Appendix A

Survey of non-linear system identification models

Linear system identification boasts with a host of widely applicable, well tried techniques, however the identification of non-linear systems has not received such attention or exposure. This can of course be attributed to the inherent complexity of non-linear systems, and the difficulty of deriving identification algorithms that can be applied to a reasonably large class on non-linear systems. This survey presents non-linear model formulations, extracted from published literature, for possible application in response reconstruction. The available models are presented in three groups, functional series methods, block-orientated methods and finally input-output model descriptions.

A.1. Functional series methods

In a survey of non-linear system identification techniques, Billings [5] indicated that functional series all stem from the analytic funtionals introduced by Volterra in 1887. Among these methods are the two formulations postulated by Weiner and the Volterra-series methods. Volterra's functional series can be represented as:

$$y(t) = \sum_{n=1}^{\infty} \int \dots \int h_n(\tau_1, \tau_2, \dots, \tau_n) \prod_{i=1}^{n} u(t - \tau_i) d\tau_i$$
$$= \sum_{n=1}^{\infty} w_i(t) \qquad \text{(A-1)}$$

The functions $h_i(\tau_1, \tau_2, \dots, \tau_i)$ are known as the Volterra kernels that form the basis of the identification of non-linear systems represented by functional series.

A.1.1. Weiner methods

One of the first authors to consider non-linear system identification was Weiner who devised two distinct approaches using functional series methods. In theory, these methods are functionally elegant, but Billings [5] indicated Weiner methods to be impractical due to the excessive number of coefficients required. Further discussions on the Weiner methods are not included in this study, as it holds no practical use in any applications.

A.1.2. Volterra-series methods

Consider a system that can be described by just the first two Volterra kernels [5][6]:

$$y(t) = \int_0^\infty h_1(\tau_1)u(t-\tau_1)d\tau_1 + \int_0^\infty\int h2(\tau_1,\tau_2)u(t-\tau_1)u(t-\tau_2)d\tau_1 d\tau_2 \qquad \text{(A-2)}$$

Many methods of identifying the kernels can be found [5][8][9]. These methods however require extensive data and great computational effort. Due to the complexity of the models, the first two kernels as shown above represent the largest practical model of this type. To specify the first two Volterra kernels for a simple quadratic non-linearity in cascade with a first order linear system Billings and Voon [8] predict 400-500 coefficients would be needed. Again the model would prove to be impractical in actual system identification techniques.

A.2. Block-orientated systems

A number of systems consisting of interconnections of linear dynamic systems and non-linear static elements have been formulated to reduce the complexity and computational effort involved with the functional-series methods. One of the most studied models, known as the General model, consists of a linear system followed by a non-linear element in cascade (Figure. A.1). Examples of such models are the Hammerstein model and the Urysons's model, shown in Figure A.2 (See Billings [5]). For the block oriented models, $h(t)$ represents the dynamic linear part, while $H(\bullet)$ or $F(\bullet)$ represents the non-linear part. Schematically such models may be represented as [55]:

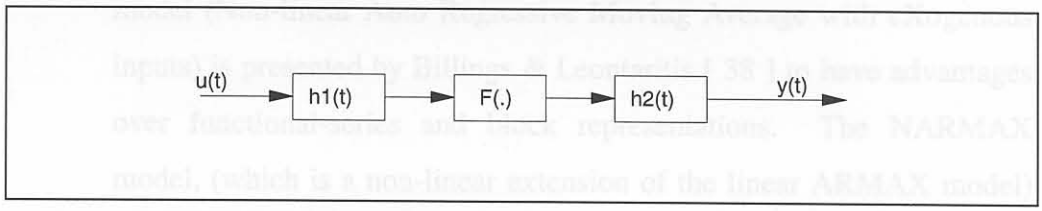

Figure. A.1The General Model

Billings [5] indicated that the Hammerstein model is found by omitting $h1(t)$ from the general model. It then represents a realisation of the Hammerstein operator, and Urysons's model consists of several Hammerstein models in parallel.

$$H_h[u(t)] = \int h(t,\tau)F[\tau,u(\tau)]d\tau \qquad \text{(A-3)}$$

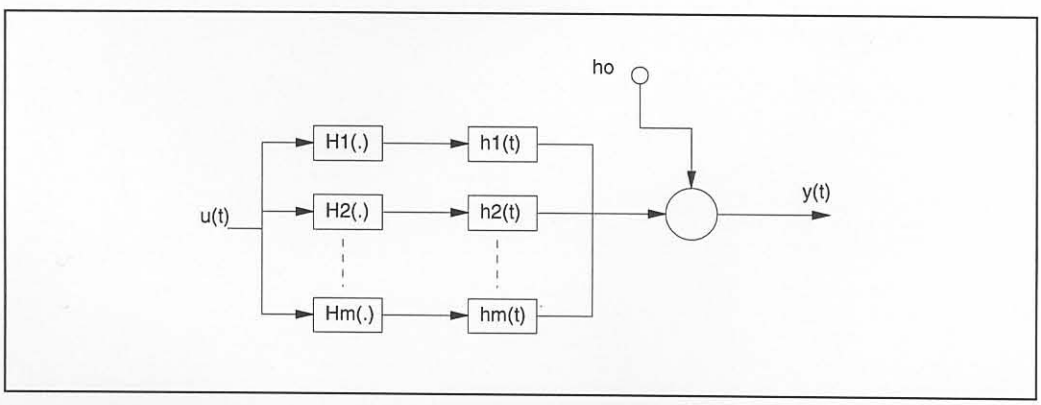

Figure A.2 Schematic representation of Urysons's model

A.3. Input - output model descriptions

Billings [5] and Sales [55] indicated that input-output model descriptions are generally applicable to systems of which little or no *a priori* information is available. Furthermore, these systems can be represented by extensions of linear models and thus form linear-in-the-parameters models. Systems of this kind form the focus of this study, to enable a general non-linear system identification technique for multiple-input, multiple-output (MIMO) systems. A special case of the input/output model description is the so-called NARMAX model.

A.3.1. The NARMAX model

The non-linear difference equation model known as the NARMAX model (Non-linear Auto Regressive Moving Average with eXogenous inputs) is presented by Billings & Leontaritis [38] to have advantages over functional-series and block representations. The NARMAX model, (which is a non-linear extension of the linear ARMAX model) may be represented as:

$$y(t) = F^{L}[y(k-1),\ldots,y(k-na),u(t),\ldots,u(t-nb),e(t-1),\ldots,e(t-nc)] + e(t) \qquad \text{(A-4)}$$

With $F^{L}[\bullet]$ some non-linear polynomial function, normally the degree of the polynomial, L=1,2,3. Various special cases of the NARMAX model have been identified, some of which are presented in the next Sections.

A.3.2. Bilinear model

The bilinear input-output model takes the following form [18]:

$$y(t) = a_0 + \sum_{i=1}^{ny} a_i \cdot y(t-i) + \sum_{i=1}^{nu} b_i \cdot u(t-i) + \sum_{i=1}^{ny}\sum_{j=1}^{nu} c_{ij} \cdot y(t-i) \cdot u(t-j) \qquad \text{(A-5)}$$

In state space formulation the model representation is:

$$\begin{aligned} x(t+1) &= A \cdot x(t) + B \cdot u(t) + u(t) \cdot C \cdot x(t) \\ y(t) &= D \cdot x(t) \end{aligned} \qquad \text{(A-6)}$$

where: $x(t)$= state vector

A = state matrix

B = input matrix

C = output matrix

D = direct transmission matrix

According to Chen & Billings [18] it is however impossible to approximate all discrete-time systems within the class of discrete-time bilinear systems.

A.3.3. Output-Affine and rational models

Chen *et al.* [18] presented rational and subsequently output-affine models for identification of non-linear dynamic systems. The rational model with polynomial order r and finite polynomials $a(\bullet)$ and $b(\bullet)$ may be written as:

$$y(t) = \frac{b \cdot \left(y(t-1), \cdots, y(t-r), u(t-1), \cdots, u(t-r)\right)}{a \cdot \left(y(t-1), \cdots, y(t-r), u(t-1), \cdots, u(t-r)\right)} \qquad \text{(A-7)}$$

The output affine model is a logical expansion of the rational model:

$$y(t) = \sum_{i=1}^{r} \frac{a_i \cdot \left(u(t-1), \cdots, u(t-r)\right)}{a_0 \cdot \left(u(t-1), \cdots, u(t-r)\right)} \cdot y(t-i) + \frac{a_{r+1} \cdot \left(u(t-1), \cdots, u(t-r)\right)}{a_0 \cdot \left(u(t-1), \cdots, u(t-r)\right)} \qquad \text{(A-8)}$$

These two models are globally valid, however, the response function is restricted to polynomial response.

A.3.4. The NARX-model

The NARMAX model may be reduced to NARX (Non-linear Auto Regressive with eXogenous input) by removal of the noise model and moving average terms (See Billings [5] and Peyton Jones [46][47]). NARX is a parametric difference equation that forms a convenient linear-in-the-parameters equation capable of describing systems with severe non-linearity. The NARX model description was selected to investigate the application of non-linear system identification in dynamic response reconstruction. The NARX model is discussed in detail in Chapter 5.

$$y(t) = F^{L}\left[y(k-1), \ldots, y(k-na), u(t), \ldots, u(t-nb)\right] \qquad \text{(A-9)}$$

With $F^{L}[\bullet]$ again some non-linear polynomial function

Appendix B

NARX Regression techniques

Various methods were developed to rewrite the NARX difference equation of (5-8) into a matrix notation for implementation within Matlab. The first and most logical method was to directly implement the difference equation in a sample point loop algorithm. This however proved extremely slow, especially for MIMO systems with large numbers of inputs and outputs. The advantages of Matlab were slightly shadowed by the inherent difficulty to execute loop structures efficiently. On the other hand lateral use of Matlab's matrix manipulation capabilities did, in a way, compensate for the lacking loop performance.

B.1. General sample point loop approach

Direct implementation of the NARX difference equation [44] resulted in an extremely slow method. The algorithm made use of a sample point loop, as well as loop functions to calculate the non-linear terms. It clearly illustrates the NARX construction and served as a solid foundation on which improved algorithms were built.

Algorithm B.1: Sample point loop regression of the NARX difference equation

INPUT:	Dynamic system input/output data:	$u_1(t),\ u_2(t) \ldots ,\ u_{nu}(t)$
		$y_1(t),\ y_2(t) \ldots ,\ y_{ny}(t)$
	Dynamic model order for each o/p channel:	n_k
	Degree of non-linearity for each o/p channel:	L_k
	Number of sample points to use in regression	N
	Number if input and output channels	$nu,\ ny$
OUTPUT:	Regression matrix for each o/p channel	$\mathbf{X}_1 , \mathbf{X}_2 , \ldots , \mathbf{X}_{ny}$

```
FOR k = 1,2,...,ny                          Main loop: output channels
   FOR  t = 1,2,...,N                       Sample point loop

                                            Linear regression
      FOR  a = 1,2,..., ny                  Output channel loop
         FOR  i = 1,2,.. n_k                Output delay loop
            X_k(t,i) = y_a(t-i)             Include response  data
      FOR  b = 1,2,...,nu                   Input channel loop
         FOR  j = 0,2,.. n_k                Input delay loop
            X_k(t,n_k+1+j) = u_b(t-i)        Include drive data

      IF L_k >= 2                           Quadratic manipulation
         c = 1                              Counter
         M_L1 = 2n_k +1                     Number of linear terms
         FOR  p = 1,2,..., M_L1
            FOR  q = p+1, p+2,..., M_L1
               c = c+1                      Increment counter
               X_k(t,M_L1+ c) = X_k(t,p) · X_k(t,q)   Calculate quadratic terms

      IF L_k >= 3                           Cubic manipulation
         M_L2 = 2n_k + 1 + c                Number of quadratic terms
         c = 1                              Reset counter
         FOR  p = 1,2,..., M_L1
            FOR  q = p+1, p+2,..., M_L1
               FOR  r = q+1, q+2,..., M_L1
                  c = c+1                   Increment counter
                  X_k(t,M_L2+ c) = X_k(t,p) · X_k(t,q) · X_k(t,r)   Calculate cubic terms
```

B.2. Column-wise Linear regression of the NARX difference equation

An improved method was devised to compute the linear part of the regression matrix without executing the sample point loop. The algorithm made use of Matlab's matrix manipulation capabilities. In each step an entire column is appended to the regression matrix by selecting the correct indices within the input-output data. This reduced calculation times from hours, to seconds. Various methods to further remove the channel and delay order loops were also developed, but proved too cumbersome to warrant practical implementation.

Algorithm B.2: Linear regression without sample point loop

FOR $k = 1,2,...,ny$	*Main loop: output channels*
	Linear regression
FOR $a = 1,2,..., ny$	*Output channel loop*
FOR $i = 1,2,..\ n_k$	*Output delay loop*
$X_k = [\ X_k \ \ y_a(\ n_k + 1 - i : N - 1\ )\ ]$	*Include response data*
FOR $b = 1,2,...,nu$	*Input channel loop*
FOR $j = 0,2,..\ n_k$	*Input delay loop*
$X_k = [\ X_k \ \ u_b(\ n_k + 1 - j : N - 1\ )\ ]$	*Include drive data*

B.3. Non-linear regression

Construction of the non-linear part of the regression matrix proved difficult to implement generally for all degrees of non-linearity, hence the cubic modelling limitation. Calculating the non-linear terms as described in Algorithm B.1 proved computationally expensive, prompting the development of more elegant methods.

B.3.1. Loop methods:

As described in Algorithm B.1 a combination of L nested loops construct the correct indices of the linear regression matrix to be multiplied for each non-linear term. This method proved extremely slow, especially if combined with a sample point loop as shown in Algorithm B.3.

Algorithm B.3: Loop method for calculation of non-linear terms

IF $L_k >= 3$
  $M_{L2} = 2n_k + 1 + c$
  $c = 1$
  FOR $p = 1,2,..., M_{L1}$
    FOR $q = p+1, p+2,..., M_{L1}$
      FOR $r = q+1, q+2,..., M_{L1}$
        $c = c+1$
        $X_k(t,M_{L2}+ c) = X_k(t,p) \cdot X_k(t,q) \cdot X_k(t,r)$

B.3.2. Matrix manipulation methods

Matlab's matrix manipulation capabilities were used in a method for finding the non-linear parts of the regression matrix and proved the forerunner of more advanced methods. This method presented a substantial improvement over the loop method of Algorithm B.3, but was still restricted by a sample point outer loop.

Algorithm B.4: Matrix manipulation of non-linear terms

FOR $k = 1,2,...,ny$
  FOR $t = 1,2,...,N$
    Find $\mathbf{XL1}_k(t)$ *the linear part of the regression matrix* $\mathbf{X}_k(t)$ *as in Algorithm B.1*

Step 1:

$$\mathbf{XL1}_k(t) = [y_1(t-1) \quad \cdots \quad y_{ny}(t-na) \quad \cdots \quad u_1(t) \quad u_1(t-1) \quad \cdots \quad u_{nu}(t-nb)]$$
$$= \left[X_{k_1}(t) \quad X_{k_2}(t) \quad \cdots \quad \cdots \quad \cdots \quad X_{k_{M1}}(t)\right]$$

Step 2: *Create the first term matrix. A square matrix called* $\boldsymbol{P_1}(t)$ *is created by multiplying a (M1x1) unity column matrix with* $\boldsymbol{XL1}_k(t)$.

$$\mathbf{P}_1(t)=\begin{bmatrix}1\\1\\\vdots\\\vdots\\1\end{bmatrix}\cdot\begin{bmatrix}X_{k_1}(t) & X_{k_2}(t) & \cdots & \cdots & X_{k_{M1}}(t)\end{bmatrix}=\begin{bmatrix}X_{k_1}(t) & X_{k_2}(t) & \cdots & \cdots & X_{k_{M1}}(t)\\X_{k_1}(t) & X_{k_2}(t) & \cdots & \cdots & X_{k_{M1}}(t)\\\vdots & & \ddots & & \vdots\\\vdots & & & \ddots & \vdots\\X_{k_1}(t) & X_{k_2}(t) & \cdots & \cdots & X_{k_{M1}}(t)\end{bmatrix}$$

Step 3: *Create the second term matrix,* $\boldsymbol{P_2}(t)$ *by simply transposing* $\boldsymbol{P_1}(t)$*:*

$$\mathbf{P}_2(t)=\mathbf{P}_1(t)^{\mathrm{T}}$$

Step 4: *The matrices* $\boldsymbol{P_1}(t)$ *and* $\boldsymbol{P_2}(t)$ *are then reduced by discarding their upper triangular parts. Taking the elements in a column-wise fashion matrix* $\boldsymbol{P_1}(t)$ *and* $\boldsymbol{P_2}(t)$ *are reshaped into vectors.*

$$\mathbf{P}_1(t)=\begin{bmatrix}X_{k_1}(t)\\X_{k_1}(t) & X_{k_2}(t)\\\vdots & & \ddots\\\vdots & & & \ddots\\X_{k_1}(t) & X_{k_2}(t) & \cdots & \cdots & X_{k_{M1}}(t)\end{bmatrix}\Rightarrow$$

$$\boldsymbol{P}_1(t)=\begin{bmatrix}X_{k_1}(t) & X_{k_1}(t) & \cdots & X_{k_1}(t) & X_{k_2}(t) & \cdots & X_{k_2}(t) & \cdots & \cdots & X_{k_{M1}}(t)\end{bmatrix}$$

Similarly

$$\mathbf{P}_2(t)=\begin{bmatrix}X_{k_1}(t)\\X_{k_2}(t) & X_{k_2}(t)\\\vdots & & \ddots\\\vdots & & & \ddots\\X_{k_{M1}}(t) & X_{k_{M1}}(t) & \cdots & \cdots & X_{k_{M1}}(t)\end{bmatrix}\Rightarrow$$

$$\boldsymbol{P}_2(t)=\begin{bmatrix}X_{k_1}(t) & X_{k_2}(t) & \cdots & X_{k_{M1}}(t) & X_{k_2}(t) & \cdots & X_{k_{M1}}(t) & \cdots & \cdots & X_{k_{M1}}(t)\end{bmatrix}$$

Step 5: *The non-linear part of* $\boldsymbol{X}_k(t)$ *is found by multiplying vectors* $\boldsymbol{P_1}(t)$ *and* $\boldsymbol{P_2}(t)$ *in an element-by-element fashion. The regression matrix* $\boldsymbol{X}_k(t)$ *if formed by appending* $\boldsymbol{XL1}_k(t)$ *and* $\boldsymbol{XL2}_k(t)$ *(.* implies element-by-element multiplication.)*

$$\boldsymbol{XL2}_k(t)=\boldsymbol{P}_1(t).*\boldsymbol{P}_2(t)=$$

$$\begin{bmatrix}X_{k_1}(t)\cdot X_{k_1}(t) & X_{k_1}(t)\cdot X_{k_2}(t) & \cdots & X_{k_2}(t)\cdot X_{k_2}(t) & X_{k_2}(t)\cdot X_{k_3}(t) & \cdots & \cdots & X_{k_{M1}}(t)\cdot X_{k_{M1}}(t)\end{bmatrix}$$

$$\boldsymbol{X}_k(t)=\begin{bmatrix}\boldsymbol{XL1}_k(t) & \boldsymbol{XL2}_k(t)\end{bmatrix}$$

B.3.3. Indexed matrix manipulation methods

The limiting factor in the matrix manipulation technique of Algorithm B.4 remained the sample point loop (the number of samples, N, is typically in the order of 10^4). A more general method in which the non-linear calculations need not be done for each sample was devised. This indexed matrix manipulation method creates a set of matrices containing the indices for non-linear combinations of the linear terms. The procedure is similar to that of Algorithm B.4, but does not manipulate the actual ARX-terms, only the indices thereof. The indexed matrix manipulation method presented in Algorithm B.5 may be used with techniques that implement sample point loops (Algorithm B.1) as well as those that do not (Algorithm B.2). If used in conjunction with a sample point loop algorithm, the index vectors $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$ must be calculated prior to initiating the loop. The non-linear terms, $\boldsymbol{XL2_k(t)}$, are then calculated for each sample by evaluating:

$$\begin{aligned} X_{L2_k}(t) &= X_{L1_k}(t, P_1) \cdot X_{L1_k}(t, P_2) \\ &= X_{L1_k}(t, [1,1,1,\ldots,2,2,\ldots, M_{L1}]) \cdot X_{L1_k}(t, [1,2,3,\ldots,2,3,\ldots, M_{L1}]) \end{aligned} \qquad \text{(B-1)}$$

The indexed matrix manipulation method is best suited for use with column-wise linear regression (Algorithm B.2). The linear regression is completed for each channel k prior to finding the non-linear combinations. Combining Algorithm B.2 and Algorithm B.5 provides a system capable of easily calculating the NARX regression matrix, with minimal use of loop structures.

Algorithm B.5: Indexed matrix manipulation of non-linear terms

FOR k = 1,2,...,ny

Step 1: *Create the first term matrix. A square matrix called* $\boldsymbol{P_1}$ *is created by multiplying a (M1x1) unity column matrix with the vector* $[1,2,3,\ldots,M_{L1}]$.

$$\mathbf{P}_1 = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ \vdots \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 & \cdots & M_{l1} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & & M_{l1} \\ 1 & 2 & 3 & & M_{l1} \\ 1 & 2 & 3 & & M_{l1} \\ \vdots & & & \ddots & \\ 1 & 2 & & & M_{l1} \end{bmatrix}$$

Step 2: *Create the second term matrix,* $\boldsymbol{P_2}$ *by simply transposing* $\boldsymbol{P_1}$

$$\mathbf{P}_2 = \mathbf{P}_1^{\mathrm{T}} = \begin{bmatrix} 1 & 1 & 1 & & 1 \\ 2 & 2 & 2 & & 2 \\ 3 & 3 & 3 & & 3 \\ \vdots & & & \ddots & \\ M_{l1} & M_{l1} & M_{l1} & & M_{l1} \end{bmatrix}$$

Step 3: *The matrices* $\boldsymbol{P_1}$ *and* $\boldsymbol{P_2}$ *are again reduced by discarding their upper triangular parts. Taking the elements in a column-wise fashion matrix* $\boldsymbol{P_1}$ *and* $\boldsymbol{P_2}$ *are reshaped into vectors.*

$$\mathbf{P}_1 = \begin{bmatrix} 1 & & & & \\ 1 & 2 & & & \\ 1 & 2 & 3 & & \\ \vdots & & & \ddots & \\ 1 & 2 & 3 & \cdots & M_{L1} \end{bmatrix} \Rightarrow \quad \mathbf{P}_2 = \begin{bmatrix} 1 & & & & \\ 2 & 2 & & & \\ 3 & 3 & 3 & & \\ \vdots & & & \ddots & \\ M_{L1} & M_{L1} & \cdots & \cdots & M_{L1} \end{bmatrix} \Rightarrow$$

$$\mathbf{P}_1 = \begin{bmatrix} 1 & 1 & \cdots & 2 & 2 & \cdots & 3 & 3 & \cdots & M_{L1} \end{bmatrix} \quad \mathbf{P}_2 = \begin{bmatrix} 1 & 2 & 3 & \cdots & 2 & 3 & \cdots & 3 & \cdots & M_{L1} \end{bmatrix}$$

Step 4: *The vectors* $\boldsymbol{P_1}$ *and* $\boldsymbol{P_2}$ *are the indices of the linear regression matrix* $\boldsymbol{XL1_k}$ *to be multiplied to form the non-linear* $\boldsymbol{XL2_k}$ *so that*

$$\mathbf{X}_{L2_k} = \mathbf{X}_{L1_k}(:, \boldsymbol{P}_1) .* \mathbf{X}_{L1_k}(:, \boldsymbol{P}_2)$$

Step 5: *Lastly the matrices* $\boldsymbol{XL1_k}$ *and* $\boldsymbol{XL2_k}$ *are appended to form the complete regression matrix:*

$$\mathbf{X}_k = \begin{bmatrix} \mathbf{X}_{L1_k} & \mathbf{X}_{L2_k} \end{bmatrix}$$

Appendix C

NARX Parameter estimation: full parameter set solutions

Parameter estimation for NARX models was presented in Section 5.3, defining the concept of orthogonal decomposition to solve the Least Squares equation. Consider again the Least Squares Equation (5-17) for finding the NARX coefficient vector Θ_k .

$$\mathbf{X}^T \cdot \mathbf{X} \cdot \Theta - \mathbf{X}^T \cdot \mathbf{Y} = 0$$

$$\Downarrow$$

$$\boxed{\Theta = \left[\mathbf{X}^T \cdot \mathbf{X}\right]^{-1} \cdot \mathbf{X}^T \cdot \mathbf{Y}} \qquad \text{(C-1)}$$

The simplest method would be to implement Matlab functions to directly solve the inverse $[\ X^T \cdot X\ ]^{-1}$ and then find the Θ_k according to (C-1).

C.1. Solving the normal Least Squares equation

Implementation within Matlab gives access to a multitude of numerical functions and toolboxes which are easily and generally applied to most engineering problems. It was thus a logical first choice to make use of Matlab's matrix *INVERSE* function to find a solution to the Least Squares equation. The algorithm for finding the NARX parameter vector for channel k, Θ_k can thus be written as shown in Algorithm C.1

Algorithm C.1: Solving the Least Squares equation using Matlab

INPUT:	Dynamic system output data:	$y_1(t),\ y_2(t) \ldots ,\ y_{ny}(t)$
	Regression matrix for each channel	X_k
	Number of output channels	ny
OUTPUT:	NARX coefficient vector for each o/p channel	$\Theta_1 , \Theta_2 , \ldots , \Theta_{ny}$

for k = 1, 2, ..., ny	*Loop channel numbers*
$\Theta k = INV(X_k^T * X_k) * X_k^T * y_k$	*Calculate NARX coefficient vector using INVERSE and matrix manipulation functions*

This simple and effective solution is unfortunately not ideally suited for application in large NARX systems. It has major limitations concerning the size of the system and ill-conditioning of matrices. It is not uncommon for the NARX matrix $[\ \mathbf{X}^T \cdot \mathbf{X}\ ]^{-1}$ to consist of more than 10^3 elements, which poses a problem for conventional matrix inversion techniques.

Alternatively the Least Squares equation may be considered as a set of linear equations and solved by a Gaussion elimination scheme. Rewrite Equation (C-1) into the familiar $\mathbf{A}x = b$ format for systems of linear equations:

$$\mathbf{X}^T \cdot \mathbf{X} \cdot \Theta - \mathbf{X}^T \cdot \mathbf{Y} = 0$$

$$\Downarrow$$

$$\left[\mathbf{X}^T \cdot \mathbf{X}\right] \cdot \Theta = \mathbf{X}^T \cdot \mathbf{Y}$$

$$\Downarrow$$

$$\mathbf{A} \cdot x = b \qquad \begin{cases} \mathbf{A} = \left[\mathbf{X}^T \cdot \mathbf{X}\right] \\ x = \Theta \\ b = \mathbf{X}^T \cdot \mathbf{Y} \end{cases} \qquad \text{(C-2)}$$

An elegant and effective method for solving Equation (C-2) is by Crout factorisation of Algorithm C.2, as presented by Burden and Faires [11].

The matrix $\mathbf{X}^T \cdot \mathbf{X}$ may further tend to be ill-conditioned, which makes inversion by conventional techniques impossible. If inversion of an ill conditioned $\mathbf{X}^T \cdot \mathbf{X}$ matrix is possible the NARX coefficient vector Θ may be inaccurate due to an accumulation of round off errors within the inversion process. The ill conditioning of a matrix $\mathbf{X}$ can be quantified by its rank $K[\mathbf{X}]$, the ratio of the largest to the smallest non-zero singular value of $\mathbf{X}$ [37]. According to Chen [18], a normal equation for solving the Least Squares problem can not be used unless $K[\mathbf{X}] \leq 2^{M/2}$, with M the number of NARX coefficients within Θ. This creates the demand for more general solution methods.

Algorithm C.2: Crout factorisation

INPUT:	Dynamic system output data:	$y_1(t),\ y_2(t) \ldots , y_{ny}(t)$
	Regression matrix for each channel	X_k
	Number of output channels	ny
OUTPUT:	NARX coefficient vector for each o/p channel	$\Theta_1 , \Theta_2 , \ldots , \Theta_{ny}$

```
for k = 1, 2, ..., ny                    Loop channel numbers
    A = X_k^T * X_k                      Create A matrix
    b = X_k^T * y_k                      Create b vector
    n = max(size(A));                    Calculate number of columns in A
    A = [A b];                           Create augmented matrix
    l(1,1) = A(1,1);                     Initialise lower triangular matrix L
    u(1,2) = A(1,2)/l(1,1);              Initialise upper triangular matrix U
        for i = 2:n-1
          l(i,i-1) = A(i,i-1);
          l(i,i) = A(i,i)-l(i,i-1)*u(i-1,i);
          u(i,i+1) = a(i,i+1)/l(i,i);
    l(n,n-1) = A(n,n-1);
    l(n,n) = A(n,n)-l(n,n-1)*u(n-1,n);
    z(1) = a(1,n+1)/l(1,1);
    for i = 2:n
        z(i) = (a(i,n+1)-l(i,i-1)*z(i-1))/l(i,i);
    x(n) = z(n);
    for i = n-1:-1:1                     Backward substitution
        x(i) = z(i) - u(i,i+1)*x(i+1);
    Θ_k = x^T;
```

C.2. Singular value decomposition

The solution to the Least Squares problem is no longer unique if the rank of **X** is less then $2^{M/2}$. The singular value decomposition offers a general solution to the Least Squares problem. The procedure starts with the singular value decomposition theorem as presented by Marcus [43]:

$$\mathbf{X}=\mathbf{U}\cdot\mathbf{S}\cdot\mathbf{V}^{T} \tag{C-3}$$

The matrix can be the product of a diagonal matrix, **S**, of the same dimension as **X** and with nonnegative diagonal elements in decreasing order, and unitary matrices **U** and **V**.

$$\mathbf{S}=\mathrm{diag}[s_1,s_2,\cdots,s_M] \tag{C-4}$$

The pseudo-inverse of **S** is defined as:

$$\mathbf{S}^{+}=\mathrm{diag}[s_1^{+},s_2^{+},\cdots,s_M^{+}] \tag{C-5}$$

$$s_i^{+}=\begin{cases}\dfrac{1}{s_i}, & \text{for } s_i>0\\ 0, & \text{for } s_i=0\end{cases}$$

So that:

$$\mathbf{X}^{+}=\mathbf{V}\cdot\mathbf{S}^{+}\cdot\mathbf{U}^{T} \tag{C-6}$$

Thus the solution to the Least Squares problem is:

$$\Theta=\mathbf{X}^{+}\cdot\mathbf{Y} \tag{C-7}$$

Finding the NARX parameters using the singular value decomposition scheme proved accurate, but time consuming due to the complexity of finding the singular values of a large matrix (C-3).

C.3. Orthogonal decomposition

Matrix inversion methods based on orthogonalization of the matrix $\mathbf{X}^T \cdot \mathbf{X}$ proved the most convenient for parameter estimation applications of large NARX systems [18][36]. These methods are generally fast and can be implemented to be insensitive to low matrix ranks and thus produce accurate results for most systems.

Recall that a matrix **X** can be transformed into an orthogonal matrix **Q** and an upper triangular matrix **R** so that [32].

$$\mathbf{X} = \mathbf{Q} \cdot \mathbf{R} \tag{C-8}$$

Transform the Least Squares equation, in order to find the parameter vector Θ.

$$\Theta = \left[\mathbf{X}^T \cdot \mathbf{X}\right]^{-1} \cdot \mathbf{X}^T \cdot \mathbf{Y}$$
$$\mathbf{X}^T \cdot \mathbf{X} \cdot \Theta = \mathbf{X}^T \cdot \mathbf{Y} \tag{C-9}$$

Substitute $\mathbf{X} = \mathbf{Q} \cdot \mathbf{R}$

$$\left[\mathbf{Q} \cdot \mathbf{R}\right]^T \cdot \mathbf{Q} \cdot \mathbf{R} \cdot \Theta = \left[\mathbf{Q} \cdot \mathbf{R}\right]^T \cdot \mathbf{Y} \tag{C-10}$$
$$\mathbf{Q}^T \cdot \mathbf{R}^T \cdot \mathbf{Q} \cdot \mathbf{R} \cdot \Theta = \mathbf{Q}^T \cdot \mathbf{R}^T \cdot \mathbf{Y}$$

but $\mathbf{Q}^T \cdot \mathbf{Q} = \mathbf{I}$

$$\mathbf{R}^T \cdot \mathbf{R} \cdot \Theta = \mathbf{Q}^T \cdot \mathbf{R}^T \cdot \mathbf{Y} \tag{C-11}$$
$$\mathbf{R} \cdot \Theta = \left[\mathbf{R}^T\right]^{-1} \cdot \mathbf{Q}^T \cdot \mathbf{R}^T \cdot \mathbf{Y}$$

thus (C-12)

$$\boxed{\Theta = \left[\mathbf{R}^T\right]^{-1} \cdot \mathbf{Q}^T \cdot \mathbf{Y}}$$

Again direct implementation of built in Matlab functions provided a simple and convenient first approach to finding the NARX parameter vector Θ_k. **Algorithm C.3** shows the use of Matlab's *QR* orthogonalization function.

Algorithm C.3: Parameter estimation using Matlab orthogonal decomposition

INPUT:	Dynamic system output data:	$y_1(t),\ y_2(t) \ldots , y_{ny}(t)$
	Regression matrix for each channel	X_k
	Number of output channels	ny
OUTPUT:	NARX coefficient vector for each o/p channel	$\Theta_1 , \Theta_2 , \ldots , \Theta_{ny}$

for k = 1, 2, ..., ny	*Loop channel numbers*
$[Q , R] = QR (X_k)$	*Orthogonal decomposition of regression vector X, using Matlab's QR function*
$\Theta k = INV(R^T) * Q^T * y_k$	*Calculate NARX coefficient vector using INVERSE and matrix manipulation functions*

The parameter estimation technique presented in Algorithm C.3 proved accurate and extremely convenient, but is unfortunately slow for large non-linear systems. A faster, more elegant approach to orthogonal decomposition was needed. Various methods are discussed: Golub [29] reviews methods applicable to matrix orthogonalization, Chen *et all.* [18] present a classical and modified Gram Schmidt process specifically for application in parameter estimation. A Gram Schmidt process as presented by Burden and Faires [11] as well as a method proposed by Householder [32] was modified by the author for application in NARX parameter estimation.

C.3.1. Gram Schmidt

The Gram Schmidt process is commonly used in numerical analysis for applications ranging from finding eigen values, solving sets of numerical equations, finding matrix inverses and more specifically orthogonalization of matrices. A typical example is that presented by Burden and Faires [11]. A more formal Gram Schmidt approach, specifically adapted for parameter estimation, is presented by Chen *et all* [18]. This procedure is an adaptation of the method presented by Golub [29] and makes use of an auxiliary model:

$$\mathbf{Y} = \mathbf{W} \cdot \mathbf{g} \tag{C-13}$$

With **W** an orthogonal regression matrix and **g** an auxiliary parameter vector. Consider the factorisation of the regression matrix **X** into a upper triangular matrix **A** and an orthogonal matrix **W**. The matrices W and A are calculated according to Algorithm C.4

$$\mathbf{X} = \mathbf{W} \cdot \mathbf{A}$$

$$\mathbf{A} = \begin{bmatrix} 1 & \alpha_{12} & \alpha_{13} & \cdots & \alpha_{1M} \\ & 1 & \alpha_{23} & \cdots & \alpha_{2M} \\ & & \ddots & \ddots & \vdots \\ & & & 1 & \alpha_{M-1M} \\ & & & & 1 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_M \end{bmatrix} \tag{C-14}$$

The N by M matrix **W** has orthogonal columns that satisfy:

$$\mathbf{W}^T \cdot \mathbf{W} = \mathbf{D} \tag{C-15}$$

With **D** a positive diagonal matrix such that:

$$\mathbf{X}^T \cdot \mathbf{X} = \mathbf{A}^T \mathbf{D} \mathbf{A} \tag{C-16}$$

Algorithm C.4: Calculation of W and A for Gram Schmidt process

INPUT:	Regression matrix for each channel	X_k
	Number of NARX parameters	M
OUTPUT:	Orthogonal regression matrix	**W**
	upper triangular matrix	**A**

$W_1 = X_1$ — *Initiate process by setting column 1 of* **W** *equal to column 1 of* **X**

for $k = 2, 3, ..., M$

for $i = 1, 2, ..., k-1$ — *Calculate the* k^{th} *column of* **W** *and* k^{th} *row of* **A**

$$\alpha_{ik} = \frac{W_i \cdot X_k}{W_i \cdot W_i}$$ — *(for* **W** *and* **X** *the subscript indicates the column number)*

$$W_k = X_k - \sum_{i=1}^{k-1} \alpha_{ik} W_i$$

The auxiliary parameter vector **g** is defined so that:

$$\mathbf{g} = \mathbf{D}^{-1} \cdot \mathbf{W}^{T} \cdot \mathbf{Y} \qquad \text{(C-17)}$$

Using back substitution, the parameter vector Θ can now readily be calculated from:

$$\mathbf{A} \cdot \Theta = \mathbf{g} \qquad \text{(C-18)}$$

This classical Gram Schmidt process proved fast, but is very sensitive to round-off errors [18].

C.3.2. *Modified Gram Schmidt*

Due to the inherent inaccuracies associated with the classical Gram Schmidt process, Chen *et all* [18] further presented a modified Gram Schmidt procedure for finding the NARX coefficients. The orthogonalization process for the modified Gram Schmidt method is presented in Algorithm C.5

Algorithm C.5: Calculation of W and A for modified Gram Schmidt process

INPUT:	Regression matrix for each channel	X_k
	Number of NARX parameters	M
OUTPUT:	Orthogonal regression matrix	$\mathbf{W}$
	upper triangular matrix	$\mathbf{A}$

$X^{(0)} = X$ — *Initiate process by setting the matrix $X^{(0)}$ equal to X*

for k =1, 2, 3, ..., M-1

$W_k^{(0)} = X_k^{(k-1)}$ — *Calculate the k^{th} column of W and k^{th} row of A*

for i =1, 2, ..., k-1 — *(for W and X the subscript indicates the column number)*

$$\mathbf{w}_k = \mathbf{x}_k^{(k-1)}$$

$$\alpha_{ki} = \frac{\mathbf{w}_k^{\mathrm{T}} \cdot \mathbf{x}_i^{(k-1)}}{\mathbf{w}_k^{\mathrm{T}} \cdot \mathbf{w}_k}$$

$$\mathbf{x}_i^{(k)} = \mathbf{x}_i^{(k-1)} - \alpha_{ki}\mathbf{w}_k$$

$$\mathbf{w}_M = \mathbf{x}_M^{(M-1)}$$

The elements of the auxiliary parameter vector **g** are calculated according to Algorithm C.6.

Algorithm C.6: Calculation of auxiliary parameter vector g for the modified Gram Schmidt process

INPUT:	Orthogonal regression matrix	W
	Number of NARX parameters	M
OUTPUT:	Auxiliary parameter vector	$\mathbf{g}$

$Y^{(0)} = Y$ *Initiate process by setting* $Y^{(0)}$ *equal to X*

for k =1, 2, 3, ..., M-1

$$\mathbf{g}_k = \frac{\mathbf{w}_k^{\mathrm{T}} \cdot \mathbf{Y}^{(k-1)}}{\mathbf{w}_k^{\mathrm{T}} \cdot \mathbf{w}_k}$$

Calculate the k^{th} *element of* $\mathbf{g}$ *and rewrite* Y^k

$$\mathbf{Y}^{(k)} = \mathbf{Y}^{(k-1)} - \mathbf{g}_k \mathbf{w}_k$$

(for $\mathbf{W}$ *the subscript indicates the column number)*

This modified Gram Schmidt procedure is not as fast as the classical method, but proved numerically superior and very accurate since round off errors are not accumulated. The modified Gram Schmidt procedure is slow due to the complex loop structures required during the orthogonalization process. An adapted version of this modified Gram Schmidt algorithm proved to be the most effective method for NARX parameter estimation. This method combined the classical Gram Schmidt technique with Matlab's *QR* function as shown in Algorithm C.7.

Algorithm C.7: Combined Matlab Gram Schmidt parameter estimation

INPUT:	Dynamic system output data:	$y_1(t),\ y_2(t) \ldots , y_{ny}(t)$
	Regression matrix for each channel	X_k
	Number of output channels	ny
OUTPUT:	NARX coefficient vector for each channel	$\Theta_1 , \Theta_2 , \ldots , \Theta_{ny}$

```
for k = 1, 2, ..., ny                          Loop channel numbers
    [Q , R ] = QR ( X_k )                      Orthogonal decomposition of regression vector X, using Matlab's QR function
    [dum,M]=size(X_k );                        Calculate number of columns in regression matrix X
    g = Q'*y;                                  Calculate NARX coefficient vector
    Θ_k (M) = g(M)/R(M,M);
    for j = M-1:-1:1
        Θ_k (j) =
            (g(j)-(R(j,j+1:M)*Θ_k (j+1:M,1)))/R(j,j)
```

C.3.3. Householder

A parameter estimation technique based on the orthogonalization proposed by Householder [32] and again presented by Golub [29] and Chen [18] for application in parameter estimation, is given in Algorithm C.8. Augment the matrix $\mathbf{Q}$ with a further N-M orthonormal columns to make up a full set of N orthonormal vectors for an N-dimensional Euclidean space:

$$\tilde{\mathbf{Q}} = \begin{bmatrix} \mathbf{Q} & \tilde{\mathbf{q}}_{M+1} & \cdots & \tilde{\mathbf{q}}_{N} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_M \tilde{\mathbf{Q}}_{N-M} \end{bmatrix} \qquad \text{(C-19)}$$

Then

$$\mathbf{X} = \tilde{\mathbf{Q}}\tilde{\mathbf{R}} = \mathbf{Q}\begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix} \qquad \text{(C-20)}$$

Where $\mathbf{R}$ is a M x M upper triangular matrix and $\tilde{\mathbf{Q}}^{\mathrm{T}}$ can be used to triangularize $\mathbf{X}$.

If $\tilde{\mathbf{Q}}^{\mathrm{T}}\mathbf{y}$ is partitioned into:

$$\tilde{\mathbf{Q}}^{\mathrm{T}}\mathbf{y} = \mathbf{Q}\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} \begin{matrix} \}M \\ \}N-M \end{matrix} \qquad \text{(C-21)}$$

We have:

$$\|\mathbf{y} - \mathbf{X}\cdot\Theta\| = \left\|\tilde{\mathbf{Q}}^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\cdot\Theta)\right\| = \|\mathbf{y}_1 - \mathbf{R}\cdot\Theta\| + \|\mathbf{y}_2\| \qquad \text{(C-22)}$$

The Least Squares estimates can therefore be obtained by solving the triangular system:

$$\mathbf{R}\cdot\Theta = \mathbf{y}_1 \qquad \text{(C-23)}$$

Algorithm C.8: Householder parameter estimation

INPUT:	Dynamic system output data:	$y_1(t),\ y_2(t) \ldots , y_{ny}(t)$
	Regression matrix for each channel	X_k
	Number of output channels	ny
OUTPUT:	NARX coefficient vector for each channel	$\Theta_1 , \Theta_2 , \ldots , \Theta_{ny}$

```
[N,M]=size(x);
x = [x y];
for j = 1:M;
    sj = norm(x(j:N,j));
    bj = 1/(sj*(sj+abs(x(j,j))));
    vj = zeros(N,1);
    vj(j,1) = x(j,j)+sign(x(j,j))*sj;
    vj(j+1:N,1) = x(j+1:N,j);
    x = x - vj*bj*vj'*x;
z1 = x(1:M,M+1);
z2 = x(M+1:N,M+1);
x = x(1:M,1:M);
th = inv(x)*z1;
```

Appendix D

NARX Parameter estimation: Reduced parameter set solutions

Because the number of possible NARX candidate terms can easily run into several thousands even for 'moderately' non-linear MIMO systems optimal multiple selection methods could reduce the computational effort involved in the system identification process. Unfortunately the model structure of real systems is rarely known *a priori* and methods of model structure determination must therefore be developed and included as a vital part of the identification procedure. Korenberg [36] indicates that "provided the significant terms in the model can be detected, models with fewer than ten terms are usually sufficient to capture the dynamics of highly non-linear processes." Various studies [38][36][6], have been undertaken into structure detection for non-linear parametric models. More specifically Chen [18] surveyed methods for finding a reduced NARX equation i.e. discarding terms in the NARX equation which do not contribute to the dynamic behaviour of the system. These structure reduction schemes are presented for the parameter estimation techniques of Appendix C. These methods all require the full set of NARX coefficients to be available for evaluation, thus an initial full set regression and parameter estimation is required prior to structure detection. Research into finding the significant terms within the NARX model prior to parameter estimation proved fruitless. The basic structure detection scheme is presented in Figure D.1.

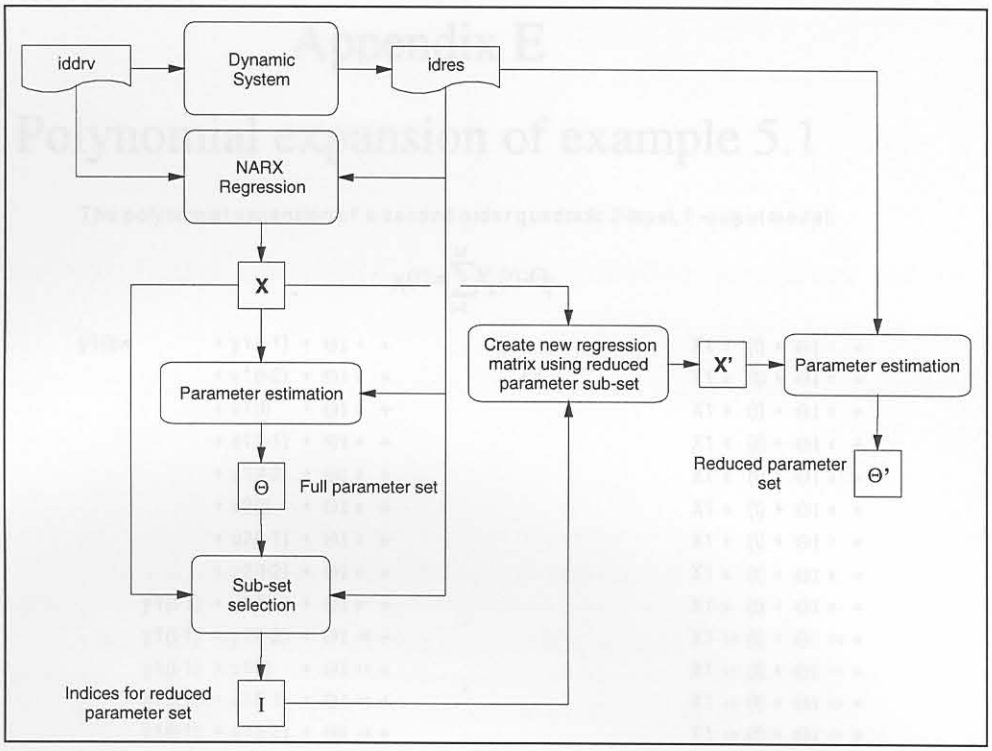

Figure D.1: Reduced sub-set selection

The structure detection processes proved computationally cumbersome. More importantly simulation of the NARX models proved insensitive to the number of terms involved. The general conclusion is that the amount of effort concerned with model reduction does not warrant the implementation thereof. Full parameter set modelling proved more practical. Some of the reduction techniques investigated are listed in Table D.1.

Table D.1 Parameter reduction methods

1	Stepwise regression
2	Singular value decomposition
3	Orthogonal decomposition
	Gram Schmidt
	Modified Gram Schmidt
	Householder
4	Diverse methods
5	Eigen value methods

Appendix E

Polynomial expansion of example 5.1

The polynomial expansion of a second order quadratic 2-input, 1 -output model.

$$y_1(t)=\sum_{i=1}^{M} X_{1_i}(t)\cdot\Theta_{1_i}$$

y1(t)=		• y1(t-1)	• $\Theta1_{1}$ +		$X1_{1}$ (t) • $\Theta1_{1}$ +
		• y1(t-2)	• $\Theta1_{2}$ +	or	$X1_{2}$ (t) • $\Theta1_{2}$ +
		• u1(t)	• $\Theta1_{3}$ +		$X1_{3}$ (t) • $\Theta1_{3}$ +
		• u1(t-1)	• $\Theta1_{4}$ +		$X1_{4}$ (t) • $\Theta1_{4}$ +
		• u1(t-2)	• $\Theta1_{5}$ +		$X1_{5}$ (t) • $\Theta1_{5}$ +
		• u2(t)	• $\Theta1_{6}$ +		$X1_{6}$ (t) • $\Theta1_{6}$ +
		• u2(t-1)	• $\Theta1_{7}$ +		$X1_{7}$ (t) • $\Theta1_{7}$ +
		• u2(t-2)	• $\Theta1_{8}$ +		$X1_{8}$ (t) • $\Theta1_{8}$ +
	y1(t-1)	• y1(t-1)	• $\Theta1_{9}$ +		$X1_{9}$ (t) • $\Theta1_{9}$ +
	y1(t-1)	• y1(t-2)	• $\Theta1_{10}$ +		$X1_{10}$ (t) • $\Theta1_{10}$ +
	y1(t-1)	• u1(t)	• $\Theta1_{11}$ +		$X1_{11}$ (t) • $\Theta1_{11}$ +
	y1(t-1)	• u1(t-1)	• $\Theta1_{12}$ +		$X1_{12}$ (t) • $\Theta1_{12}$ +
	y1(t-1)	• u1(t-2)	• $\Theta1_{13}$ +		$X1_{13}$ (t) • $\Theta1_{13}$ +
	y1(t-1)	• u2(t)	• $\Theta1_{14}$ +		$X1_{14}$ (t) • $\Theta1_{14}$ +
	y1(t-1)	• u2(t-1)	• $\Theta1_{15}$ +		$X1_{15}$ (t) • $\Theta1_{15}$ +
	y1(t-1)	• u2(t-2)	• $\Theta1_{16}$ +		$X1_{16}$ (t) • $\Theta1_{16}$ +
	y1(t-2)	• y1(t-2)	• $\Theta1_{17}$ +		$X1_{17}$ (t) • $\Theta1_{17}$ +
	y1(t-2)	• u1(t)	• $\Theta1_{18}$ +		$X1_{18}$ (t) • $\Theta1_{18}$ +
	y1(t-2)	• u1(t-1)	• $\Theta1_{19}$ +		$X1_{19}$ (t) • $\Theta1_{19}$ +
	y1(t-2)	• u1(t-2)	• $\Theta1_{20}$ +		$X1_{20}$ (t) • $\Theta1_{20}$ +
	y1(t-2)	• u2(t)	• $\Theta1_{21}$ +		$X1_{21}$ (t) • $\Theta1_{21}$ +
	y1(t-2)	• u2(t-1)	• $\Theta1_{22}$ +		$X1_{22}$ (t) • $\Theta1_{22}$ +
	y1(t-2)	• u2(t-2)	• $\Theta1_{23}$ +		$X1_{23}$ (t) • $\Theta1_{23}$ +
	u1(t)	• u1(t)	• $\Theta1_{24}$ +		$X1_{24}$ (t) • $\Theta1_{24}$ +
	u1(t)	• u1(t-1)	• $\Theta1_{25}$ +		$X1_{25}$ (t) • $\Theta1_{25}$ +
	u1(t)	• u1(t-2)	• $\Theta1_{26}$ +		$X1_{26}$ (t) • $\Theta1_{26}$ +
	u1(t)	• u2(t)	• $\Theta1_{27}$ +		$X1_{27}$ (t) • $\Theta1_{27}$ +
	u1(t)	• u2(t-1)	• $\Theta1_{28}$ +		$X1_{28}$ (t) • $\Theta1_{28}$ +
	u1(t)	• u2(t-2)	• $\Theta1_{29}$ +		$X1_{29}$ (t) • $\Theta1_{29}$ +
	u1(t-1)	• u1(t-1)	• $\Theta1_{30}$ +		$X1_{30}$ (t) • $\Theta1_{30}$ +
	u1(t-1)	• u1(t-2)	• $\Theta1_{31}$ +		$X1_{31}$ (t) • $\Theta1_{31}$ +
	u1(t-1)	• u2(t)	• $\Theta1_{32}$ +		$X1_{32}$ (t) • $\Theta1_{32}$ +
	u1(t-1)	• u2(t-1)	• $\Theta1_{33}$ +		$X1_{33}$ (t) • $\Theta1_{33}$ +
	u1(t-1)	• u2(t-2)	• $\Theta1_{34}$ +		$X1_{34}$ (t) • $\Theta1_{34}$ +
	u1(t-2)	• u1(t-2)	• $\Theta1_{35}$ +		$X1_{35}$ (t) • $\Theta1_{35}$ +
	u1(t-2)	• u2(t)	• $\Theta1_{36}$ +		$X1_{36}$ (t) • $\Theta1_{36}$ +
	u1(t-2)	• u2(t-1)	• $\Theta1_{37}$ +		$X1_{37}$ (t) • $\Theta1_{37}$ +
	u1(t-2)	• u2(t-2)	• $\Theta1_{38}$ +		$X1_{38}$ (t) • $\Theta1_{38}$ +
	u2(t)	• u2(t)	• $\Theta1_{39}$ +		$X1_{39}$ (t) • $\Theta1_{39}$ +
	u2(t)	• u2(t-1)	• $\Theta1_{40}$ +		$X1_{40}$ (t) • $\Theta1_{40}$ +
	u2(t)	• u2(t-2)	• $\Theta1_{41}$ +		$X1_{41}$ (t) • $\Theta1_{41}$ +
	u2(t-1)	• u2(t-1)	• $\Theta1_{42}$ +		$X1_{42}$ (t) • $\Theta1_{42}$ +
	u2(t-1)	• u2(t-2)	• $\Theta1_{43}$ +		$X1_{43}$ (t) • $\Theta1_{43}$ +
	u2(t-2)	• u2(t-2)	• $\Theta1_{44}$		$X1_{44}$ (t) • $\Theta1_{44}$

Appendix F

Summary of functions

The research into application of NARX system identification procedures led to the development of a toolbox of Matlab M-functions for non-linear system identification, and response reconstruction. Some of the M-functions, together with a short description, are listed below:

NLID.M	Non-linear system identification to condensed NARX formulation
NLSIM.M	Simulation of SISO condensed NARX models
NLMIMO.M	Simulation of MIMO condensed NARX models
NLINVID.M	Inverse Non-linear system identification to condensed NARX formulation for use in response reconstruction
NLINVSIM.M	Inverse simulation of MIMO condensed NARX models for use in response reconstruction
IDARX.M	Linear ARX system identification. User friendly implementation of Matlab's ARX function
SIMARX.M	Linear simulation of ARX models. User friendly implementation of Matlab's IDSIM function
THQR.M	NARX parameter estimation using Orthogonal decomposition
REGRES.M	Regression of time history data sets for use with NLID.M and NLINVID.M

REPEAT.M	Repeatability function
MAKEMOD.M	Random non-linear model generator
NEWPSD.M	Rig specific identification PSD generator
QFIT.M	Modified QanTiM error function

Appendix G

QanTiM: a practical solution to response reconstruction

Figure G.1 is a simplified schematic representation of the QanTiM simulation process (See [34] & [35]). The flow diagram shows relevant file names, and their locations within the process. The general response processing icon represents possible high-pass and low-pass filtering, as well as DC-offset removal as applied to all response data. The IDDRV processing icon represents filtering operations to identification drive data. Both these processing functions make use of parameters set up in the data processing window and are implemented into QanTiM as a direct result of the empirical research presented in Chapter 3.

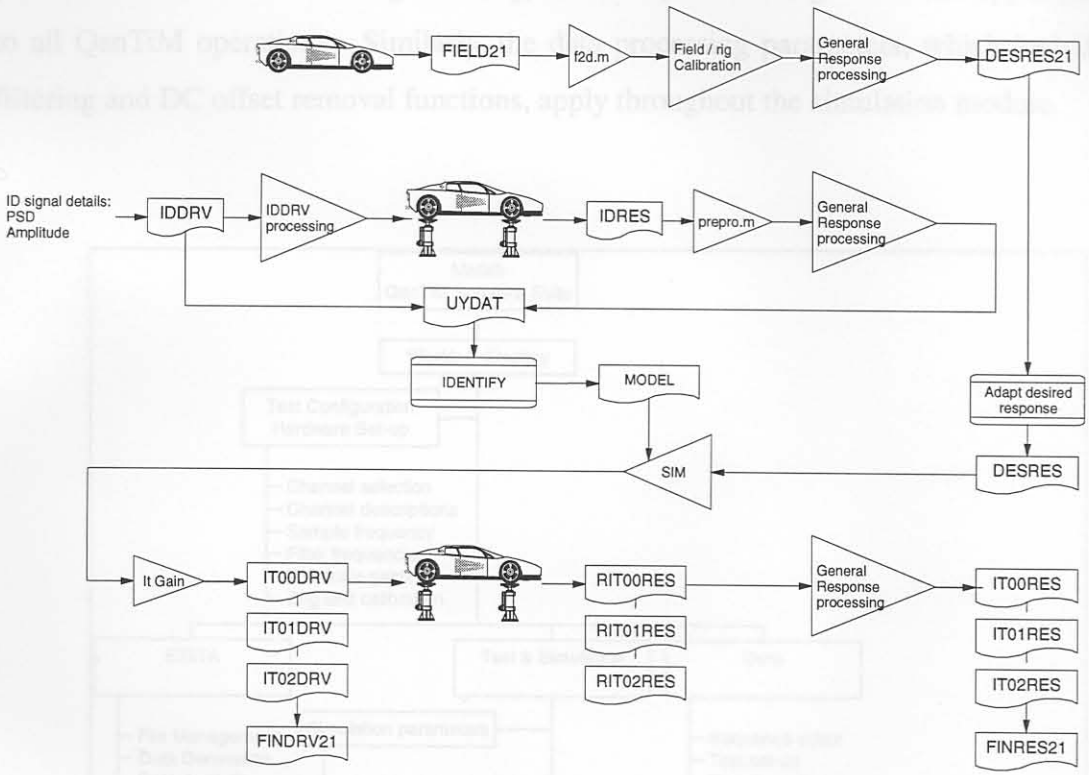

Figure G.1 QanTiM process

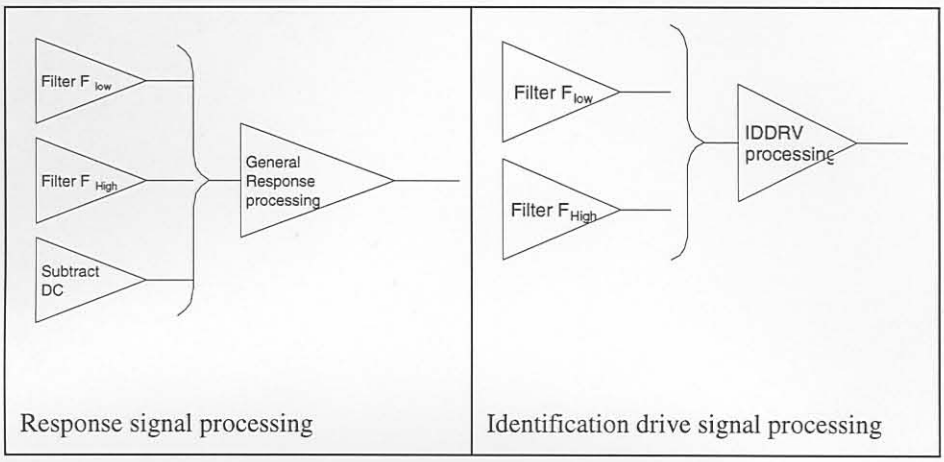

Figure G.2 General processing functions

Figure G.3 shows a map of QanTiM windows and relevant operations. Initial test parameters such as the working directory, and the system configuration are applicable to all QanTiM operations. Similarly, the data processing parameters, which include filtering and DC offset removal functions, apply throughout the simulation module.

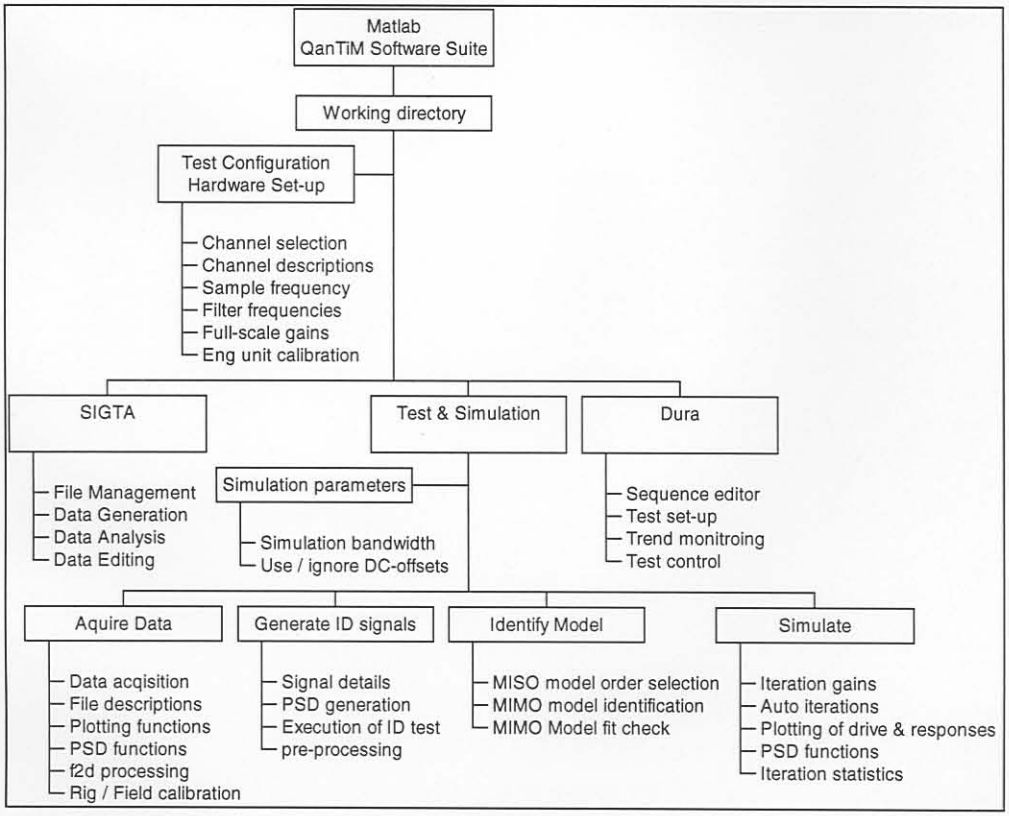

Figure G.3 QanTiM software map