e., the Dimension of the Actual/Working Optimisation Control Vector, NOT to be confused with the Dimension of the CV(IJK) Array) **for Neural-Net Update/Optimisation** during the **Controlled Trajectory Phase** (INTEGER*4).

Internally Set Parameters Group C (Continued)

Internally Set Parameters for the Optimisation Processes

Parameter	Default or Initial Value	Definition
NIJKCVL	0	**Total Number of Elements** in the **Actual/Working Optimisation Control Vector** CV(IJK) (i.e., the Dimension of the Actual/Working Optimisation Control Vector, NOT to be confused with the Dimension of the CV(IJK) Array) **for Neural-Net Update/Optimisation** during the **Learning Trajectory Phase** (INTEGER*4).
NJEC	0	**Total Number of Elements** in the **Actual/Working Optimisation End Conditions Vector** EC(JJ) (i.e., the Dimension of the Actual/Working Optimisation End Conditions Vector, NOT to be confused with the Dimension of the EC(JJ) Array) **for Control Update/Optimisation** during the **Controlled Trajectory Phase** (INTEGER*4).
NJJECC	0	**Total Number of Elements** in the **Actual/Working Optimisation End Conditions Vector** EC(JJJ) (i.e., the Dimension of the Actual/Working Optimisation End Conditions Vector, NOT to be confused with the Dimension of the EC(JJJ) Array) **for Neural-Net Update/Optimisation** during the **Controlled Trajectory Phase** (INTEGER*4).
NJJECL	0	**Total Number of Elements** in the **Actual/Working Optimisation End Conditions Vector** EC(JJJ) (i.e., the Dimension of the Actual/Working Optimisation End Conditions Vector, NOT to be confused with the Dimension of the EC(JJJ) Array) **for Neural-Net Update/Optimisation** during the **Learning Trajectory Phase** (INTEGER*4).
PINDX	0.000	The **Performance Index** (REAL*8). $$\text{PINDX} \equiv \text{SUMSQ}$$

Internally Set Parameters Group C (Continued)

Internally Set Parameters for the Optimisation Processes

Parameter	Default or Initial Value	Definition
SUMSQ	0.000	**Sum** of the **Product** of the **Weighting Coefficients** with the **Squares** of the **Elements** of the **Optimisation End Conditions Vector EC(•)** (REAL*8).

For **Neural-Network Optimisation** during the **Learning Trajectory**,

$$\text{SUMSQ} = \sum_{L=1}^{LMAX} \text{WTSNNL(L)} * \text{SUMSQW(L)}$$

Where

$$\text{SUMSQW(L)} = \sum_{JJJ} \text{WTNNL(JJJ)} * \text{EC(JJJ)} * \text{EC(JJJ)}$$

For **Neural-Network Optimisation** during the **Controlled Trajectory**,

$$\text{SUMSQ} = \sum_{L=1}^{LMAX} \text{WTSNNC(L)} * \text{SUMSQW(L)}$$

Where

$$\text{SUMSQW(L)} = \sum_{JJJ} \text{WTNNC(JJJ)} * \text{EC(JJJ)} * \text{EC(JJJ)}$$

For **Control Optimisation** during the **Controlled Trajectory Phase**,

$$\text{SUMSQ} = \sum_{JJ} \text{WTC(JJ)} * \text{EC(JJ)} * \text{EC(JJ)}$$

Internally Set Parameters Group C (Continued)

Internally Set Parameters for the Optimisation Processes

Parameter	Default or Initial Value	Definition
SUMSQW(L)	0.000	**Sum** of the **Product** of the **Weighting Coefficients** with the **Squares** of the **Elements** of the **Optimisation End Conditions Vector** during **Neural-Net Optimisation EC(•)** (REAL*8).

Where

during the **Learning Trajectory**,

$$\text{SUMSQW(L)} = \sum_{\text{JJJ}} \text{WTNNL(JJJ)} * \text{EC(JJJ)} \bullet \text{EC(JJJ)}$$

during the **Controlled Trajectory**,

$$\text{SUMSQW(L)} = \sum_{\text{JJJ}} \text{WTNNC(JJJ)} * \text{EC(JJJ)} * \text{EC(JJJ)}$$

Parameter	Default or Initial Value	Definition
WC(JJ)		**Weighting Coefficient** element in the **WC(JJ)*EC(JJ)^2** term in **SUMSQ** and the Performance Index **PINDX** (REAL* 8).
WNNC(JJJ)		**Weighting Coefficient** element in the **WNNC(JJJ)*EC(JJJ)^2** term in **SUMSQ** and the Performance Index **PINDX** (REAL* 8).
WNNL(JJJ)		**Weighting Coefficient** element in the **WNNL(JJJ)*EC(JJJ)^2** term in **SUMSQ** and the Performance Index **PINDX** (REAL* 8).

Internally Set Parameters Group D

Internally Set Constants

Internal Value	Internal Name	Definition
0.000	ZERO	**0.000** (REAL*8).
1.0 D-08	TENM8	**0.000,000,01** (REAL*8).
1.0 D-06	TENM6	**0.000,001** (REAL*8).
1.0 D-03	TENM3	**0.001** (REAL*8).
1.0 D-02	TENM2	**0.010** (REAL*8).
0.100	PT100	**0.100** (REAL*8).
0.200	PT200	**0.200** (REAL*8).
0.300	PT300	**0.300** (REAL*8).
0.500	PT500	**0.500** (REAL*8).
0.800	PT800	**0.800** (REAL*8).
1.000	ONE	**1.000** (REAL*8).
2.000	TWO	**2.000** (REAL*8).
e	EBASE	***e*** **(2.71828182845904523536)** (REAL*8).
3.000	THREE	**3.000** (REAL*8).
π	PI	π **(3.14159265358979323846)** (REAL*8).
5.000	FIVE	**5.000** (REAL*8).
2π	TWOPI	2π **(6.28318530717958647693)** (REAL*8).
8.000	EIGHT	**8.000** (REAL*8).
10.000	TEN	**10.000** (REAL*8).
$360/2\pi$	RTD	$360/2\pi$ ***Degrees/Radian*** (REAL*8).
1.0 D+02	TENP2	**100.000** (REAL*8).
1.0 D+03	TENP3	**1000.000** (REAL*8).
1.0 D+06	TENP6	**1000,000.000** (REAL*8).
1.0 D+08	TENP8	**100,000,000.000** (REAL*8).

Appendix B

Principal Routines

in the

OPTIMNN Code

Appendix B

Table of Contents for the Principal Routines in the OPTIMNN Code

Routines List

Routines Group 1

Principal OPTIMNN Peculiar Routines

Routine	Purpose of Routine
ASTATE	***Synthesis*** of the ***"Actual" (Reference) Plant Model*** by ***Combining Selected Individual Analytic Models*** (i.e., ASTATE01, ASTATE02, ASTATE03, •, •, •, •).
ASTATE01	The ***Linear Function*** (i.e., the ***Ramp Function***) Individual Analytic Model Element defined by: $$y-y_0 = A(x-x_0) + C$$
ASTATE02	The ***Serpentine Curve*** Individual Analytic Model Element defined by: $$y-y_0 = \frac{ab(x-x_0)}{a^2+(x-x_0)^2}$$
ASTATE03	The ***Witch of Agnesi Curve*** Individual Analytic Model Element defined by: $$y-y_0 = \frac{a^3}{b^2(x-x_0)^2+a^2}$$
ASTATE04	The ***Inverted Witch of Agnesi Curve*** Individual Analytic Model Element defined by: $$y-y_0 = a - \frac{a^3}{b^2(x-x_0)^2+a^2}$$
ASTATE05	The ***Enveloped Sinusoid Function*** Individual Analytic Model Element defined by: $$y-y_0 = C\,Exp_e\left[\alpha(x-x_0-\psi)\right]Cos\left[n\omega(x-x_0-\phi)\right]$$
ASTATE06	The ***Hyperbolic Tangent Function*** (i.e., the ***Threshold Function***) Individual Analytic Model Element defined by: $$y-y_0 = C\,Tanh\left[A(x-x_0)\right]$$
ASTATE07	The ***Derivative of the Threshold Function*** (i.e., the ***Pulse Function***) Individual Analytic Model Element defined by: $$y-y_0 = \frac{d}{dx}\left\{C\,Tanh\left[A(x-x_0)\right]\right\} = AC\,Sech^2\left[A(x-x_0)\right]$$

Routines Group 1 (Continued)

Principal OPTIMNN Peculiar Routines

Routine	Purpose of Routine
ASTATRAN	The ***Uniform Distribution Function*** Individual Analytic Model Element defined by: $$y - y_0 = [A + B\,Uran(ISEED)] + [C + D\,Uran(JSEED)]f(x - x_0)$$ where: Uran(•) is the *Uniformly Random Distribution Function* such that $$-1.00000 \le \text{Uran}(\bullet) \le +1.00000$$ $f(\bullet)$ is any of the functions defined by ASTATE01, ASTATE02, ASTATE03, •, •, •, ASTATE07. *ISEED* and *JSEED* are the seeds required by the VAX/VMS System Subroutine RAN(•).
CVVCTR	Defines the ***Control Vector*** for the Optimisation Processes.
DSTATE	Defines the ***"Actual" (Reference) Plant Model*** from ***On-Line Test Data***.
ECVCTR	Defines ***End Conditions*** (Conditions-of-Interest during the Optimisation Process) used to evaluate the Performance Index and Constraint Functions.
GRADC	Defines the ***Analytic Gradient*** of the ***Performance Index and the Constraint Functions*** with respect to the ***Control θs*** for the ***Optimisation Processes***.
GRADW	Defines the ***Analytic Gradient*** of the ***Error Metric and the Constraint Functions*** with respect to the ***Neural-Net Signal Coefficients Ws*** for the ***Neural-Net Learning Processes***.
INIT	Reads the **Input Data** defined by **"NAMELIST CDATA"** and then **initialises** the data for the case to be processed.
INITDAT	**FORTRAN Code** (<u>not</u> a complete routine) which is **included** in the **OPTIMNN** Peculiar Routine **INIT** by means of an **INCLUDE** Statement to define the initially set **Default Values** of the **"NAMELIST CDATA" INPUT** Parameters and the **Values** of the **Internally Set Constants** of the **OPTIMNN** System.
JCTRL	Defines the ***Performance Index*** and ***Constraint Functions*** for the ***Control Optimisation Processes***.
JNNW	Defines the ***Error Metric (Performance Index)*** and ***Constraint Functions*** for the ***Neural-Net Learning Processes***.

Routines Group 1 (Continued)
Principal OPTIMNN Peculiar Routines

Routine	Purpose of Routine
OPTIMNN	**Main Driver Routine**: **Executes** the Code by first calling Subroutine **INIT** to cause the **Input Data** defined by **NAMELIST "CDATA"** to be **read** and **initialised** for the case to be processed, and then by subsequently calling Subroutine **TRAJ** to cause **execution** of the options and propagation of the trajectories defined by the input.
PFNCT00	The ***No-Pass*** (i.e., the ***Constant Function***) Node Filter Function defined by: $$y - y_0 = C$$
PFNCT01	The ***Direct-Pass*** (i.e., the ***Linear Function***) Node Filter Function defined by: $$y - y_0 = A(x - x_0) + C$$
PFNCT02	The ***Hyperbolic Tangent (Threshold Function)*** Node Filter Function defined by: $$y - y_0 = C\,Tanh[A(x - x_0)]$$
PFNCT03	The ***First Derivative of the Hyperbolic Tangent (Pulse Function)*** Node Filter Function defined by: $$y - y_0 = \frac{d}{dx}\{C\,Tanh[A(x - x_0)]\} = AC\,Sech^2[A(x - x_0)]$$
STATE	Defines the ***Input Vector*** (i.e., the ***Control Vector***) and the ***Output Vector*** (i.e., the ***Measurement/State Vector***) to/from the ***"Actual" (Reference) Plant*** at a specific time point by **Selecting** the ***"Actual" (Reference) Plant Model*** from amongst Routines **ASTATE**, **DSTATE**, **TSTATE**, and **USTATE**.
STATENN	Defines the ***Neural-Net State*** using the Current ***NN W*** and ***Control θ*** Values.
TSTATE	Defines the ***"Actual" (Reference) Plant Model*** from ***Stored Data Tables***.
TRAJ	***Propagates (Integrates)*** the ***Trajectory*** by ***Incrementing*** the ***Time***.

<u>Routines Group 1 (Continued)</u>

<u>Principal OPTIMNN Peculiar Routines</u>

Routine	Purpose of Routine
TYPECOM	**FORTRAN Code** (<u>not</u> a complete routine) which is **included** in the **OPTIMNN** Peculiar Routines by means of an **INCLUDE** Statement to establish and define: 1) the **Principal COMMON Blocks**; 2) the **Data TYPE** of the **Principal Parameters, Arrays,** and **Vectors**; and 3) the **DIMENSION** of the **Principal Arrays** and **Vectors** of the **OPTIMNN** System.
USTATE	Defines the ***"Actual" (Reference) Plant Model*** from a ***User Supplied Model.***

Routines Group 2

Principal IMSL MATH/LIBRARY Routines used by OPTIMNN

Routine	Purpose of Routine
DNCONF	IMSL MATH/LIBRARY Routine which solves a general non-linear programming problem using a successive quadratic programming algorithm and a finite-difference approximation gradient. See pages 895-902 in Chapter 8 of Reference D-3
DN4ONF1	Modified IMSL MATH/LIBRARY DN4ONF Routine which is called during the computation process initiated when the IMSL MATH/LIBRARY Routine DNCONF is called. DN4ONF was modified to provide better mathematical conditioning for the controller problems considered.
DNCONG	IMSL MATH/LIBRARY Routine which solves a general non-linear programming problem using a successive quadratic programming algorithm and a user-supplied (analytic) gradient routine. See pages 903-908 in Chapter 8 of Reference D-3
DN9ONG1	Modified IMSL MATH/LIBRARY DN9ONG Routine which is called during the computation process initiated when the IMSL MATH/LIBRARY Routine DNCONG is called. DN9ONG was modified to provide better mathematical conditioning for the controller problems considered.
ERSET	IMSL MATH/LIBRARY Error Handling Routine which sets actions to be taken (changes the default actions) when errors occur during the execution of IMSL MATH/LIBRARY Routines. See pages 1130-1134 in Chapter 8 of Reference D-3.
IERCD	IMSL MATH/LIBRARY Error Handling Routine which retrieves the integer code defined when an informational error occurs during the execution of IMSL MATH/LIBRARY Routines. See pages 1130-1134 in Chapter 8 of Reference D-3.

Routines Group 3

Principal VAX/VMS FORTRAN Routines used by OPTIMNN

Routine	Purpose of Routine
DABS	***Absolute Value*** VAX/VMS FORTRAN Double Precision Intrinsic Function. Y = DABS(ARG) = Abs(ARG) = |ARG| where: Y is REAL*8, ARG is REAL*8
DATAN2D	***Arc Tangent*** VAX/VMS FORTRAN Double Precision Intrinsic Function. Y = DATAN2D(ARG1/ARG2) + Tan^{-1}(ARG1/ARG2) where: Y is REAL*8, Y is in <u>Degrees</u>, -180 Degrees < Y < +180 Degrees, ARG1 is REAL*8 and ARG1 = Sin(Y), ARG2 is REAL*8 and ARG2 = Cos(Y)
DCOS	***Cosine*** VAX/VMS FORTRAN Double Precision Intrinsic Function. Y = DCOS(ARG) = Cos(ARG) where: Y is REAL*8, Y is in <u>Radians.</u> ARG is REAL*8
DCOSD	***Cosine (Degrees)*** VAX/VMS FORTRAN Double Precision Intrinsic Function. Y = DCOSD(ARG) = Cos(ARG) where: Y is REAL*8, Y is in <u>Degrees</u>, ARG is REAL*8
DCOSH	***Hyperbolic Cosine*** VAX/VMS FORTRAN Double Precision Intrinsic Function. Y = DCOSH(ARG) = Cosh(ARG) where: Y is REAL*8, ARG is REAL*8
DEXP	***Exponential*** VAX/VMS FORTRAN Double Precision Intrinsic Function. Y = DEXP(ARG) = Exp_e(ARG) where: Y is REAL*8, ARG is REAL*8

Routines Group 3 (Continued)

Principal VAX/VMS FORTRAN Routines used by OPTIMNN

Routine	Purpose of Routine
DFLOTJ	***INTEGER*4 to REAL*8 Conversion*** VAX/VMS FORTRAN Double Precision Intrinsic Function. This function converts the INTEGER*4 argument to the floating point REAL*8 equivalent which is returned as the function value. Y = DFLOTJ(IARG) = Float(IARG) where: Y is REAL*8, IARG is INTEGER*4
DINT	***Truncation (REAL*8 to REAL*8)*** VAX/VMS FORTRAN Double Precision Intrinsic Function This function converts the floating point REAL*8 argument ARG to the truncated floating point REAL*8 Y which is returned as the function value. Y is defined as the largest integral value whose magnitude does not exceed the magnitude of ARG and whose sign is the same as that of ARG. For example, DINT(7.9) equals 7.000 and JIDINT(-7.9) equals -7.000. Y = DINT(ARG) = Trunc(ARG) where: Y is REAL*8, ARG is REAL*8, Trunc(•) is the *Truncation Function.*
DLOG	***Natural Logarithm*** VAX/VMS FORTRAN Double Precision Intrinsic Function. Y = DLOG(ARG) = Log_e(ARG) = Ln(ARG) where: Y is REAL*8, ARG is REAL*8
DMAX1	***Selection of Maximum*** VAX/VMS FORTRAN Double Precision Intrinsic Function. This function returns the value of the argument in the argument list (ARG1, ARG2, ARG3, •, •, •, •) which has the greatest value. There must be at least two arguments in the argument list. Y = DMAX1(ARG1, ARG2, ARG3, •, •, •, •) where: Y is REAL*8, ARG1, ARG2, ARG3, •, •, •, • are REAL*8
DMIN1	***Selection of Minimum*** VAX/VMS FORTRAN Double Precision Intrinsic Function. This function returns the value of the argument in the argument list (ARG1, ARG2, ARG3, •, •, •, •) which has the least value. There must be at least two arguments in the argument list. Y = DMIN1(ARG1, ARG2, ARG3, •, •, •, •) where: Y is REAL*8, ARG1, ARG2, ARG3, •, •, •, • are REAL*8

Routines Group 3 (Continued)

Principal VAX/VMS FORTRAN Routines used by OPTIMNN

Routine	Purpose of Routine
DMOD	***Remainder*** VAX/VMS FORTRAN Intrinsic Function. This function returns the remainder when the first argument is divided by the second.

$$Y = DMOD(ARG1,ARG2)$$

$$Y = ARG1 - ARG2*Trunc(ARG1/ARG2)$$

$$Y = ARG1 - ARG2*DINT(ARG1/ARG2)$$

where: Y is REAL*8,
ARG1 is REAL*8,
ARG2 is REAL*8
Trunc(•) is the *Truncation Function.*

DSIGN — ***Transfer of Sign*** VAX/VMS FORTRAN Double Precision Intrinsic Function. This function assigns the sign of the second argument (ARG2) to the absolute value of the first argument (ARG1).

$$Y = DSIGN(ARG1,ARG2) = |ARG1|*Sgn(ARG2)$$

where: Y is REAL*8,
ARG1 is REAL*8, ARG2 is REAL*8

DSIN — ***Sine*** VAX/VMS FORTRAN Double Precision Intrinsic Function.

$$Y = DSIN(ARG) = Sin(ARG)$$

where: Y is REAL*8, Y is in Radians,
ARG is REAL*8

DSIND — ***Sine (Degrees)*** VAX/VMS FORTRAN Double Precision Intrinsic Function.

$$Y = DSIND(ARG) = Sin(ARG)$$

where: Y is REAL*8, Y is in Degrees,
ARG is REAL*8

DSQRT — ***Square Root*** VAX/VMS FORTRAN Double Precision Intrinsic Function.

$$Y = DSQRT(ARG) = \sqrt{ARG}$$

where: Y is REAL*8,
ARG is REAL*8, $ARG \geq 0.00000$

Routines Group 3 (Continued)

Principal VAX/VMS FORTRAN Routines used by OPTIMNN

Routine	Purpose of Routine
DTAN	***Tangent*** VAX/VMS FORTRAN Double Precision Intrinsic Function. Y = DTAN(ARG) = Tan(ARG) where: Y is REAL*8, Y is in Radians, ARG is REAL*8
DTAND	***Tangent (Degrees)*** VAX/VMS FORTRAN Double Precision Intrinsic Function. Y = DTAND(ARG) = Tan(ARG) where: Y is REAL*8, Y is in Degrees, ARG is REAL*8
DTANH	***Hyperbolic Tangent*** VAX/VMS FORTRAN Double Precision Intrinsic Function. Y = DTANH(ARG) = Tanh(ARG) where: Y is REAL*8, ARG is REAL*8
JIDINT	***Truncation (REAL*8 to INTEGER*4 Conversion)*** VAX/VMS FORTRAN Double Precision Intrinsic Function This function converts the floating point REAL*8 argument ARG to the truncated INTEGER*4 IY which is returned as the function value. IY is defined as the largest integer whose magnitude does not exceed the magnitude of ARG and whose sign is the same as that of ARG. For example, JIDINT(7.9) equals 7 and JIDINT(-7.9) equals -7. IY = JIDINT(ARG) = Trunc(ARG) where: IY is INTEGER*4, ARG is REAL*8, Trunc(•) is the *Truncation Function*.
JMAX0	***Selection of Maximum*** VAX/VMS FORTRAN Intrinsic Function. This function returns the value of the argument in the argument list (IARG1, IARG2, IARG3, •, •, •, •) which has the greatest value. There must be at least two arguments in the argument list. IY = JMAX0(IARG1, IARG2, IARG3, •, •, •, •) where: IY is INTEGER*4, IARG1, IARG2, IARG3, •, •, •, • are INTEGER*4

Principal VAX/VMS FORTRAN Routines used by OPTIMNN

Routine	Purpose of Routine

JMOD

Remainder VAX/VMS FORTRAN Intrinsic Function. This function returns the remainder when the first argument is divided by the second.

IY = JMOD(IARG1,IARG2)

IY = IARG1 − IARG2*Trunc(IARG1/IARG2)

IY = IARG1 − IARG2*(IARG1/IARG2)

where: IY is INTEGER*4,

IARG1 is INTEGER*4,

IARG2 is INTEGER*4

Trunc(•) is the *Truncation Function.*

RAN

Uniformly Distributed Random Number Generator VAX/VMS FORTRAN System Subroutine. RAN is a general random number generator of the multiplicative congruential type. RAN produces a Single Precision Floating Point (REAL*4) number that is uniformly distributed in the range between 0.00000 inclusive and 1.00000 inclusive ([0.00000, 1.00000]) from an input seed (ISEED).

Y = RAN(ISEED) = Urand(ISEED)

where: Y is REAL*4, $0.00000 \le Y \le +1.00000$

ISEED is INTEGER*4,

Urand(•) is the *Uniformly Random Distribution Function* such that

$$0.00000 \le \text{Urand}(\bullet) \le +1.00000$$

References

1. IMSL MATH/LIBRARY User's Manual, ***FORTRAN Subroutines for Mathematical Applications, Version 1.1, Volume 1***, MALB-USM-UNBND- EN8908-1.1, August 1989

2. IMSL MATH/LIBRARY User's Manual, ***FORTRAN Subroutines for Mathematical Applications, Version 1.1, Volume 2***, MALB-USM-UNBND- EN8901-1.1, January 1989

3. IMSL MATH/LIBRARY User's Manual, ***FORTRAN Subroutines for Mathematical Applications, Version 1.1, Volume 3***, MALB-USM-UNBND- EN8901-1.1, January 1989

4. ***Programming in VAX FORTRAN, Software Version: V4.0***, VAX/VMS Manual No AA-D034D-TE, Digital Equipment Corporation (DEC), Maynard, Mass, September 1984

Appendix C

Listing of the OPTIMNN Code

	Code Blocks	Category
A.	OPTIMNN.COM	DCL COMMAND Procedure
B.	TYPECOM	Data Type INCLUDE File
C.	INITDAT	Default Data Values INCLUDE File
00.	DN4ONF1	Modified IMSL DN4ONF Routine
0.	DN9ONG1	Modified IMSL DN9ONG Routine
1.	OPTIMNN	OPTIMNN Main Driver
2.	INIT	Initialisation Function
3.	TRAJ	Trajectory Propagation Procedure
4.	JNNW	Optimisation Support Function
5.	JCTRL	Optimisation Support Function
6.	CVVCTR	Optimisation Support Function
7.	ECVCTR	Optimisation Support Function
8.	STATENN	Neural-Network Modelling Function
9.	PFNCT00	Neural-Network Modelling Function
10.	PFNCT01	Neural-Network Modelling Function
11.	PFNCT02	Neural-Network Modelling Function
12.	PFNCT03	Neural-Network Modelling Function
13.	STATE	Trajectory Data Function
14.	ASTATE	Trajectory Data Function
15.	ASTATRAN	Trajectory Data Function
16.	ASTATE01	Trajectory Data Function
17.	ASTATE02	Trajectory Data Function
18.	ASTATE03	Trajectory Data Function
19.	ASTATE04	Trajectory Data Function
20.	ASTATE05	Trajectory Data Function
21.	ASTATE06	Trajectory Data Function
22.	ASTATE07	Trajectory Data Function
23.	DSTATE	Trajectory Data Function
24.	TSTATE	Trajectory Data Function
25.	USTATE	Trajectory Data Function

Appendix C

```
$ ASSIGN  SYS$COMMAND:  SYS$INPUT
$ ASSIGN  SYS$INPUT     FOR005
$ ASSIGN  SYS$OUTPUT    FOR006
$       SET TERM/WIDTH=80
$       SET VERIFY
$       SET NOVERIFY
$ !
$ ! *****   OPTIM COMMAND PROCEDURE:    OPTIM.COM      *****
$ !
$ !     ON WARNING THEN GOTO _______
$ !     ON ERROR THEN GOTO _______
$ !     ON SEVERE THEN GOTO _______
$ !
$ START:
$ !
$       INQUIRE EXPRAA "Express to RUN OPTIMNN?  (Y/N)"
$       IF EXPRAA .EQS. "N" THEN GOTO EXPR01
$       GOTO EXPR04
$ EXPR01:
$       INQUIRE EXPRBB "Express to LINK OPTIMNN?  (Y/N)"
$       IF EXPRBB .EQS. "N" THEN GOTO TYPE01
$       GOTO EXPR03
$ !
$ TYPE01:
$       INQUIRE TYPEAA "TYPE a File?  (Y/N)"
$       IF TYPEAA .EQS. "N" THEN GOTO EDIT01
$ !
$ ! *****   TYPE a File    *****
$ !
$       INQUIRE TYPEBB "ENTER NAME of File to be TYPED."
$       ON ERROR THEN GOTO TYPE02
$       TYPE 'TYPEBB'
$       GOTO TYPE01
$ TYPE02:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "ERROR Specifying File to be TYPED; Try Again."
$       WRITE SYS$OUTPUT " "
$       GOTO TYPE01
$ !
$ EDIT01:
$       INQUIRE EDITAA "EDIT a File?  (Y/N)"
$       IF EDITAA .EQS. "N" THEN GOTO CMPL01
$ !
$ ! *****   EDIT a File    *****
$ !
$       INQUIRE EDITBB "ENTER NAME of File to be EDITED."
$       ON ERROR THEN GOTO EDIT02
$       EDT 'EDITBB'
$       GOTO PURGE14
$ EDIT02:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "ERROR Specifying File to be EDITED; Try Again."
$       WRITE SYS$OUTPUT " "
$       GOTO EDIT01
$ !
$ CMPL01:
$       INQUIRE CMPLAA "COMPILE a File?  (Y/N)"
$       IF CMPLAA .EQS. "N" THEN GOTO LINK01
$ !
$ ! *****   COMPILE a File    *****
$ !
$       INQUIRE CMPLBB "ENTER NAME of File to be COMPILED."
$       INQUIRE CMPLCC "COMPILE a FORTRAN File?  (Y/N)"
$       IF CMPLCC .EQS. "N" THEN GO TO CMPL06
$ !
$ ! *****   FORTRAN Compilation    *****
$ !
$       INQUIRE CMPLDD "Specify the /LIST Qualifier?  (Y/N)"
$       IF CMPLDD .EQS. "N" THEN GOTO CMPL03
```

```
$       INQUIRE CMPLEE "Specify the /SHOW=INCLUDE Qualifier?  (Y/N)"
$       IF CMPLEE .EQS. "N" THEN GOTO CMPL02
$       ON ERROR THEN GOTO CMPL05
$ !     FOR/LIST/SHOW=INCLUDE/CROSS_REFERENCE 'CMPLBB'.FOR
$       FOR/LIST/SHOW=INCLUDE/CROSS_REFERENCE/NOWARNINGS 'CMPLBB'.FOR
$       GOTO PURGE01
$ CMPL02:
$       ON ERROR THEN GOTO CMPL05
$ !     FOR/LIST/CROSS_REFERENCE 'CMPLBB'.FOR
$       FOR/LIST/CROSS_REFERENCE/NOWARNINGS 'CMPLBB'.FOR
$       GOTO PURGE01
$ CMPL03:
$       INQUIRE CMPLFF "Specify the /SHOW=INCLUDE Qualifier?  (Y/N)"
$       IF CMPLFF .EQS. "N" THEN GOTO CMPL04
$       ON ERROR THEN GOTO CMPL05
$ !     FOR/SHOW=INCLUDE 'CMPLBB'.FOR
$       FOR/SHOW=INCLUDE/NOWARNINGS 'CMPLBB'.FOR
$       GOTO PURGE01
$ CMPL04:
$       ON ERROR THEN GOTO CMPL05
$ !     FOR 'CMPLBB'.FOR
$       FOR/NOWARNINGS 'CMPLBB'.FOR
$       GOTO PURGE01
$ CMPL05:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "ERROR in FORTRAN Compilation."
$       WRITE SYS$OUTPUT " "
$       GOTO TYPE01
$ !
$ ! *****   C  Compilation   *****
$ !
$ CMPL06:
$       INQUIRE CMPLGG "Specify the /LIST Qualifier?  (Y/N)"
$       IF CMPLGG .EQS. "N" THEN GOTO CMPL08
$       INQUIRE CMPLHH "Specify the /SHOW=INCLUDE Qualifier?  (Y/N)"
$       IF CMPLHH .EQS. "N" THEN GOTO CMPL07
$       ON ERROR THEN GOTO CMPL10
$       CC/LIST/SHOW=INCLUDE/CROSS_REFERENCE 'CMPLBB'.C
$       GOTO PURGE01
$ CMPL07:
$       ON ERROR THEN GOTO CMPL10
$       CC/LIST/CROSS_REFERENCE 'CMPLBB'.C
$       GOTO PURGE01
$ CMPL08:
$       INQUIRE CMPLII "Specify the /SHOW=INCLUDE Qualifier?  (Y/N)"
$       IF CMPLII .EQS. "N" THEN GOTO CMPL09
$       ON ERROR THEN GOTO CMPL10
$       CC/SHOW=INCLUDE 'CMPLBB'.C
$       GOTO PURGE01
$ CMPL09:
$       ON ERROR THEN GOTO CMPL10
$       CC 'CMPLBB'.C
$ CMPL10:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "ERROR in  CC  Compilation."
$       WRITE SYS$OUTPUT " "
$       GOTO TYPE01
$ !
$ PURGE01:
$       INQUIRE PURGEAA "Automatic PURGE?  (Y/N)"
$       IF PURGEAA .EQS. "N" THEN GOTO CMPL01
$ !
$ !   *****   Automatic PURGE of Previous Files   *****
$ !
$       IF CMPLCC .EQS. "N" THEN GO TO PURGE05
$       INQUIRE PURGEBB "PURGE  .FOR  Files?  (Y/N)"
$       IF PURGEBB .EQS. "N" THEN GOTO PURGE02
$ !
$ ! *****   PURGE  .FOR  Files   *****
```

```
$ !
$       DIR 'CMPLBB'.FOR
$       INQUIRE VERSAA "Are There EXACTLY Versions 1, 2, 3, and 4?  (Y/N)"
$       IF VERSAA .EQS. "N" THEN GOTO VERS01
$       DELETE 'CMPLBB'.FOR;1
$       RENAME 'CMPLBB'.FOR;4 'CMPLBB'.FOR;1
$       DELETE 'CMPLBB'.FOR;2
$       DELETE 'CMPLBB'.FOR;3
$       COPY 'CMPLBB'.FOR;1 'CMPLBB'.FOR;2
$       COPY 'CMPLBB'.FOR;1 'CMPLBB'.FOR;3
$       GOTO PURGE02
$ VERS01:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "PURGE NOT Executed for  ",CMPLBB,".FOR  Files."
$       WRITE SYS$OUTPUT " "
$ PURGE02:
$       DIR 'CMPLBB'.FOR
$       INQUIRE PURGECC "Continue?  (Y/N)"
$       IF PURGECC .EQS. "N" THEN GOTO PURGE03
$       GOTO PURGE04
$ PURGE03:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "Pause 7 Seconds."
$       WRITE SYS$OUTPUT " "
$       WAIT 00:00:07
$ PURGE04:
$       IF CMPLDD .EQS. "N"  THEN GOTO PURGE11
$       GOTO PURGE08
$ PURGE05:
$       INQUIRE PURGEDD "PURGE  .C  Files?  (Y/N)"
$       IF PURGEDD .EQS. "N" THEN GOTO PURGE06
$ !
$ ! *****   PURGE  .C  Files   *****
$ !
$       DIR 'CMPLBB'.C
$       INQUIRE VERSBB "Are There EXACTLY Versions 1, 2, 3, and 4?  (Y/N)"
$       IF VERSBB .EQS. "N" THEN GOTO VERS02
$       DELETE 'CMPLBB'.C;1
$       RENAME 'CMPLBB'.C;4 'CMPLBB'.C;1
$       DELETE 'CMPLBB'.C;2
$       DELETE 'CMPLBB'.C;3
$       COPY 'CMPLBB'.C;1 'CMPLBB'.C;2
$       COPY 'CMPLBB'.C;1 'CMPLBB'.C;3
$       GOTO PURGE06
$ VERS02:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "PURGE NOT Executed for  ",CMPLBB,".C  Files."
$       WRITE SYS$OUTPUT " "
$ PURGE06:
$       DIR 'CMPLBB'.C
$       INQUIRE PURGEEE "Continue?  (Y/N)"
$       IF PURGEEE .EQS. "N" THEN GOTO PURGE07
$       GOTO PURGE08
$ PURGE07:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "Pause 7 Seconds."
$       WRITE SYS$OUTPUT " "
$       WAIT 00:00:07
$       IF CMPLGG .EQS. "N"  THEN GOTO PURGE11
$ PURGE08:
$       INQUIRE PURGEFF "PURGE  .LIS  Files?  (Y/N)"
$       IF PURGEFF .EQS. "N" THEN GOTO PURGE09
$ !
$ ! *****   PURGE  .LIS  Files   *****
$ !
$       DIR 'CMPLBB'.LIS
$       INQUIRE VERSCC "Are There EXACTLY Versions 1, 2, 3, and 4?  (Y/N)"
$       IF VERSCC .EQS. "N" THEN GOTO VERS03
$       DELETE 'CMPLBB'.LIS;1
```

```
$       RENAME 'CMPLBB'.LIS;4 'CMPLBB'.LIS;1
$       DELETE 'CMPLBB'.LIS;2
$       DELETE 'CMPLBB'.LIS;3
$       COPY 'CMPLBB'.LIS;1 'CMPLBB'.LIS;2
$       COPY 'CMPLBB'.LIS;1 'CMPLBB'.LIS;3
$       GOTO PURGE09
$ VERS03:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "PURGE NOT Executed for  ",CMPLBB,".LIS  Files."
$       WRITE SYS$OUTPUT " "
$ PURGE09:
$       DIR 'CMPLBB'.LIS
$       INQUIRE PURGEGG "Continue?  (Y/N)"
$       IF PURGEGG .EQS. "N" THEN GOTO PURGE10
$       GOTO PURGE11
$ PURGE10:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "Pause 7 Seconds."
$       WRITE SYS$OUTPUT " "
$       WAIT 00:00:07
$ PURGE11:
$       INQUIRE PURGEHH "PURGE  .OBJ  Files?  (Y/N)"
$       IF PURGEHH .EQS. "N" THEN GOTO PURGE12
$ !
$ ! *****   PURGE  .OBJ  Files   *****
$ !
$       DIR 'CMPLBB'.OBJ
$       INQUIRE VERSDD "Are There EXACLY Versions 1, 2, 3, and 4?  (Y/N)"
$       IF VERSDD .EQS. "N" THEN GOTO VERS04
$       DELETE 'CMPLBB'.OBJ;1
$       RENAME 'CMPLBB'.OBJ;4 'CMPLBB'.OBJ;1
$       DELETE 'CMPLBB'.OBJ;2
$       DELETE 'CMPLBB'.OBJ;3
$       COPY 'CMPLBB'.OBJ;1 'CMPLBB'.OBJ;2
$       COPY 'CMPLBB'.OBJ;1 'CMPLBB'.OBJ;3
$       GOTO PURGE12
$ VERS04:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "PURGE NOT Executed for  ",CMPLBB,".OBJ  Files."
$       WRITE SYS$OUTPUT " "
$ PURGE12:
$       DIR 'CMPLBB'.OBJ
$       INQUIRE PURGEII "Continue?  (Y/N)"
$       IF PURGEII .EQS. "N" THEN GOTO PURGE13
$       GOTO CMPL01
$ PURGE13:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "Pause 7 Seconds."
$       WRITE SYS$OUTPUT " "
$       WAIT 00:00:07
$       GOTO CMPL01
$ PURGE14:
$       INQUIRE PURGEJJ "Automatic PURGE of Compile-Only File?  (Y/N)"
$       IF PURGEJJ .EQS. "N" THEN GOTO PURGE15
$ !
$ !   *****   Automatic PURGE of "Compile-Only" File   *****
$ !
$       DIR 'EDITBB'
$       INQUIRE VERSEE "Are There EXACTLY Versions 1, 2, 3, and 4?  (Y/N)"
$       IF VERSEE .EQS. "N" THEN GOTO VERS05
$       DELETE 'EDITBB';1
$       RENAME 'EDITBB';4 'EDITBB';1
$       DELETE 'EDITBB';2
$       DELETE 'EDITBB';3
$       COPY 'EDITBB';1 'EDITBB';2
$       COPY 'EDITBB';1 'EDITBB';3
$       GOTO PURGE15
$ VERS05:
$       WRITE SYS$OUTPUT " "
```

```
$       WRITE SYS$OUTPUT "PURGE NOT Executed for  ",EDITBB,"  Files."
$       WRITE SYS$OUTPUT " "
$ PURGE15:
$       DIR 'EDITBB'
$       INQUIRE PURGEKK "Continue?  (Y/N)"
$       IF PURGEKK .EQS. "N" THEN GOTO PURGE16
$       GOTO EXPR02
$ PURGE16:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "Pause 7 Seconds."
$       WRITE SYS$OUTPUT " "
$       WAIT 00:00:07
$ EXPR02:
$       INQUIRE EXPRCC "Express to RUN OPTIMNN?  (Y/N)"
$       IF EXPRCC .EQS. "N" THEN GOTO TYPE01
$       GOTO EXPR04
$ !
$ LINK01:
$       INQUIRE LINKAA "LINK the OPTIMNN Code? (Y/N)"
$       IF LINKAA .EQS. "N" THEN GOTO RUN01
$ !
$ ! *****   LINK the OPTIMNN Routines    *****
$ !
$ EXPR03:
$       INQUIRE LINKBB "LINK with IMSL Optimisation System? (Y/N)"
$       IF LINKBB .EQS. "N" THEN GOTO LINK03
$       INQUIRE LINKCC "LINK with /MAP/CROSS_REFERENCE Qualifiers? (Y/N)"
$       IF LINKCC .EQS. "N" THEN GOTO LINK02
$ !
$ ! *****   LINK Code with the IMSL Shared Library and the
$ !                                   /MAP/CROSS_REFERENCE Qualifiers    *****
$ !
$       ON ERROR THEN GOTO LINK05
$ !     LINK/MAP/CROSS_REFERENCE        OPTIMNN,  INIT,     TRAJ,-
$ !       JNNW,     JCTRL,    CVVCTR,   ECVCTR,   STATENN,  PFNCT00,-
$ !       PFNCT01,  PFNCT02,  PFNCT03,  STATE,    ASTATE,   ASTATRAN,-
$ !       ASTATE01, ASTATE02, ASTATE03, ASTATE04, ASTATE05, ASTATE06,-
$ !       ASTATE07, DSTATE,   TSTATE,   USTATE,   IMSLIBG_SHARE/OPT
$ !
$ ! *****   LINK Code with the IMSL Static Library and the
$ !                                   /MAP/CROSS_REFERENCE Qualifiers    *****
$ !
$       LINK/MAP/CROSS_REFERENCE        OPTIMNN,  INIT,     TRAJ,-
          JNNW,     JCTRL,    CVVCTR,   ECVCTR,   STATENN,  PFNCT00,-
          PFNCT01,  PFNCT02,  PFNCT03,  STATE,    ASTATE,   ASTATRAN,-
          ASTATE01, ASTATE02, ASTATE03, ASTATE04, ASTATE05, ASTATE06,-
          ASTATE07, DSTATE,   TSTATE,   USTATE,   DN4ONF1,  DN9ONG1,-
          IMSLIBG_STATIC/OPT,IMSLPSECT/OPT
$       GOTO RUN01
$ LINK02:
$ !
$ ! *****   LINK Code with the IMSL Shared Library with NO
$ !                                   /MAP/CROSS_REFERENCE Qualifiers    *****
$ !
$       ON ERROR THEN GOTO LINK05
$ !     LINK                            OPTIMNN,  INIT,     TRAJ,-
$ !       JNNW,     JCTRL,    CVVCTR,   ECVCTR,   STATENN,  PFNCT00,-
$ !       PFNCT01,  PFNCT02,  PFNCT03,  STATE,    ASTATE,   ASTATRAN,-
$ !       ASTATE01, ASTATE02, ASTATE03, ASTATE04, ASTATE05, ASTATE06,-
$ !       ASTATE07, DSTATE,   TSTATE,   USTATE,   IMSLIBG_SHARE/OPT
$ !
$ ! *****   LINK Code with the IMSL Static Library with NO
$ !                                   /MAP/CROSS_REFERENCE Qualifiers    *****
$ !
$       LINK                            OPTIMNN,  INIT,     TRAJ,-
          JNNW,     JCTRL,    CVVCTR,   ECVCTR,   STATENN,  PFNCT00,-
          PFNCT01,  PFNCT02,  PFNCT03,  STATE,    ASTATE,   ASTATRAN,-
          ASTATE01, ASTATE02, ASTATE03, ASTATE04, ASTATE05, ASTATE06,-
          ASTATE07, DSTATE,   TSTATE,   USTATE,   DN4ONF1,  DN9ONG1,-
```

```
         IMSLIBG_STATIC/OPT,IMSLPSECT/OPT
$        GOTO RUN01
$ LINK03:
$        INQUIRE LINKDD "LINK with /MAP/CROSS_REFERENCE Qualifiers? (Y/N)"
$        IF LINKDD .EQS. "N" THEN GOTO LINK04
$ !
$ ! *****   LINK Code with KJAG and the  /MAP/CROSS_REFERENCE Qualifiers    *****
$ !
$        ON ERROR THEN GOTO LINK05
$        LINK/MAP/CROSS_REFERENCE        OPTIMNN,  INIT,      TRAJ,-
           JNNW,      JCTRL,    CVVCTR,   ECVCTR,   STATENN,  PFNCT00,-
           PFNCT01,  PFNCT02,  PFNCT03,  STATE,    ASTATE,   ASTATRAN,-
           ASTATE01, ASTATE02, ASTATE03, ASTATE04, ASTATE05, ASTATE06,-
           ASTATE07, DSTATE,   TSTATE,   USTATE,   DN4ONF1,  DN9ONG1,-
           WORK,     [LEYLAND.OPTIMNN.OPTIMNN1]SRCVLIB.OLB
$        GOTO RUN01
$ LINK04:
$ !
$ ! *****   LINK Code with KJAG with NO  /MAP/CROSS_REFERENCE Qualifiers    *****
$ !
$        ON ERROR THEN GOTO LINK05
$        LINK                            OPTIMNN,  INIT,      TRAJ,-
           JNNW,      JCTRL,    CVVCTR,   ECVCTR,   STATENN,  PFNCT00,-
           PFNCT01,  PFNCT02,  PFNCT03,  STATE,    ASTATE,   ASTATRAN,-
           ASTATE01, ASTATE02, ASTATE03, ASTATE04, ASTATE05, ASTATE06,-
           ASTATE07, DSTATE,   TSTATE,   USTATE,   DN4ONF1,  DN9ONG1,-
           WORK,     [LEYLAND.OPTIMNN.OPTIMNN1]SRCVLIB.OLB
$        GOTO RUN01
$ LINK05:
$        WRITE SYS$OUTPUT " "
$        WRITE SYS$OUTPUT "ERROR in Linking. Terminate Process."
$        WRITE SYS$OUTPUT " "
$        GO TO TERMINATE
$ RUN01:
$        INQUIRE RUNAA "RUN OPTIMNN? (Y/N)"
$        IF RUNAA .EQS. "N" THEN GOTO TERMINATE
$ !
$ ! *****   RUN  OPTIMNN   *****
$ !
$ ! *****   Clear INPUT (FOR007.DAT) and OUTPUT (FOR008.DAT)
$ !         Data   *****
$ !
$ EXPR04:
$        ON ERROR THEN GOTO RUN02 .
$        DELETE FOR007.*;*
$ RUN02:
$        ON ERROR THEN GOTO RUN03
$        DELETE FOR008.*;*
$ RUN03:
$ !
$ ! *****   COPY INPUT  CDATA.DAT  File to  FOR007.DAT    *****
$ !
$        ON ERROR THEN GOTO RUN04
$        COPY CDATA.DAT FOR007.DAT
$        GOTO RUN05
$ RUN04:
$        WRITE SYS$OUTPUT " "
$        WRITE SYS$OUTPUT "ERROR with the INPUT. Terminate Process."
$        WRITE SYS$OUTPUT " "
$        GOTO TERMINATE
$ RUN05:
$        INQUIRE RUNBB "Delete Previous EDATA.DAT;*  OUTPUT Files? (Y/N)"
$        IF RUNBB .EQS. "N" THEN GOTO RUN07
$ !
$ ! *****   Delete Previous EDATA.DAT;*  OUTPUT Files.   *****
$ !
$        ON ERROR THEN GOTO RUN06
$        DELETE EDATA.*;*
$        GOTO RUN07
```

```
$ RUN06:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "ERROR Clearing OUTPUT (EDATA.DAT;*);-
 Continue Process."
$       WRITE SYS$OUTPUT " "
$ RUN07:
$       INQUIRE RUNCC "TYPE INPUT Data (CDATA.DAT)? (Y/N)"
$       IF RUNCC .EQS. "N" THEN GOTO RUN09
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "INPUT Data File CDATA.DAT."
$       WRITE SYS$OUTPUT " "
$       ON ERROR THEN GOTO RUN08
$ !
$ ! *****   TYPE INPUT File CDATA.DAT/FOR007.DAT Before Execution.   *****
$ !
$       TYPE FOR007.DAT
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "End of INPUT Data File CDATA.DAT."
$       WRITE SYS$OUTPUT " "
$       GOTO RUN09
$ RUN08:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "ERROR with INPUT (CDATA.DAT/FOR007.DAT).-
 Terminate Process."
$       WRITE SYS$OUTPUT " "
$       GOTO TERMINATE
$ RUN09:
$       ASSIGN EDATA.DAT SYS$OUTPUT
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "START RUN."
$       WRITE SYS$OUTPUT " "
$       SET TERM/WIDTH=132
$       ON ERROR THEN GOTO RUN10
$ !
$ ! *****   Execute  OPTIMNN   *****
$ !
$       RUN OPTIMNN
$ !
$       SET TERM/WIDTH=80
$       GOTO RUN11
$ RUN10:
$       SET TERM/WIDTH=80
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "INPUT (ERROR in Running OPTIMNN. Continue."
$       WRITE SYS$OUTPUT " "
$ RUN11:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "END of RUN."
$       WRITE SYS$OUTPUT " "
$       DEASSIGN SYS$OUTPUT
$       ON ERROR THEN GOTO RUN12
$       DELETE FOR007.*;*
$ RUN12:
$ !     ON ERROR THEN GOTO RUN13
$ !     RENAME FOR008.DAT EDATA.DAT
$ RUN13:
$       INQUIRE RUNDD "TYPE INPUT Data (CDATA.DAT)? (Y/N)"
$       IF RUNDD .EQS. "N" THEN GOTO RUN15
$ !
$ ! *****   TYPE INPUT File CDATA.DAT   *****
$ !
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "INPUT Data File CDATA.DAT."
$       WRITE SYS$OUTPUT " "
$       ON ERROR THEN GOTO RUN14
$       TYPE CDATA.DAT
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "End of INPUT Data File CDATA.DAT."
$       WRITE SYS$OUTPUT " "
```

```
$       GOTO RUN13
$ RUN14:
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "ERROR with INPUT (CDATA.DAT).  Continue Process."
$       WRITE SYS$OUTPUT " "
$       GOTO RUN13
$ RUN15:
$       INQUIRE RUNEE "TYPE OUTPUT Data (EDATA.DAT)? (Y/N)"
$       IF RUNEE .EQS. "N" THEN GOTO TERMINATE
$ !
$ ! *****   TYPE OUTPUT File EDATA.DAT    *****
$ !
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "OUTPUT Data File EDATA.DAT."
$       WRITE SYS$OUTPUT " "
$       SET TERM/WIDTH=132
$       ON ERROR THEN GOTO RUN16
$       TYPE EDATA.DAT
$       SET TERM/WIDTH=80
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "End of OUTPUT Data File EDATA.DAT."
$       WRITE SYS$OUTPUT " "
$       GOTO RUN15
$ RUN16:
$       SET TERM/WIDTH=80
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "ERROR with OUTPUT (EDATA.DAT).  Continue Process."
$       WRITE SYS$OUTPUT " "
$       GOTO RUN15
$ !
$ TERMINATE:
$ !
$ ! *****   TERMINATE RUN.    *****
$ !
$       WRITE SYS$OUTPUT " "
$       WRITE SYS$OUTPUT "TERMINATE RUN."
$       WRITE SYS$OUTPUT " "
$ EXIT
```

```
C
C
C   *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C           This file contains the statements which establish and define:
C           1) the Principal COMMON Blocks;  2) the Data TYPE of the
C           Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C           DIMENSION of the Principal  Arrays  and  Vectors  of the
C           OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
```

```
C
C
C
C     *****   Start TYPECOM.INC   *****
C
C
C     *****   These statements establish and define:   1) the Principal
C             COMMON Blocks;   2) the Data TYPE of the Principal  Parameters,
C             Arrays,  and Vectors;   and 3) the DIMENSION of the Principal
C             Arrays  and  Vectors  of the OPTIMNN System.
C
C
C
C
      IMPLICIT NONE
C
C
C
C     *****   Data Type for the Group 1 Parameters   *****
C
      INTEGER*4   NCON,     NCV,      NEC,      NIDIM,    NIJKDIM,
     1  NJDIM,    NJKDIM,   NKDIM,    NL1DIM,   NL21,     NL2DIM,
     2  NL321,    NL3DIM,   NLDIM,    NLTBL
C
C     *****   Dimensions for Arrays and Vectors   *****
C
      PARAMETER (NIDIM=16, NJDIM=16, NKDIM=4, NL1DIM=2, NL2DIM=12,
     1 NL3DIM=7, NLDIM=300, NLTBL=600)
C
      PARAMETER (NCV=JMAX0(NL2DIM, NIDIM*NJDIM*NKDIM), NEC=NL2DIM,
     1 NCON=NL2DIM)
C
      PARAMETER (NJKDIM=NJDIM*NKDIM, NIJKDIM=NIDIM*NJDIM*NKDIM)
C
      PARAMETER (NL21=NL2DIM*NL1DIM, NL321=NL3DIM*NL2DIM*NL1DIM)
C
C
C
C     *****   Data Type, Dimension, and COMMON for the Group 2 Parameters   *****
C
      INTEGER*4  MULT, TBLMAX
C
      REAL*8   CONST1, CONST2, CONST3, CONST4, CONST5, LARGE1, LARGE2,
     1 LARGE3, LARGE4, SMALL1, SMALL2, SMALL3, SMALL4, TD(NLDIM),
     2 TINIT,  TFINL,          TTBL(NLTBL),     XD(NL2DIM,NLDIM),
     3 XTBL(NL2DIM,NLTBL),     YD(NL2DIM,NLDIM),
     4 YTBL(NL2DIM,NLTBL)
C
      COMMON / GRP2 /  CONST1, CONST2, CONST3, CONST4, CONST5, LARGE1,
     1 LARGE2, LARGE3, LARGE4, MULT,   SMALL1, SMALL2, SMALL3, SMALL4,
     2 TBLMAX, TD,     TINIT,  TFINL,  TTBL,   XD,     XTBL,   YD,
     3 YTBL
C
C
C
C     *****   Data Type, Dimension, and COMMON for the Group 3 Parameters   *****
C
      INTEGER*4  DLFREQ,  DLLGTH,  LDELAY,  NNLID,   STMODL,  TLTYPE
C
      REAL*8  TLINIT,  TLFINL,  TLSTEP,  WTSNNL(NLDIM)
C
      COMMON / GRP3 /  DLFREQ, DLLGTH, LDELAY, NNLID, STMODL, TLINIT,
     1 TLFINL, TLSTEP, TLTYPE, WTSNNL
C
C
C
C     *****   Data Type, Dimension, and COMMON for the Group 4 Parameters   *****
C
      INTEGER*4  CDELAY, CVTID,  DCFREQ, DCLGTH, ISTEP0, NNCID,  STMODC,
```

```
     1           TCTYPE, UPDATE
C
      REAL*8  TCINIT, TCFINL, TCSTEP,  WTSNNC(NLDIM)
C
      COMMON / GRP4 /  CDELAY, CVTID,  DCFREQ, DCLGTH, ISTEP0, NNCID,
     1 STMODC, TCINIT, TCFINL, TCSTEP, TCTYPE, UPDATE, WTSNNC
C
C
C
C  *****  Data Type, Dimension, and COMMON for the Group 5 Parameters    *****
C
      INTEGER*4  NFUNCT(NJDIM,NKDIM),  NI(NKDIM),  NJ(NKDIM),  NK
C
      REAL*8  AN(NJDIM,NKDIM),        BN(NJDIM,NKDIM),  CN(NJDIM,NKDIM),
     1        CW(NIDIM,NJDIM,NKDIM),  DN(NJDIM,NKDIM),  XN0(NJDIM,NKDIM),
     2        YN0(NJDIM,NKDIM)
C
      COMMON / GRP5 / AN, BN, CN, CW, DN, NFUNCT, NI, NJ, NK, XN0, YN0
C
C
C
C  *****  Data Type, Dimension, and COMMON for the Group 6 Parameters    *****
C
      INTEGER*4                              IFUNCT(NL3DIM,NL2DIM,NL1DIM),
     1   ISEED1(NL3DIM,NL2DIM,NL1DIM),       ISEED2(NL3DIM,NL2DIM,NL1DIM),
     2   ISEED3(NL2DIM,NL1DIM),              JSEED1(NL3DIM,NL2DIM,NL1DIM),
     2   JSEED2(NL3DIM,NL2DIM,NL1DIM),       JSEED3(NL2DIM,NL1DIM),
     3   NL2(NL1DIM),                        NL3(NL2DIM,NL1DIM)
C
      REAL*8                                 A(NL3DIM,NL2DIM,NL1DIM),
     1   A1(NL3DIM,NL2DIM,NL1DIM),           A2(NL3DIM,NL2DIM,NL1DIM),
     2   A3(NL2DIM,NL1DIM),                  ALPHA(NL3DIM,NL2DIM,NL1DIM),
     3   B(NL3DIM,NL2DIM,NL1DIM),            B1(NL3DIM,NL2DIM,NL1DIM),
     4   B2(NL3DIM,NL2DIM,NL1DIM),           B3(NL2DIM,NL1DIM),
     5   C(NL3DIM,NL2DIM,NL1DIM),            C1(NL3DIM,NL2DIM,NL1DIM),
     6   C2(NL3DIM,NL2DIM,NL1DIM),           C3(NL2DIM,NL1DIM),
     7   D(NL3DIM,NL2DIM,NL1DIM),            D1(NL3DIM,NL2DIM,NL1DIM),
     8   D2(NL3DIM,NL2DIM,NL1DIM),           D3(NL2DIM,NL1DIM),
     9   NN(NL3DIM,NL2DIM,NL1DIM),           OMEGA(NL3DIM,NL2DIM,NL1DIM),
     o   PERIOD(NL3DIM,NL2DIM,NL1DIM),       PHASE(NL3DIM,NL2DIM,NL1DIM),
     1   PHI(NL3DIM,NL2DIM,NL1DIM),          PSI(NL3DIM,NL2DIM,NL1DIM),
     2   TWOPI0,                             X0(NL3DIM,NL2DIM,NL1DIM),
     3   Y0(NL3DIM,NL2DIM,NL1DIM),           YR1(NL3DIM,NL2DIM,NL1DIM),
     4   YR2(NL3DIM,NL2DIM,NL1DIM),          YR3(NL2DIM,NL1DIM)
C
      COMMON / GRP6 /   A,       A1,      A2,      A3,      ALPHA,   B,
     1   B1,      B2,      B3,      C,       C1,      C2,      C3,      D,
     2   D1,      D2,      D3,      IFUNCT,  ISEED1,  ISEED2,  ISEED3,  JSEED1,
     3   JSEED2,  JSEED3,  NL2,     NL3,     NN,      OMEGA,   PERIOD,  PHASE,
     4   PHI,     PSI,     X0,      Y0,      YR1,     YR2,     YR3
C
C
C
C  *****  Data Type, Dimension, and COMMON for the Group 7 Parameters    *****
C
      INTEGER*4  IJKCVL(NIDIM,NJDIM,NKDIM),    JJECL(NL2DIM),
     1           ICONNNL(NIDIM,NJDIM,NKDIM),   IOPTNNL,
     2           MITNNNL,                      OUTNNL
C
      REAL*8   SCVNNL(NIDIM,NJDIM,NKDIM),    WTNNL(NL2DIM),
     1         AMAXNNL(NIDIM,NJDIM,NKDIM),   AMINNNL(NIDIM,NJDIM,NKDIM)
C
      COMMON / GRP7 /   IJKCVL,   SCVNNL,   JJECL,    WTNNL,    AMAXNNL,
     1         AMINNNL,  ICONNNL,  IOPTNNL,  MITNNNL,  OUTNNL
C
C
C
C  *****  Data Type, Dimension, and COMMON for the Group 8 Parameters    *****
C
```

```
      INTEGER*4  IJKCVC(NIDIM,NJDIM,NKDIM),    JJECC(NL2DIM),
     1           ICONNNC(NIDIM,NJDIM,NKDIM),   IOPTNNC,
     2           MITNNNC,                      OUTNNC
C
      REAL*8   SCVNNC(NIDIM,NJDIM,NKDIM),   WTNNC(NL2DIM),
     1         AMAXNNC(NIDIM,NJDIM,NKDIM),  AMINNNC(NIDIM,NJDIM,NKDIM)
C
      COMMON / GRP8 /   IJKCVC,   SCVNNC,   JJECC,   WTNNC,   AMAXNNC,
     1       AMINNNC,   ICONNNC,  IOPTNNC,  MITNNNC,  OUTNNC
C
C
C
C   *****   Data Type, Dimension, and COMMON for the Group 9 Parameters   *****
C
      INTEGER*4  ICV(NL2DIM),   JEC(NL2DIM),   ICONC(NL2DIM),   IOPTC,
     1           MITNC,         OUTC
C
      REAL*8   SMAXC(NL2DIM),   WTC(NL2DIM),    AMAXC(NL2DIM),
     1         AMINC(NL2DIM),   SCVC(NL2DIM)
C
      COMMON / GRP9 /   ICV,   SCVC,   JEC,   WTC,   AMAXC,   AMINC,
     1 ICONC,  SMAXC,  IOPTC,  MITNC,  OUTC
C
C
C
C   *****   Data Type, Dimension, and COMMON for the Group A Parameters   *****
C
      INTEGER*4   CVUP,   DATAR,   DELAY,   DFREQ,   DFREQ0,  DLGTH,  ICUT,
     1   IPHASE,  ISTEP,  LMAX,    LSTEP,   LTBL,    NNID,    NNUP,   NNUP0
C
      REAL*8    T,        TABS,        TCUT,        TREL,        TSTEP,
     1 XA(NL2DIM),        XN(NL2DIM),  YA(NL2DIM),  YN(NL2DIM)
C
      COMMON / GRPA /  CVUP,  DATAR,  DELAY,  DFREQ,  DFREQ0,  DLGTH,  ICUT,
     1 IPHASE,  ISTEP,   LMAX,   LSTEP,  LTBL,   NNID,   NNUP,   NNUP0,  T,
     2 TABS,    TCUT,    TREL,   TSTEP,  XA,     XN,     YA,     YN
C
C
C
C   *****   Data Type, Dimension, and COMMON for the Group B Parameters   *****
C
      REAL*8  UNN(NJDIM,NKDIM),  XNN(NIDIM,NJDIM,NKDIM),  YNN(NJDIM,NKDIM)
C
      COMMON / GRPB / UNN, XNN, YNN
C
C
C
C   *****   Data Type, Dimension, and COMMON for the Group C Parameters   *****
C
      INTEGER*4     CVBDC,      CVBDNNC,     CVBDNNL,    ICVDEF,     IECDEF,
     1  II,         III,        IIJK,        IJK,        JJ,         JJJ,
     2  NCONC,      NCONNNC,    NCONNNL,     NICV,       NIJKCVC,    NIJKCVL,
     3  NJEC,       NJJECC,     NJJECL
C
      REAL*8                   CMAXC(NCV),     CMAXNNC(NCV),   CMAXNNL(NCV),
     1  CMINC(NCV),            CMINNNC(NCV),   CMINNNL(NCV),   CON(NCON),
     2  CV(NCV),               CV0(NCV),       CVSC(NCV),      CVSNNC(NCV),
     3  CVSNNL(NCV),           EC(NEC),        PINDX,          SUMSQ,
     4  SUMSQW(NLDIM),         WC(NEC),        WNNC(NEC),      WNNL(NEC)
C
      COMMON / GRPC /        CMAXC,     CMAXNNC,  CMAXNNL,  CMINC,    CMINNNC,
     1  CMINNNL,  CON,       CV,        CV0,      CVBDC,    CVBDNNC,  CVBDNNL,
     2  CVSC,     CVSNNC,    CVSNNL,    EC,       ICVDEF,   IECDEF,   II,
     3  III,      IIJK,      IJK,       JJ,       JJJ,      NCONC,    NCONNNC,
     4  NCONNNL,  NICV,      NIJKCVC,   NIJKCVL,  NJEC,     NJJECC,   NJJECL,
     5  PINDX,    SUMSQ,     SUMSQW,    WC,       WNNC,     WNNL
C
C
C
```

```
C     *****   Data Type, Dimension, and COMMON for the Group D Parameters    *****
C
      REAL*8           ZERO,   TENM8, TENM6, TENM3, TENM2, PT100, PT200,
     1  PT300, PT500, PT800, ONE,    TWO,   EBASE, THREE, PI,    FIVE,
     2  TWOPI, EIGHT, TEN,   RTD,    TENP2, TENP3, TENP6, TENP8
C
      COMMON / GRPD / ZERO,   TENM8, TENM6, TENM3, TENM2, PT100, PT200,
     1  PT300, PT500, PT800, ONE,    TWO,   EBASE, THREE, PI,    FIVE,
     2  TWOPI, EIGHT, TEN,   RTD,    TENP2, TENP3, TENP6, TENP8
C
C
C
C     *****   End TYPECOM.INC   *****
C
C
C
```

```
C
C
C     *****  The  "[LEYLAND.OPTIMNN]INITDAT.INC"  File is Included here.
C            This file contains the statements which define the initially
C            set Default Values of the  "NAMELIST CDATA"  INPUT Parameters
C            and the Values of the Internally Set Constants of the OPTIMNN
C            System.
C
      INCLUDE '[LEYLAND.OPTIMNN]INITDAT.INC'
C
C
C
```

```
C
C
C
C   *****   Start INITDAT.INC   *****
C
C
C   *****   These statements define the initially set Default Values of
C           the "NAMELIST CDATA" INPUT Parameters and the Values of the
C           Internally Set Constants of the OPTIMNN System.
C
C
C
C
C   *****   DATA Set Values for the Group 1 Parameters   *****
C
C                      NONE
C
C
C
C   *****   DATA Set Values for the Group 2 Parameters   *****
C
      DATA    CONST1,         CONST2,         CONST3,         CONST4,
     1        CONST5,         LARGE1,         LARGE2,         LARGE3,
     2        LARGE4,         SMALL1,         SMALL2,         SMALL3,
     3        SMALL4,         TBLMAX,         TINIT,          TFINL   /
     o        0.200,          0.500,          0.800,          1.200,
     1        1.500,          1.0D+03,        1.0D+06,        1.0D+09,
     2        1.0D+12,        1.0D-03,        1.0D-06,        1.0D-09,
     3        1.0D-12,        1,              0.000,          0.000   /
C
C
C
C   *****   DATA Set Values for the Group 3 Parameters   *****
C
      DATA    DLFREQ,         DLLGTH,         LDELAY,         NNLID,
     1        STMODL,         TLINIT,         TLFINL,         TLSTEP,
     2        TLTYPE,         WTSNNL                                  /
     o        1,              10,             0,              1,
     1        1,              0.000,          0.000,          1.000,
     2        0,              NLDIM*1.000                             /
C
C
C
C   *****   DATA Set Values for the Group 4 Parameters   *****
C
      DATA    CDELAY,         CVTID,          DCFREQ,         DCLGTH,
     1        ISTEP0,         NNCID,          STMODC,         TCINIT,
     2        TCFINL,         TCSTEP,         TLTYPE,         UPDATE,
     3        WTSNNC                                                  /
     o        0,              1,              1,              10,
     1        1,              1,              1,              0.000,
     2        0.000,          1.000,          0,              1,
     3        NLDIM*1.000                                             /
C
C
C   *****   DATA Set Values for the Group 5 Parameters   *****
C
      DATA    AN,             BN,             CN,             CW,
     1        DN,             NFUNCT,         NI,             NJ,
     2        NK,             XN0,            YN0                     /
     o    NJKDIM*0.500,  NJKDIM*0.500,  NJKDIM*1.000,  NIJKDIM*1.000,
     1    NJKDIM*-1.0D+06,  NJKDIM*0,     NKDIM*3,        NKDIM*1,
     2        2,          NJKDIM*0.000,  NJKDIM*0.000                 /
C
C
C
C   *****   DATA Set Values for the Group 6 Parameters   *****
C
      PARAMETER (TWOPI0=6.28318530717958647693D+00)
```

```
C
      DATA      A,                  A1,                 A2,                 A3,
     1          ALPHA,              B,                  B1,                 B2,
     2          B3,                 C,                  C1,                 C2,
     3          C3,                 D,                  D1,                 D2,
     4          D3,                 IFUNCT,             ISEED1,             ISEED2,
     5          ISEED3,             JSEED1,             JSEED2,             JSEED3,
     6          NL2,                NL3,                NN,                 OMEGA,
     7          PERIOD,             PHASE,              PHI,                PSI,
     8          X0,                 Y0,                 YR1,                YR2,
     9          YR3                                                               /
     o      NL321*0.500,     NL321*0.000,     NL321*0.000,       NL21*0.000,
     1      NL321*1.000,     NL321*0.500,     NL321*0.000,      NL321*0.000,
     2       NL21*0.000,     NL321*0.250,     NL321*0.000,      NL321*0.000,
     3       NL21*0.000, NL321*-1.0D+06,      NL321*0.000,      NL321*0.000,
     4       NL21*0.000,        NL321*0,   NL321*78985723, NL321*81692875,
     5    NL21*72919329, NL321*95428381,  NL321*68377297,   NL21*89672847,
     6        NL1DIM*1,         NL21*1,      NL321*1.000,    NL321*TWOPI0,
     7   NL321*1.0D+10,     NL321*0.000,     NL321*0.000,      NL321*0.000,
     8     NL321*0.000,     NL321*0.000,     NL321*0.000,      NL321*0.000,
     9      NL21*0.000                                                            /
C
C
C
C    *****    DATA Set Values for the Group 7 Parameters     *****
C
      DATA     IJKCVL,             SCVNNL,             JJECL,              WTNNL,
     1          AMAXNNL,            AMINNNL,            ICONNNL,            IOPTNNL,
     2          MITNNNL,            OUTNNL                                        /
     o       NIJKDIM*0,     NIJKDIM*1.000,       NL2DIM*0,       NL2DIM*1.000,
     1    NIJKDIM*100.0,  NIJKDIM*-100.0,       NIJKDIM*0,                 0,
     2          200,                0                                             /
C
C
C
C    *****    DATA Set Values for the Group 8 Parameters     *****
C
      DATA     IJKCVC,             SCVNNC,             JJECC,              WTNNC,
     1          AMAXNNC,            AMINNNC,            ICONNNC,            IOPTNNC,
     2          MITNNNC,            OUTNNC                                        /
     o       NIJKDIM*0,     NIJKDIM*1.000,       NL2DIM*0,       NL2DIM*1.000,
     1    NIJKDIM*100.0,  NIJKDIM*-100.0,       NIJKDIM*0,                 0,
     2          200,                0                                             /
C
C
C
C    *****    DATA Set Values for the Group 9 Parameters     *****
C
      DATA     ICV,                SCVC,               JEC,                WTC,
     1          AMAXC,              AMINC,              ICONC,              SMAXC,
     2          IOPTC,              MITNC,              OUTC                      /
     o        NL2DIM*0,     NL2DIM*1.000,        NL2DIM*0,       NL2DIM*1.000,
     1    NL2DIM*10.00,   NL2DIM*-10.00,         NL2DIM*0,       NL2DIM*10.00,
     2            0,              200,                0                           /
C
C
C
C    *****    DATA Set Values for the Group A Parameters     *****
C
C                        NONE
C
C
C
C    *****    DATA Set Values for the Group B Parameters     *****
C
C                        NONE
C
C
C
```

```
C     *****   DATA Set Values for the Group C Parameters    *****
C
       DATA     CON,     CVBDC,   CVBDNNC,    CVBDNNL,    PINDX,     SUMSQ,
      1          SUMSQW                                                     /
      o      NCON*0.000,   0,          0,          0,       0.000,     0.000,
      1      NLDIM*0.000                                                    /
C
C
C
C     *****   DATA Set Values for the Group D Parameters    *****
C
       DATA          ZERO,           TENM8,          TENM6,          TENM3,
      1              TENM2,          PT100,          PT200,          PT300,
      2              PT500,          PT800,          ONE,            TWO,
      3              EBASE,                          THREE,
      4              PI,                             FIVE,
      5              TWOPI,                          EIGHT,          TEN,
      6              TENP2,          TENP3,          TENP6,          TENP8  /
      o              0.000,        1.0D-08,        1.0D-06,        1.0D-03,
      1            1.0D-02,          0.100,          0.200,          0.300,
      2              0.500,          0.800,          1.000,          2.000,
      3            2.71828182845904523536,           3.000,
      4            3.14159265358979323846,           5.000,
      5            6.28318530717958647693,           8.000,         10.000,
      6            1.0D+02,        1.0D+03,        1.0D+06,        1.0D+08 /
C
C
C
C     *****   End INITDAT.INC    *****
C
C
C
```

```
C-----------------------------------------------------------------------
C  KJAG Name:  N4ONF/DN4ONF (Single/Double precision version)
C
C  Computer:   CRAY/DOUBLE
C
C  Revised:    December 2, 1985
C
C  Purpose:    Main driver for the successive quadratic programming
C              algorithm.
C
C  Usage:      CALL N4ONF (FCNS, MMAX, N, NMAX, X, XS, G, DF, DG, LDDG,
C                    U, XL, XU, DCL, LDDCL, CD, CWK, VMU, DEL,
C                    DLA, DCLF, BDEL, ETA, XOLD, DLAOLD, V, W,
C                    VMUOLD, DPHI, RPEN, SCG, FBEST, DFBEST,
C                    GBEST, DGBEST, WA, LWA, MNN2,
C                    MO1, NFUNC, NGRAD, ITER, NQL, ILINE,
C                    IFLISE, NOPT, IW, LIW, PHI, DFDEL, DBD,
C                    ALPHAM, ALPHAO, SCF, PRD, ACTIVE, L7)
C
C  Arguments:
C     FCNS   - User-supplied SUBROUTINE to evaluate the functions at
C              a given point.  The usage is
C              CALL FCNS (M, ME, N, X, ACTIVE, F, G), where
C              M      - Total number of constraints.  (Input)
C              ME     - Number of equality constraints.  (Input)
C              N      - Number of variables.  (Input)
C              X      - The point at which the function is evaluated.
C                       (Input)
C                       X should not be changed by FCNS.
C              ACTIVE - Logical vector of length MMAX indicating the
C                       active constraints.  (Input)
C              F      - The computed function value at the point X.
C                       (Output)
C              G      - Vector of length MMAX containing the values of
C                       constraints at point X.  (Output)
C              FCNS must be declared EXTERNAL in the calling program.
C     MMAX   - Order of the array DG.  (Input)
C              MMAX must be at least MAX(1,M).
C     N      - Number of variables.  (Input)
C     NMAX   - Order of DCL where NMAX must be at least MAX(2,N+1).
C              (Input)
C     X      - Vector of length N containing the initial guesses to the
C              solution on input and the solution on output.
C              (Input/Output)
C     XS     - Vector of length N containing the diagonal scaling
C              matrix.  (Input)
C     G      - Vector of length MMAX containing constraint values.
C              (Output)
C     DF     - Vector of length N+1 containing the gradient of the
C              of the objective function.  (Output)
C     DG     - Array of dimension MMAX by MMAX containing the gradient
C              of the constraints.  (Output)
C     LDDG   - Leading dimension of DG exactly as specified in the
C              dimension statement in the calling program.  (Input)
C     U      - Vector of length MNN2 containing the multipliers of the
C              nonlinear constraints and the bounds.  (Output)
C     XL     - Vector of length N containing the lower bounds for the
C              variables.  (Input)
C     XU     - Vector of length N containing the upper bounds for the
C              variables.  (Input)
C     DCL    - Array of dimension NMAX by NMAX containing an the final
C              approximation to the Hessian.  (Output)
C     LDDCL  - Leading dimension of DCL exactly as specified in the
C              dimension statement in the calling program.  (Input)
C     CD     - Vector of length NMAX containing the diagonal elements of
C              the Hessian.  (Output)
C     CWK    - Work vector of length M used in gradient evaluation.
C     VMU    - Work vector of length M + 2*N.
C     DEL    - Work vector of length N + 1.
```

```
C     DLA    - Work vector of length N.
C     DCLF   - Work vector of length N + 1.
C     BDEL   - Work vector of length N.
C     ETA    - Work vector of length N.
C     XOLD   - Work vector of length N.
C     DLAOLD - Work vector of length N.
C     V      - Work vector of length N + 1.
C     W      - Work vector of length N + 1.
C     VMUOLD - Work vector of length M + 2*N.
C     DPHI   - Work vector of length M + 3*N.
C     RPEN   - Work vector of length M + 2*N.
C     SCG    - Work vector of length MMAX.
C     FBEST  - Work scalar.
C     DFBEST - Work vector of length NMAX.
C     GBEST  - Work vector of length MMAX.
C     DGBEST - Work array of dimension MO1 by N.
C     WA     - Work vector of length LWA.
C     LWA    - Length of WA where LWA = N*(2*N+13) + M + MMAX + 12.
C              (Input)
C     N1     - Scalar containing the value N + 1.  (Input)
C     MNN    - Scalar containing the value M + 2*N.  (Input)
C     MNN2   - Scalar containing the value M + 2*N + 2.  (Input)
C     NMNN   - Scalar containing the value M + 3*N.  (Input)
C     NO1    - Scalar containing the value 1 when LLISE is true or N
C              when LLISE is false.  (Input)
C     MO1    - Scalar containing the value 1 when LLISE is true or MMAX
C              when LLISE is false.  (Input)
C     NFUNC  - Number of function evaluations.  (Output)
C     NGRAD  - Number of gradient evaluations.  (Output)
C     ITER   - Number of iterations.  (Output)
C     NQL    - Number of QL algorithm evaluations.  (Output)
C     ILINE  - Number of line search evaluations.  (Output)
C     IFLISE - Error parameter for line search algorithm.  (Output)
C     NOPT   - Number of optimality iterations.  (Output)
C     IW     - Work vector of length LIW.
C     LIW    - Length of IW where LIW = 12.  (Input)
C     PHI    - Scalar variable.
C     DFDEL  - Scalar variable.
C     DBD    - Scalar variable.
C     ALPHAM - Scalar variable.
C     ALPHAO - Scalar variable.
C     SCF    - Scalar variable.
C     PRD    - Scalar variable.
C     ACTIVE - Logical vector of length LACTIV indicating which
C              constraints are active.  (Output)
C     LACTIV - Length of ACTIVE where LACTIV must be at least 200.
C              (Input)
C     L7     - Logical vector of length 7.
C
C  Remark:
C     The NLPQL algorithm was designed by K. Schittkowski.
C
C  Topic:      MATH Optimization
C
C-----------------------------------------------------------------------
C
      SUBROUTINE DN4ONF (FCNS, MMAX, N, NMAX, X, XS, G, DF, DG, LDDG,
     &                   U, XL, XU, DCL, LDDCL, CD, CWK, VMU, DEL,
     &                   DLA, DCLF, BDEL, ETA, XOLD, DLAOLD, V, W,
     &                   VMUOLD, DPHI, RPEN, SCG, FBEST, DFBEST,
     &                   GBEST, DGBEST, WA, LWA, MNN2, MO1, NFUNC,
     &                   NGRAD, ITER, NQL, ILINE, IFLISE, NOPT, IW,
     &                   LIW, PHI, DFDEL, DBD, ALPHAM, ALPHAO, SCF,
     &                   PRD, ACTIVE, L7)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    MMAX, N, NMAX, LDDG, LDDCL, LWA, MNN2, MO1, NFUNC,
     &           NGRAD, ITER, NQL, ILINE, IFLISE, NOPT, LIW, IW(*)
      DOUBLE PRECISION FBEST, PHI, DFDEL, DBD, ALPHAM, ALPHAO, SCF,
     &           PRD, X(N), XS(*), G(MMAX), DF(*), DG(LDDG,*),
```

```
     &          U(MNN2), XL(*), XU(*), DCL(LDDCL,*), CD(*), CWK(*),
     &          VMU(1), DEL(*), DLA(*), DCLF(*), BDEL(*), ETA(*),
     &          XOLD(*), DLAOLD(*), V(1), W(1), VMUOLD(*), DPHI(*),
     &          RPEN(1), SCG(*), DFBEST(*), GBEST(*), DGBEST(MO1,*),
     &          WA(*)
      LOGICAL   ACTIVE(*), L7(7)
      EXTERNAL  FCNS
C                                SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER   I, IFAIL1, ILWLS, IMERIT, IOUT, IPR, IRPMAX, J,
     &          LIWQL, LWAQL, ME1, MMAX2, MN, MN1, MNN1, N2, NACT
      DOUBLE PRECISION DBD1, DBDI, DCL11, DELNM, DLAN, EDEL, EDELI,
     &          EPS0, FACT, FF, OF, ON, PHIOLD, RPMAX, SDCL11, SQACC,
     &          SQD, SRES, SUM, THETA, THETA1, TW, UAD, UF, XNM, ZE
C                                SPECIFICATIONS FOR COMMON /DN10NF/
      COMMON    /DN10NF/ F, ACC, SCBOU, DBDFAC, ZEFAC, RPENO, RPENS,
     &          RPENU, ZEFACU, DELTA, BETA, AMUE, ALM, M, ME, MAXFUN,
     &          MAXIT, IPRINT, MODE, IFAIL, LLISE, LQL, LMERIT
      INTEGER   M, ME, MAXFUN, MAXIT, IPRINT, MODE, IFAIL
      DOUBLE PRECISION F, ACC, SCBOU, DBDFAC, ZEFAC, RPENO, RPENS,
     &          RPENU, ZEFACU, DELTA, BETA, AMUE, ALM
      LOGICAL   LLISE, LQL, LMERIT
C                                SPECIFICATIONS FOR COMMON /DN11NF/
      COMMON    /DN11NF/ N1, LACT, NO1, MNN, NMNN
      INTEGER   N1, LACT, NO1, MNN, NMNN
C                                SPECIFICATIONS FOR INTRINSICS
C     INTRINSIC DABS,DMAX1,DMIN1,DBLE,DSQRT
C                                SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL  E1USR, DAXPY, DCOPY, DSCAL, DSET, DVCAL, UMACH,
     &          DCSFRG, DN5ONF, DN5ONG, DN6ONG, DN7ONG, DN8ONG
C                                SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL  DMACH, IDAMAX, IDMAX, DDOT, DA1OT
      INTEGER   IDAMAX, IDMAX
      DOUBLE PRECISION DMACH, DDOT, DA1OT
C                                CONSTANT DATA
      ZE = 0.0D0
      ON = 1.0D0
      TW = 2.0D0
      EPS0 = 100.0D0*DMACH(4)
      UF = EPS0*EPS0
      OF = ON/UF
      CALL UMACH (2, IOUT)
C                                INITIAL DEFINITIONS
      MN = M + N
      ME1 = ME + 1
      N2 = N + N
      LWAQL = LWA - MMAX - 40
      LIWQL = LIW - 10
      ILWLS = 2*MMAX + 1
      IMERIT = 0
      IF (.NOT.LMERIT) IMERIT = 4
      L7(6) = .FALSE.
      L7(4) = .FALSE.
      L7(5) = .FALSE.
      SQACC = DSQRT(ACC)
      IF (MODE.EQ.2 .OR. MODE.EQ.7 .OR. MODE.EQ.3 .OR. MODE.EQ.8) THEN
         L7(6) = .TRUE.
         IF (IFAIL .EQ. -1) GO TO 610
         IF (IFAIL .EQ. -2) GO TO 650
      END IF
      ILINE = 0
      ALPHAO = ZE
      NFUNC = 0
      NGRAD = 0
      ITER = 0
      NQL = 0
      NOPT = 0
      IF (M .NE. 0) THEN
         MMAX2 = MMAX + MMAX
         DO 10  J=1, MMAX2
```

```
            ACTIVE(J) = .TRUE.
   10    CONTINUE
      END IF
      IF (.NOT.L7(6)) THEN
         CALL E1USR ('ON')
         CALL FCNS (M, ME, N, X, ACTIVE(MMAX+1), F, G)
         CALL E1USR ('OFF')
         CALL DN5ONF (FCNS, M, ME, MMAX, N, X, XS, ACTIVE, F, G, DF,
     &                DG, CWK)
      END IF
      L7(1) = .FALSE.
      L7(2) = .FALSE.
      IF (DABS(F) .GE. SCBOU) THEN
         L7(1) = .TRUE.
         IF (SCBOU .GT. ZE) SCF = 1.0D0/DSQRT(DABS(F))
         F = SCF*F
         CALL DSCAL (N, SCF, DF, 1)
      END IF
      IF (M .NE. 0) THEN
         DO 20  J=1, M
            IF (DABS(G(J)) .GE. SCBOU) L7(2) = .TRUE.
   20    CONTINUE
      END IF
C
      IF (L7(2)) THEN
         DO 30  J=1, M
            IF (SCBOU .GT. ZE) SCG(J) = 1.0D0/DMAX1(1.0D0,
     &          DSQRT(DABS(G(J))))
            G(J) = SCG(J)*G(J)
            CALL DSCAL (N, SCG(J), DG(J,1), LDDG)
   30    CONTINUE
      END IF
C
      IF (IPRINT .GE. 1) THEN
         IF (L7(1) .AND. .NOT.L7(2)) WRITE (IOUT,99963)
99963    FORMAT (/, 5X, 'OBJECTIVE FUNCTION WILL BE SCALED')
         IF (L7(1) .AND. L7(2)) WRITE (IOUT,99964)
99964    FORMAT (/, 5X, 'OBJECTIVE AND CONSTRAINT FUNCTIONS WILL BE ',
     &          'SCALED')
         IF (.NOT.L7(1) .AND. L7(2)) WRITE (IOUT,99965)
99965    FORMAT (/, 5X, 'CONSTRAINT FUNCTIONS WILL BE SCALED')
      END IF
C
      NFUNC = NFUNC + 1
      NGRAD = NGRAD + 1
      DCLF(N1) = 0.0D0
      CALL DCOPY (N, DF, 1, DEL, 1)
      CALL DSCAL (N, -1.0D0, DEL, 1)
      CALL DSET (N, 0.0D0, DCL(N1,1), LDDCL)
      CALL DSET (N, 0.0D0, DCL(1,N1), 1)
      DCL(N1,N1) = ZEFAC
      IF (MODE.EQ.1 .OR. MODE.EQ.6 .OR. MODE.EQ.3 .OR. MODE.EQ.8) THEN
         IF (LQL) GO TO 50
         GO TO 750
      END IF
C
      CALL DSET (N, 1.0D0, CD, 1)
      DO 40  I=1, N
         CALL DSET (N, 0.0D0, DCL(1,I), 1)
   40 CONTINUE
      CALL DSET (N, 1.0D0, DCL(1,1), LDDCL+1)
   50 CALL DSET (MNN, RPENS, RPEN, 1)
      CALL DSET (MNN, 0.0D0, VMU, 1)
      IF (MODE.EQ.1 .OR. MODE.EQ.6 .OR. MODE.EQ.3 .OR. MODE.EQ.8)
     &    CALL DCOPY (MNN, U, 1, VMU, 1)
      CALL DN5ONG (IMERIT+3, M, ME, N, MNN, NMNN, ACC, RPEN, F, DF, G,
     &             DG, LDDG, VMU, U, X, XL, XU, PHI, DPHI, ACTIVE, WA,
     &             4)
C                                START MAIN LOOP, PRINT INTERMEDIATE
```

```
C                                        ITERATES
   60 CONTINUE
      L7(3) = .FALSE.
      IF (IPRINT .LT. 3) GO TO 90
      IF (L7(1)) F = F/SCF
      WRITE (IOUT,99966) ITER, F, (X(I),I=1,N)
99966 FORMAT (//5X, 'ITERATION', I3, //8X, 'FUNCTION VALUE:  F(X) =',
     &        D16.8, /8X, 'VARIABLE:  X =', /, (9X,4D16.8))
      IF (L7(1)) F = F*SCF
      IF (M.NE.0 .AND. (L7(1).OR.L7(2))) THEN
         IF (L7(1)) CALL DSCAL (M, 1.0D0/SCF, VMU, 1)
         IF (L7(2)) THEN
            DO 70  J=1, M
               VMU(J) = VMU(J)*SCG(J)
               G(J) = G(J)/SCG(J)
   70       CONTINUE
         END IF
      END IF
      WRITE (IOUT,99967) (VMU(J),J=1,MNN)
99967 FORMAT (8X, 'MULTIPLIERS:  U =', /, (9X,4D16.8))
      IF (M .NE. 0) THEN
         WRITE (IOUT,99968) (G(J),J=1,M)
99968    FORMAT (8X, 'CONSTRAINTS: G(X) =', /, (9X,4D16.8))
         IF (L7(1) .OR. L7(2)) THEN
            IF (L7(1)) CALL DSCAL (M, SCF, VMU, 1)
            IF (L7(2)) THEN
               DO 80  J=1, M
                  VMU(J) = VMU(J)/SCG(J)
                  G(J) = G(J)*SCG(J)
   80          CONTINUE
            END IF
         END IF
      END IF
   90 ITER = ITER + 1
      IF (ITER .LT. MAXIT) GO TO 100
      IFAIL = 1
      IF (IPRINT .EQ. 0) GO TO 350
      WRITE (IOUT,99969)
99969 FORMAT (8X, '**MORE THAN MAXIT ITERATIONS')
      GO TO 350
  100 CONTINUE
C                                        SEARCH DIRECTION
      CALL DCOPY (N, DF, 1, DCLF, 1)
      DO 110  I=1, N
         V(I) = XL(I) - X(I)
  110 CONTINUE
      DO 120  I=1, N
         W(I) = XU(I) - X(I)
  120 CONTINUE
      IPR = 0
      IF (IPRINT.GT.10 .AND. IPRINT.LT.1000) IPR = IPRINT - 10
      IF (MODE .GE. 5) GO TO 130
      IFAIL1 = ITER
      IF (L7(4) .OR. L7(5)) IFAIL1 = 1
      IW(11) = 0
      IF (LQL) IW(11) = 1
      IW(12) = 0
      CALL DN6ONG (M, ME, MMAX, N, NMAX, MNN, DCL, LDDCL, DCLF, DG,
     &             LDDG, G, V, W, DEL, U, IFAIL1, IPR, WA(MMAX+41),
     &             LWAQL, IW(11), LIWQL)
      DEL(N1) = ZE
      NQL = NQL + 1
      L7(4) = .FALSE.
      IF (IFAIL1 .EQ. 0) GO TO 220
  130 CONTINUE
      IF (ITER .EQ. 1) GO TO 140
      FACT = TW*DABS(DBD*DFDEL)/(DSQRT(DBD)*(ON-DEL(N1)))
      IF (LQL) FACT = FACT*FACT
      DCL11 = DMAX1(ZEFAC,FACT)
```

```
      DCL(N1,N1) = DMIN1(ZEFACU,DCL11)
  140 CONTINUE
      CALL DSET (N, 0.0D0, DEL, 1)
      DEL(N1) = 1.0D0
C
      IF (M .NE. 0) THEN
         CALL DCOPY (M, G, 1, DG(1,N1), 1)
         CALL DSCAL (M, -1.0D0, DG(1,N1), 1)
         DO 150  J=1, M
            IF (.NOT.ACTIVE(J)) DG(J,N1) = 0.0D0
  150    CONTINUE
      END IF
C
      V(N1) = 0.0D0
      W(N1) = 1.0D0
      IFAIL1 = -ITER
      IF (.NOT.L7(4) .OR. L7(5)) IFAIL1 = -1
      IW(11) = 0
      IF (LQL) IW(11) = 1
      IW(12) = 1
      CALL DN6ONG (M, ME, MMAX, N1, NMAX, MNN2, DCL, LDDCL, DCLF, DG,
     &             LDDG, G, V, W, DEL, U, IFAIL1, IPR, WA(MMAX+41),
     &             LWAQL, IW(11), LIWQL)
      NQL = NQL + 1
      MN1 = M + N1 + 1
      MNN1 = M + N1 + N
      L7(4) = .TRUE.
      IF (IFAIL1 .EQ. 0) GO TO 170
  160 IFAIL = 10 + IFAIL1
      IF (IPRINT .EQ. 0) GO TO 350
      WRITE (IOUT,99970) IFAIL1
99970 FORMAT (8X, '**ERROR IN QL. IFAIL(QL) =', I3)
      GO TO 350
  170 CONTINUE
      CALL DCOPY (N+1, U(MN1), 1, U(MN1-1), 1)
      IF (IPRINT .LT. 3) GO TO 180
      WRITE (IOUT,99971) DEL(N1)
99971 FORMAT (8X, 'ADDITIONAL VARIABLE TO PREVENT INCONSISTENCY:',
     &        '  DELTA =', D13.4)
      SDCL11 = DCL(N1,N1)
      IF (.NOT.LQL) SDCL11 = DSQRT(SDCL11)
      WRITE (IOUT,99972) SDCL11
99972 FORMAT (8X, 'PENALTY PARAMETER FOR DELTA:  RHO =', D13.4)
  180 CONTINUE
      DCL11 = DCL(N1,N1)
      IF (DEL(N1) .LT. DELTA) GO TO 220
      DCL(N1,N1) = DCL11*RPENO
      IF (LQL) DCL(N1,N1) = DCL(N1,N1)*RPENO
      IF (DCL11 .LT. ZEFACU) GO TO 140
C                                   AUGMENTED LAGRANGIAN TYPE SEARCH
C                                   DIRECTION
  190 L7(5) = .TRUE.
      IF (IPRINT .LT. 3) GO TO 200
      WRITE (IOUT,99973)
99973 FORMAT (8X, '**WARNING: AUGMENTED LAGRANGIAN SEARCH DIRECTION')
  200 CALL DN5ONG (4, M, ME, N, MNN, NMNN, ACC, RPEN, F, DF, G, DG,
     &             LDDG, VMU, U, X, XL, XU, PHI, DPHI, ACTIVE, WA, 4)
      CALL DCOPY (N, DPHI, 1, WA(41), 1)
      CALL DCOPY (N, DPHI, 1, DCLF, 1)
      IFAIL1 = 1
      IW(11) = 0
      IF (LQL) IW(11) = 1
      IW(12) = 0
      CALL DN6ONG (0, 0, MMAX, N, NMAX, MNN2, DCL, LDDCL, DCLF, DG,
     &             LDDG, G, V, W, DEL, U, IFAIL1, IPR, WA(MMAX+41),
     &             LWAQL, IW(11), LIWQL)
      IF (IFAIL1 .GT. 0) GO TO 160
      IF (M .EQ. 0) GO TO 230
      CALL DCOPY (N2, U, -1, U(M+1), -1)
```

```
C
      DO 210  J=1, M
         U(J) = VMU(J) - DPHI(N+J)
  210 CONTINUE
      GO TO 230
  220 L7(5) = .FALSE.
C                                 PROJECTION OF DEL, MAXIMAL
C                                 STEPLENGTH, AND NORM OF X,DEL
  230 ALPHAM = OF
      XNM = 0.0D0
      DELNM = 0.0D0
      DO 240  I=1, N
         IF (W(I) .LT. DEL(I)) DEL(I) = W(I)
         IF (V(I) .GT. DEL(I)) DEL(I) = V(I)
         UAD = DABS(DEL(I))
C        IF (DEL(I) .GT. UF) ALPHAM = DMIN1(ALPHAM,W(I)/DEL(I))
CMOD-------------------------------------------R.K.Owen,PhD, 12/17/91---
         IF (DEL(I) .GT. UF) THEN                                       RKO
           IF (ALPHAM*DEL(I) .GT. W(I)) THEN                            RKO
             ALPHAM = W(I)/DEL(I)                                       RKO
           ENDIF                                                        RKO
         ENDIF                                                          RKO
C        IF (DEL(I) .LT. -UF) ALPHAM = DMIN1(ALPHAM,V(I)/DEL(I))
CMOD-------------------------------------------R.K.Owen,PhD, 12/17/91---
         IF (DEL(I) .LT. -UF) THEN                                      RKO
           IF (ALPHAM*DEL(I) .GT. V(I)) THEN                            RKO
             ALPHAM = V(I)/DEL(I)                                       RKO
           ENDIF                                                        RKO
         ENDIF                                                          RKO
         XNM = DMAX1(DABS(X(I)),XNM)
  240 DELNM = DMAX1(UAD,DELNM)
      ALPHAM = DMAX1(ON,ALPHAM)
      ALPHAM = DMIN1(ALPHAM,ALM)
C                                 GRADIENT OF LAGRANGIAN
  250 DO 260  I=1, N
         UAD = DF(I)
         IF (L7(5)) UAD = DPHI(I)
         DLA(I) = UAD - U(M+I) - U(MN+I)
  260 CONTINUE
C
      IF (M.NE.0 .AND. .NOT.L7(5)) THEN
         DO 270  J=1, M
            IF (U(J) .NE. ZE) CALL DAXPY (N, -U(J), DG(J,1), LDDG,
     &            DLA, 1)
  270    CONTINUE
      END IF
      IF (L7(3)) GO TO 680
C                                 STORE SOME DATA
      CALL DCOPY (N, DLA, 1, DLAOLD, 1)
      CALL DCOPY (N, X, 1, XOLD, 1)
      DFDEL = DDOT(N,DF,1,DEL,1)
      IRPMAX = IDAMAX(N,DLA,1)
      DLAN = DABS(DLA(IRPMAX))
      CALL DCOPY (MNN, VMU, 1, VMUOLD, 1)
C                                 DETERMINE B*D AND D(T)*B*D
      IF (.NOT.LQL) THEN
         DO 280  I=1, N
            ETA(I) = DDOT(N-I+1,DCL(I,I),LDDCL,DEL(I),1)
  280    CONTINUE
         DBD = DDOT(N,ETA,1,ETA,1)
         DO 290  I=1, N
            BDEL(I) = DDOT(I,DCL(1,I),1,ETA,1)
  290    CONTINUE
      ELSE
         DO 300  I=1, N
            BDEL(I) = DDOT(N,DCL(I,1),LDDCL,DEL,1)
  300    CONTINUE
         DBD = DDOT(N,BDEL,1,DEL,1)
      END IF
```

```
C                                     TEST FOR OPTIMALITY AND FINAL OUTPUT
      SRES = ZE
      SUM = DABS(DFDEL)
      IF (L7(1)) SUM = SUM/SCF
      NACT = 0
      IF (M .NE. 0) THEN
         DO 310  J=1, M
            IF (ACTIVE(J)) NACT = NACT + 1
            UAD = DABS(G(J))
            IF (L7(2)) UAD = UAD/SCG(J)
            IF (J.LE.ME .OR. G(J).LT.ZE) SRES = SRES + UAD
  310    CONTINUE
         SUM = SUM + DA1OT(M,U,1,G,1)
         IF (IPRINT .EQ. 3) THEN
            WRITE (IOUT,99974) SRES
99974       FORMAT (8X, 'SUM OF CONSTRAINT VIOLATIONS: ', 19X,
     &              'SCV =', D13.4)
            WRITE (IOUT,99975) NACT
99975       FORMAT (8X, 'NUMBER OF ACTIVE CONSTRAINTS: ', 19X,
     &              'NAC =', I4)
         END IF
      END IF
C
      DO 320  I=1, N
         SUM = SUM + DABS(U(M+I)*V(I)) + DABS(U(MN+I)*W(I))
  320 CONTINUE
      IF (IPRINT .EQ. 2) THEN
         FF = F
         IF (L7(1)) FF = F/SCF
         WRITE (IOUT,99976) ITER, FF, SRES, NACT, ILINE, ALPHAO,
     &                      DEL(N1), DLAN, SUM
99976    FORMAT (1X, I3, D16.8, D10.2, I4, I3, 4D10.2)
      END IF
C
      IF (IPRINT .EQ. 3) THEN
         WRITE (IOUT,99977) SUM
99977    FORMAT (8X, 'KUHN-TUCKER OPTIMALITY CONDITION:        ', 9X,
     &           'KTO =', D13.4)
         WRITE (IOUT,99978) DLAN
99978    FORMAT (8X, 'NORM OF LAGRANGIAN GRADIENT:             ', 9X,
     &           'NLG =', D13.4)
      END IF
      IF (DBD .GE. UF) GO TO 330
      IF (SRES .LT. SQACC) GO TO 340
      IF (DBD .GT. ZE) GO TO 390
      IF (.NOT.L7(5)) GO TO 190
      IFAIL = 7
      IF (IPRINT .EQ. 0) GO TO 350
      WRITE (IOUT,99979)
99979 FORMAT (8X, '**UNDERFLOW IN D(T)*B*D AND INFEASIBLE ITERATE X')
      GO TO 350
  330 CONTINUE
      IF (SUM.GE.ACC .OR. SRES.GT.SQACC) GO TO 390
      IF (DLAN.LE.DSQRT(SQACC) .OR. DBD.LE.ACC) GO TO 340
      NOPT = NOPT + 1
      IF (NOPT .LT. 3) GO TO 390
  340 IFAIL = 0
  350 CONTINUE
      IF (L7(1)) F = F/SCF
      IF (M.EQ.0 .OR. (.NOT.L7(1).AND..NOT.L7(2))) GO TO 370
      IF (L7(1)) CALL DSCAL (N, 1.0D0/SCF, U, 1)
      IF (L7(2)) THEN
         DO 360  J=1, M
            U(J) = U(J)*SCG(J)
            G(J) = G(J)/SCG(J)
  360    CONTINUE
      END IF
  370 CONTINUE
C
```

```
      IF (IPRINT .EQ. 0) GO TO 9000
      WRITE (IOUT,99980)
99980 FORMAT (//, 5X, '* FINAL CONVERGENCE ANALYSIS', /)
      WRITE (IOUT,99981) F
99981 FORMAT (8X, 'OBJECTIVE FUNCTION VALUE:  F(X) =', D16.8)
      WRITE (IOUT,99982) (X(I),I=1,N)
99982 FORMAT (8X, 'APPROXIMATION OF SOLUTION:  X =', /, (9X,4D16.8))
      WRITE (IOUT,99983) (U(J),J=1,MNN)
99983 FORMAT (8X, 'APPROXIMATION OF MULTIPLIERS:  U =', /, (9X,4D16.8))
      IF (M .EQ. 0) GO TO 380
      WRITE (IOUT,99984) (G(J),J=1,M)
99984 FORMAT (8X, 'CONSTRAINT VALUES:  G(X) =', /, (9X,4D16.8))
  380 WRITE (IOUT,99985) (V(I),I=1,N)
99985 FORMAT (8X, 'DISTANCE FROM LOWER BOUND:  XL-X =', /, (9X,4D16.8))
      WRITE (IOUT,99986) (W(I),I=1,N)
99986 FORMAT (8X, 'DISTANCE FROM UPPER BOUND:  XU-X =', /, (9X,4D16.8))
      IF (.NOT.LLISE) WRITE (IOUT,99987) ITER
99987 FORMAT (8X, 'NUMBER OF ITERATIONS:  ITER =', I4)
      WRITE (IOUT,99988) NFUNC
99988 FORMAT (8X, 'NUMBER OF FUNC-CALLS:  NFUNC =', I4)
      WRITE (IOUT,99989) NGRAD
99989 FORMAT (8X, 'NUMBER OF GRAD-CALLS:  NGRAD =', I4)
      WRITE (IOUT,99990) NQL
99990 FORMAT (8X, 'NUMBER OF QL-CALLS:    NQL   =', I4, ///)
      GO TO 9000
  390 CONTINUE
C                                   CORRECT PENALTY PARAMETER
      IF (L7(5)) GO TO 400
      WA(1) = DBD
      WA(2) = DEL(N1)
      WA(3) = RPENU
      WA(4) = DBLE(ITER)
      CALL DN5ONG (IMERIT+2, M, ME, N, MNN, NMNN, ACC, RPEN, F, DF, G,
     &             DG, LDDG, VMU, U, X, XL, XU, PHI, DPHI, ACTIVE, WA,
     &             4)
      GO TO 430
  400 SUM = ZE
      DO 410  I=1, N
  410 SUM = SUM + DPHI(I)*DEL(I) + DABS(U(M+I)*V(I)) +
     &      DABS(U(MN+I)*W(I))
      IF (SUM .GT. DSQRT(SQACC)) GO TO 430
      DO 420  J=1, MNN
  420 RPEN(J) = DMIN1(ZEFACU,RPEN(J)*RPENO)
      CALL DN5ONG (IMERIT+4, M, ME, N, MNN, NMNN, ACC, RPEN, F, DF, G,
     &             DG, LDDG, VMU, U, X, XL, XU, PHI, DPHI, ACTIVE, WA,
     &             4)
  430 IF (IPRINT .LT. 3) GO TO 440
      WRITE (IOUT,99991) DBD
99991 FORMAT (8X, 'PRODUCT OF SEARCH DIRECTION WITH BFGS-MATRIX: ',
     &       '  DBD =', D13.4)
      WRITE (IOUT,99992) (RPEN(J),J=1,MNN)
99992 FORMAT (8X, 'PENALTY PARAMETER:  R =', /, (9X,4D16.8))
  440 CONTINUE
C                                   EVALUATION OF MERIT FUNCTION
C
  450 CALL DN5ONG (IMERIT+3, M, ME, N, MNN, NMNN, ACC, RPEN, F, DF, G,
     &             DG, LDDG, VMU, U, X, XL, XU, PHI, DPHI, ACTIVE, WA,
     &             4)
      IF (.NOT.L7(5)) CALL DN5ONG (IMERIT+4, M, ME, N, MNN, NMNN, ACC,
     &   RPEN, F, DF, G, DG, LDDG, VMU, U, X, XL, XU, PHI, DPHI,
     &   ACTIVE, WA, 4)
      PRD = DDOT(N,DPHI,1,DEL,1)
      DO 460  J=1, MNN
  460 PRD = PRD + DPHI(J+N)*(U(J)-VMU(J))
      PHIOLD = PHI
      IF (PRD .LT. ZE) GO TO 480
      CALL DSCAL (MNN, RPENO, RPEN, 1)
      IRPMAX = IDMAX(MNN,RPEN,1)
      RPMAX = DMAX1(RPEN(IRPMAX),0.0D0)
```

```
      IF (RPMAX .LT. RPENU) GO TO 450
      IF (L7(5)) GO TO 470
      IF (.NOT.L7(4) .OR. DBD.LT.ACC) GO TO 190
      DCL11 = DCL(N1,N1)
      IF (DCL11 .GE. ZEFACU) GO TO 190
      DCL11 = DCL11*RPENO
      IF (LQL) DCL11 = DCL11*RPENO
      DCL(N1,N1) = DCL11
      GO TO 140
  470 CONTINUE
      IFAIL = 2
      IF (IPRINT .EQ. 0) GO TO 350
      WRITE (IOUT,99993) PRD
99993 FORMAT (8X, '**SEARCH DIRECTION NOT PROFITABLE:  DPHI*P =',
     &       D13.4)
      GO TO 350
  480 CONTINUE
      IF (IPRINT .LT. 3) GO TO 490
      WRITE (IOUT,99994) PRD
99994 FORMAT (8X, 'PRODUCT LAGRANGIAN GRADIENT WITH ', 'SEARCH ',
     &       'DIRECTION:  DLP =', D13.4)
  490 CONTINUE
C                                     LINE SEARCH
      WA(6) = XNM
      WA(7) = DELNM
      L7(7) = .FALSE.
      IFLISE = 0
  500 IPR = 0
      IF (IPRINT .GE. 1000) IPR = IPRINT - 1000
      CALL DN8ONG (ALPHAO, ALPHAM, PHI, PRD, AMUE, BETA, ILINE,
     &             MAXFUN, IFLISE, IPR, WA(6), 35, IW, 10,
     &             ACTIVE(ILWLS), 5)
      IF (IFLISE .GT. -2) GO TO 520
      L7(7) = .TRUE.
      FBEST = F
      CALL DCOPY (M, G, 1, GBEST, 1)
      IF (LLISE) GO TO 500
      CALL DCOPY (N, DF, 1, DFBEST, 1)
      DO 510  I=1, N
         CALL DCOPY (M, DG(1,I), 1, DGBEST(1,I), 1)
  510 CONTINUE
      GO TO 500
  520 CONTINUE
      DO 530  I=1, N
  530 X(I) = XOLD(I) + ALPHAO*DEL(I)
      DO 540  J=1, MNN
  540 VMU(J) = VMUOLD(J) + ALPHAO*(U(J)-VMUOLD(J))
      IF (IFLISE .EQ. 0) GO TO 570
      IF (IFLISE .EQ. 1) GO TO 560
      IF (IFLISE .GT. 1) GO TO 550
      GO TO 600
  550 IFAIL = 1000 + IFLISE
      IF (IPRINT .EQ. 0) GO TO 350
      WRITE (IOUT,99995) IFLISE
99995 FORMAT (8X, '**ERROR IN LINE SEARCH. IFLISE =', I4)
      GO TO 350
  560 IFAIL = 4
      IF (IPRINT .EQ. 0) GO TO 350
      WRITE (IOUT,99996)
99996 FORMAT (8X, '**MORE THAN MAXFUN FUNC-CALLS IN LINE SEARCH')
      GO TO 350
  570 L7(3) = .TRUE.
      IF (IPRINT .LT. 3) GO TO 580
      IF (ILINE .EQ. 1) WRITE (IOUT,99997)
99997 FORMAT (8X, 'LINE SEARCH SUCCESSFUL AFTER ONE STEP:  ALPHA = 1.')
      IF (ILINE .GT. 1) WRITE (IOUT,99998) ILINE, ALPHAO
99998 FORMAT (8X, 'LINE SEARCH SUCCESSFUL AFTER', I3, ' STEPS:',
     &        '  ALPHA =', D13.4)
  580 CONTINUE
```

```
      IF (.NOT.L7(7) .AND. LLISE) GO TO 630
      IF (.NOT.L7(7) .AND. .NOT.LLISE) GO TO 250
      F = FBEST
      CALL DCOPY (M, GBEST, 1, G, 1)
      IF (LLISE) GO TO 630
      CALL DCOPY (N, DFBEST, 1, DF, 1)
      DO 590  I=1, N
         CALL DCOPY (M, DGBEST(1,I), 1, DG(1,I), 1)
  590 CONTINUE
      CALL DN5ONG (IMERIT+1, M, ME, N, MNN, NMNN, ACC, RPEN, F, DF, G,
     &             DG, LDDG, VMU, U, X, XL, XU, PHI, DPHI, ACTIVE, WA,
     &             4)
      GO TO 250
  600 CONTINUE
C                                  NEW FUNCTION AND GRADIENT VALUES
      IF (L7(6)) THEN
         IFAIL = -1
         GO TO 9000
      END IF
      CALL E1USR ('ON')
      CALL FCNS (M, ME, N, X, ACTIVE(MMAX+1), F, G)
      CALL E1USR ('OFF')
  610 CONTINUE
      IF (L7(1)) F = F*SCF
C
      IF (M.NE.0 .AND. L7(2)) THEN
         DO 620  J=1, M
            G(J) = SCG(J)*G(J)
  620    CONTINUE
      END IF
      NFUNC = NFUNC + 1
      CALL DN5ONG (IMERIT+3, M, ME, N, MNN, NMNN, ACC, RPEN, F, DF, G,
     &             DG, LDDG, VMU, U, X, XL, XU, PHI, DPHI, ACTIVE, WA,
     &             4)
      IF (LLISE .AND. .NOT.L7(3)) GO TO 500
  630 CONTINUE
      CALL DN5ONG (IMERIT+1, M, ME, N, MNN, NMNN, ACC, RPEN, F, DF, G,
     &             DG, LDDG, VMU, U, X, XL, XU, PHI, DPHI, ACTIVE, WA,
     &             4)
      IF (L7(1)) F = F/SCF
      IF (M.NE.0 .AND. L7(2)) THEN
         DO 640  J=1, M
            G(J) = G(J)/SCG(J)
  640    CONTINUE
      END IF
C
      IF (L7(6)) THEN
         IFAIL = -2
         GO TO 9000
      END IF
C
      CALL DN5ONF (FCNS, M, ME, MMAX, N, X, XS, ACTIVE, F, G, DF, DG,
     &             CWK)
  650 CONTINUE
      NGRAD = NGRAD + 1
      IF (L7(1)) THEN
         F = F*SCF
         CALL DSCAL (N, SCF, DF, 1)
      END IF
C
      IF (M.NE.0 .AND. L7(2)) THEN
         DO 660  J=1, M
            G(J) = G(J)*SCG(J)
            IF (ACTIVE(J)) CALL DSCAL (N, SCG(J), DG(J,1), LDDG)
  660    CONTINUE
      END IF
C
      IF (L7(3)) GO TO 250
      CALL DN5ONG (IMERIT+4, M, ME, N, MNN, NMNN, ACC, RPEN, F, DF, G,
```

```
     &              DG, LDDG, VMU, U, X, XL, XU, PHI, DPHI, ACTIVE, WA,
     &              4)
      PRD = DDOT(N,DPHI,1,DEL,1)
      DO 670  J=1, MNN
  670 PRD = PRD + DPHI(N+J)*(U(J)-VMUOLD(J))
      GO TO 500
C                                  UPDATE HESSIAN OF LAGRANGIAN
  680 DBD = DBD*ALPHAO*ALPHAO
      CALL DSCAL (N, ALPHAO, BDEL, 1)
      DO 690  I=1, N
         ETA(I) = DLA(I) - DLAOLD(I)
  690 CONTINUE
      EDEL = ALPHAO*DDOT(N,DEL,1,ETA,1)
      DBD1 = DBDFAC*DBD
      IF (EDEL .GE. DBD1) GO TO 720
      THETA = (DBD-DBD1)/(DBD-EDEL)
      THETA1 = ON - THETA
      DO 700  I=1, N
  700 ETA(I) = THETA*ETA(I) + THETA1*BDEL(I)
  710 EDEL = DBD1
  720 CONTINUE
      DBDI = DSQRT(ON/DBD)
      EDELI = DSQRT(ON/EDEL)
C                                  UPDATE FACTORIZATION
      CALL DSCAL (N, DBDI, BDEL, 1)
      CALL DSCAL (N, EDELI, ETA, 1)
      IF (LQL) THEN
         DO 740  I=1, N
            DO 730  J=1, I
               DCL(J,I) = DCL(J,I) + ETA(I)*ETA(J) - BDEL(I)*BDEL(J)
  730       CONTINUE
  740    CONTINUE
         CALL DCSFRG (N, DCL, LDDCL)
         GO TO 60
      END IF
      CALL DN7ONG (N, DCL, LDDCL, CD, ETA, BDEL)
C                                  CORRECT DATA FOR QL-SOLUTION
  750 DO 770  I=1, N
         SQD = DSQRT(CD(I))
         IF (SQD .GT. UF) GO TO 760
         IFAIL = 3
         IF (IPRINT .EQ. 0) GO TO 350
         WRITE (IOUT,99999)
99999    FORMAT (8X, '**UNDERFLOW IN BFGS-UPDATE')
         GO TO 350
  760    CONTINUE
         IF (I .LT. N) CALL DVCAL (N-I, SQD, DCL(I+1,I), 1, DCL(I,I+1)
     &        , LDDCL)
         DCL(I,I) = SQD
  770 CONTINUE
      IF (ITER .EQ. 0) GO TO 50
C                                  PERFORM NEXT ITERATION
      GO TO 60
 9000 RETURN
      END
```

```
C-----------------------------------------------------------------------
C  KJAG Name:  N9ONG/DN9ONG (Single/Double precision version)
C
C  Computer:   CRAY/DOUBLE
C
C  Revised:    September 24, 1987
C
C  Purpose:    Compute minimum of the unconstrained problem.
C
C  Usage:      CALL N9ONG (N, M, MEQ, MMAX, MN, MNN, NMAX, LQL, A, B,
C                          GRAD, G, XL, XU, X, NACT, IACT, INFO, DIAG,
C                          W, LW)
C
C  Arguments:
C     N      - Number of variables.  (Input)
C     M      - Number of constraints.  (Input)
C     MEQ    - Number of equality constraints.  (Input)
C     MMAX   - Leading dimension of A.  (Input)
C              MMAX must be at least MAX(1,M).
C     MN     - Scalar variable suxh that MN = M + N.  (Input)
C     MNN    - Scalar variable suxh that MNN = M + 2*N.  (Input)
C     NMAX   - Leading dimension of G.  (Input)
C              NMAX must be at least MAX(2,N).
C     LQL    - Logical scalar determining the initial decomposition.
C              (Input)
C              If LQL is true, the initial Cholesky-factorization of G
C              is performed.  If LQL is false, the upper triangle of G
C              contains the Cholesky-factor of a suitable decomposition.
C     A      - Array of dimension MMAX by NMAX containing the constraint
C              normals in the columns.  (Output)
C     LDA    - Leading dimension of A exactly as specified in the
C              dimension statement of the calling program.  (Input)
C     B      - Vector of length MMAX containing the right-hand-sides of
C              the constraints.  (Input)
C     GRAD   - Vector of length N containing the objective function
C              gradient.  (Input)
C     G      - Array of dimension NMAX by N containing symmetric
C              objective function matrix.  (Input)
C     XL     - Vector of length N containing the lower bounds for the
C              variables.  (Input)
C     XU     - Vector of length N containing the upper bounds for the
C              variables.  (Input)
C     X      - Vector of length N containing the current point being
C              evaluated.  (Input)
C     NACT   - Number of active constraints.  (Output)
C     IACT   - Vector of length NACT indicating the final active
C              constraints.  (Output)
C     INFO   - Scalar containing exiting information.  (Output)
C     DIAG   - Scalar containing multiple of the unit matrix that was
C              added to G to achieve positive definiteness.  (Output)
C     W      - Work vector of length LW.
C     LW     - Length of W where LW = NMAX*(2*NMAX+10) + M.
C              (Input)
C
C  Topic:      MATH Optimization
C
C-----------------------------------------------------------------------
C
      SUBROUTINE DN9ONG (N, M, MEQ, MMAX, MN, MNN, NMAX, LQL, A, LDA,
     &                   B, GRAD, G, LDG, XL, XU, X, NACT, IACT, INFO,
     &                   DIAG, W, LW)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N, M, MEQ, MMAX, MN, MNN, NMAX, LDA, LDG, NACT, INFO,
     &           LW, IACT(*)
      DOUBLE PRECISION DIAG, A(LDA,*), B(*), GRAD(*), G(LDG,*), XL(*),
     &           XU(*), X(*), W(*)
      LOGICAL    LQL
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, IA, ID, IFINC, IFLAG, II, IL, IP, IPP, IR, IRA,
```

```
     &           IRB, IS, ITERC, ITREF, IU, IW, IWA, IWD, IWR, IWS,
     &           IWW, IWWN, IWX, IWY, IWZ, IX, IY, IZ, IZA, J, JFINC,
     &           JFLAG, JL, K, K1, KDROP, KFINC, KFLAG, KK, KNEXT,
     &           LFLAG, MFLAG, NFLAG, NM, NU
      DOUBLE PRECISION BIG, CVMAX, DIAGR, FDIFF, FDIFFA, GA, GB,
     &           PARINC, PARNEW, RATIO, RES, SMALL, STEP, SUM, SUMA,
     &           SUMB, SUMC, SUMX, SUMY, TEMP, TEMPA, VFACT, VSMALL,
     &           XMAG, XMAGR
      LOGICAL    LOWER
C                                 SPECIFICATIONS FOR INTRINSICS
C     INTRINSIC  DABS,DMAX1,DMIN1,MAX0,MIN0,DSQRT
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   DCOPY, DSET
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   DMACH, DDOT, DSUM, DA1OT
      DOUBLE PRECISION DMACH, DDOT, DSUM, DA1OT
C                                 INITIAL ADDRESSES
C
C    *****   Start Debug 1   *****
C
C     IBUG = 0
C
C    *****    End Debug 1    *****
C
      VSMALL = DMACH(4)
      SMALL = DMACH(1)
      BIG = DMACH(2)
      IF (SMALL*BIG .LT. 1.0D0) SMALL = 1.0D0/BIG
C
      IWZ = NMAX
      IWR = IWZ + NMAX*NMAX
      IWW = IWR + (NMAX*(NMAX+3))/2
      IWD = IWW + NMAX
      IWX = IWD + NMAX
      IWA = IWX + NMAX
C                                 SET SOME CONSTANTS.
      VFACT = 1.0D0
C                                 SET SOME PARAMETERS. NUMBER LESS
C                                 THAN VSMALL ARE ASSUMED TO BE
C                                 NEGLIGIBLE. THE MULTIPLE OF I THAT
C                                 IS ADDED TO G IS AT MOST DIAGR
C                                 TIMES THE LEAST MULTIPLE OF I THAT
C                                 GIVES POSITIVE DEFINITENESS. X IS
C                                 RE-INITIALISED IF ITS MAGNITUDE IS
C                                 REDUCED BY THE FACTOR XMAGR. A
C                                 CHECK IS MADE FOR AN INCREASE IN F
C                                 EVERY IFINC ITERATIONS, AFTER
C                                 KFINC ITERATIONS ARE COMPLETED.
      DIAGR = 2.0D0
      XMAGR = 1.0D-2
      IFINC = 3
      KFINC = MAX0(10,N)
C                                 FIND THE RECIPROCALS OF THE LENGTHS
C                                 OF THE CONSTRAINT NORMALS. RETURN
C                                 IF A CONSTRAINT IS INFEASIBLE DUE
C                                 TO A 0.0E0 NORMAL.
      NACT = 0
      DO 30  K=1, M
         SUM = DDOT(N,A(K,1),LDA,A(K,1),LDA)
         IF (SUM .GT. 0.0D0) GO TO 10
         IF (B(K) .EQ. 0.0D0) GO TO 20
         INFO = -K
         IF (K .LE. MEQ) GO TO 1020
         IF (B(K)) 20,20,1020
   10    SUM = 1.0D0/DSQRT(SUM)
   20    IA = IWA + K
         W(IA) = SUM
   30 CONTINUE
      CALL DSET (N, 1.0D0, W(IWA+M+1), 1)
```

```
C                                      IF NECESSARY INCREASE THE DIAGONAL
C                                      ELEMENTS OF G.
      IF (.NOT.LQL) GO TO 150
      DIAG = 0.0D0
      DO 50  I=1, N
         ID = IWD + I
         W(ID) = G(I,I)
         DIAG = DMAX1(DIAG,VSMALL-W(ID))
C
C    *****   Start Debug 2   *****
C
C        IF (I .EQ. N) GO TO 50
C
         IF (I .EQ. N  .AND.  N .NE. 1) GO TO 50
         IF (N .NE. 1) II = I + 1
         IF (N .EQ. 1) II = 1
C
C        II = I + 1
C
C    *****    End Debug 2    *****
C
         DO 40  J=II, N
            GA = -DMIN1(W(ID),G(J,J))
            GB = DABS(W(ID)-G(J,J)) + DABS(G(I,J))
            IF (GB .GT. SMALL) GA = GA + G(I,J)*G(I,J)/GB
   40    DIAG = DMAX1(DIAG,GA)
   50 CONTINUE
      IF (DIAG .GT. 0.0D0) GO TO 80
   60 DIAG = DIAGR*DIAG
      DO 70  I=1, N
         ID = IWD + I
         G(I,I) = DIAG + W(ID)
   70 CONTINUE
C                                      FORM THE CHOLESKY FACTORISATION OF
C                                      G. THE TRANSPOSE OF THE FACTOR
C                                      WILL BE PLACED IN THE R-PARTITION
C                                      OF W.
   80 IR = IWR
      DO 110  J=1, N
         IRA = IWR
         IRB = IR + 1
         DO 100  I=1, J
            TEMP = G(I,J)
C
C    *****   Start Debug 3    *****
C
C           IF (I .NE. 1) THEN
C
            IF (I .NE. 1  .OR.  N .EQ. 1) THEN
C
C    *****    End Debug 3    *****
C

               TEMP = TEMP - DDOT(IR-IRB+1,W(IRB),1,W(IRA+1),1)
               IRA = IRA + (IR-IRB+1)
            END IF
   90       IR = IR + 1
            IRA = IRA + 1
            IF (I .LT. J) W(IR) = TEMP/W(IRA)
  100    CONTINUE
C
C    *****   Start Debug 4    *****
C
C6000 FORMAT(2H0 )
C6001 FORMAT(10I8)
C6002 FORMAT(4D20.11)
C6010 FORMAT(2H0 ,2X,4H IWR,4X,4H IRB,5X,3H IR,4X,4H IRA,6X,2H I,
C    1 6X,2H J/8X,5H TEMP,13X,7H VSMALL)
C     WRITE(6,6010)
```

```
C     WRITE(6,6001) IWR, IRB, IR, IRA, I, J
C     WRITE(6,6002) TEMP, VSMALL
C     WRITE(6,6000)
C
C   *****     End Debug 4     *****
C
          IF (TEMP .LT. VSMALL) GO TO 120
          W(IR) = DSQRT(TEMP)
  110 CONTINUE
      GO TO 170
C                                      INCREASE FURTHER THE DIAGONAL
C                                      ELEMENT OF G.
  120 W(J) = 1.0D0
      SUMX = 1.0D0
      K = J
  130 SUM = 0.0D0
      IRA = IR - 1
      DO 140  I=K, J
         SUM = SUM - W(IRA)*W(I)
         IRA = IRA + I
  140 CONTINUE
      IR = IR - K
C
C   *****    Start Debug 5    *****
C
      IF (K .LE. 1) GO TO 7700
C
C   *****     End Debug 5     *****
C
      K = K - 1
C
C   *****    Start Debug 6    *****
C
C7000 FORMAT(2H0 )
C7001 FORMAT(10I8)
C7002 FORMAT(4D20.11)
C7010 FORMAT(2H0 ,1X,5H NMAX,4X,4H IWZ,4X,4H IWR,4X,4H IWW,4X,4H IWD,
C    1 4X,4H IWX,4X,4H IWA,5X,3H IA,5X,3H ID,5X,3H II/6X,2H M,6X,2H N,
C    2 6X,2H I,5X,3H IR,6X,2H J,4X,4H IRB,4X,4H IRA,6X,2H K/9X,4H SUM,
C    3 14X,6H W(IR)/)
C     WRITE(6,7010)
C     WRITE(6,7001) NMAX, IWZ, IWR, IWW, IWD, IWX, IWA, IA, ID, II
C     WRITE(6,7001) M, N, I, IR, J, IRB, IRA, K
C     WRITE(6,7002) SUM, W(IR)
C     WRITE(6,7000)
C
C   *****     End Debug 6     *****
C
      W(K) = SUM/W(IR)
      SUMX = SUMX + W(K)*W(K)
      IF (K .GE. 2) GO TO 130
C
C   *****    Start Debug 7    *****
C
C     GO TO 7701
C7700 IBUG = IBUG + 1
 7700 CONTINUE
C8010 FORMAT(20H    *****    IBUG  =  ,I5)
C     WRITE(6,8010) IBUG
C7701 CONTINUE
C
C   *****     End Debug 7     *****
C
      DIAG = DIAG + VSMALL - TEMP/SUMX
      GO TO 60
C                                      STORE THE CHOLESKY FACTORISATION IN
C                                      THE R-PARTITION OF W.
  150 IR = IWR
      DO 160  I=1, N
```

```
         CALL DCOPY (I, G(1,I), 1, W(IR+I*(I-1)/2+1), 1)
  160 CONTINUE
C                                  SET Z THE INVERSE OF THE MATRIX IN
C                                  R.
  170 NM = N - 1
      DO 190  I=1, N
         IZ = IWZ + I
         CALL DSET (I-1, 0.0D0, W(IZ), N)
         IZ = IZ + N*(I-1)
         IR = IWR + (I+I*I)/2
         W(IZ) = 1.0D0/W(IR)
         IF (I .EQ. N) GO TO 190
         IZA = IZ
         DO 180  J=I, NM
            IR = IR + I
            SUM = DDOT((IZ-IZA)/N+1,W(IZA),N,W(IR),1)
            IR = IR + (IZ-IZA)/N + 1
            IZ = IZ + N
            W(IZ) = -SUM/W(IR)
  180    CONTINUE
  190 CONTINUE
C                                  SET THE INITIAL VALUES OF SOME
C                                  VARIABLES. ITERC COUNTS THE NUMBER
C                                  OF ITERATIONS. ITREF IS SET TO
C                                  1.0E0 WHEN ITERATIVE REFINEMENT IS
C                                  REQUIRED. JFINC INDICATES WHEN TO
C                                  TEST FOR AN INCREASE IN F.
      ITERC = 1
      ITREF = 0
      JFINC = -KFINC
C                                  SET X TO 0.0E0 AND SET THE
C                                  CORRESPONDING RESIDUALS OF THE
C                                  KUHN-TUCKER CONDITIONS.
  200 IFLAG = 1
      IWS = IWW - N
      CALL DSET (N, 0.0D0, X, 1)
      DO 230  I=1, N
         IW = IWW + I
         W(IW) = GRAD(I)
         IF (I .GT. NACT) GO TO 230
         W(I) = 0.0D0
         IS = IWS + I
         K = IACT(I)
         IF (K .LE. M) GO TO 220
         IF (K .GT. MN) GO TO 210
         K1 = K - M
         W(IS) = XL(K1)
         GO TO 230
  210    K1 = K - MN
         W(IS) = -XU(K1)
         GO TO 230
  220    W(IS) = B(K)
  230 CONTINUE
      XMAG = 0.0D0
      VFACT = 1.0D+0
      IF (NACT) 390,390,340
C                                  SET THE RESIDUALS OF THE KUHN-TUCKER
C                                  CONDITIONS FOR GENERAL X.
  240 IFLAG = 2
      IWS = IWW - N
      DO 290  I=1, N
         IW = IWW + I
         W(IW) = GRAD(I)
         IF (LQL) GO TO 270
         ID = IWD + I
         W(ID) = 0.0D0
         DO 250  J=I, N
  250    W(ID) = W(ID) + G(I,J)*X(J)
         DO 260  J=1, I
```

```
         ID = IWD + J
 260    W(IW) = W(IW) + G(J,I)*W(ID)
        GO TO 290
 270    DO 280  J=1, N
 280    W(IW) = W(IW) + G(I,J)*X(J)
 290 CONTINUE
     IF (NACT .EQ. 0) GO TO 390
     DO 330  K=1, NACT
        KK = IACT(K)
        IS = IWS + K
        IF (KK .GT. M) GO TO 310
        W(IS) = B(KK)
        DO 300  I=1, N
           IW = IWW + I
           W(IW) = W(IW) - W(K)*A(KK,I)
 300    W(IS) = W(IS) - X(I)*A(KK,I)
        GO TO 330
 310    IF (KK .GT. MN) GO TO 320
        K1 = KK - M
        IW = IWW + K1
        W(IW) = W(IW) - W(K)
        W(IS) = XL(K1) - X(K1)
        GO TO 330
 320    K1 = KK - MN
        IW = IWW + K1
        W(IW) = W(IW) + W(K)
        W(IS) = -XU(K1) + X(K1)
 330 CONTINUE
C                                 PRE-MULTIPLY THE VECTOR IN THE
C                                 S-PARTITION OF W BY THE INVERS OF
C                                 R TRANSPOSE.
 340 IR = IWR
     IP = IWW + 1
     IPP = IWW + N
     IL = IWS + 1
     IU = IWS + NACT
     DO 350  I=IL, IU
        SUM = DDOT(I-IL,W(IR+1),1,W(IL),1)
        IR = IR + I - IL + 1
        W(I) = (W(I)-SUM)/W(IR)
 350 CONTINUE
C                                 SHIFT X TO SATISFY THE ACTIVE
C                                 CONSTRAINTS AND MAKE THE
C                                 CORRESPONDING CHANGE TO THE
C                                 GRADIENT RESIDUALS.
     DO 380  I=1, N
        IZ = IWZ + I
        SUM = DDOT(IU-IL+1,W(IL),1,W(IZ),N)
        IZ = IZ + (IU-IL+1)*N
        X(I) = X(I) + SUM
        IF (.NOT.LQL) THEN
           ID = IWD + I
           W(ID) = SUM*DSUM(N-I+1,G(I,I),LDG)
           IW = IWW + I
           DO 360  J=1, I
              ID = IWD + J
              W(IW) = W(IW) + G(J,I)*W(ID)
 360       CONTINUE
        ELSE
           DO 370  J=1, N
              IW = IWW + J
              W(IW) = W(IW) + SUM*G(I,J)
 370       CONTINUE
        END IF
 380 CONTINUE
C                                 FORM THE SCALAR PRODUCT OF THE
C                                 CURRENT GRADIENT RESIDUALS WITH
C                                 EACH COLUMN OF Z.
 390 KFLAG = 1
```

```
      GO TO 1260
  400 IF (NACT .NE. N) THEN
C                                 SHIFT X SO THAT IT SATISFIES THE
C                                 REMAINING KUHN-TUCKER CONDITIONS.
         IL = IWS + NACT + 1
         IZA = IWZ + NACT*N
         DO 410  I=1, N
            IZ = IZA + I
            SUM = DDOT(IWW-IL+1,W(IZ),N,W(IL),1)
            IZ = IZ + (IWW-IL+1)*N
            X(I) = X(I) - SUM
  410    CONTINUE
         INFO = ITERC
         IF (NACT .EQ. 0) GO TO 440
      END IF
C                                 UPDATE THE LAGRANGE MULTIPLIERS.
      LFLAG = 3
      GO TO 1030
  420 DO 430  K=1, NACT
         IW = IWW + K
         W(K) = W(K) + W(IW)
  430 CONTINUE
C                                 REVISE THE VALUES OF XMAG. BRANCH IF
C                                 ITERATIVE REFINEMENT IS REQUIRED.
  440 JFLAG = 1
      GO TO 1230
  450 IF (IFLAG .EQ. ITREF) GO TO 240
C                                 DELETE A CONSTRAINT IF A LAGRANGE
C                                 MULTIPLIER OF AN INEQUALITY
C                                 CONSTRAINT IS NEGATIVE.
      KDROP = 0
      GO TO 470
  460 KDROP = KDROP + 1
      IF (W(KDROP) .GE. 0.0D0) GO TO 470
      IF (IACT(KDROP) .LE. MEQ) GO TO 470
      NU = NACT
      MFLAG = 1
      GO TO 1120
  470 IF (KDROP .LT. NACT) GO TO 460
C                                 SEEK THE GREATEAST NORMALISED
C                                 CONSTRAINT VIOLATION, DISREGARDING
C                                 ANY THAT MAY BE DUE TO COMPUTER
C                                 ROUNDING ERRORS.
  480 CVMAX = 0.0D0
C
      DO 490  K=1, M
         IA = IWA + K
         IF (W(IA) .GT. 0.0D0) THEN
            SUM = DDOT(N,X,1,A(K,1),LDA) - B(K)
            SUMX = -SUM*W(IA)
            IF (K .LE. MEQ) SUMX = DABS(SUMX)
            IF (SUMX .GT. CVMAX) THEN
               TEMP = DABS(B(K)) + DA1OT(N,X,1,A(K,1),LDA)
               TEMPA = TEMP + DABS(SUM)
               IF (TEMPA .GT. TEMP) THEN
                  TEMP = TEMP + 1.5D0*DABS(SUM)
                  IF (TEMP .GT. TEMPA) THEN
                     CVMAX = SUMX
                     RES = SUM
                     KNEXT = K
                  END IF
               END IF
            END IF
         END IF
  490 CONTINUE
C
      DO 520  K=1, N
         LOWER = .TRUE.
         IA = IWA + M + K
```

```
         IF (W(IA) .LE. 0.0D0) GO TO 520
         SUM = XL(K) - X(K)
         IF (SUM) 500,520,510
  500    SUM = X(K) - XU(K)
         LOWER = .FALSE.
  510    IF (SUM .LE. CVMAX) GO TO 520
         CVMAX = SUM
         RES = -SUM
         KNEXT = K + M
         IF (LOWER) GO TO 520
         KNEXT = K + MN
  520 CONTINUE
C                                   TEST FOR CONVERGENCE
      INFO = ITERC
      IF (CVMAX .LE. VSMALL) GO TO 990
C                                   RETURN IF, DUE TO ROUNDING ERRORS,
C                                   THE ACTUAL CHANGE IN X MAY NOT
C                                   INCREASE THE OBJECTIVE FUNCTION
      JFINC = JFINC + 1
      IF (JFINC .EQ. 0) GO TO 590
      IF (JFINC .NE. IFINC) GO TO 610
      FDIFF = 0.0D0
      FDIFFA = 0.0D0
      DO 580  I=1, N
         SUM = 2.0D0*GRAD(I)
         SUMX = DABS(SUM)
         IF (LQL) GO TO 550
         ID = IWD + I
         W(ID) = 0.0D0
         DO 530  J=I, N
            IX = IWX + J
            W(ID) = W(ID) + G(I,J)*(W(IX)+X(J))
  530    CONTINUE
C
         DO 540  J=1, I
            ID = IWD + J
            TEMP = G(J,I)*W(ID)
            SUM = SUM + TEMP
            SUMX = SUMX + DABS(TEMP)
  540    CONTINUE
         GO TO 570
  550    DO 560  J=1, N
            IX = IWX + J
            TEMP = G(I,J)*(W(IX)+X(J))
            SUM = SUM + TEMP
            SUMX = SUMX + DABS(TEMP)
  560    CONTINUE
  570    IX = IWX + I
         FDIFF = FDIFF + SUM*(X(I)-W(IX))
         FDIFFA = FDIFFA + SUMX*DABS(X(I)-W(IX))
  580 CONTINUE
      INFO = 0
      SUM = FDIFFA + FDIFF
      IF (SUM .LE. FDIFFA) GO TO 990
      TEMP = FDIFFA + 1.5D0*FDIFF
      IF (TEMP .LE. SUM) GO TO 990
      JFINC = 0
  590 DO 600  I=1, N
         IX = IWX + I
         W(IX) = X(I)
  600 CONTINUE
C                                   FORM THE SCALAR PRODUCT OF THE NEW
C                                   CONSTRAINT NORMAL WITH EACH COLUMN
C                                   OF Z. PARNEW WILL BECOME THE
C                                   LAGRANGE MULTIPLIER OF THE NEW
C                                   CONSTRAINT.
  610 ITERC = ITERC + 1
      IWS = IWR + (NACT+NACT*NACT)/2
      IF (KNEXT .GT. M) GO TO 630
```

```
      DO 620  I=1, N
         IW = IWW + I
         W(IW) = A(KNEXT,I)
  620 CONTINUE
      GO TO 680
  630 DO 640  I=1, N
         IW = IWW + I
         W(IW) = 0.0D0
  640 CONTINUE
      K1 = KNEXT - M
      IF (K1 .GT. N) GO TO 660
      IW = IWW + K1
      W(IW) = 1.0D0
      IZ = IWZ + K1
      DO 650  I=1, N
         IS = IWS + I
         W(IS) = W(IZ)
         IZ = IZ + N
  650 CONTINUE
      GO TO 690
  660 K1 = KNEXT - MN
      IW = IWW + K1
      W(IW) = -1.0D0
      IZ = IWZ + K1
      DO 670  I=1, N
         IS = IWS + I
         W(IS) = -W(IZ)
         IZ = IZ + N
  670 CONTINUE
      GO TO 690
  680 KFLAG = 2
      GO TO 1260
  690 PARNEW = 0.0D0
C                                  APPLY GIVENS ROTATIONS TO MAKE THE
C                                  LAST (N-NACT-2) SCALAR PRODUCTS
C                                  EQUAL TO 0.0E0.
      IF (NACT .EQ. N) GO TO 740
      NU = N
      NFLAG = 1
      GO TO 1180
C                                  BRANCH IF THERE IS NO NEED TO DELETE
C                                  A CONSTRAINT.
  700 IS = IWS + NACT
      IF (NACT .EQ. 0) GO TO 930 .
      SUMA = 0.0D0
      SUMB = 0.0D0
      IZ = IWZ + NACT*N
      SUMC = DDOT(N,W(IZ+1),1,W(IZ+1),1)
      DO 710  I=1, N
         IZ = IZ + 1
         IW = IWW + I
         SUMA = SUMA + W(IW)*W(IZ)
         SUMB = SUMB + DABS(W(IW)*W(IZ))
  710 CONTINUE
      TEMP = SUMB + .1D+0*DABS(SUMA)
      TEMPA = SUMB + .2D+0*DABS(SUMA)
      IF (TEMP .LE. SUMB) GO TO 740
      IF (TEMPA .LE. TEMP) GO TO 740
      IF (SUMB .GT. VSMALL) GO TO 720
      GO TO 740
  720 SUMC = DSQRT(SUMC)
      IA = IWA + KNEXT
      IF (KNEXT .LE. M) SUMC = SUMC/W(IA)
      TEMP = SUMC + .1D+0*DABS(SUMA)
      TEMPA = SUMC + .2D+0*DABS(SUMA)
      IF (TEMP .LE. SUMC) GO TO 730
      IF (TEMPA .LE. TEMP) GO TO 730
      GO TO 930
C                                  CALCULATE THE MULTIPLIERS FOR THE
```

```
C                                     NEW CONSTRAINT NORMAL EXPRESSED IN
C                                     TERMS OF THE ACTIVE CONSTRAINT
C                                     NORMALS. THEN WORK OUT WHICH
C                                     CONTRAINT TO DROP.
  730 LFLAG = 4
      GO TO 1030
  740 LFLAG = 1
      GO TO 1030
C                                     COMPLETE THE TEST FOR LINEARLY
C                                       DEPENDENT CONSTRAINTS.
  750 IF (KNEXT .GT. M) GO TO 790
      DO 780  I=1, N
         SUMA = A(KNEXT,I)
         SUMB = DABS(SUMA)
         IF (NACT .EQ. 0) GO TO 770
         DO 760  K=1, NACT
            KK = IACT(K)
            IW = IWW + K
            TEMP = W(IW)*A(KK,I)
            SUMA = SUMA - TEMP
            SUMB = SUMB + DABS(TEMP)
  760    CONTINUE
  770    IF (SUMA .LE. VSMALL) GO TO 780
         TEMP = SUMB + .1D+0*DABS(SUMA)
         TEMPA = SUMB + .2D+0*DABS(SUMA)
         IF (TEMP .LE. SUMB) GO TO 780
         IF (TEMPA .LE. TEMP) GO TO 780
         GO TO 920
  780 CONTINUE
      LFLAG = 1
      GO TO 1080
  790 K1 = KNEXT - M
      IF (K1 .GT. N) K1 = K1 - N
      DO 850  I=1, N
         SUMA = 0.0D0
         IF (I .NE. K1) GO TO 800
         SUMA = 1.0D0
         IF (KNEXT .GT. MN) SUMA = -1.0D0
  800    SUMB = DABS(SUMA)
         IF (NACT .EQ. 0) GO TO 840
         DO 830  K=1, NACT
            KK = IACT(K)
            IF (KK .LE. M) GO TO 810
            KK = KK - M
            TEMP = 0.0D0
            IF (KK .EQ. I) TEMP = W(IWW+KK)
            KK = KK - N
            IF (KK .EQ. I) TEMP = -W(IWW+KK)
            GO TO 820
  810       IW = IWW + K
            TEMP = W(IW)*A(KK,I)
  820       SUMA = SUMA - TEMP
  830    SUMB = SUMB + DABS(TEMP)
  840    TEMP = SUMB + .1D+0*DABS(SUMA)
         TEMPA = SUMB + .2D+0*DABS(SUMA)
         IF (TEMP .LE. SUMB) GO TO 850
         IF (TEMPA .LE. TEMP) GO TO 850
         GO TO 920
  850 CONTINUE
      LFLAG = 1
      GO TO 1080
C                                     BRANCH IF THE CONTRAINTS ARE
C                                       INCONSISTENT.
  860 INFO = -KNEXT
      IF (KDROP .EQ. 0) GO TO 990
      PARINC = RATIO
      PARNEW = PARINC
C                                     REVISE THE LAGRANGE MULTIPLIERS OF
C                                       THE ACTIVE CONSTRAINTS.
```

```
  870 IF (NACT .EQ. 0) GO TO 890
      DO 880  K=1, NACT
         IW = IWW + K
         W(K) = W(K) - PARINC*W(IW)
         IF (IACT(K) .GT. MEQ) W(K) = DMAX1(0.0D0,W(K))
  880 CONTINUE
  890 IF (KDROP .EQ. 0) GO TO 970
C                                   DELETE THE CONSTRAINT TO BE DROPPED.
C                                     SHIFT THE VECTOR OF SCALAR
C                                     PRODUCTS. THEN, IF APPROPRIATE,
C                                     MAKE ONE MORE SCALAR PRODUCT
C                                     0.0E0.
      NU = NACT + 1
      MFLAG = 2
      GO TO 1120
  900 IWS = IWS - NACT - 1
      NU = MIN0(N,NU)
      DO 910  I=1, NU
         IS = IWS + I
         J = IS + NACT
  910 W(IS) = W(J+1)
      NFLAG = 2
      GO TO 1180
C                                   CALCULATE THE STEP TO THE VIOLATED
C                                     CONSTRAINT.
  920 IS = IWS + NACT
  930 SUMY = W(IS+1)
      STEP = -RES/SUMY
      PARINC = STEP/SUMY.
      IF (NACT .EQ. 0) GO TO 950
C                                   CALCULATE THE CHANGES TO THE
C                                     LAGRANGE MULTIPLIERS, AND REDUCE
C                                     THE STEP ALONG THE NEW SEARCH
C                                     DIRECTION IF NECESSARY.
      LFLAG = 2
      GO TO 1030
  940 IF (KDROP .EQ. 0) GO TO 950
      TEMP = 1.0D0 - RATIO/PARINC
      IF (TEMP .LE. 0.0D0) KDROP = 0
      IF (KDROP .EQ. 0) GO TO 950
      STEP = RATIO*SUMY
      PARINC = RATIO
      RES = TEMP*RES
C                                   UPDATE X AND THE LAGRANGE
C                                     MULTIPIERS. DROP A CONSTRAINT IF
C                                     THE FULL STEP IS NOT TAKEN.
  950 IWY = IWZ + NACT*N
      DO 960  I=1, N
         IY = IWY + I
  960 X(I) = X(I) + STEP*W(IY)
      PARNEW = PARNEW + PARINC
      IF (NACT .GE. 1) GO TO 870
C                                   ADD THE NEW CONSTRAINT TO THE ACTIVE
C                                     SET.
  970 NACT = NACT + 1
      W(NACT) = PARNEW
      IACT(NACT) = KNEXT
      IA = IWA + KNEXT
      IF (KNEXT .GT. MN) IA = IA - N
      W(IA) = -W(IA)
C                                   ESTIMATE THE MAGNITUDE OF X. THEN
C                                     BEGIN A NEW ITERATION,
C                                     RE-INITILISING X IF THIS MAGNITUDE
C                                     IS SMALL.
      JFLAG = 2
      GO TO 1230
  980 IF (SUM .LT. (XMAGR*XMAG)) GO TO 200
      IF (ITREF) 480,480,240
C                                   INITIATE ITERATIVE REFINEMENT IF IT
```

```
C                                       HAS NOT YET BEEN USED, OR RETURN
C                                       AFTER RESTORING THE DIAGONAL
C                                       ELEMENTS OF G.
  990 IF (ITERC .EQ. 0) GO TO 1000
      ITREF = ITREF + 1
      JFINC = -1
      IF (ITREF .EQ. 1) GO TO 240
 1000 IF (.NOT.LQL) RETURN
      DO 1010  I=1, N
         ID = IWD + I
 1010 G(I,I) = W(ID)
 1020 RETURN
C                                     THE REMAINIG INSTRUCTIONS ARE USED
C                                       AS SUBROUTINES. CALCULATE THE
C                                       LAGRANGE MULTIPLIERS BY
C                                       PRE-MULTIPLYING THE VECTOR IN THE
C                                       S-PARTITION OF W BY THE INVERSE OF
C                                       R.
 1030 IR = IWR + (NACT+NACT*NACT)/2
      I = NACT
      SUM = 0.0D0
      GO TO 1070
 1040 IRA = IR - 1
      SUM = 0.0D0
      IF (NACT .EQ. 0) GO TO 1060
      DO 1050  J=I, NACT
         IW = IWW + J
         SUM = SUM + W(IRA)*W(IW)
 1050 IRA = IRA + J
 1060 IR = IR - I
      I = I - 1
 1070 IW = IWW + I
      IS = IWS + I
      W(IW) = (W(IS)-SUM)/W(IR)
      IF (I .GT. 1) GO TO 1040
      IF (LFLAG .EQ. 3) GO TO 420
      IF (LFLAG .EQ. 4) GO TO 750
C                                     CALCULATE THE NEXT CONSTRAINT TO
C                                       DROP.
 1080 IP = IWW + 1
      IPP = IWW + NACT
      KDROP = 0
      IF (NACT .EQ. 0) GO TO 1110
      DO 1100  K=1, NACT
         IF (IACT(K) .LE. MEQ) GO TO 1100
         IW = IWW + K
         IF ((RES*W(IW)) .GE. 0.0D0) GO TO 1100
         TEMP = W(K)/W(IW)
         IF (KDROP .EQ. 0) GO TO 1090
         IF (DABS(TEMP) .GE. DABS(RATIO)) GO TO 1100
 1090    KDROP = K
         RATIO = TEMP
 1100 CONTINUE
 1110 GO TO (860, 940), LFLAG
C                                     DROP THE CONSTRAINT IN POSITION
C                                       KDROP IN THE ACTIVE SET.
 1120 IA = IWA + IACT(KDROP)
      IF (IACT(KDROP) .GT. MN) IA = IA - N
      W(IA) = -W(IA)
      IF (KDROP .EQ. NACT) GO TO 1170
C                                     SET SOME INDICES AND CALCULATE THE
C                                       ELEMENTS OF THE NEXT GIVENS
C                                       ROTATION.
      IZ = IWZ + KDROP*N
      IR = IWR + (KDROP+KDROP*KDROP)/2
 1130 IRA = IR
      IR = IR + KDROP + 1
      TEMP = DMAX1(DABS(W(IR-1)),DABS(W(IR)))
      SUM = TEMP*DSQRT((W(IR-1)/TEMP)**2+(W(IR)/TEMP)**2)
```

```
      GA = W(IR-1)/SUM
      GB = W(IR)/SUM
C                                     EXCHANGE THE COLUMNS OF R.
      DO 1140  I=1, KDROP
         IRA = IRA + 1
         J = IRA - KDROP
         TEMP = W(IRA)
         W(IRA) = W(J)
 1140 W(J) = TEMP
      W(IR) = 0.0D0
C                                     APPLY THE ROTATION TO THE ROWS OF R.
      W(J) = SUM
      KDROP = KDROP + 1
      DO 1150  I=KDROP, NU
         TEMP = GA*W(IRA) + GB*W(IRA+1)
         W(IRA+1) = GA*W(IRA+1) - GB*W(IRA)
         W(IRA) = TEMP
 1150 IRA = IRA + I
C                                     APPLY THE ROTATION TO THE COLUMNS OF
C                                       Z.
      DO 1160  I=1, N
         IZ = IZ + 1
         J = IZ - N
         TEMP = GA*W(J) + GB*W(IZ)
         W(IZ) = GA*W(IZ) - GB*W(J)
 1160 W(J) = TEMP
C                                     REVISE IACT AND THE LAGRANGE
C                                       MULTIPLIERS.
      IACT(KDROP-1) = IACT(KDROP)
      W(KDROP-1) = W(KDROP)
      IF (KDROP .LT. NACT) GO TO 1130
 1170 NACT = NACT - 1
      GO TO (240, 900), MFLAG
C                                     APPLY GIVENS ROTATION TO REDUCE SOME
C                                       OF THE SCALAR PRODUCTS IN THE
C                                       S-PARTITION OF W TO 0.0E0.
 1180 IZ = IWZ + NU*N
 1190 IZ = IZ - N
 1200 IS = IWS + NU
      NU = NU - 1
      IF (NU .EQ. NACT) GO TO 1220
      IF (W(IS) .EQ. 0.0D0) GO TO 1190
      TEMP = DMAX1(DABS(W(IS-1)),DABS(W(IS)))
      SUM = TEMP*DSQRT((W(IS-1)/TEMP)**2+(W(IS)/TEMP)**2)
      GA = W(IS-1)/SUM
      GB = W(IS)/SUM
      W(IS-1) = SUM
      DO 1210  I=1, N
         K = IZ + N
         TEMP = GA*W(IZ) + GB*W(K)
         W(K) = GA*W(K) - GB*W(IZ)
         W(IZ) = TEMP
 1210 IZ = IZ - 1
      GO TO 1200
 1220 GO TO (700, 920), NFLAG
C                                     CALCULATE THE MAGNITUDE OF X AN
C                                       REVISE XMAG.
 1230 SUM = 0.0D0
      DO 1240  I=1, N
         SUM = SUM + DABS(X(I))*VFACT*(DABS(GRAD(I))+DABS(G(I,I)*X(I)))
         IF (LQL) GO TO 1240
         IF (SUM .LT. 1.0D-30) GO TO 1240
         VFACT = 1.0D-10*VFACT
         SUM = 1.0D-10*SUM
         XMAG = 1.0D-10*XMAG
 1240 CONTINUE
 1250 XMAG = DMAX1(XMAG,SUM)
      GO TO (450, 980), JFLAG
C                                     PRE-MULTIPLY THE VECTOR IN THE
```

```
C                                          W-PARTITION OF W BY Z TRANSPOSE.
 1260 JL = IWW + 1
      IZ = IWZ
      DO 1270  I=1, N
         IS = IWS + I
         W(IS) = 0.0D0
         IWWN = IWW + N
         W(IS) = DDOT(IWWN-JL+1,W(JL),1,W(IZ+1),1)
         IZ = IZ + (IWWN-JL+1)
 1270 CONTINUE
      GO TO (400, 690), KFLAG
      RETURN
      END
```

```
      PROGRAM OPTIMNN
C
C
C
C                       By Permission of Her Most Gracious Majesty
C
C                                       Elizabeth R
C
C
C                                   Jane Leyland presents
C
C                       "Ye Olde Crystal Balle Magic Incantation"
C
C                     An Elizabethan witchcraft and sorcery product
C                     which defines Ye Olde Magic Control Wand.
C
C
C   *****  This is a Main Driver Program for the Optimal Neural-Network
C          Closed-Loop Trajectory Controller described in:
C
C        1.  Leyland, Jane A.,   "A Closed-Loop Optimal Neural-Network
C              Controller to Optimise Rotorcraft Aeromechanical Behaviour",
C            to be published as a NASA Technical Memorandum.
C
C        2.  Leyland, Jane A.,  "A Higher Harmonic Optimal Controller
C            to Optimise Rotorcraft Aeromechanical Behaviour",   NASA
C              Technical Memorandum 110390,  March 1996.
C
C
C   *****   Start PROGRAM OPTIMNN   *****
C
C
C
C   *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C           This file contains the statements which establish and define:
C           1) the Principal COMMON Blocks;   2) the Data TYPE of the
C           Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C           DIMENSION of the Principal  Arrays  and  Vectors  of the
C           OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 ICASE, JERR
C
      EXTERNAL INIT, TRAJ
      REAL*8   INIT, TRAJ
C
C
 1071 FORMAT(43H0  *****   NORMAL EXIT FROM OPTIMNN   *****/)
 1072 FORMAT(42H0  *****   ERROR EXIT FROM OPTIMNN   *****//)
C
C
C  *****   Initialisation   *****
C
      RTD   = 360.000/TWOPI
      ICASE = 1
      JERR  = 0
C
C  *****   Run Case Number "ICASE"
C
  100 CONTINUE
C
      CALL INIT(JERR)
C
      CALL TRAJ(JERR)
C
      IF (JERR .NE. 0) GO TO 996
```

```
      GO TO 997
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      JERR = 0
      GO TO 998
C
C   *****   Normal Exit   *****
C
  997 WRITE(6,1071)
      WRITE(8,1071)
C
C  *****   Check for subsequent case   *****
C
  998 IF (MULT .LE. 0) GO TO 999
      MULT = 0
      ICASE = ICASE + 1
      GO TO 100
  999 STOP
      END

C2345678901234567890123456789012345678901234567890123456789012345678901234567890
C2345678901234567890123456789012345678901234567890123456789012345678901234567890
C2345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE INIT(JERR)
C
C
C   *****    This subroutine: 1) reads changes to the Data Set Values of
C            the INPUT DATA via NAMELIST CDATA,  and then 2) initialises
C            the data required to execute the trajectory simulation.
C
C
C   *****    Start SUBROUTINE INIT   *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]INITDAT.INC"  File is Included here.
C            This file contains the statements which define the initially
C            set Default Values of the  "NAMELIST CDATA"  INPUT Parameters
C            and the Values of the Internally Set Constants of the OPTIMNN
C            System.
C
      INCLUDE '[LEYLAND.OPTIMNN]INITDAT.INC'
C
C
C
      INTEGER*4 JERR
C
C
      NAMELIST / CDATA / A, A1, A2, A3, ALPHA, AMAXC, AMAXNNC, AMAXNNL,
     1 AMINC, AMINNNC, AMINNNL, AN, B, B1, B2, B3, BN, C, C1, C2, C3,
     2 CDELAY, CN, CONST1, CONST2, CONST3, CONST4, CONST5, CVTID, CW,
     3 D, D1, D2, D3, DCFREQ, DCLGTH, DLFREQ, DLLGTH, DN, ICONC,
     4 ICONNNC, ICONNNL, ICV, IFUNCT, IJKCVC, IJKCVL, IOPTC, IOPTNNC,
     5 IOPTNNL, ISEED1, ISEED2, ISEED3, ISTEP0, JEC, JJECC, JJECL,
     6 JSEED1, JSEED2, JSEED3, LARGE1, LARGE2, LARGE3, LARGE4, LDELAY,
     7 MITNC, MITNNNC, MITNNNL, MULT, NFUNCT, NI, NJ, NK, NL2, NL3, NN,
     8 NNCID, NNLID, OMEGA, OUTC, OUTNNC, OUTNNL, PERIOD, PHASE, PHI,
     9 PSI, SCVC, SCVNNC, SCVNNL, SMALL1, SMALL2, SMALL3, SMALL4, SMAXC,
     o STMODC, STMODL, TBLMAX, TCINIT, TCFINL, TCSTEP, TCTYPE, TD,
     1 TINIT, TFINL, TLINIT, TLFINL, TLSTEP, TLTYPE, TTBL, UPDATE, WTC,
     2 WTNNC, WTNNL, WTSNNC, WTSNNL, X0, XD, XN0, XTBL, Y0, YD, YN0,
     3 YR1, YR2, YR3, YTBL
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(40H0  *****   NORMAL EXIT FROM INIT    *****//)
 1072 FORMAT(39H0  *****   ERROR EXIT FROM INIT   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Pre-Input Data Initialisation   *****
C
      JERR = 0
C
C   *****    Read INPUT Data with NAMELIST CDATA   *****
C
      READ(7,CDATA)
C
C   *****    Write INPUT Data   *****
C
```

```
C
C   *****   Post-Input Data Initialisation   *****
C
      IF (JERR .NE. 0) GO TO 996
      GO TO 997
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit   *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
```

```
      SUBROUTINE TRAJ(JERR)
C
C
C   *****   This subroutine:  1) initialises for trajectory propagation,
C           2) provides phase cut-logic,  3) reads reference trajectory
C           data,  4) updates Neural-Network Parameters,  5) updates the
C           control Vector,  and 6) propagates the trajectory.
C
C
C   *****   Start SUBROUTINE TRAJ   *****
C
C
C
C   *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C           This file contains the statements which establish and define:
C           1) the Principal COMMON Blocks;   2) the Data TYPE of the
C           Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C           DIMENSION of the Principal  Arrays  and  Vectors  of the
C           OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 I, ICODE, J, JERR, K, L, LL
      INTEGER*4 IBUG
C
      REAL*8 XDUM(NL2DIM)
C
      EXTERNAL CVVCTR, DNCONF, DNCONG, ECVCTR, ERSET, IERCD, JCTRL,
     1 JNNW, STATE
      REAL*8   CVVCTR, DNCONF, DNCONG, ECVCTR, ERSET,        JCTRL,
     1 JNNW, STATE
      INTEGER*4 IERCD
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1010 FORMAT(35H1  *****   START TRAJECTORY   *****//)
 1011 FORMAT(20H0  *****   PHASE  = ,I2,3X,11H   TABS  = ,D13.5,3X,
     1 11H   TREL  = ,D13.5/25X,11H   PINDX = ,D13.5)
 1071 FORMAT(40H0  *****   NORMAL EXIT FROM TRAJ   *****//)
 1072 FORMAT(39H0  *****   ERROR EXIT FROM TRAJ   *****//)
 1096 FORMAT(31H0  *****   TRAJ DEBUG POINT  = ,I3,15H,     NNID   = ,
     1 I3,8H   *****)
 1097 FORMAT(31H0  *****   TRAJ DEBUG POINT  = ,I3,15H,     PHASE  = ,
     1 I3,15H,     ICODE  = ,I3,8H   *****)
 1098 FORMAT(2X,6I12)
 1099 FORMAT(31H0  *****   TRAJ DEBUG POINT  = ,I3,15H,     PHASE  = ,
     1 I3,8H   *****)
 7011 FORMAT(4D20.8)
C
C
C   *****   INITIALISATION   *****
C
      JERR = 0
      IF (TFINL-TINIT) 996, 996, 101
      GO TO 996
  101 WRITE(6,1010)
      WRITE(8,1010)
C     DFREQ0 = 0
      LMAX   = 0
C     NNUP0  = 0
      TABS   = TINIT
      ISTEP  = 1
      IF (TLFINL-TLINIT) 141, 141, 102
C
C   *****   Initialisation for the Learning Trajectory Phase   *****
C
```

```
  102 IPHASE = 1
      DELAY  = LDELAY
      DFREQ  = DLFREQ
      DLGTH  = DLLGTH
      NNID   = NNLID
      IF (NNID-1) 115, 114, 115
  114 NNUP   = 0
      GO TO 116
  115 NNUP   = 1
  116 TCUT   = TLFINL
      TSTEP  = TLSTEP
      IF (TLTYPE) 103, 103, 104
  103 TABS   = TLINIT
      TREL   = ZERO
      T      = TABS
      GO TO 105
  104 TREL   = TLINIT
      TABS   = TINIT + TLINIT
      T      = TREL
C
  105 TD(1)    = T
C
      IBUG = 105
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)   T, TABS, TREL
      WRITE(8,7011)   T, TABS, TREL
      WRITE(6,1000)
      WRITE(8,1000)
C
      IJK     = 0
      NIJKCVL = 0
      DO 109 K=1,NK
      DO 108 I=1,NI(K)
      DO 107 J=1,NJ(K)
      IF (IJKCVL(I,J,K)) 107, 107, 106
  106 IJK = IJK + 1
      CMAXNNL(IJK) = AMAXNNL(I,J,K)
      CMINNNL(IJK) = AMINNNL(I,J,K)
      CVSNNL(IJK)  = SCVNNL(I,J,K)
  107 CONTINUE
  108 CONTINUE
  109 CONTINUE
      NIJKCVL = IJK
C
      JJJ    = 0
      NJJECL = 0
      DO 111 J=1,NL2(2)
      IF (JJECL(J)) 111, 111, 110
  110 JJJ = JJJ + 1
      WNNL(JJJ) = WTNNL(J)
  111 CONTINUE
      NJJECL = JJJ
C
      NCONNNL = 0
C
C   *****   Definition of the Initial Data Values for the Sliding Window
C           Table (L = 1).
C
      CALL STATE(XD(1,1),YD(1,1),JERR)
      IF (JERR .NE. 0) GO TO 996
      TD(1) = T
C
C
      IBUG = 111
```

```
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)   T, TABS, TREL
      WRITE(8,7011)   T, TABS, TREL
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)   (XD(I,1), I=1,NL2(1))
      WRITE(8,7011)   (XD(I,1), I=1,NL2(1))
      WRITE(6,7011)   (YD(J,1), J=1,NL2(2))
      WRITE(8,7011)   (YD(J,1), J=1,NL2(2))
      WRITE(6,1000)
      WRITE(8,1000)
C
      GO TO 181
C
  141 IF (TCFINL-TCINIT) 997, 997, 142
C
C   *****   Initialisation for the Controlled Trajectory Phase   *****
C
  142 IPHASE = 5
      GO TO (171,173,173,171), STMODC
  171 IF (ISTEP0) 173, 172, 173
  172 ISTEP = 1
  173 ISTEP = ISTEP + ISTEP0
      DELAY  = CDELAY
      DFREQ  = DCFREQ
      DLGTH  = DCLGTH
      NNID   = NNCID
      IF (NNID-1) 175, 174, 175
  174 NNUP   = 0
      GO TO 176
  175 NNUP   = 1
  176 TCUT   = TCFINL
      TSTEP  = TCSTEP
      IF (TCTYPE) 143, 143, 144
  143 TABS   = TCINIT
      TREL   = ZERO
      T      = TABS
      GO TO 145
  144 TREL   = TCINIT
      TABS   = TABS + TREL
      T      = TREL
C
  145 TD(1)   = T
C
      IBUG = 145
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)   T, TABS, TREL
      WRITE(8,7011)   T, TABS, TREL
      WRITE(6,1000)
      WRITE(8,1000)
C
      IJK     = 0
      NIJKCVC = 0
      DO 149 K=1,NK
      DO 148 I=1,NI(K)
      DO 147 J=1,NJ(K)
      IF (IJKCVC(I,J,K)) 147, 147, 146
  146 IJK = IJK + 1
      CMAXNNC(IJK) = AMAXNNC(I,J,K)
```

```
      CMINNNC(IJK) = AMINNNC(I,J,K)
      CVSNNC(IJK)  = SCVNNC(I,J,K)
  147 CONTINUE
  148 CONTINUE
  149 CONTINUE
      NIJKCVC = IJK
C
      JJJ    = 0
      NJJECC = 0
      DO 151 J=1,NL2(2)
      IF (JJECC(J)) 151, 151, 150
  150 JJJ = JJJ + 1
      WNNC(JJJ) = WTNNC(J)
  151 CONTINUE
      NJJECC = JJJ
C
      NCONNNC = 0
C
      II   = 0
      NICV = 0
      DO 157 I=1,NL2(1)
      IF (ICV(I)) 157, 157, 156
  156 II = II + 1
      CMAXC(II) = AMAXC(I)
      CMINC(II) = AMINC(I)
      CVSC(II)  = SCVC(I)
  157 CONTINUE
      NICV = II
C
      JJ   = 0
      NJEC = 0
      DO 159 J=1,NL2(2)
      IF (JEC(J)) 159, 159, 158
  158 JJ = JJ + 1
      WC(JJ) = WTC(J)
  159 CONTINUE
      NJEC = JJ
C
      III   = 0
      NCONC = 0
      DO 161 I=1,NL2(1)
      IF (ICONC(I)) 161, 161, 160
  160 III = III + 1
  161 CONTINUE
      NCONC = III
C
C   *****   Definition of the Initial Data Values for the Sliding Window
C           Table (L = 1).
C
      CALL STATE(XD(1,1),YD(1,1),JERR)
      IF (JERR .NE. 0) GO TO 996
      TD(1)   = T
C
C
      IBUG = 161
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)   T, TABS, TREL
      WRITE(8,7011)   T, TABS, TREL
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)    (XD(I,1), I=1,NL2(1))
      WRITE(8,7011)    (XD(I,1), I=1,NL2(1))
      WRITE(6,7011)    (YD(J,1), J=1,NL2(2))
      WRITE(8,7011)    (YD(J,1), J=1,NL2(2))
```

```
      WRITE(6,1000)
      WRITE(8,1000)
C
  181 ICUT  = 0
C
C   *****   CUT LOGIC   *****
C
C
  200 IBUG = 200
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
C     WRITE(6,1000)
C     WRITE(8,1000)
C     WRITE(6,7011)   T, TABS, TREL
C     WRITE(8,7011)   T, TABS, TREL
C     WRITE(6,1000)
C     WRITE(8,1000)
C
      IF (T-TCUT) 300, 202, 201
  201 T = TCUT
  202 ICUT = 1
      NNUP = 0
      IF (IPHASE) 996, 996, 203
  203 GO TO (204,204,205,996,206,206,207,996), IPHASE
  204 IPHASE = 3
C 205 DFREQ0 = 1
C     DLGTH  = DLLGTH
C     NNUP0  = 1
  205 DLGTH  = DLLGTH
      GO TO 300
  206 IPHASE = 7
  207 CVUP   = 0
      DLGTH  = DCLGTH
C
C   *****   READ REFERENCE TRAJECTORY DATA   *****
C
C
  300 IBUG = 300
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
      WRITE(6,1000)
      WRITE(8,1000)
C     WRITE(6,7011)   T, TABS, TREL
C     WRITE(8,7011)   T, TABS, TREL
C     WRITE(6,1000)
C     WRITE(8,1000)
C     WRITE(6,7011)    (XD(I,1), I=1,NL2(1))
C     WRITE(8,7011)    (XD(I,1), I=1,NL2(1))
C     WRITE(6,7011)    (YD(J,1), J=1,NL2(2))
C     WRITE(8,7011)    (YD(J,1), J=1,NL2(2))
C     WRITE(6,1000)
C     WRITE(8,1000)
C
      IF (DFREQ) 996, 996, 301
  301 DATAR = JMOD(ISTEP-1,DFREQ)
C     IF (DATAR)  302, 304, 302
C 302 IF (IPHASE) 996, 996, 303
C 303 GO TO (600,600,600,996,511,511,511,996), IPHASE
C 304 IF (IPHASE.NE.5 .OR. NNUP0.EQ.0) GO TO 305
C     GO TO 500
C
C 305 LSTEP = 1 + (ISTEP - 1)/DFREQ
C     IF (DELAY-LSTEP) 306, 302, 302
C 306 IF (IPHASE.NE.5 .OR. DFREQ0.EQ.0) GO TO 321
C     DFREQ0 = 0
```

```
C     GO TO 500
C
      IF (DATAR)  302, 305, 302
  302 IF (IPHASE) 996, 996, 303
  303 GO TO (600,600,600,996,511,511,511,996), IPHASE
C
  305 LSTEP = 1 + (ISTEP - 1)/DFREQ
      IF (DELAY-LSTEP) 321, 302, 302
C
  321 IF (LMAX-DLGTH) 323, 324, 322
  322 LMAX = DLGTH
      GO TO 324
  323 LMAX = LMAX + 1
      IF (LMAX-1) 996, 370, 324
C
C   *****   Advance the Data Values for the Sliding Window Table
C           (L = 1 to LMAX).
C
  324 DO 330 L = 1, LMAX-1
      LL = LMAX - L
      TD(LL+1)    = TD(LL)
      DO 325 I = 1, NL2(1)
      XD(I,LL+1) = XD(I,LL)
  325 CONTINUE
      DO 326 J = 1, NL2(2)
      YD(J,LL+1) = YD(J,LL)
  326 CONTINUE
  330 CONTINUE
C
C   *****   Definition of the First Set (L = 1) of Data Values for the
C           Sliding Window Table.
C
  370 CALL STATE(XD(1,1),YD(1,1),JERR)
      IF (JERR .NE. 0) GO TO 996
      TD(1)    = T
C
      IBUG = 370
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)   T, TABS, TREL
      WRITE(8,7011)   T, TABS, TREL
      WRITE(6,1000)
      WRITE(8,1000)
C
C  *****   Determine State from Neural-Net Model   *****
C
      DO 373 I = 1,NL2(1)
      XN(I) = XD(I,1)
  373 CONTINUE
      CALL STATENN(XN,YN,JERR)
      WRITE(6,7011)    (XN(I), I=1,NL2(1))
      WRITE(8,7011)    (XN(I), I=1,NL2(1))
      WRITE(6,7011)    (YN(J), J=1,NL2(2))
      WRITE(8,7011)    (YN(J), J=1,NL2(2))
      WRITE(6,1000)
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(8,1000)
      DO 969 L=1,LMAX
      WRITE(6,7011)    (XD(I,L), I=1,NL2(1))
      WRITE(8,7011)    (XD(I,L), I=1,NL2(1))
      WRITE(6,7011)    (YD(J,L), J=1,NL2(2))
      WRITE(8,7011)    (YD(J,L), J=1,NL2(2))
      WRITE(6,1000)
      WRITE(8,1000)
```

```
  969 CONTINUE
      WRITE(6,1000)
      WRITE(8,1000)
C
      IF  (NNID)  371, 371, 400
  371 IF (IPHASE) 996, 996, 372
  372 GO TO (600,600,600,996,500,500,500,996), IPHASE
C
C   *****   NEURAL-NETWORK UPDATE   *****
C
C
  400 IBUG = 400
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
C     WRITE(6,1000)
C     WRITE(8,1000)
C     WRITE(6,1098)   NCON,    NCV,     NEC,     NIDIM,    NIJKDIM, NJDIM
C     WRITE(6,1098)   NJKDIM,  NKDIM,   NL1DIM,  NL21,     NL2DIM,  NL321
C     WRITE(6,1098)   NL3DIM,  NLDIM,   NLTBL
C     WRITE(8,1098)   NCON,    NCV,     NEC,     NIDIM,    NIJKDIM, NJDIM
C     WRITE(8,1098)   NJKDIM,  NKDIM,   NL1DIM,  NL21,     NL2DIM,  NL321
C     WRITE(8,1098)   NL3DIM,  NLDIM,   NLTBL
C     WRITE(6,1000)
C     WRITE(8,1000)
C     WRITE(6,1098)   ISTEP,  LSTEP, ICUT,  NNID,  NNUP0,   NNUP,
C    1                DFREQ0, DFREQ, DATAR, DLGTH, IPHASE,  LMAX
C     WRITE(8,1098)   ISTEP,  LSTEP, ICUT,  NNID,  NNUP0,   NNUP,
C    1                DFREQ0, DFREQ, DATAR, DLGTH, IPHASE,  LMAX
C     WRITE(6,1098)   ISTEP,  LSTEP, ICUT,  NNID,   NNUP,
C    1                DFREQ,  DATAR, DLGTH, IPHASE, LMAX
C     WRITE(8,1098)   ISTEP,  LSTEP, ICUT,  NNID,   NNUP,
C    1                DFREQ,  DATAR, DLGTH, IPHASE, LMAX
C     WRITE(6,7011)   T,      TABS,  TREL,  TCUT
C     WRITE(8,7011)   T,      TABS,  TREL,  TCUT
C
      IF (NNID) 500, 500, 401
C 401 IF (NNUP) 500, 402, 500
C 402 IF (IPHASE.NE.5 .OR. NNUP0.EQ.0) GO TO 403
C     NNUP0 = 0
C     GO TO 500
  401 IF (NNUP) 500, 403, 500
  403 IF (IPHASE) 996, 996, 404
  404 GO TO (411,411,411,996,421,421,421,996), IPHASE
C
C  *****   Neural-Net Optimisation During the Learning Trajectory Phase   *****
C
  411 ICVDEF = 1
      IECDEF = 1
      CALL CVVCTR(XDUM,JERR)
      IF (JERR .NE. 0) GO TO 996
      DO 412 IJK = 1,NIJKCVL
      CV0(IJK) = CV(IJK)
  412 CONTINUE
C
      IBUG = 412
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
C     WRITE(6,1000)
C     WRITE(8,1000)
C     WRITE(6,7011)   T, TABS, TREL
C     WRITE(8,7011)   T, TABS, TREL
C     WRITE(6,1000)
C     WRITE(8,1000)
C     DO 971 L=1,LMAX
C     WRITE(6,7011)   (XD(I,L), I=1,NL2(1))
```

```
C     WRITE(8,7011)    (XD(I,L), I=1,NL2(1))
C     WRITE(6,7011)    (YD(J,L), J=1,NL2(2))
C     WRITE(8,7011)    (YD(J,L), J=1,NL2(2))
C     WRITE(6,1000)
C     WRITE(8,1000)
C 971 CONTINUE
C     WRITE(6,7011)    (CV0(IJK), IJK=1,NIJKCVL)
C     WRITE(8,7011)    (CV0(IJK), IJK=1,NIJKCVL)
C     WRITE(6,1000)
C     WRITE(8,1000)
C
C
C  *****   ICODE is the IMSL Informational Error Code Number   *****
C
C                ICODE = 1  indicates that the Search Direction is Uphill.
C                ICODE = 2  indicates that the Line Search required more
C                           than  5  Function Calls.
C                ICODE = 3  indicates that the Maximum Number of Iterations
C                           were Exceeded.
C                ICODE = 4  indicates that the Search Direction vector is
C                           close to being a Zero vector.
C
      CALL ERSET(0,1,0)
      ICODE = 0
      CALL DNCONF(JNNW,NCONNNL,0,NIJKCVL,CV0,CVBDNNL,CMINNNL,CMAXNNL,
     1 CVSNNL,OUTNNL,MITNNNL,CV,PINDX)
      ICODE = IERCD()
C
      IBUG = 489
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1097) IBUG, IPHASE, ICODE
      WRITE(8,1097) IBUG, IPHASE, ICODE
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)    T, TABS, TREL
      WRITE(8,7011)    T, TABS, TREL
      WRITE(6,1000)
      WRITE(8,1000)
C
C  *****   Determine State from Neural-Net Model   *****
C
      DO 413 I = 1,NL2(1)
      XN(I) = XD(I,1)
  413 CONTINUE
      CALL STATENN(XN,YN,JERR)
      WRITE(6,7011)    (XN(I), I=1,NL2(1))
      WRITE(8,7011)    (XN(I), I=1,NL2(1))
      WRITE(6,7011)    (YN(J), J=1,NL2(2))
      WRITE(8,7011)    (YN(J), J=1,NL2(2))
      WRITE(6,1000)
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(8,1000)
      DO 972 L=1,LMAX
      WRITE(6,7011)    (XD(I,L), I=1,NL2(1))
      WRITE(8,7011)    (XD(I,L), I=1,NL2(1))
      WRITE(6,7011)    (YD(J,L), J=1,NL2(2))
      WRITE(8,7011)    (YD(J,L), J=1,NL2(2))
      WRITE(6,1000)
      WRITE(8,1000)
  972 CONTINUE
      WRITE(6,7011)    (CV(IJK), IJK=1,NIJKCVL)
      WRITE(8,7011)    (CV(IJK), IJK=1,NIJKCVL)
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)    PINDX
      WRITE(8,7011)    PINDX
      WRITE(6,1000)
```

```
      WRITE(8,1000)
C
      GO TO 500
C
C  *****   Neural-Net Optimisation During the Controlled Trajectory Phase   ***
C
  421 ICVDEF = 3
      IECDEF = 2
      CALL CVVCTR(XDUM,JERR)
      IF (JERR .NE. 0) GO TO 996
      DO 422 IJK = 1,NIJKCVC
      CV0(IJK) = CV(IJK)
  422 CONTINUE
C
      IBUG = 422
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
C     WRITE(6,1000)
C     WRITE(8,1000)
C     WRITE(6,7011)   T, TABS, TREL
C     WRITE(8,7011)   T, TABS, TREL
C     WRITE(6,1000)
C     WRITE(8,1000)
C     DO 973 L=1,LMAX
C     WRITE(6,7011)    (XD(I,L), I=1,NL2(1))
C     WRITE(8,7011)    (XD(I,L), I=1,NL2(1))
C     WRITE(6,7011)    (YD(J,L), J=1,NL2(2))
C     WRITE(8,7011)    (YD(J,L), J=1,NL2(2))
C     WRITE(6,1000)
C     WRITE(8,1000)
C 973 CONTINUE
C     WRITE(6,7011)    (CV0(IJK), IJK=1,NIJKCVL)
C     WRITE(8,7011)    (CV0(IJK), IJK=1,NIJKCVL)
C     WRITE(6,1000)
C     WRITE(8,1000)
C
C
C  *****   ICODE is the IMSL Informational Error Code Number   *****
C
C               ICODE = 1  indicates that the Search Direction is Uphill.
C               ICODE = 2  indicates that the Line Search required more
C                          than  5  Function Calls.
C               ICODE = 3  indicates that the Maximum Number of Iterations
C                          were Exceeded.
C               ICODE = 4  indicates that the Search Direction vector is
C                          close to being a Zero vector.
C
      CALL ERSET(0,1,0)
      ICODE = 0
      CALL DNCONF(JNNW,NCONNNC,0,NIJKCVC,CV0,CVBDNNC,CMINNNC,CMAXNNC,
     1 CVSNNC,OUTNNC,MITNNNC,CV,PINDX)
      ICODE = IERCD()
C
      IBUG = 499
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1097) IBUG, IPHASE, ICODE
      WRITE(8,1097) IBUG, IPHASE, ICODE
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)   T, TABS, TREL
      WRITE(8,7011)   T, TABS, TREL
      WRITE(6,1000)
      WRITE(8,1000)
C
C  *****   Determine State from Neural-Net Model   *****
C
```

```
      DO 423 I = 1,NL2(1)
      XN(I) = XD(I,1)
  423 CONTINUE
      CALL STATENN(XN,YN,JERR)
      WRITE(6,7011)   (XN(I), I=1,NL2(1))
      WRITE(8,7011)   (XN(I), I=1,NL2(1))
      WRITE(6,7011)   (YN(J), J=1,NL2(2))
      WRITE(8,7011)   (YN(J), J=1,NL2(2))
      WRITE(6,1000)
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(8,1000)
      DO 974 L=1,LMAX
      WRITE(6,7011)   (XD(I,L), I=1,NL2(1))
      WRITE(8,7011)   (XD(I,L), I=1,NL2(1))
      WRITE(6,7011)   (YD(J,L), J=1,NL2(2))
      WRITE(8,7011)   (YD(J,L), J=1,NL2(2))
      WRITE(6,1000)
      WRITE(8,1000)
  974 CONTINUE
      WRITE(6,7011)   (CV(IJK), IJK=1,NIJKCVL)
      WRITE(8,7011)   (CV(IJK), IJK=1,NIJKCVL)
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)   PINDX
      WRITE(8,7011)   PINDX
      WRITE(6,1000)
      WRITE(8,1000)
C
C
C  *****   CONTROL VECTOR UPDATE   *****
C
C
  500 IBUG = 500
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
C
      IF (IPHASE) 996, 996, 501
  501 GO TO (600,600,600,996,502,502,502,996), IPHASE
  502 IF (CVTID) 511, 511, 503
  503 IF (CVUP) 511, 504, 511
C
C *****   Control Optimisation During the Controlled Trajectory Phase   *****
C
  504 ICVDEF = 5
      IECDEF = 3
      CALL CVVCTR(XD(1,1),JERR)
      IF (JERR .NE. 0) GO TO 996
      DO 505 II = 1,NICV
      CV0(II) = CV(II)
  505 CONTINUE
C
      IBUG = 505
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
C     WRITE(6,1000)
C     WRITE(8,1000)
C     WRITE(6,7011)   T, TABS, TREL
C     WRITE(8,7011)   T, TABS, TREL
C     WRITE(6,1000)
C     WRITE(8,1000)
C     DO 975 L=1,LMAX
C     WRITE(6,7011)   (XD(I,L), I=1,NL2(1))
C     WRITE(8,7011)   (XD(I,L), I=1,NL2(1))
C     WRITE(6,7011)   (YD(J,L), J=1,NL2(2))
```

```
C     WRITE(8,7011)    (YD(J,L), J=1,NL2(2))
C     WRITE(6,1000)
C     WRITE(8,1000)
C 975 CONTINUE
C     WRITE(6,7011)    (CV0(II), II=1,NICV)
C     WRITE(8,7011)    (CV0(II), II=1,NICV)
C     WRITE(6,1000)
C     WRITE(8,1000)
C
C
C  *****    ICODE is the IMSL Informational Error Code Number    *****
C
C                 ICODE = 1  indicates that the Search Direction is Uphill.
C                 ICODE = 2  indicates that the Line Search required more
C                            than  5  Function Calls.
C                 ICODE = 3  indicates that the Maximum Number of Iterations
C                            were Exceeded.
C                 ICODE = 4  indicates that the Search Direction vector is
C                            close to being a Zero vector.
C
      CALL ERSET(0,1,0)
      ICODE = 0
      CALL DNCONF(JCTRL,NCONC,0,NICV,CV0,CVBDC,CMINC,CMAXC,CVSC,OUTC,
     1 MITNC,CV,PINDX)
      ICODE = IERCD()
C
      IBUG = 599
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1097) IBUG, IPHASE, ICODE
      WRITE(8,1097) IBUG, IPHASE, ICODE
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)    T, TABS, TREL
      WRITE(8,7011)    T, TABS, TREL
      WRITE(6,1000)
      WRITE(8,1000)
C
C  *****    Determine State from Neural-Net Model    *****
C
      DO 509 I = 1,NL2(1)
      XN(I) = XD(I,1)
  509 CONTINUE
      CALL STATENN(XN,YN,JERR)
      WRITE(6,7011)    (XN(I), I=1,NL2(1))
      WRITE(8,7011)    (XN(I), I=1,NL2(1))
      WRITE(6,7011)    (YN(J), J=1,NL2(2))
      WRITE(8,7011)    (YN(J), J=1,NL2(2))
      WRITE(6,1000)
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(8,1000)
      DO 976 L=1,LMAX
      WRITE(6,7011)    (XD(I,L), I=1,NL2(1))
      WRITE(8,7011)    (XD(I,L), I=1,NL2(1))
      WRITE(6,7011)    (YD(J,L), J=1,NL2(2))
      WRITE(8,7011)    (YD(J,L), J=1,NL2(2))
      WRITE(6,1000)
      WRITE(8,1000)
  976 CONTINUE
      WRITE(6,7011)    (CV(II), II=1,NICV)
      WRITE(8,7011)    (CV(II), II=1,NICV)
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)    PINDX
      WRITE(8,7011)    PINDX
      WRITE(6,1000)
      WRITE(8,1000)
C
```

```
C
      IF (UPDATE) 511, 511, 506
C
C   *****   Update the First Set (L = 1) of the Sliding Window Table
C           (i.e., XD(I,1) and YD(J,1)) to those values determined by
C           the Current Control Optimisation (i.e., XN(I) and YN(J)).   *****
C
  506 DO 507 I = 1, NL2(1)
      XD(I,1) = XN(I)
  507 CONTINUE
      DO 508 J = 1, NL2(2)
      YD(J,1) = YN(J)
  508 CONTINUE
C
  511 IBUG = 511
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1097) IBUG, IPHASE, ICODE
      WRITE(8,1097) IBUG, IPHASE, ICODE
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)   T, TABS, TREL
      WRITE(8,7011)   T, TABS, TREL
      WRITE(6,1000)
      WRITE(8,1000)
C
C  *****   Determine State from Neural-Net Model   *****
C
      DO 510 I = 1,NL2(1)
      XN(I) = XD(I,1)
  510 CONTINUE
      CALL STATENN(XN,YN,JERR)
      WRITE(6,7011)   (XN(I), I=1,NL2(1))
      WRITE(8,7011)   (XN(I), I=1,NL2(1))
      WRITE(6,7011)   (YN(J), J=1,NL2(2))
      WRITE(8,7011)   (YN(J), J=1,NL2(2))
      WRITE(6,1000)
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(8,1000)
      DO 970 L=1,LMAX
      WRITE(6,7011)   (XD(I,L), I=1,NL2(1))
      WRITE(8,7011)   (XD(I,L), I=1,NL2(1))
      WRITE(6,7011)   (YD(J,L), J=1,NL2(2))
      WRITE(8,7011)   (YD(J,L), J=1,NL2(2))
      WRITE(6,1000)
      WRITE(8,1000)
  970 CONTINUE
      WRITE(6,7011)   (CV(II), II=1,NICV)
      WRITE(8,7011)   (CV(II), II=1,NICV)
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,7011)   PINDX
      WRITE(8,7011)   PINDX
      WRITE(6,1000)
      WRITE(8,1000)
C
      IF (ICUT) 997, 512, 997
  512 IPHASE = 6
C
      IBUG = 512
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1097) IBUG, IPHASE, ICODE
      WRITE(8,1097) IBUG, IPHASE, ICODE
      WRITE(6,1000)
      WRITE(8,1000)
C
      GO TO 602
```

```
C
C   *****    TRAJECTORY PROPAGATION    *****
C
C
  600 IBUG = 600
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
      WRITE(6,1000)
      WRITE(8,1000)
C
      IF (ICUT) 141, 601, 141
  601 IPHASE = 2
C
  602 IBUG = 602
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1099) IBUG, IPHASE
      WRITE(8,1099) IBUG, IPHASE
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1096) IBUG, NNID
      WRITE(8,1096) IBUG, NNID
      WRITE(6,1000)
      WRITE(8,1000)
      WRITE(6,1011) IPHASE, TABS, TREL, PINDX
      WRITE(8,1011) IPHASE, TABS, TREL, PINDX
C
      ISTEP = ISTEP + 1
      IF (NNID) 604, 604, 603
  603 NNUP  = JMOD(ISTEP-1,NNID)
  604 IF (IPHASE) 996, 996, 605
  605 GO TO (608,608,608,996,608,606,608,996), IPHASE
  606 IF (CVTID) 608, 608, 607
  607 CVUP = JMOD(ISTEP-1,CVTID)
  608 T     = T + TSTEP
      TABS  = TABS + TSTEP
      TREL  = TREL + TSTEP
      GO TO 200
C
C   *****    Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****    Normal Exit    *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****    EXIT    *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE JNNW(M,ME,N,X,ACTIVE,F,G)
C
C
C   *****   This subroutine computes the Performance Index PINDX and the
C           constraints CON(III) for Neural-Net Optimisation/Update during
C           both the Learning and Controlled Trajectory Phases.
C
C
C   *****   Start SUBROUTINE JNNW   *****
C
C
C
C   *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C           This file contains the statements which establish and define:
C           1) the Principal COMMON Blocks;   2) the Data TYPE of the
C           Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C           DIMENSION of the Principal  Arrays  and  Vectors  of the
C           OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 I, IARG, J, JERR, L, M, ME, N, NIJKCV
C
      REAL*8   F, G(NCON), SARG, X(NCV), XDUM(NL2DIM), YYA(NL2DIM),
     1         YYN(NL2DIM)
C
      LOGICAL  ACTIVE(NCON)
C
      EXTERNAL CVVCTR, ECVCTR, STATE, STATENN
      REAL*8   CVVCTR, ECVCTR, STATE, STATENN
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(40H0  *****   NORMAL EXIT FROM JNNW   *****//)
 1072 FORMAT(39H0  *****   ERROR EXIT FROM JNNW   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****   Initialisation   *****
C
      JERR   = 0
      IF (IPHASE) 996, 996, 10
   10 GO TO (11,11,11,996,12,12,12,996), IPHASE
   11 ICVDEF = 2
      IECDEF = 1
      NIJKCV = NIJKCVL
      GO TO 13
   12 ICVDEF = 4
      IECDEF = 2
      NIJKCV = NIJKCVC
   13 DO 14 IJK=1,NIJKCV
      CV(IJK) = X(IJK)
   14 CONTINUE

C
C   *****   Unload the Control Vector CV(II)   *****
C
      CALL CVVCTR(XDUM,JERR)
      IF (JERR .NE. 0) GO TO 996
C
C   *****   Determine State from Neural-Net Model   *****
C
      SARG = ZERO
      DO 30 L=1,LMAX
      CALL STATENN(XD(1,L),YYN,JERR)
      IF (JERR .NE. 0) GO TO 996
```

```
      DO 15 J=1,NL2(2)
      YYA(J) = YD(J,L)
   15 CONTINUE
C
C   *****   Load the End Conditions Vextor EC(JJ)    *****
C
      CALL ECVCTR(L,YYA,YYN,JERR)
      IF (JERR .NE. 0) GO TO 996
      SUMSQW(L) = SUMSQ
      GO TO (21,21,21,996,22,22,22,996), IPHASE
   21 SARG = SARG + WTSNNL(L)*SUMSQW(L)
      GO TO 30
   22 SARG = SARG + WTSNNC(L)*SUMSQW(L)
   30 CONTINUE
      SUMSQ = SARG
C
C   *****   Define the Performance Index PINDX    *****
C
      PINDX = SUMSQ
      F     = PINDX
      CON(1) = ZERO
      G(1)   = CON(1)
      GO TO 997
C
C   *****   Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit    *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT    *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
3      SUBROUTINE JCTRL(M,ME,N,X,ACTIVE,F,G)
C
C
C   *****   This subroutine computes the Performance Index PINDX and the
C           constraints CON(III) for Control Optimisation/Update during
C           the Controlled Trajectory Phase.
C
C
C   *****   Start SUBROUTINE JCTRL   *****
C
C
C
C   *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C           This file contains the statements which establish and define:
C           1) the Principal COMMON Blocks;   2) the Data TYPE of the
C           Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C           DIMENSION of the Principal  Arrays  and  Vectors  of the
C           OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 I, IARG, JERR, LDUM, M, ME, N
C
      REAL*8   F, G(NCON), X(NCV), YDUM(NL2DIM)
C
      LOGICAL  ACTIVE(NCON)
C
      EXTERNAL CVVCTR, ECVCTR, STATENN
      REAL*8   CVVCTR, ECVCTR, STATENN
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(41H0  *****   NORMAL EXIT FROM JCTRL   *****//)
 1072 FORMAT(40H0  *****   ERROR EXIT FROM JCTRL   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****   Initialisation   *****
C
      JERR   = 0
      ICVDEF = 6
      IECDEF = 3
      DO 10 II=1,NICV
      CV(II) = X(II)
   10 CONTINUE
      DO 11 I=1,NL2(1)
      XN(I) = XD(I,1)
   11 CONTINUE
C
C   *****   Unload the Control Vector CV(II)   *****
C
      CALL CVVCTR(XN,JERR)
      IF (JERR .NE. 0) GO TO 996
C
C   *****   Determine State from Neural-Net Model   *****
C
      CALL STATENN(XN,YN,JERR)
      IF (JERR .NE. 0) GO TO 996
C
C   *****   Load the End Conditions Vextor EC(JJ)   *****
C
      CALL ECVCTR(LDUM,YDUM,YN,JERR)
      IF (JERR .NE. 0) GO TO 996
C
C   *****   Define the Performance Index PINDX   *****
C
```

```
      PINDX = SUMSQ
      F     = PINDX
      IF (NCONC) 997, 997, 100
C
C   *****   Compute Constraint Vector Function CON(III)   *****
C
  100 III   = 0
C     NCONC = 0
      DO 102 I=1,NL2(1)
      IF (ICONC(I)) 102, 102, 101
  101 III   = III + 1
      IARG = ICONC(I)
      CON(III) = SMAXC(I)*SMAXC(I) - XA(I)*XA(I) - XA(IARG)*XA(IARG)
      G(III)    = CON(III)
  102 CONTINUE
C     NCONC = III
      GO TO 997
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit   *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
```

```
      SUBROUTINE CVVCTR(X,JERR)
C
C
C   *****    This subroutine either Loads the Control Vector FROM the
C            Principal Parameters in the OPTIMNN System if ICVDEF = 1,
C            3, or 5, or Unloads the Control Vector TO the appropriate
C            Principal Parameters in the OPTIMNN System if ICVDEF = 2,
C            4, or 6.
C
C
C   *****    Start SUBROUTINE CVVCTR    *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 I, J, JERR, K, NIJKCV
C
      REAL*8 X(NL2DIM)
C
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(42H0  *****   NORMAL EXIT FROM CVVCTR   *****//)
 1072 FORMAT(41H0  *****   ERROR EXIT FROM CVVCTR   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Initialisation    *****
C
      JERR   = 0
      GO TO (100,100,100,100,200,200,996), ICVDEF
C
C   *****    Load or Unload the Control Vector CV(IJK) during Neural-Net
C            Optimisation/Update.
C
  100 IJK    = 0
      NIJKCV = 0
      DO 130 K = 1,NK
      DO 120 I = 1,NI(K)
      DO 110 J = 1,NJ(K)
      GO TO (101,101,102,102,996,996,996), ICVDEF
  101 IF (IJKCVL(I,J,K)) 110,110,103
  102 IF (IJKCVC(I,J,K)) 110,110,103
  103 IJK = IJK + 1
      GO TO (104,105,104,105,996,996,996), ICVDEF
C
C   *****    Load the Control Vector CV(IJK)    *****
C
  104 CV(IJK) = CW(I,J,K)
      GO TO 110
C
C   *****    Unload the Control Vector CV(IJK)    *****
C
  105 CW(I,J,K) = CV(IJK)
  110 CONTINUE
  120 CONTINUE
  130 CONTINUE
      NIJKCV = IJK
```

```
      GO TO (141,141,142,142,996,996,996), ICVDEF
  141 NIJKCVL = IJK
      GO TO 997
  142 NIJKCVC = IJK
      GO TO 997
C
C   *****   Load or Unload the Control Vector CV(II) during Control
C           Optimisation/Update.
C
  200 II   = 0
      NICV = 0
      DO 210 I = 1,NL2(1)
      IF (ICV(I)) 210, 210, 201
  201 II = II + 1
      GO TO (996,996,996,996,202,203,996), ICVDEF
C
C   *****   Load the Control Vector CV(II)    *****
C
  202 CV(II) = X(I)
      GO TO 210
C
C   *****   Unload the Control Vector CV(II)    *****
C
  203 X(I) = CV(II)
  210 CONTINUE
      NICV = II
      GO TO 997
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit   *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789
C234567890123456789012345678901234567890123456789012345678901234567890123456789
C234567890123456789012345678901234567890123456789012345678901234567890123456789
```

```
      SUBROUTINE ECVCTR(L,YYA,YYN,JERR)
C
C
C   *****   This subroutine Loads the End Conditions Vector FROM the
C           appropriate Principal Parameters in the OPTIMNN System
C           ans Sums the Squares of selected End Conditions to define
C           the Core of the Performance Index if IECDEF = 1, 2, or 3.
C
C
C   *****   Start SUBROUTINE ECVCTR    *****
C
C
C
C   *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C           This file contains the statements which establish and define:
C           1) the Principal COMMON Blocks;  2) the Data TYPE of the
C           Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C           DIMENSION of the Principal  Arrays  and  Vectors  of the
C           OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 J, JERR, L, NJJEC
C
      REAL*8 WT, YYA(NL2DIM), YYN(NL2DIM)
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(42H0  *****   NORMAL EXIT FROM ECVCTR   *****//)
 1072 FORMAT(41H0  *****   ERROR EXIT FROM ECVCTR   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****   Initialisation    *****
C
      JERR   = 0
      GO TO (100,100,200,996), IECDEF
C
C   *****   Load the End Conditions Vector EC(JJJ) during Neural-Net
C           Optimisation/Update.
C
  100 JJJ   = 0
      NJJEC = 0
      SUMSQ = ZERO
      DO 110 J = 1,NL2(2)
      GO TO (101,102,200,996), IECDEF
  101 WT = WTNNL(J)
      IF (JJECL(J)) 110,110,103
  102 WT = WTNNC(J)
      IF (JJECC(J)) 110,110,103
  103 JJJ = JJJ + 1
C
C   *****   Load the End Conditions Vector EC(JJJ)    *****
C
      EC(JJJ) = YYN(J) - YYA(J)
      SUMSQ   = SUMSQ + WT*EC(JJJ)*EC(JJJ)
  110 CONTINUE
      GO TO (111,112,996,996), IECDEF
  111 NJJECL = JJJ
      GO TO 997
  112 NJJECC = JJJ
      GO TO 997
C
C   *****   Load the End Conditions Vector EC(JJ) during Control
C           Optimisation/Update.
C
```

```
  200 JJ   = 0
      NJEC = 0
      SUMSQ = ZERO
      DO 210 J = 1,NL2(2)
      IF (JEC(J)) 210, 210, 201
  201 JJ = JJ + 1
C
C   *****   Load the End Conditions Vector EC(JJ)    *****
C
      EC(JJ) = YYN(J)
      SUMSQ  = SUMSQ + WTC(J)*EC(JJ)*EC(JJ)
  210 CONTINUE
      NJEC = JJ
      GO TO 997
C
C   *****   Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit    *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT    *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
```

```
      SUBROUTINE STATENN(X,Y,JERR)
C
C
C   *****    This subroutine Determines the State as a Function of the
C            Control and the Neural-Net Parameters using the Neural-Net
C            Model.
C
C
C   *****    Start SUBROUTINE STATENN   *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 I, J, JERR, K
C
      REAL*8 X(NL2DIM), Y(NL2DIM)
C
      EXTERNAL PFNCT00, PFNCT01, PFNCT02, PFNCT03
      REAL*8   PFNCT00, PFNCT01, PFNCT02, PFNCT03
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(43H0  *****   NORMAL EXIT FROM STATENN     *****//)
 1072 FORMAT(42H0  *****   ERROR EXIT FROM STATENN    *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Initialisation    *****
C
      JERR   = 0
C
C   *****    Evaluate for Each Layer.
C
      DO 310 K=1,NK
C
C   *****    Determine the Origin Signals for Each Neural-Net Layer,
C            Origin Position, and Destination Position.
C
C   *****    Evaluate for Each Origin Position.
C
      DO 120 I=1,NI(K)
C
C   *****    Evaluate for Each Destination Position.
C
      DO 110 J=1,NJ(K)
C
      IF (K-1) 111,111,112
  111 XNN(I,J,K) = X(I)
      GO TO 110
  112 XNN(I,J,K) = YNN(I,K-1)
C
  110 CONTINUE
C
  120 CONTINUE
C
C   *****    Determine the Destination Signals for Each Neural-Net Layer,
C            Origin Position, and Destination Position.
C
```

```
C   *****   Evaluate for Each Destination Position.
C
      DO 210 J=1,NJ(K)
C
      UNN(J,K) = ZERO
C
C   *****   Evaluate for Each Origin Position.
C
      DO 220 I=1,NI(K)
      UNN(J,K) = UNN(J,K) + CW(I,J,K)*XNN(I,J,K)
  220 CONTINUE
C
C   *****   Input the Destination Signal to the Selected Neural-Net
C           Pass-Through Function (i.e.,  Neural-Net Node Filter).
C
      GO TO (231,232,233,234), NFUNCT(J,K)+1
C
C   *****   The No-Pass (i.e., the Constant Function) Neural-Net Node
C           Filter Function.
C
  231 CALL PFNCT00(J,K,JERR)
      IF (JERR) 996,210,996
C
C   *****   The Direct-Pass (i.e., the Linear Function) Neural-Net Node
C           Filter Function.
C
  232 CALL PFNCT01(J,K,JERR)
      IF (JERR) 996,210,996
C
C   *****   The Hyperbolic Tangent (i.e., the Threshold Function)
C           Neural-Net Node Filter Function.
C
  233 CALL PFNCT02(J,K,JERR)
      IF (JERR) 996,210,996
C
C   *****   The First Derivative of the Hyperbolic Tangent (i.e., the
C           Pulse Function) Neural-Net Node Filter Function.
C
  234 CALL PFNCT03(J,K,JERR)
      IF (JERR) 996,210,996
C
  210 CONTINUE
C
  310 CONTINUE
C
C   *****   Determine the Neural-Net Model Output Vector
C
      DO 410 J=1,NJ(NK)
      Y(J) = YNN(J,NK)
  410 CONTINUE
C
      GO TO 997
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit   *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C   *****   EXIT   *****
C
  999 RETURN
      END
```

```
C23456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
C23456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
C23456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE PFNCT00(J,K,JERR)
C
C
C   *****    This subroutine Defines the No-Pass (i.e., the Constant
C            Function) Neural-Net Pass-Through Function (i.e., Node
C            Filter).
C
C
C   *****    Start SUBROUTINE PFNCT00    *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 J, JERR, K
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(43H0   *****   NORMAL EXIT FROM PFNCT00    *****//)
 1072 FORMAT(42H0   *****   ERROR EXIT FROM PFNCT00    *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Initialisation    *****
C
      JERR   = 0
C
C   *****    Evaluate the Destination Signal to the J-th Destination
C            Position of the K-th Neural-Net Layer.
C
      YNN(J,K) = YN0(J,K) + CN(J,K)
C
      IF (JERR) 996,997,996
C
C   *****    Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****    Normal Exit    *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****    EXIT    *****
C
  999 RETURN
      END

C23456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
C23456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
C23456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE PFNCT01(J,K,JERR)
C
C
C   *****   This subroutine Defines the Direct-Pass (i.e., the Linear
C           Function) Neural-Net Pass-Through Function (i.e., Node Filter).
C
C
C   *****   Start SUBROUTINE PFNCT01    *****
C
C
C
C   *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C           This file contains the statements which establish and define:
C           1) the Principal COMMON Blocks;   2) the Data TYPE of the
C           Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C           DIMENSION of the Principal  Arrays  and  Vectors  of the
C           OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 J, JERR, K
C
      REAL*8  AA, ARG, BB, CC, DD, TMOD, YY
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(43H0  *****   NORMAL EXIT FROM PFNCT01    *****//)
 1072 FORMAT(42H0  *****   ERROR EXIT FROM PFNCT01    *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****   Initialisation    *****
C
      JERR   = 0
C
C   *****   Select Method of Defining Model Constants.    *****
C
      DD = DN(J,K)
      IF (DD+TENP6-TENM2) 100,200,200
C
C   *****   Input Model Constants Directly.    *****
C
  100 AA = AN(J,K)
      CC = CN(J,K)
      GO TO 202
C
C   *****   Define Model Constants from Geometrical Considerations.    *****
C
  200 ARG = CN(J,K) - AN(J,K)
      IF (DABS(ARG)-TENM6) 996,201,201
  201 AA  = (DN(J,K) - BN(J,K))/ARG
      CC = DN(J,K) - YN0(J,K) - AA*(CN(J,K) - XN0(J,K))
C
C   *****   Evaluate the Destination Signal to the J-th Destination
C           Position of the K-th Neural-Net Layer.
C
  202 YNN(J,K) = YN0(J,K) + AA*(UNN(J,K) - XN0(J,K)) + CC
C
      GO TO 997
C
C   *****   Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
```

```
C   *****   Normal Exit   *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
```

```
      SUBROUTINE PFNCT02(J,K,JERR)
C
C
C   *****    This subroutine Defines the Hyperbolic Tangent (i.e., the
C            Threshold Function) Neural-Net Pass-Through Function (i.e.,
C            Node Filter).
C
C
C   *****    Start SUBROUTINE PFNCT02    *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 J, JERR, K
C
      REAL*8 AA, ARG, PT990
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(43H0  *****   NORMAL EXIT FROM PFNCT02    *****//)
 1072 FORMAT(42H0  *****   ERROR EXIT FROM PFNCT02   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Initialisation    *****
C
      JERR   = 0
C
C   *****    Select Method of Defining Model Constants.    *****
C
      IF (BN(J,K)-TENM2) 100,200,200
C
C   *****    Input Model Constants Directly.    *****
C
  100 AA = AN(J,K)
      GO TO 204
C
C   *****    Define Model Constants from Geometrical Considerations.    *****
C
  200 IF (AN(J,K)-TENM2) 996,201,201
  201 PT990 = ONE - TENM2
      IF (PT990-AN(J,K)) 996,202,202
  202 IF (BN(J,K)-TENM2) 996,203,203
  203 ARG = (ONE + AN(J,K))/(ONE - AN(J,K))
      AA  = (PT500/BN(J,K))*DLOG(ARG)
C
C   *****    Function Evaluation    *****
C
  204 YNN(J,K) = YN0(J,K) + CN(J,K)*DTANH(AA*(UNN(J,K)-XN0(J,K)))
      GO TO 997
C
C   *****    Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****    Normal Exit    *****
```

```
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****    EXIT    *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
```

```
      SUBROUTINE PFNCT03(J,K,JERR)
C
C
C   *****    This subroutine Defines the First Derivative of the Hyperbolic
C            Tangent (i.e., the Pulse Function) Neural-Net Pass-Through
C            Function (i.e., Node Filter).
C
C
C   *****    Start SUBROUTINE PFNCT03    *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;  2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 J, JERR, K
C
      REAL*8 AA, ARG, PT990
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(43H0  *****   NORMAL EXIT FROM PFNCT03   *****//)
 1072 FORMAT(42H0  *****   ERROR EXIT FROM PFNCT03   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Initialisation    *****
C
      JERR   = 0
C
C   *****    Select Method of Defining Model Constants.    *****
C
      IF (BN(J,K)) 100,100,200
C
C   *****    Input Model Constants Directly.    *****
C
  100 AA = AN(J,K)
      GO TO 204
C
C   *****    Define Model Constants from Geometrical Considerations.    *****
C
  200 IF (AN(J,K)-TENM2) 996,201,201
  201 PT990 = ONE - TENM2
      IF (PT990-AN(J,K)) 996,202,202
  202 IF (BN(J,K)-TENM2) 996,203,203
  203 ARG = TWO/DSQRT(AN(J,K)) - ONE
      AA  = (PT500/BN(J,K))*DLOG(ARG)
C
C   *****    Function Evaluation    *****
C
  204 ARG = ONE/DCOSH(AA*(UNN(J,K)-XN0(J,K)))
      YNN(J,K) = YN0(J,K) + AA*CN(J,K)*ARG*ARG
      GO TO 997
C
C   *****    Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
```

```
C   *****   Normal Exit   *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C2345678901234567890123456789012345678901234567890123456789012345678901234567890
C2345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE STATE(X,Y,JERR)
C
C
C   *****    This subroutine Determines the "Actual" (i.e., Reference)
C            Plane Model (i.e., Definition of the Control and State as
C            a Function of Time).
C
C
C   *****    Start SUBROUTINE STATE    *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 I, J, JERR, K
C
      REAL*8 X(NL2DIM), Y(NL2DIM)
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(41H0  *****   NORMAL EXIT FROM STATE    *****//)
 1072 FORMAT(40H0  *****   ERROR EXIT FROM STATE    *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Initialisation    *****
C
      JERR   = 0
C
C   *****    Select Source for Control/State Definition.
C
      IF (IPHASE) 996,996,100
  100 GO TO (101,101,101,996,102,102,102,996), IPHASE
  101 GO TO (111,112,113,114), STMODL
  102 GO TO (111,112,113,114), STMODC
C
C   *****    Synthesis the "Actual" (i.e., Reference) Plant Model by
C            Combining Selected Individual Analytic Models.
C
  111 CALL ASTATE(X,Y,JERR)
      IF (JERR) 996,997,996
C
C   *****    Defines the "Actual" (i.e., Reference) Plant Model from
C            On-Line Test Data.
C
  112 CALL DSTATE(X,Y,JERR)
      IF (JERR) 996,997,996
C
C   *****    Defines the "Actual" (i.e., Reference) Plant Model from
C            Stored Data Tables
C
  113 CALL TSTATE(X,Y,JERR)
      IF (JERR) 996,997,996
C
C   *****    Defines the "Actual" (i.e., Reference) Plant Model from
C            a User Supplied Model.
C
  114 CALL USTATE(X,Y,JERR)
      IF (JERR) 996,997,996
```

```
C
      GO TO 997
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit    *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
```

```
      SUBROUTINE ASTATE(X,Y,JERR)
C
C
C   *****   This subroutine Synthesises (i.e., Defines) the "ACTUAL"
C           (i.e., the Reference) Plant Model including both Input and
C           Output Signals by Combining Selected Individual Analytic
C           Models  (i.e., ASTATE01, ASTATE02, ASTATE03, *, *, *, *)
C
C
C   *****   Start SUBROUTINE ASTATE   *****
C
C
C
C   *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C           This file contains the statements which establish and define:
C           1) the Principal COMMON Blocks;   2) the Data TYPE of the
C           Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C           DIMENSION of the Principal  Arrays  and  Vectors  of the
C           OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 IARG, JERR, L1, L2, L3
C
      REAL*8  ARG, X(NL2DIM), Y(NL2DIM), YY
C
      EXTERNAL   ASTATE01, ASTATE02, ASTATE03, ASTATE04, ASTATE05,
     1 ASTATE06, ASTATE07, ASTATRAN
      REAL*8     ASTATE01, ASTATE02, ASTATE03, ASTATE04, ASTATE05,
     1 ASTATE06, ASTATE07, ASTATRAN
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(42H0  *****   NORMAL EXIT FROM ASTATE   *****//)
 1072 FORMAT(41H0  *****   ERROR EXIT FROM ASTATE   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****   Initialisation   *****
C
      JERR = 0
C
C   *****   Evaluate for Both the Plant Input and Plant Output Vectors   *****
C
      DO 310 L1=1,2
C
C   *****   Evaluate for Each Vector Element   *****
C
      IF (L1-2) 373, 371, 996
  371 IF (NNID) 372, 372, 373
  372 CALL STATENN(X,Y,JERR)
      IF (JERR .NE. 0) GO TO 996
      GO TO 310
  373 DO 210 L2=1,NL2(L1)
      ARG = ZERO
      IF (NL3(L2,L1)) 200,200,180
C
C   *****   Evaluate Each Individual Primary Analytic Model   *****
C
  180 DO 190 L3=1,NL3(L2,L1)
C
C   *****   Select the Primary Analytic Model   *****
C
      IARG = IFUNCT(L3,L2,L1) + 1
      GO TO (100,101,102,103,104,105,106,107,996), IARG
C
```

```
C    *****   The Random Uniform Distribution Function    *****
C
  100 CALL ASTATRAN(L3,L2,L1,1,YY,JERR)
      IF (JERR) 996,150,996
C
C    *****   The Linear Function (i.e., the Ramp Function)   *****
C
  101 CALL ASTATE01(L3,L2,L1,YY,JERR)
      IF (JERR) 996,150,996
C
C    *****   The Serpentine Curve Function    *****
C
  102 CALL ASTATE02(L3,L2,L1,YY,JERR)
      IF (JERR) 996,150,996
C
C    *****   The Witch of Agnesi Function    *****
C
  103 CALL ASTATE03(L3,L2,L1,YY,JERR)
      IF (JERR) 996,150,996
C
C    *****   The Inverted Witch of Agnesi Function    *****
C
  104 CALL ASTATE04(L3,L2,L1,YY,JERR)
      IF (JERR) 996,150,996
C
C    *****   The Evneloped Sinusoidal Function    *****
C
  105 CALL ASTATE05(L3,L2,L1,YY,JERR)
      IF (JERR) 996,150,996
C
C    *****   The Hyperbolic Tangent Function (i.e., the Threshold Function)
C
  106 CALL ASTATE06(L3,L2,L1,YY,JERR)
      IF (JERR) 996,150,996
C
C    *****   The First Derivative of the Hyperbolic Tangent Function (i.e.,
C            the Pulse Function)
C
  107 CALL ASTATE07(L3,L2,L1,YY,JERR)
      IF (JERR) 996,150,996
C
C    *****   Randomise the Primary Analytic Function Just Evaluated    *****
C
  150 IF(DABS(A2(L3,L2,L1))-TENM8) 151,151,154
  151 IF(DABS(B2(L3,L2,L1))-TENM8) 152,152,154
  152 IF(DABS(C2(L3,L2,L1))-TENM8) 153,153,154
  153 IF(DABS(D2(L3,L2,L1))-TENM8) 155,155,154
  154 CALL ASTATRAN(L3,L2,L1,2,YY,JERR)
      IF (JERR) 996,155,996
C
C    *****   Sum the Primary Analytic Functions Evaluated To-Date    *****
C
  155 ARG = ARG + YY
C
  190 CONTINUE
C
C    *****   Randomise the Combined Primary Analytic Models to Yield the
C            Final Result.
C
  200 IF(DABS(A3(L2,L1))-TENM8) 201,201,204
  201 IF(DABS(B3(L2,L1))-TENM8) 202,202,204
  202 IF(DABS(C3(L2,L1))-TENM8) 203,203,204
  203 IF(DABS(D3(L2,L1))-TENM8) 205,205,204
  204 CALL ASTATRAN(L3,L2,L1,3,ARG,JERR)
      IF (JERR) 996,205,996
  205 GO TO (206,207), L1
  206 X(L2) = ARG
      GO TO 210
  207 Y(L2) = ARG
```

```
C
  210 CONTINUE
C
  310 CONTINUE
C
      GO TO 997
C
C   *****    Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****    Normal Exit    *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****    EXIT    *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE ASTATRAN(L3,L2,L1,LCALL,YY,JERR)
C
C
C    *****   This subroutine Defines the Uniform Distribution Function
C            which is One of the Individual Analytic Models available
C            to be used in the Synthesis (i.e., the Definition) of the
C            "ACTUAL" (i.e., the Reference) Plant Model including both
C            Input and Output Signals.
C
C
C    *****   Start SUBROUTINE ASTATRAN    *****
C
C
C
C    *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 ISEED, JERR, JSEED, L1, L2, L3, LCALL
C
      REAL*8  AA, ARG1, ARG2, BB, CC, DD, YR, YY
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(44H0   *****   NORMAL EXIT FROM ASTATRAN    *****//)
 1072 FORMAT(43H0   *****   ERROR EXIT FROM ASTATRAN    *****//)
 7011 FORMAT(4D20.7)
C
C
C    *****   Initialisation    *****
C
      JERR = 0
      GO TO (10,20,30,996), LCALL
C
C    *****   The Random Uniform Distribution Function    *****
C
   10 ISEED = ISEED1(L3,L2,L1)
C     JSEED = JSEED1(L3,L2,L1)
      AA    = A1(L3,L2,L1)
      BB    = B1(L3,L2,L1)
C     CC    = C1(L3,L2,L1)
C     DD    = D1(L3,L2,L1)
      YR    = YR1(L3,L2,L1)
      GO TO 103
   20 ISEED = ISEED2(L3,L2,L1)
      JSEED = JSEED2(L3,L2,L1)
      AA    = A2(L3,L2,L1)
      BB    = B2(L3,L2,L1)
      CC    = C2(L3,L2,L1)
      DD    = D2(L3,L2,L1)
      YR    = YR2(L3,L2,L1)
      GO TO 100
   30 ISEED = ISEED3(L2,L1)
      JSEED = JSEED3(L2,L1)
      AA    = A3(L2,L1)
      BB    = B3(L2,L1)
      CC    = C3(L2,L1)
      DD    = D3(L2,L1)
      YR    = YR3(L2,L1)
C
C    *****   Determine ARG2    *****
```

```
C
  100 IF (TENM6-DABS(YY)) 103,101,101
  101 IF (TENM6-DABS(DD)) 104,102,102
  102 IF (JSEED) 104,104,105
  103 ARG1 = ZERO
      GO TO 200
  104 ARG2 = CC
      GO TO 200
  105 ARG2 = CC + DD*(TWO*RAN(JSEED) - ONE)*YY
C
C   *****   Determine ARG1   *****
C
  200 IF (TENM6-DABS(BB)) 202,201,201
  201 IF (ISEED) 202,202,203
  202 ARG1 = AA
      GO TO 300
  203 ARG1 = AA + BB*(TWO*RAN(ISEED) - ONE)
C
C   *****   Determine YY   *****
C
  300 YY = YR + ARG1 + ARG2
C
      GO TO 997
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit   *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C2345678901234567890123456789012345678901234567890123456789012345678901234567890
C2345678901234567890123456789012345678901234567890123456789012345678901234567890
C2345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE ASTATE01(L3,L2,L1,YY,JERR)
C
C
C   *****    This subroutine Defines the Linear i.e., the Ramp Function)
C            Function which is One of the Individual Analytic Models
C            available to be used in the Synthesis (i.e., the Definition)
C            of the "ACTUAL" (i.e., the Reference) Plant Model including
C            both Input and Output Signals.
C
C
C   *****    Start SUBROUTINE ASTATE01    *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 JERR, L1, L2, L3
C
      REAL*8  AA, ARG, BB, CC, DD, TMOD, YY
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(44H0  *****   NORMAL EXIT FROM ASTATE01   *****//)
 1072 FORMAT(43H0  *****   ERROR EXIT FROM ASTATE01   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Initialisation    *****
C
      JERR = 0
      ARG  = T - X0(L3,L2,L1) - PHASE(L3,L2,L1)
      IF(PERIOD(L3,L2,L1)-TENM6) 996,996,11
   11 IF(PERIOD(L3,L2,L1)-TENP6) 13,12,12
   12 TMOD = ARG
      GO TO 14
   13 TMOD = DMOD(ARG,PERIOD(L3,L2,L1))
C
C   *****    Select Method of Defining Model Constants.    *****
C
   14 DD = D(L3,L2,L1)
      IF (DD+TENP6-TENM2) 100,200,200
C
C   *****    Input Model Constants Directly.    *****
C
  100 AA = A(L3,L2,L1)
      CC = C(L3,L2,L1)
      GO TO 202
C
C   *****    Define Model Constants from the Co-ordinates of Two Points.    *****
C
  200 ARG = C(L3,L2,L1) - A(L3,L2,L1)
      IF (DABS(ARG)-TENM6) 996,201,201
  201 AA  = (D(L3,L2,L1) - B(L3,L2,L1))/ARG
      CC = D(L3,L2,L1) - Y0(L3,L2,L1) - AA*(C(L3,L2,L1) - X0(L3,L2,L1))
C
C   *****    Function Evaluation    *****
C
  202 YY = Y0(L3,L2,L1) + AA*(TMOD - X0(L3,L2,L1)) + CC
      GO TO 997
```

```
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit   *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789 0
C2345678901234567890123456789012345678901234567890123456789012345678901234567890
C2345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE ASTATE02(L3,L2,L1,YY,JERR)
C
C
C   *****    This subroutine Defines the Serpentine Curve Function which
C            is One of the Individual Analytic Models available to be used
C            in the Synthesis (i.e., the Definition) of the "ACTUAL" (i.e.,
C            the Reference) Plant Model including both Input and Output
C            Signals.
C
C
C   *****    Start SUBROUTINE ASTATE02   *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;  2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 JERR, L1, L2, L3
C
      REAL*8  AA, ARG, BB, TMOD, YY
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(44H0  *****   NORMAL EXIT FROM ASTATE02   *****//)
 1072 FORMAT(43H0  *****   ERROR EXIT FROM ASTATE02   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****   Initialisation   *****
C
      JERR = 0
      ARG  = T - X0(L3,L2,L1) - PHASE(L3,L2,L1)
      IF(PERIOD(L3,L2,L1)-TENM6) 996,996,11
   11 IF(PERIOD(L3,L2,L1)-TENP6) 13,12,12
   12 TMOD = ARG
      GO TO 14
   13 TMOD = DMOD(ARG,PERIOD(L3,L2,L1))
C
   14 IF (DABS(A(L3,L2,L1))-TENM6) 996,996,15
   15 IF (DABS(B(L3,L2,L1))-TENM6) 996,996,100
C
C   *****   Input Model Constants Directly.   *****
C
  100 AA = A(L3,L2,L1)
      BB = B(L3,L2,L1)
C
C   *****   Function Evaluation   *****
C
      YY = Y0(L3,L2,L1) + AA*BB*TMOD/(AA*AA + TMOD*TMOD)
      GO TO 997
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit   *****
C
  997 CONTINUE
```

```
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C *****    EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789
C234567890123456789012345678901234567890123456789012345678901234567890123456789
C234567890123456789012345678901234567890123456789012345678901234567890123456789
```

```
      SUBROUTINE ASTATE03(L3,L2,L1,YY,JERR)
C
C
C   *****   This subroutine Defines the Witch of Agnesi Function which
C           is One of the Individual Analytic Models available to be used
C           in the Synthesis (i.e., the Definition) of the "ACTUAL" (i.e.,
C           the Reference) Plant Model including both Input and Output
C           Signals.
C
C
C   *****   Start SUBROUTINE ASTATE03    *****
C
C
C
C   *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C           This file contains the statements which establish and define:
C           1) the Principal COMMON Blocks;   2) the Data TYPE of the
C           Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C           DIMENSION of the Principal  Arrays  and  Vectors  of the
C           OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 JERR, L1, L2, L3
C
      REAL*8  AA, ARG, BB, CC, PT990, TMOD, YY
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(44H0  *****   NORMAL EXIT FROM ASTATE03    *****//)
 1072 FORMAT(43H0  *****   ERROR EXIT FROM ASTATE03    *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****   Initialisation   *****
C
      JERR = 0
      ARG  = T - X0(L3,L2,L1) - PHASE(L3,L2,L1)
      IF(PERIOD(L3,L2,L1)-TENM6) 996,996,11
   11 IF(PERIOD(L3,L2,L1)-TENP6) 13,12,12
   12 TMOD = ARG
      GO TO 14
   13 TMOD = DMOD(ARG,PERIOD(L3,L2,L1))
C
C   *****   Select Method of Defining Model Constants.   *****
C
   14 AA = A(L3,L2,L1)
      CC = C(L3,L2,L1)
      IF (CC-TENM2) 100,100,200
C
C   *****   Input Model Constants Directly.   *****
C
  100 BB = B(L3,L2,L1)
      GO TO 202
C
C   *****   Define Model Constants from Geometrical Considerations.   *****
C
  200 PT990 = ONE - TENM2
      IF (PT990-CC) 996,201,201
  201 BB  = DSQRT((ONE - CC)/CC)
C
C   *****   Function Evaluation   *****
C
  202 YY = Y0(L3,L2,L1) + AA*AA*AA/(BB*BB*TMOD*TMOD + AA*AA)
      GO TO 997
C
```

```
C    *****    Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C    *****    Normal Exit    *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****    EXIT    *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
```

```
      SUBROUTINE ASTATE04(L3,L2,L1,YY,JERR)
C
C
C   *****    This subroutine Defines the Inverted Witch of Agnesi Function
C            which is One of the Individual Analytic Models available to be
C            used in the Synthesis (i.e., the Definition) of the "ACTUAL"
C            (i.e., the Reference) Plant Model including both Input and
C            Output Signals.
C
C
C   *****    Start SUBROUTINE ASTATE04    *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 JERR, L1, L2, L3
C
      REAL*8  AA, ARG, BB, CC, PT990, TMOD, YY
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(44H0  *****   NORMAL EXIT FROM ASTATE04   *****//)
 1072 FORMAT(43H0  *****   ERROR EXIT FROM ASTATE04   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Initialisation    *****
C
      JERR = 0
      ARG  = T - X0(L3,L2,L1) - PHASE(L3,L2,L1)
      IF(PERIOD(L3,L2,L1)-TENM6) 996,996,11
   11 IF(PERIOD(L3,L2,L1)-TENP6) 13,12,12
   12 TMOD = ARG
      GO TO 14
   13 TMOD = DMOD(ARG,PERIOD(L3,L2,L1))
C
C   *****    Select Method of Defining Model Constants.    *****
C
   14 AA = A(L3,L2,L1)
      CC = C(L3,L2,L1)
      IF (CC-TENM2) 100,100,200
C
C   *****    Input Model Constants Directly.    *****
C
  100 BB = B(L3,L2,L1)
      GO TO 202
C
C   *****    Define Model Constants from Geometrical Considerations.    *****
C
  200 PT990 = ONE - TENM2
      IF (PT990-CC) 996,201,201
  201 BB  = DSQRT(CC/(ONE - CC))
C
C   *****    Function Evaluation    *****
C
  202 YY = Y0(L3,L2,L1) + AA*(ONE - AA*AA/(BB*BB*TMOD*TMOD + AA*AA))
      GO TO 997
C
```

```
C   *****   Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit    *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
C234567890123456789012345678901234567890123456789012345678901234567890123456789O
```

```
      SUBROUTINE ASTATE05(L3,L2,L1,YY,JERR)
C
C
C   *****   This subroutine Defines the Enveloped Sinusoidal Function which
C           is One of the Individual Analytic Models available to be used
C           in the Synthesis (i.e., the Definition) of the "ACTUAL" (i.e.,
C           the Reference) Plant Model including both Input and Output
C           Signals.
C
C
C   *****   Start SUBROUTINE ASTATE05    *****
C
C
C
C   *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C           This file contains the statements which establish and define:
C           1) the Principal COMMON Blocks;   2) the Data TYPE of the
C           Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C           DIMENSION of the Principal  Arrays  and  Vectors  of the
C           OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 JERR, L1, L2, L3
C
      REAL*8  AA, ALP, ARG, ARG1, ARG2, BB, CC, NW, TCOS, TEXP, TMOD, YY
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(44H0  *****   NORMAL EXIT FROM ASTATE05    *****//)
 1072 FORMAT(43H0  *****   ERROR EXIT FROM ASTATE05    *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****   Initialisation   *****
C
      JERR = 0
      ARG  = T - X0(L3,L2,L1) - PHASE(L3,L2,L1)
      IF(PERIOD(L3,L2,L1)-TENM6) 996,996,11
   11 IF(PERIOD(L3,L2,L1)-TENP6) 13,12,12
   12 TMOD = ARG
      GO TO 100
   13 TMOD = DMOD(ARG,PERIOD(L3,L2,L1))
C
C   *****   Evaluation of the Exponential Part (i.e., ARG1) of the
C           Enveloped Sinusoidal Function.
C
  100 TEXP = TMOD - PSI(L3,L2,L1)
      AA = A(L3,L2,L1)
      BB = B(L3,L2,L1)
      CC = C(L3,L2,L1)
      IF (DABS(AA)-TENM6) 101,101,110
C
C   *****   Input Model Constants (i.e., ALPHA(L3,L2,L1)) Directly.   *****
C
  101 ALP = ALPHA(L3,L2,L1)
      GO TO 113
C
C   *****   Define Model Constants from Geometrical Considerations.   *****
C
  110 IF(DABS(BB)-TENM6) 996,996,111
  111 IF(DABS(CC)-TENM6) 996,996,112
  112 ARG = DABS(BB/CC)
      ALP = (DLOG(ARG))/AA
  113 IF(DABS(ALP)-TENM6) 115,115,114
  114 ARG1 = CC*DEXP(ALP*TEXP)
```

```
      GO TO 200
  115 ARG1 = CC
C
C   *****   Evaluation of the Sinusoidal Part (i.e., ARG2) of the
C           Enveloped Sinusoidal Function.
C
  200 TCOS = TMOD - PHI(L3,L2,L1)

      IF (NN(L3,L2,L1)-TENP8) 201,203,203
C
C   *****   Input the Harmonic Number [NN(L3,L2,L1)] and Two-Pi times the
C           Primary Frequency [OMEGA(L3,L2,L1)] Directly.
C
  201 NW = NN(L3,L2,L1)*OMEGA(L3,L2,L1)
      IF (NW-TENM8) 205,205,202
  202 IF (NW-TENP8) 207,205,205
C
C   *****   Input Sinusoidal Period [OMEGA(L3,L2,L1)] Directly.   *****
C
  203 IF (OMEGA(L3,L2,L1)-TENP8) 204,205,205
  204 IF (OMEGA(L3,L2,L1)-TENM8) 996,996,206
  205 ARG2 = ONE
      GO TO 300
  206 NW = TWOPI/OMEGA(L3,L2,L1)
C
C   *****   Evaluation of the Sinusoidal Part (i.e., ARG2) of the
C           Enveloped Sinusoidal Function.
C
  207 ARG2 = DCOS(NW*TCOS)
C
C   *****   Function Evaluation   *****
C
  300 YY = Y0(L3,L2,L1) + ARG1*ARG2
      GO TO 997
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit   *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE ASTATE06(L3,L2,L1,YY,JERR)
C
C
C    *****   This subroutine Defines the Hyperbolic Tangent (i.e., the
C            Threshold Function) Function which is One of the Individual
C            Analytic Models available to be used in the Synthesis (i.e.,
C            the Definition) of the "ACTUAL" (i.e., the Reference) Plant
C            Model including both Input and Output Signals.
C
C
C    *****   Start SUBROUTINE ASTATE06    *****
C
C
C
C    *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 JERR, L1, L2, L3
C
      REAL*8  AA, ARG, PT990, TMOD, YY
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(44H0  *****   NORMAL EXIT FROM ASTATE06    *****//)
 1072 FORMAT(43H0  *****   ERROR EXIT FROM ASTATE06   *****//)
 7011 FORMAT(4D20.7)
C
C
C    *****   Initialisation   *****
C
      JERR = 0
      ARG  = T - X0(L3,L2,L1) - PHASE(L3,L2,L1)
      IF(PERIOD(L3,L2,L1)-TENM6) 996,996,11
   11 IF(PERIOD(L3,L2,L1)-TENP6) 13,12,12
   12 TMOD = ARG
      GO TO 14
   13 TMOD = DMOD(ARG,PERIOD(L3,L2,L1))
C
C    *****   Select Method of Defining Model Constants.   *****
C
   14 IF (B(L3,L2,L1)-TENM2) 100,200,200
C
C    *****   Input Model Constants Directly.   *****
C
  100 AA = A(L3,L2,L1)
      GO TO 204
C
C    *****   Define Model Constants from Geometrical Considerations.   *****
C
  200 IF (A(L3,L2,L1)-TENM2) 996,201,201
  201 PT990 = ONE - TENM2
      IF (PT990-A(L3,L2,L1)) 996,202,202
  202 IF (B(L3,L2,L1)-TENM2) 996,203,203
  203 ARG = (ONE + A(L3,L2,L1))/(ONE - A(L3,L2,L1))
      AA  = (PT500/B(L3,L2,L1))*DLOG(ARG)
C
C    *****   Function Evaluation   *****
C
  204 YY = Y0(L3,L2,L1) + C(L3,L2,L1)*DTANH(AA*TMOD)
      GO TO 997
```

```
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit   *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE ASTATE07(L3,L2,L1,YY,JERR)
C
C
C   *****    This subroutine Defines the First Derivative of the Hyperbolic
C            Tangent (i.e., the Pulse Function) Function which is One of
C            the Individual Analytic Models available to be used in the
C            Synthesis (i.e., the Definition) of the "ACTUAL" (i.e., the
C            Reference) Plant Model including both Input and Output Signals.
C
C
C   *****    Start SUBROUTINE ASTATE07    *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 JERR, L1, L2, L3
C
      REAL*8  AA, ARG, PT990, TMOD, YY
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(44H0  *****   NORMAL EXIT FROM ASTATE07   *****//)
 1072 FORMAT(43H0  *****   ERROR EXIT FROM ASTATE07   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Initialisation    *****
C
      JERR = 0
      ARG  = T - X0(L3,L2,L1) - PHASE(L3,L2,L1)
      IF(PERIOD(L3,L2,L1)-TENM6) 996,996,11
   11 IF(PERIOD(L3,L2,L1)-TENP6) 13,12,12
   12 TMOD = ARG
      GO TO 14
   13 TMOD = DMOD(ARG,PERIOD(L3,L2,L1))
C
C   *****    Select Method of Defining Model Constants.    *****
C
   14 IF (B(L3,L2,L1)) 100,100,200
C
C   *****    Input Model Constants Directly.    *****
C
  100 AA = A(L3,L2,L1)
      GO TO 204
C
C   *****    Define Model Constants from Geometrical Considerations.    *****
C
  200 IF (A(L3,L2,L1)-TENM2) 996,201,201
  201 PT990 = ONE - TENM2
      IF (PT990-A(L3,L2,L1)) 996,202,202
  202 IF (B(L3,L2,L1)-TENM2) 996,203,203
  203 ARG = TWO/DSQRT(A(L3,L2,L1)) - ONE
      AA  = (PT500/B(L3,L2,L1))*DLOG(ARG)
C
C   *****    Function Evaluation    *****
C
  204 ARG = ONE/DCOSH(AA*TMOD)
      YY  = Y0(L3,L2,L1) + AA*C(L3,L2,L1)*ARG*ARG
```

```
      GO TO 997
C
C   *****   Error Exit   *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****   Normal Exit   *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789
C234567890123456789012345678901234567890123456789012345678901234567890123456789
C234567890123456789012345678901234567890123456789012345678901234567890123456789
```

```
      SUBROUTINE DSTATE(X,Y,JERR)
C
C
C   *****    This subroutine Defines the "ACTUAL" (i.e., the Reference)
C            Plant Model including both Input and Output Signals from
C            On-Line Test Data.
C
C
C   *****    Start SUBROUTINE DSTATE    *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 JERR
C
      REAL*8  X(NL2DIM), Y(NL2DIM)
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(42H0  *****   NORMAL EXIT FROM DSTATE    *****//)
 1072 FORMAT(41H0  *****   ERROR EXIT FROM DSTATE   *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Initialisation    *****
C
C   *****    Subroutine DSTATE has NOT been defined yet.
C
      IF (JERR) 996,997,996
C
C   *****    Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****    Normal Exit    *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****   EXIT   *****
C
  999 RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890
C234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE TSTATE(X,Y,JERR)
C
C
C   *****   This subroutine Defines the "ACTUAL" (i.e., the Reference)
C           Plant Model including both Input and Output Signals from a
C           Stored Data Table.
C
C
C   *****   Start SUBROUTINE TSTATE   *****
C
C
C
C   *****   The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C           This file contains the statements which establish and define:
C           1) the Principal COMMON Blocks;  2) the Data TYPE of the
C           Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C           DIMENSION of the Principal  Arrays  and  Vectors  of the
C           OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 JERR, L1, L2
C
      REAL*8  X(NL2DIM), Y(NL2DIM)
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(42H0  *****   NORMAL EXIT FROM TSTATE   *****//)
 1072 FORMAT(41H0  *****   ERROR EXIT FROM TSTATE   *****//)
 1073 FORMAT(66H0  *****   ERROR EXIT FROM TSTATE WHEN THE MAXIMUM NUMBE
     1R OF TABLE/11X,66H VALUES DEFINED BY  "TBLMAX"  IS EXCEEDED.
     2                  *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****   Initialisation   *****
C
      JERR = 0
      LTBL = ISTEP
      IF (LTBL-TBLMAX) 10, 10, 995
   10 T = TTBL(LTBL)
C
C   *****   Evaluate for Both the Plant Input and Plant Output Vectors   *****
C
      DO 310 L1=1,2
C
C   *****   Evaluate for Each Vector Element   *****
C
      IF (L1-2) 373, 371, 996
  371 IF (NNID) 372, 372, 373
  372 CALL STATENN(X,Y,JERR)
      IF (JERR .NE. 0) GO TO 996
      GO TO 310
  373 DO 210 L2=1,NL2(L1)
      GO TO (201,202), L1
  201 X(L2) = XTBL(L2,LTBL)
      GO TO 203
  202 Y(L2) = YTBL(L2,LTBL)
  203 IF (JERR) 996,210,996
C
  210 CONTINUE
C
  310 CONTINUE
C
C   *****   Error Exit when the Maximum Number of Table Values defined
C           by "TBLMAX" is Exceeded.   *****
```

```
C
      GO TO 997
  995 JERR = 1
      WRITE(6,1073)
      WRITE(8,1073)
      GO TO 999
C
C   *****    Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****    Normal Exit    *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****    EXIT    *****
C
  999 RETURN
      END

C2345678901234567890123456789012345678901234567890123456789012345678901234567890
C2345678901234567890123456789012345678901234567890123456789012345678901234567890
C2345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
      SUBROUTINE USTATE(X,Y,JERR)
C
C
C   *****    This subroutine Defines the "ACTUAL" (i.e., the Reference)
C            Plant Model including both Input and Output Signals from a
C            User Supplied Model.
C
C
C   *****    Start SUBROUTINE USTATE    *****
C
C
C
C   *****    The  "[LEYLAND.OPTIMNN]TYPECOM.INC"  File is Included here.
C            This file contains the statements which establish and define:
C            1) the Principal COMMON Blocks;   2) the Data TYPE of the
C            Principal  Parameters,  Arrays,  and Vectors;   and 3) the
C            DIMENSION of the Principal  Arrays  and  Vectors  of the
C            OPTIMNN System.
C
      INCLUDE '[LEYLAND.OPTIMNN]TYPECOM.INC'
C
C
C
      INTEGER*4 JERR
C
      REAL*8  X(NL2DIM), Y(NL2DIM)
C
C
 1000 FORMAT(2H0 )
 1001 FORMAT(2H1 )
 1071 FORMAT(42H0  *****   NORMAL EXIT FROM USTATE    *****//)
 1072 FORMAT(41H0  *****   ERROR EXIT FROM USTATE    *****//)
 7011 FORMAT(4D20.7)
C
C
C   *****    Initialisation    *****
C
C   *****    Subroutine USTATE has NOT been defined yet.
C
      IF (JERR) 996,997,996
C
C   *****    Error Exit    *****
C
  996 WRITE(6,1072)
      WRITE(8,1072)
      GO TO 999
C
C   *****    Normal Exit    *****
C
  997 CONTINUE
C     WRITE(6,1071)
C     WRITE(8,1071)
C
C  *****    EXIT    *****
C
  999 RETURN
      END

C23456789012345678901234567890123456789012345678901234567890123456789012345678900
C23456789012345678901234567890123456789012345678901234567890123456789012345678900
C23456789012345678901234567890123456789012345678901234567890123456789012345678900
```

REPORT DOCUMENTION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2001	Technical Memorandum

4. TITLE AND SUBTITLE

A Closed-Loop Optimal Neural-Network Controller to Optimise Rotorcraft Aeromechanical Behaviour: Volume 2, Output from Two Sample Cases

5. FUNDING NUMBERS

712-10-12

6. AUTHOR(S)

Jane Anne Leyland

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ames Research Centre
Moffett Field, CA 94035-1000

8. PERFORMING ORGANIZATION REPORT NUMBER

A-00V0033

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA TM-2001-209623

11. SUPPLEMENTARY NOTES

Point of Contact: Jane Anne Leyland, Ames Research Centre, MS T12-B, Moffett Field, CA 94035-1000
Tel No (650) 604-4750

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified --- Unlimited Distribution---Standard
Subject Category 08

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A closed-loop optimal neural-network controller technique was developed to optimise rotorcraft aeromechanical behaviour. This technique utilises a neural-network scheme to provide a general non-linear model of the rotorcraft. A modern constrained optimisation method is used to determine and update the constants in the neural-network plant model as well as to determine the optimal control vector. Current data is read, weighted, and added to a sliding data window. When the specified maximum number of data sets allowed in the data window is exceeded, the oldest data set is purged and the remaining data sets are re-weighted. This procedure provides at least four additional degrees-of-freedom in addition to the size and geometry of the neural-network itself with which to optimise the overall operation of the controller. These additional degrees-of-freedom are: 1. the maximum length of the sliding data window, 2. the frequency of neural-network updates, 3. the weighting of the individual data sets within the sliding window, and 4. the maximum number of optimisation iterations used for the neural-network updates.

The output of two sample cases using this technique are presented in this volume, Volume 2.

14. SUBJECT TERMS

Optimal Neural-Network Controller, Optimal Closed-Loop Controller, Neural-Network Controller

15. NUMBER OF PAGES

268

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		