DYNAMIC MATRIX CONTROL OF MILLING CIRCUITS

by

Sikhumbuzo Cazwell Mkhize
(B. Sc. Chem Eng)

Submitted to

The University of Cape Town

in fulfillment for the degree of

Master of Science in Engineering (Chemical)

July 1992

ACKNOWLEDGEMENTS

The author wishes to thank Dr C. L. E Swartz for his supervision, Messrs J. Young and R. Leeuw, Mrs P. Bettison and all UCT Chemical Engineering staff for their assistance. The author also wishes to thank Mintek for their financial assistance.

Lastly, the understanding, support and motivation from my wife, Bonani, is greatfully acknowledged. Above all people the Honor and Thanx goes to the Lord Jesus Christ who makes all things possible.

SYNOPSIS

This thesis sets out the results of a research project on Dynamic Matrix Control (DMC) of milling circuits.

The main aim of the study was to investigate the suitability of DMC for milling circuit control. This was conducted through simulation studies using models of two different milling circuits. A generalised software package was developed for the application and analysis of DMC.

DMC was developed in the United States of America by Shell Oil Company (Cutler and Ramaker, 1979; Prett and Gillete, 1979). It falls under the class of controllers which is termed Model Predictive Control (MPC). The control algorithms falling into this category are multivariable and model-based, and as such are expected to improve control of processes which exhibit strong interactions, non-minimum phase behavior, and operate at constraints. Other control schemes falling into this category are Model Algorithmic Control (MAC) and Internal Model Control (IMC). Details of MAC are largely proprietary, while the frequency-based IMC method does not permit direct handling of constraints. Thus the focus of this project was on the DMC algorithm and its variants; Linear Dynamic Matrix Control (LDMC) and Quadratic Dynamic Matrix Control (QDMC).

The function of a milling or grinding circuit in a mineral processing plant is to prepare the ore for concentration by liberating the valuable mineral from the waste minerals or for chemical reaction by increasing the surface area of the valuable mineral. Grinding circuits have been found to be energy intensive. Rajamani and Herbst (1991) estimate that of the total energy consumed in the mill, less than 10% is utilised in the size reduction of ore particles, the rest is used to drive the mill. Wills (1985) has shown that as ore grades decrease, mill energy

consumptions could become the most important factor in deciding whether the ore is processed or not. For this reason, grinding circuits have been of considerable interest in the mineral processing industry, particularly with respect to their control.

Grinding circuits are multivariable in nature with large interactions and time delays and demand that they be operated at constraints for profitable operation. These dynamics and demands have proved single-loop control inadequate. DMC with its features appeared suitable for milling circuit control and was therefore investigated.

The first model used was a linear model derived by Hulbert and Braae (1981) from the number 1 East Driefontein Gold Mine. The model is a three input - three ouput system with a disturbance. DMC was succesfully implemented showing minor interaction effects. Unmodelled disturbance handling capabilities of DMC were investigated. The DMC unmodelled disturbance compensation proved to be sufficient even though it assumed disturbances were of a step type whereas the disturbance associated with this model was more of a ramp type.

The flexible DMC algorithm allows for the inclusion of feedforward control. Since there was a model for the disturbance, a feedforward controller was readily and succesfully included in the control algorithm. The results showed that DMC used with feedforward control effectively rejects known disturbances.

DMC allows for constraints to be placed on input changes, the inputs and the controlled and uncontrolled outputs, restricting their movemement within specified limits. Constraints were imposed on the process. As can be expected, the performance deteriorated as the constraints were made tighter; however, the most important point demonstrated was that constraints which are typical in milling circuit control can be readily implemented in QDMC.

Since the basic DMC algorithm assumes that the process is linear, a nonlinear model of the grinding circuit derived by Rajamani and Herbst (1991) for a test rig was used to investigate suitability of DMC with nonlinear systems. The model has two independent inputs and therefore can

control two outputs. DMC was found to be robust enough to reject model errors associated with nonlinearities even in the face of unknown disturbances and constraints.

In conclusion, the simulation studies demonstrated that DMC can in principle be succesfully applied in milling circuit control as easily as is used in other industries. DMC performance has demonstrated its abilities to handle interactions and time delays inherent in the process. It is recommended that on-line optimization of milling circuits be investigated for maximum benefit from the control algorithm. Stability, in particular stability under constraints, should be investigated. Investigations involving real-time control should lead to reasonable conclusions as to the extent of the benefits of using DMC in milling circuit control.

CONTENTS

LIST OF FIGURES

NOMENCLATURE

A the dynamic matrix.

A^T the transpose of A.

$\underline{b}$ a vector representing the influence of past inputs and disturbance predictions on $\underline{y}$.

C_1 the fraction of plus 44μm material in the cyclone feed that reports to underflow.

C_2 the fraction of minus 44μm material in the cyclone feed that reports to underflow.

C_S the concentration of solids in the sump expressed as mass of solids per unit volume of slurry (kg/m^3).

CPU central processing unit.

d unmeasured disturbance.

D the dynamic matrix of the disturbance.

d_{50} the size at which 50% of the solids report to overflow and 50% to underflow.

FCCU fluid catalytic cracking unit.

F_{OF} the percentage of solids less than 44μm in the product stream.

f_v the volume fraction of solids in the slurry feed.

$\underline{g}$ a QP gradient vector.

H a QP hessian matrix.

H_M the mill hold-up mass (kg).

I the identity matrix.

I_L the lower triangular identity matrix.

k_0 a kinetic rate parameter (min^{-1}).

Λ an N X N matrix of move suppression factors.

λ_i the i-th move suppression factor.

LP a linear programming.

M the number of intervals to reach steady state.

M_{FF} the fresh feed solids rate (kg/min).

M_{MF} the mill solids feed rate (kg/min).

M_{UF} the cyclone underflow solids rate (kg/min).

N the number of future control moves (Δu).

ODEs ordinary differential equations.

P the prediction horizon.

PCD the primary cyclone feed density (kg/m^3).

PCF the flow-rate of feed to primary cyclone (kg/min).

PD the dilution of primary sump (kg/min).

PEB the feed rate of pebbles (kg/min).

PID proportional, integral and derivative.

PSM the particle-size measurement (%).

Q a P X P output weighting matrix.

Q_C the volumetric feed rate to the cyclone (m^3/min).

q_i the i-th output weight.

Q_M the total volume of slurry discharging from the mill (m^3/min).

QP a quadratic programming.

R_{FF} the fraction of material above 44μm in the mill fresh feed.

R_M the fraction of material above 44μm in the mill.

RMFD the feed rate of water to rod mill (kg/min).

R_S the fraction of material above 44μm in the sump.

ρ_S the solids density (kg/m^3).

R_{UF} the fraction of material above 44μm in the cyclone underflow.

ρ_W water density (kg/m^3).

SD the dilution of secondary sump (kg/min).

ΔT the discretization step.

$\Delta \underline{u}$ an N-vector of future moves.

u the input.

u_i the i^{th} input.

u_{max} the maximum input.

u_{min} the minimum input.

y the output.

$\underline{y}$ the output vector.

yi the output prediction.

ym the plant measured output.

y_{max} the maximum output.

y_{min} the minimum output.

y_s the setpoint.

$\underline{y}_s$ the setpoint vector.

V_S the volume of slurry in the sump.

W_F the feed water rate to the cyclone (kg/min).

W_{FF} fresh water feed to the mill (kg/min).

W_{OF} the water in the cyclone overflow (kg/min).

W_{SF} the sump water addition rate (kg/min).

Δx the disturbance input.

CHAPTER 1

INTRODUCTION

The main aim of the study is to investigate the suitability of Dynamic Matrix Control (DMC) for milling circuit control. The study was conducted through simulation studies using two different milling circuit models. A generalised software package was developed for the application and analysis of DMC.

DMC was developed in the United States of America by Shell Oil Company (Cutler and Ramaker, 1979; Prett and Gillete, 1979). This technique is based on a linear model prediction. The development of DMC marked the first time a control technique was devised which could successfully handle process constraints, thus allowing the realization of achievable profits via implementation of results of on-line constrained optimizations.

DMC falls under the class of controllers which is termed Model Predictive Control (MPC). The control algorithms falling into this category are multivariable and model-based, and as such are expected to improve control of processes which exhibit strong interactions, non-minimum phase behavior, and operate at constraints. Other control schemes falling into this category are Model Algorithmic Control (MAC) and Internal Model Control (IMC). Details of MAC are largely proprietary, while the frequency-based IMC method does not permit direct handling of constraints. Thus the focus of this project was on DMC algorithm and its variants; Linear Dynamic Matrix Control (LDMC) and Quadratic Dynamic Matrix Control (QDMC).

Model Predictive Control methods and in particular DMC and its extensions such as QDMC, have generated an increasing interest in both academia and industry. This class of methods is still however, under development. Some of the industrial applications of MPC include implementation on fluid catalytic cracking units (Cutler and Hawkins, 1987; Caldwell and Dearwater, 1991), circulating fluidized bed combustor (Venugopal and Gupta, 1991), heat exchange network (Grimm et al., 1989), moderate- and high-purity distillation towers (McDonald and McAvoy, 1987), furnaces (Garcia and Morshedi, 1985), evaporator process in a kraft pulp mill (Ricker et al., 1989), and many more. The diverse applications of MPC demonstrate its flexibility in model specification and ease in performance specification.

The function of a grinding circuit in a mineral processing plant is to prepare the ore for concentration by liberating the valuable mineral from the waste minerals or for chemical reaction by increasing the surface area of the valuable mineral.

Grinding is an inherently energy-intensive process in which less than 10% of the total electrical energy input is utilised in the size reduction of ore particles (Rajamani and Herbst, 1991). Wills (1985) estimates that for a typical copper ore containing 0.6% metal, the total energy required to produce the metal is 11.0×10^3 kWh per tonne of metal, a third of which is consumed in the mill. On the other hand, the total energy requirement for primary iron from an ore of 24% metal is about 2.5×10^3 kWh per tonne of metal, of which the milling requirement is about 10% of the total. It is significant therefore that as ore grades decrease, mill energy consumptions could become the most important factor in deciding whether the ore is processed or not. It is for this reason that grinding circuits are of considerable interest to researchers in mineral processing industry, particularly with respect to their control.

Grinding circuits are multivariable in nature and are not very well understood because of complicated breakage and selection that occurs in the circuit. They also have large interactions and time delays which render single-loop control insufficient. Mill efficiency increases as its throughput increases. The optimum point has been found to be just before

the mill overloads, at which point it becomes inoperable (Lynch, 1977; Wills, 1985; Rajamani and Herbst, 1991). Therefore, for maximum profit, the control system has to operate near the constraint, again single-loop control has proven inadequate. DMC does not require a detailed explicit knowledge of the process. The process model is commonly derived empirically through step tests on the plant. DMC with its multivariable features, inherent interaction and time delay handling capabilities, and constraint handling features, appeared suitable for control of grinding circuits and as such was investigated. Following below is the summary of each chapter of the thesis.

Chapter 2 discusses in detail the theoretical background of DMC and MPC. A single-input single-output (SISO) formulation of DMC is derived in detail and the multi-input multi-output (MIMO) extension is also shown. The tuning parameters of DMC are listed and their impact on performance discussed. A constrained formulation of DMC, QDMC, is described and constraint equations derived. DMC is then compared with other MPC controllers namely, IMC and MAC.

Chapter 3 firstly defines grinding and gives a general background on grinding circuits. It also looks at the control of grinding circuits and the typical control objectives of grinding circuits that are encountered in practice.

In Chapter 4 DMC is applied to a linear milling model. The model is a representation of the number one East Driefontein grinding circuit. The performance of both the unconstrained and constrained DMC is evaluated through simulation studies. Basic DMC disturbance compensation is demonstrated and a feedfoward controller is designed into the DMC algorithm to counter the disturbance.

Since the basic DMC algorithm assumes the plant is linear, while most plants are nonlinear, Chapter 5 evaluates the performance of linear DMC to a nonlinear milling circuit model. The model used was based on a literature model derived from a laboratory scale test rig. The evaluation was also done through simulation studies.

Conclusions were then drawn and some recommendations made to possible future research. These are given in Chapter 6.

CHAPTER 2

THEORY AND FORMULATION OF DYNAMIC MATRIX CONTROL

2.1 BACKGROUND THEORY OF MODEL PREDICTIVE CONTROL

DMC falls under the class of controllers commonly refered to as Model Predictive Controllers (MPC). Garcia et al (1989) define MPC as a family of controllers in which there is a direct use of an explicit and seperately identifiable model. Amongst the design techniques emanating from MPC are Model Algorithmic Control (MAC), and Internal Model Control (IMC). In all the algorithms an explicit dynamic model of the plant is used to predict the effect of future actions of the manipulated variables on the output and thus the name Model Predictive Control. The future moves are determined by optimization with the objective of minimizing the predicted error (deviation between output and set point) subject to operating constraints. The optimization is repeated at each sampling time based on updated information/measurements from the plant.

Key features of MPC

MPC has performed reliably in its applications because of its

- multivariable features,
- stability properties,
- inherent deadtime compensation, and
- constraint handling capabilities.

The DMC formulation is furthermore in the time domain which makes it appealing to plant operators.

In its most general form, MPC is not restricted in terms of the model, objective function and/or constraint functionality. For these reasons, it is the only methodology that currently can reflect most directly the many performance criteria of relevance to the process industries and is capable of utilizing any available process model.

The MPC algorithm

The MPC methodology consists of the following elements:

- a linear model relating the manipulated variables and measurable disturbances to the outputs of interest,

- prediction of the outputs of interest over the future time horizon corrected via feedback,

- computation of future manipulated variable moves to make the prediction of the outputs and manipulated variables satisfy some performance criterion.

It is assumed that the model is open-loop stable, and that the algorithm is implemented in a sampled-data system.

At the present time *k* the behavior of the process over a horizon (P) is considered. Using a model, the process response to changes in the manipulated variable is predicted. The moves of the manipulated variables are selected such that the predicted response has certain desirable characteristics. Only the first computed change in the manipulated variable is implemented. At time *k+1* the computation is repeated with the horizon moved by one time interval. The "moving horizon" approach of MPC is depicted in Figure 2.1 below.

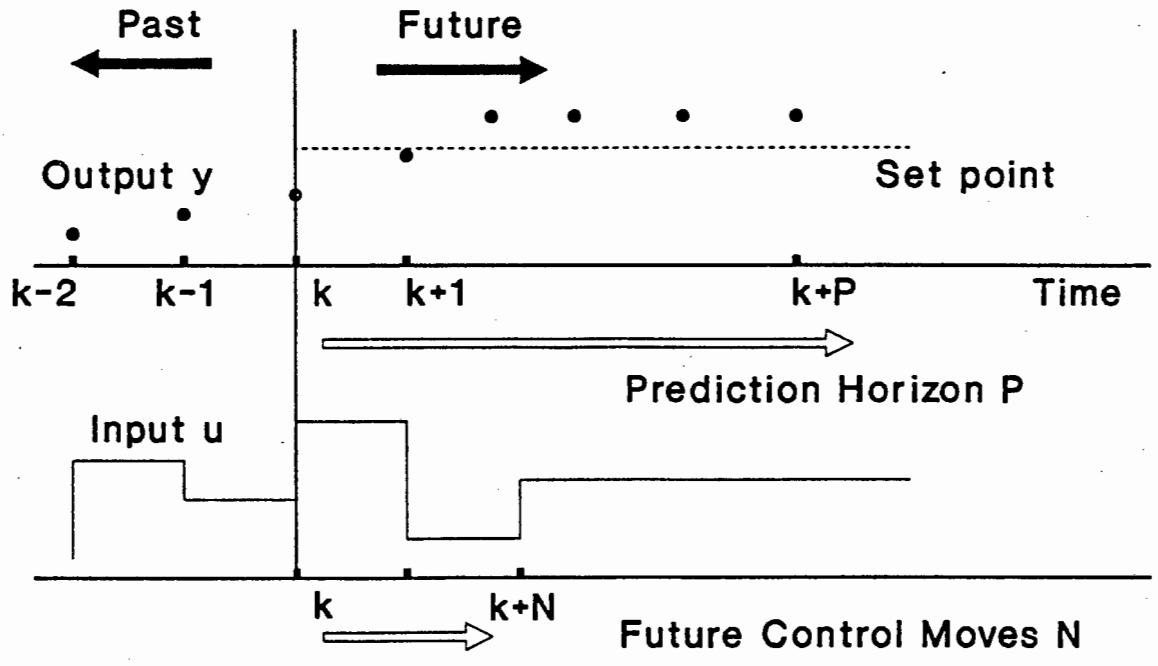


Figure 2.1 The "moving horizon" approach of MPC. N is the number of control moves and P is the prediction horizon.

2.2 The Formulation of Basic DMC

Model Representation

The DMC algorithm is based on the following linear step response model:

$$y(k) = \sum_{i=0}^{\infty} a_i \Delta u(k-i) + d(k) \qquad (2.1)$$

where $y(k)$ is the output at time k,
$\Delta u(k-i)$ is the change in the manipulated input at time k-i,
$d(k)$ is the unmeasured disturbance contribution,
a_i, i=1, 2, ..., are the open-loop step response coefficients.

One can truncate the sum after M steps which gives

$$y(k) = \sum_{i=0}^{M-1} a_i \, \Delta u(k-i) + a_M \, u(k-M) + d(k) \qquad (2.2)$$

where M is number of intervals required for the output to reach steady state after a step change in the input.

Equations (2.2) may be seperated into three terms:

$$y(k+1) = \sum_{i=1}^{l} a_i \, \Delta u(k+1-i) \qquad \text{effect of future moves}$$

$$+ \sum_{i=l+1}^{\infty} a_i \, \Delta u(k+1-i) \qquad \text{effect of past moves} \qquad (2.3)$$

$$+ \; d(k+1) \qquad \text{predicted disturbance}$$

where k denotes present time step.

Equation (2.3) can be compactly represented as:

$$\underline{y} = A \, \Delta \underline{u} + \underline{b} \qquad (2.4)$$

Here,

$\underline{y}$ is a P-vector of predicted outputs,

$$\underline{y} = \begin{bmatrix} y(k+1) \\ \vdots \\ y(k+P) \end{bmatrix}$$

$\Delta \underline{u}$ is an N-vector of future control moves,

$$\Delta \underline{u} = \begin{bmatrix} \Delta u(k) \\ \vdots \\ \Delta u(k+N-1) \end{bmatrix}$$

A is the dynamic matrix of step response coefficients,

$$A = \begin{bmatrix} a_1 & 0 & 0 & \cdots & 0 \\ a_2 & a_1 & 0 & \cdots & 0 \\ a_3 & a_2 & a_1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ a_N & a_{N-1} & a_{N-2} & \cdots & a_1 \\ \vdots & \vdots & \vdots & & \vdots \\ a_P & a_{P-1} & a_{P-2} & \cdots & a_{P-N+1} \end{bmatrix}$$

N is number of future control moves. We assumed that $\Delta u(k) = 0$ for $k > N$

P is the prediction horizon,

$\underline{b}$ is a vector representing the influence of past moves and disturbance predictions on $\underline{y}$, given by

$\underline{b} = \underline{y}^* + \underline{d}$ which follows directly from equation (2.3),

$\underline{y}^*$ is a P-vector of past outputs,

$$\underline{y}^* = \begin{bmatrix} y^*(k+1) \\ \vdots \\ y^*(k+P) \end{bmatrix}$$

$\underline{d}$ is a P-vector of the unknown disturbance,

$$\underline{d} = \begin{bmatrix} d(k+1) \\ \vdots \\ d(k+P) \end{bmatrix}$$

Equation (2.3) above requires a prediction of unmodelled effects of disturbance d(k). Since future values of disturbance are not available an estimate must be used. The current value of the disturbance can be estimated using the current feedback measurement y_m, together with equation (2.1). In the absence of additional knowledge of the disturbance

over future intervals, the predicted disturbance is commonly assumed to be equal to the present estimate:

$$d(k+l) = d(k) \qquad \text{for } l = 1 \text{ to } P$$

where $$d(k) = y_m - \sum_{i=1}^{\infty} a_i \, \Delta u(k-i)$$

Controller Algorithm - Solution of the DMC Equations

The DMC control problem is defined as finding the N future input moves so that the sum of squared deviations between the predicted output values and the setpoint are minimized. One can also include a penalty term on input changes. This is equivalent to the following least-squares problem:

$$\underset{\Delta \underline{u}}{\text{Minimize}} \quad \phi = (\underline{y}_s - \underline{y})^T Q \, (\underline{y}_s - \underline{y}) + \Delta \underline{u}^T \Lambda \, \Delta \underline{u}$$

$$\text{subject to:} \quad \underline{y} = A \, \Delta \underline{u} + \underline{b}$$

Solution to this minimization problem can be found from Garcia and Moshedi (1986) and is

$$\Delta \underline{u} = (A^T Q A + \Lambda)^{-1} A^T Q \, (\underline{y}_s - \underline{b}) \qquad (2.5)$$

where,

$\underline{y}_s$ is a P-dimensioned setpoint vector,
Q is (PXP) positive definite output weigthing matrix and
Λ is (NXN) positive definite matrix of move suppression factors.

Q and Λ are commonly taken to be diagonal matrices:

$$Q = \text{diag}(q_1, q_2, \ldots, q_P) \text{ and}$$
$$\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$$

Usually q_i and λ_i are taken to be constant at q and λ, respectively. These are useful in the Multi-Input Multi-Output (MIMO) case to weight outputs and inputs.

Equation (2.5) is the DMC equation. Only the first computed move is implemented. The computation is repeated at every sampling time when a new feedback measurement is obtained and used to update b. Failure to compute a move at each sampling time could impair the disturbance handling features of the algorithm (Garcia and Morshedi, 1986).

While the basic DMC formulation as seen above does not include feedforward control on known disturbances, this facility can be readily included in the algorithm. Taking equation (2.4) above, feedfoward control is accomplished by adding measured disturbances to the right hand side of the equation:

$$\underline{y} = A\,\Delta\underline{u} + \underline{b} + D\,\Delta x \qquad (2.6)$$

where D is a (PX1) dynamic matrix of the disturbance and
x is the disturbance input.

2.3 Multi-Input Multi-Output (MIMO) Systems

The MIMO case takes the same form as the single-input single-output (SISO) case with the input and output vectors defined as composites of the individual inputs and outputs, respectively. The appropriate definitions are derived below.

2.3.1 The DMC Formulation for MIMO Systems

The control algorithm for multivariable systems is derived in the same way as for the SISO case. For an r-output, s-input system, a linear dynamic representation is given by

$$y_i(k) = \sum_{l=1}^{\infty} \sum_{j=1}^{S} a_{ij}(l)\ \Delta u_j(k-l) + d_i(k) \quad \text{or}$$

$$\underline{Y}(k) = \sum_{l=1}^{\infty} \underline{\underline{a}}(l)\ \Delta\underline{U}(k-l) + \underline{d}(k) \qquad (2.7)$$

where

$$\underline{Y}(k) = \begin{bmatrix} y_1(k) \\ \vdots \\ y_r(k) \end{bmatrix} \qquad \Delta\underline{U}(k-1) = \begin{bmatrix} \Delta u_1(k-1) \\ \vdots \\ \Delta u_s(k-1) \end{bmatrix}$$

$$\underline{\underline{a}}(1) = \begin{bmatrix} a_{11}(1) & \cdots & a_{1s}(1) \\ a_{21}(1) & \cdots & a_{2s}(1) \\ \vdots & & \vdots \\ a_{r1}(1) & \cdots & a_{rs}(1) \end{bmatrix}$$

Equation (2.7) can be represented in a compact form analogous to that for SISO case i.e.

$$\underline{y} = A\,\Delta\underline{u} + \underline{b}$$

here,

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1s} \\ A_{21} & & \cdots & A_{2s} \\ \vdots & & & \vdots \\ A_{r1} & & \cdots & A_{rs} \end{bmatrix},$$

$$\underline{y} = \begin{bmatrix} y_1(k+1) \\ y_1(k+2) \\ \vdots \\ y_1(k+P) \\ \hline \vdots \\ \hline y_r(k+1) \\ y_r(k+2) \\ \vdots \\ y_r(k+P) \end{bmatrix}, \qquad \Delta\underline{u} = \begin{bmatrix} \Delta u_1(k) \\ \Delta u_1(k+1) \\ \vdots \\ \Delta u_1(k+N-1) \\ \hline \vdots \\ \hline \Delta u_s(k) \\ \Delta u_s(k+1) \\ \vdots \\ \Delta u_s(k+N-1) \end{bmatrix},$$

$\underline{b} = \underline{y}^* + \underline{d}$ where,

$$\underline{y}^* = \begin{bmatrix} y_1^*(k+1) \\ y_1^*(k+2) \\ \vdots \\ y_1^*(k+P) \\ \hline \vdots \\ \hline y_r^*(k+1) \\ y_r^*(k+2) \\ \vdots \\ y_r^*(k+P) \end{bmatrix}, \qquad \underline{d} = \begin{bmatrix} d_1(k+1) \\ d_1(k+2) \\ \vdots \\ d_1(k+P) \\ \hline \vdots \\ \hline d_r(k+1) \\ d_r(k+1) \\ \vdots \\ d_r(k+P) \end{bmatrix}$$

A_{ij} is the "Dynamic Matrix" relating output i to input j, its form corresponding to that of the SISO case,

$\underline{y}$, $\underline{y}^*$, $\Delta\underline{u}$ and $\underline{d}$ are composite vectors of outputs, past outputs, control moves and unmeasured disturbances, respectively.

As for the SISO case, the function to be minimized with respect to $\Delta\underline{u}$ is

$$\Phi = (\underline{y}_s - \underline{y})^T Q (\underline{y}_s - \underline{y}) + \Delta\underline{u}^T \Lambda \Delta\underline{u}$$

Here,

$Q = \text{diag}(q_1 \ldots\ldots q_1\ q_2 \ldots\ldots q_2 \ldots\ldots q_r \ldots\ldots q_r)$
Each q_i is repeated P times and corresponds to the different outputs.

$\Lambda = \text{diag}(\lambda_1 \ldots\ldots \lambda_1\ \lambda_2 \ldots\ldots \lambda_2 \ldots\ldots \lambda_s \ldots\ldots \lambda_s)$
Each λ_i is repeated N times and corresponds to the different inputs.

Solution to this minimization is

$$\Delta\underline{u} = (A^T Q A + \Lambda)^{-1} A^T Q (\underline{y}_s - \underline{b})$$

as before.

Again the formulation that includes feedfoward control is similar to that of the SISO case (equation 2.6) with D being possibly a composite matrix representing more than one disturbances and x being a composite vector of disturbances. In general, DMC can allow any number of feedforward disturbances and will yield zero offset for step-like inputs and disturbances.

2.4 The Formulation of QDMC

It is known that in practice the operating point of a plant that satisfies the overall economic goals of the process will generally lie at the intersection of constraints (Garcia et al, 1989). A succesful controller must therefore anticipate constraint violations and correct for them in a systematic way. Violations must not be allowed while keeping the operation of the plant close to these constraints.

A QDMC controller may be designed to:

(i) constrain manipulated variable changes to a maximum and/or minimum.

(ii) constrain the manipulated variables to a maximum and/or a minimum.

(iii) constrain controlled variables.

(iv) constrain uncontrolled variables.

As in the original DMC formulation the error between the setpoint and the predicted outputs over the time horizon is minimized over the future changes in the manipulated variables. The minimization of the predicted error is based on least-square (L_2) norm. The quadratic programming (QP) optimization technique allows for the minimization of a quadratic objective function subject to linear constraints; hence the acronym QDMC.

In practice, the moves computed in unconstrained DMC may not be implementable due to process operating limit violations. Three types of process constraints are usually encountered:

- manipulated variable constraints: e.g. valve saturation.
- controlled variable constraints: overshoots in the controlled variables past allowable limits must be avoided.
- constraints on associated variables i.e. key process variables which are *not* directly controlled but that must be kept within bounds.

In QDMC, we wish to add constraints to the minimization problem. The objective function may be solved as a QP of the standard form (Garcia and Moshedi, 1986):

$$\underset{\Delta \underline{u}}{\text{Minimize}}\ \Psi = 1/2\ \Delta \underline{u}^T\ H\ \Delta \underline{u}\ -\ \underline{g}^T\ \Delta \underline{u}$$

subject to: $C\ \Delta \underline{u}\ \geq \underline{0}$

where:

$H = (A^T\ Q\ A + \Lambda)$ (the QP Hessian matrix)

$\underline{g} = A^T\ Q\ (\underline{y}_s - \underline{b})$ (the QP gradient vector)

The above problem may be solved using a standard QP routine such as is available in the standard numerical software libraries. The formulation of constraints is discussed in the next section.

2.4.1 QDMC Constraint Equations

Manipulated variables

Bounds on manipulated variable changes can be included as follows:

$$\Delta u_{min} \leq \Delta u(k+l) \leq \Delta u_{max} \qquad \text{for } l=0, \ldots, N-1 \quad \text{(for SISO case)}$$

where, Δu_{min} is the minimum allowed manipulated variable change,

Δu_{max} is the maximum allowed manipulated variable change,

$\Delta u(k+l)$ is the manipulated variable change at time step k+l.

Similarly for the MIMO case:

$$\Delta \underline{u}_{min} \leq \Delta \underline{u}(k+l) \leq \Delta \underline{u}_{max} \qquad \text{for } l=0, \ldots, N-1$$

where $\Delta \underline{u}_{min}$, $\Delta \underline{u}(k+l)$, $\Delta \underline{u}_{max}$ are composite vectors of manipulated variable changes.

Manipulated variables can also be similarly bound. Consider first bounds on a single input, u:

$$u_{min} \leq u(k+l-1) \leq u_{max}, \quad l=1,\ldots, N \tag{2.8}$$

We need to represent this in terms of the input changes, $\Delta \underline{u}$, to be compatible with the above QP.

Recognizing that $u(k+l-1) = u(k-1) + \sum_{i=1}^{l} \Delta u(k+i-1)$

Equation (2.8) above can be written as:

$$I_L \, \Delta \underline{u} \geq (u_{min} - u(k-1)) \, \underline{1}$$

$$-I_L \, \Delta \underline{u} \geq (u(k-1) - u_{max}) \, \underline{1}$$

where $\underline{1} = (1\ 1\ 1\ \ldots 1)^T$ and I_L is an N X N lower triangular matrix:

$$I_L = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & 1 & 0 & & 0 \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ 1 & 1 & 1 & \ldots & 1 \end{bmatrix}$$

For s-inputs, these constraints become:

$$\begin{bmatrix} -I_L & & \\ & \ddots & \\ & & -I_L \\ I_L & & \\ & \ddots & \\ & & I_L \end{bmatrix} \Delta\underline{u} \quad \geq \quad \begin{bmatrix} (u_1(k-1)-u_{1,max})\,\underline{1} \\ \vdots \\ (u_s(k-1)-u_{s,max})\,\underline{1} \\ (u_{1,min}-u_1(k-1))\,\underline{1} \\ \vdots \\ (u_{s,min}-u_s(k-1)\,\underline{1} \end{bmatrix}$$

where $\Delta\underline{u}$ now represent the composite vector of future moves for all the inputs.

$$\Delta\underline{u} = \begin{bmatrix} \Delta\underline{u}_1(k) \\ \vdots \\ \Delta\underline{u}_1(k+N-1) \\ \hline \Delta\underline{u}_2(k) \\ \vdots \\ \Delta\underline{u}_2(k+N-1) \\ \hline \vdots \\ \hline \Delta\underline{u}_s(k) \\ \vdots \\ \Delta\underline{u}_s(k+N-1) \end{bmatrix}$$

Controlled variables

The QP can be made to prescribe moves so that projections of the controlled variable responses lie within bounds. The output variable can be bound using the same concepts described above. For a SISO system, with respective maximum and minimum limits y_{max} and y_{min}, the constraint equations are formulated as:

$$y_{min} \leq y(k+l) \leq y_{max}, \quad l=1,\ldots,P$$

which may be written in terms of $\Delta\underline{u}$ as:

$$\begin{bmatrix} -A \\ \hline A \end{bmatrix} \Delta\underline{u}(k) \geq \begin{bmatrix} \underline{b} - y_{max}\,\underline{1} \\ \hline y_{min}\,\underline{1} - \underline{b} \end{bmatrix}$$

This follows directly from the prediction equation (2.4), and is readily extended to multiple outputs.

Associated variables

As with controlled variables it is possible to have the QP keep projections of associated variables within limits. However, a new projection vector must be created since there is no setpoint on these variables (Garcia and Morshedi, 1986).

Controller software can be found in the Appendix. The programs are coded in standard Fortran-77 and run on the UCT Vax 6330.

2.5 Model Identification

The dynamic matrix is formed from step test data which may be obtained by applying a unit step change to the input and sampling the output as it changes with time until it reaches steady state. The dynamic matrix for the MIMO case is found in the same way; a unit step change is applied to one input while others are kept constant. Generally, a plant would be allowed to settle at a steady state before step tests or a new input change can be applied. However, Cutler and Yocum (1991) present a method whereby the process does not have to be at steady state to start the test or come to steady state during the testing, and more than one independent variable is allowed to change at the same time. They use the inverse of the Dynamic Matrix Control algorithm to identify the plant model. The time to steady state for each independent and dependent variable pair must be assumed and the step response curve analysed at a number of times to steady state to determine the best fit to the data. A good dynamic matrix or model of the plant will result in good control.

In reality, model errors exist for a number of reasons:

- The model is assumed linear when the process is nonlinear.

- The equipment degrades or is changed.

- Techniques used for identification are not accurate enough or the measurements are not sufficiently accurate.

Attempts have been made to improve model accuracy. Ogunnaike and Adewale (1986) have expanded the scope of the DMC algorithm for SISO systems to include time-varying delays and steady state gains. Li et al. (1989) have shown that a state space formulation of MPC with a Kalman filter is better than the standard MPC formulation. McDonald and McAvoy (1987) proposed a gain and time constant scheduling extension to DMC for control of a highly nonlinear distillation process.

2.6 Tuning Parameters in DMC

The objective of any controller is to find the moves of the manipulated variables which would make the output best match a target value y_s in the face of disturbances. Assuming the present time interval to be k^*, in DMC a projection of the output y(k) over P future time intervals (k^*+1 to k^*+P) is matched to the setpoint y_s by prescribing a sequence of future moves. The projection of the output to the future P time intervals commonly refered to as predictions, offers capabilities of off-line tuning of the controller yet unequalled by any other control scheme. It allows a control engineer to predict where the outputs will be in P time intervals to the future for any given set of tuning parameters. A control engineer can therefore compare plant performance for each set of tuning parameters and adjust them accordingly. This information is invaluable in situations when there are constraints on the control variable(s) which are to be strictly enforced. Control decisions which lead to constraint violations in the future can be detected and appropriately adjusted for ahead of time. In practice, it is unlikely that the plant will reach the predicted outputs since plant inputs often change before the end of the prediction horizon. Predictions will then be calculated on-line as well,

and be updated each time the inputs change to give an operator an idea of where the plant is going.

From the DMC formulation above, it is clear that as the number of control moves (N) increases, the more freedom DMC has in matching the output projections to the setpoint. In other words DMC produces tighter control although at the expense of larger moves. As N is increased, the moves get larger and often result in an unacceptable oscillatory response and if made even larger can render the system unstable (Garcia and Morari, 1982). In the DMC solution, an inverse of an NXN matrix has to be computed. The inverse matrix is often called a controller matrix. For an inverse to exist, the matrix must be nonsingular. An increase in N leads to poor conditioning of this matrix, and hence a greater chance of a reliable inverse not being able to be computed. On the other hand, if N is made small enough, stability can be guaranteed, unfortunately at the expense of performance. So there must be a balance between good performance and stability. It has been found that N should be about a quarter of M, the steady state horizon.

Garcia and Morari (1982) have proven that for a perfect model, stability for DMC is guaranteed if the prediction horizon P is greater than N+M-1. Garcia and Morshedi (1986) have found that stability is ensured for a perfect model, by selecting P such that the steady state effect of the most future move shows in the projections i.e P=N+M. Therefore, DMC is capable of handling non-minimum phase behaviour such as inverse response and dead-time.

The move suppression factor is used to restrict or suppress the magnitude of the input moves. As λ is increased, the input moves get more and more suppressed as can be deduced from the objective function. An increase in input penalties is equivalent to reducing controller gain and therefore improves the stability of DMC, particularly in the face of model inaccuracies (Morari, 1983). An increase in λ improves the conditioning of the inverse of the controller matrix. A move suppression of zero means that input moves are not penalized. λ also allows for control move scaling with respect to other manipulated variables.

In DMC, it is possible to give tighter control to a particular controlled variable or variables by increasing the relative weight of the corresponding least-squares residual. This is achieved by matrix Q described above. For equal weighting q for each output will have to be the same, and also Q cannot be 0.

2.7 Tuning of QDMC

All tuning parameters for DMC still apply for the constrained case. However, in QDMC control quality is further influenced by the selection of the projection interval to be constrained (Garcia and Morshedi, 1986). In practice only a subset of all P projections are constrained in the case of controlled and associated variables. This subset form a "constraint window" of future intervals of time over which QDMC will prevent violations from occuring.

In the presence of non-minimum phase behavior of controlled and associated variables much improvement in performance is achieved by moving the "constraint window" further down the horizon. The reason is that any projected violation inside the "constraint window" is handled rigorously by the QP, not unlike a tightly tuned controller. Therefore, if the QP is asked to correct for violations in the earlier projections, severe input moves might be required in the face of non-minimum phase characteristics. Further insight into stability under constraints can be found from Zafiriou (1991).

2.8 Model Forms in MPC

There are various models that can be used in the MPC algorithm. The most common in applications are sampled-data, linear, time-invariant (LTI) models which have been mentioned above. There are three types of LTI models used in MPC: state-space, transfer-function matrix, and finite step- (FSR) or pulse-response (FIR) models. Morari and Lee (1990) have used the state-space approach from which they show among other findings,

that the approach can be extended to processes with pure integrators. Although FSR and FIR models can be derived from state-space or transfer-function models, the usual practice in application is to identify an FSR or FIR model directly from the plant data (Ricker, 1991).

Alternatives to the above LTI models have been of considerable interest in nonlinear, time varying processes (Ricker, 1991). One approach is to combine a nonlinear, steady-state model with low-order LTI dynamic model. Another is to describe the plant by a set of nonlinear ODEs in continuous time. These can either be linearised and discretized repeatedly, or used directly in the formulation of a nonlinear programming problem. A third approach is to use a model in one of the LTI forms, but in which the parameters are assumed to be varying. An on-line parameter estimator then tries to track these variations so that the linear model represents the current operating point as well as possible (Ricker, 1991).

2.9 Comparison of IMC with DMC and MAC

Garcia and Morari (1982) found that the structure shown in Figure 2.2 below is inherent in all MPC and other control schemes. They referred to this structure as the IMC structure. IMC framework allows easy parameterization of all stabilizing controllers and has also been found to be useful for analyzing inherent limitations to control performance. For any given IMC controller, an equivalent classical feedback controller can be found. IMC however, does not permit direct inclusion of constraints as does QDMC.

MAC is distinctive from DMC in the following aspects:

(i) MAC uses an impulse rather than step response model. If the input is penalized in the quadratic objective, then the controller does not remove offset. This can be corrected by a static offset compensator (Garcia and Morari, 1982). If the input is not penalized then extremely awkward procedures are necessary to treat non-minimum phase systems (Mehra and Rouhani, 1980).

(ii) The number of input moves is not used for tuning i.e the number of input moves and prediction horizon are chosen to be same.

(iii) The disturbance estimate is filtered.

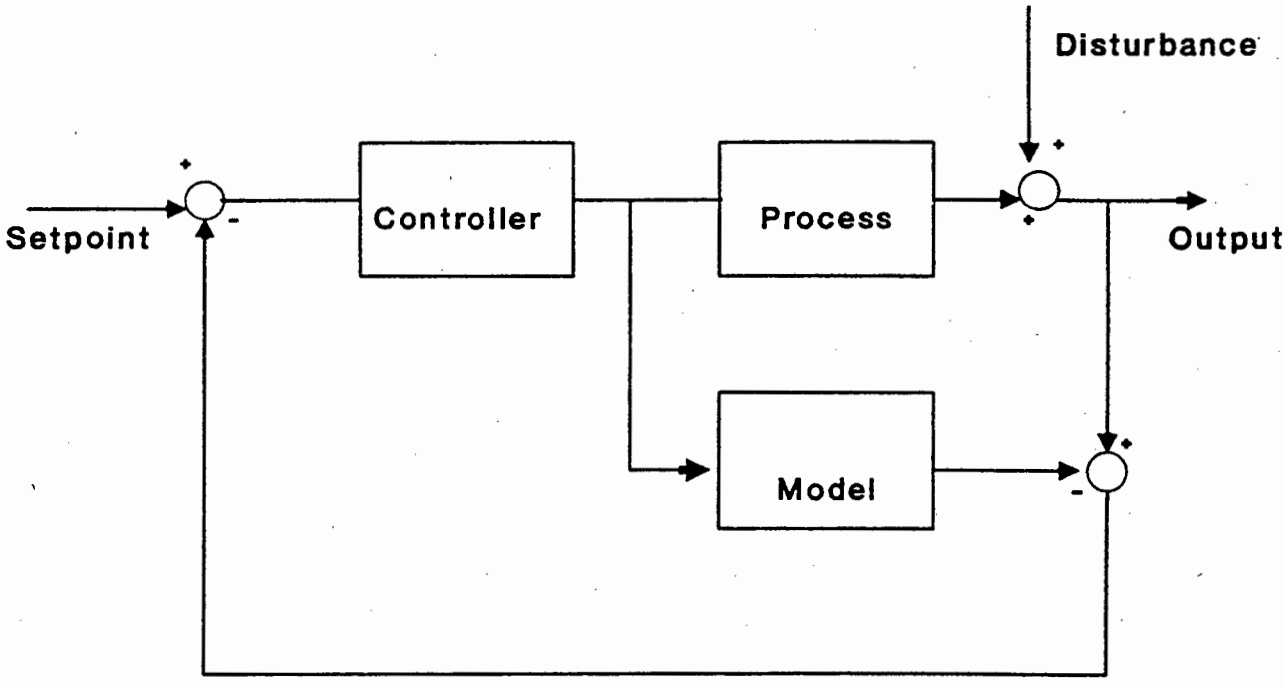


Figure 2.2 The IMC structure inherent in all MPC controllers.

It is for these reasons that DMC and its variants, QDMC and LDMC, found popularity amongst practitioners and academics alike. QDMC not only allows for explicit constraint handling but at the same time leaves the basic underlying DMC structure unchanged. LDMC is an optimal multivariable control algorithm which combines linear programming (LP) method with DMC. LDMC is not covered in the literature as much as QDMC. Therefore this study involved the use of DMC and QDMC.

CHAPTER 3

CONTROL OF GRINDING CIRCUITS

3.1 BACKGROUND THEORY OF GRINDING CIRCUITS

Mineral processing, sometimes called ore dressing, or milling, follows mining and prepares the ore for extraction of the valuable minerals. Liberation of the valuable minerals from gangue is accomplished by communition, which involves crushing, and if necessary grinding to such a particle size that the product mixture is of the correct size for subsequent seperation processes. The correct degree of liberation is a key to success in minerals industry (Wills, 1985). The valuable mineral should be freed from the gangue, but only just freed. Overground ore is wasteful since it needlessly consume grinding power and in some processes makes efficient recovery difficult to attain. On the other hand, inadequately ground ore makes the recovery of the mineral in subsequent separation processes low since some of it remains locked up in the ore. After minerals have been liberated from the gangue, the ore is subjected to some process of concentration, which essentially seperates the ore into pure streams; the valuable mineral and the waste mineral or gangue. Sometimes grinding is used to increase the surface area of the valuable mineral. In gold ore treatment for instance, leaching with cyanide solution follows the grinding process. Leaching is much more efficient on particles with large surface areas in relation to their mass (Wills, 1985). Figure 3.1 is a flowsheet which shows the sequence of operations in a typical mineral processing plant.

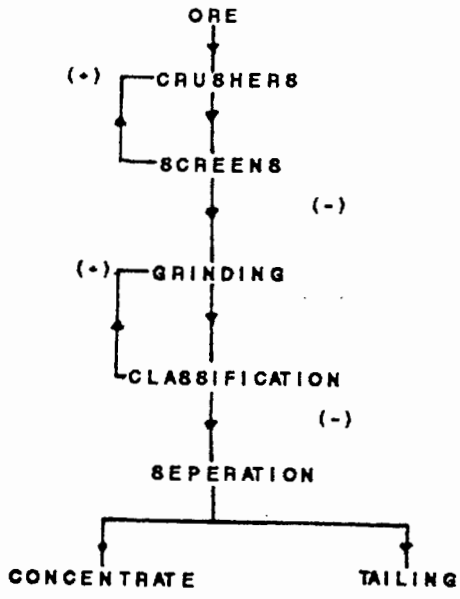


Figure 3.1 A flowsheet of a typical mineral processing plant.

Grinding is performed in tumbling mills. These contain charge of loose crushing bodies called the grinding medium, which is free to move inside the mill, thus comminuting the ore particles. The grinding medium may be steel rods (rod mills), balls (ball mills), hard rock (pebble mills) or the ore itself (autogenous mills). In the grinding process, particles between 5 and 250mm are reduced in size to between 10 and 300μm (Wills, 1985). The actual grinding mechanism is a complicated one. It can take place in several ways; it can happen by impact or compression, by chipping and by abrasion.

The function of a grinding circuit in a processing plant is to prepare the ore either for concentration by liberating the valuable minerals from the waste minerals or for chemical reaction by exposing the surfaces of the valuable minerals.

Grinding is an inherently energy-intensive process in which less than 10% of the total electrical energy input is utilised in the size reduction of ore particles (Rajamani and Herbst, 1991). Considering the fact that thousands of tonnes of ore are processed every day in a typical plant, the cost of operation of a grinding plant often dictates the overall cost of metal production. Hence, it is essential that such circuits be run as efficiently as possible. A ball mill draws a certain level of power just

to keep the ball charge in motion and a fraction thereof to grind the ore. Therefore the cost of operation is minimised by maintaining the ore feed rate to the mill at the maximum design capacity at all times. But at the same time the product from the circuit must meet the size specification for metal extraction effiency in subsequent processing.

Grinding circuits are divided into two broad classifications: open and closed circuits. In open circuits the material is fed into the mill at a rate calculated to produce the correct product in one pass. There is therefore no control on product size in these circuits, thus they are seldom used in mineral processing applications. Grinding in mineral processing industry is almost always in closed circuit (Wills, 1985), in which material of the required size is removed by a classifier, which returns oversize ore to the mill. Grinding circuits can be run either dry or wet, that is, solid only or a mixture of solids and water (slurry). Dry circuits have two major problems; dust from the mills is a major environmental hazard, and transportation and control are difficult since conventional equipment like pumps and flow measuring devices cannot be used, as a result, wet grinding circuits are more popular since they are easier to control automatically.

3.2 Selection of Control Variables

In implementing control for grinding circuits, the control objective is first defined, which may include:

1. the sizing analysis of the circuit product is to be maintained constant at constant feed rate;

2. the sizing analysis of the circuit product is to be maintained constant at maximum feed rate; or

3. both sizing analysis and solids content of the circuit product are to remain constant

Figure 3.2 shows a typical grinding circuit with some of the important variables.

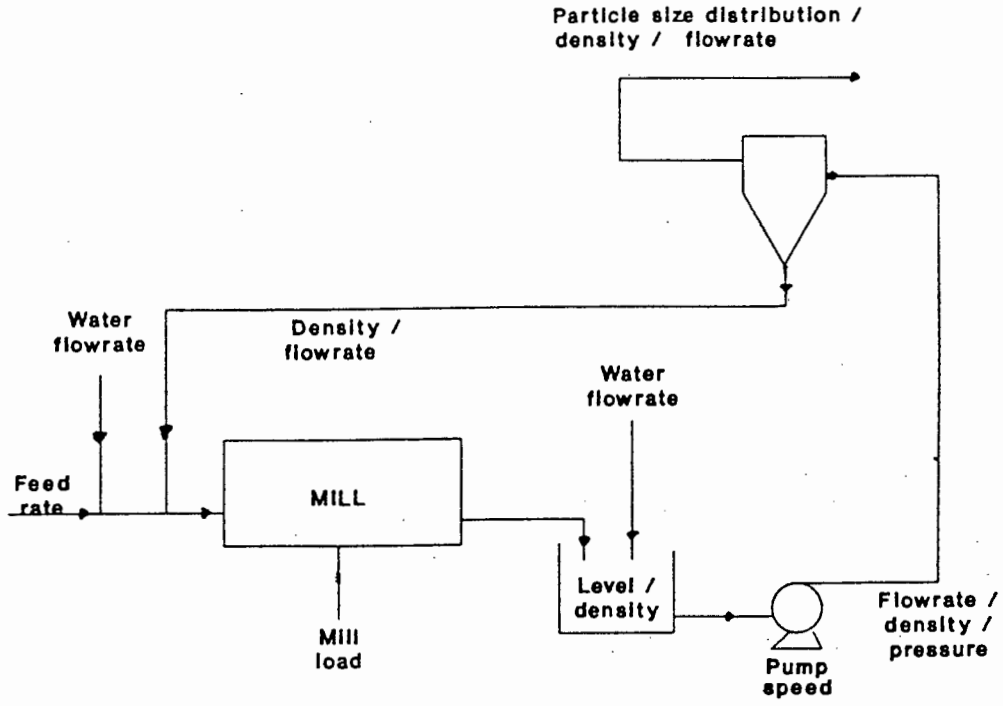


Figure 3.2 A typical grinding circuit with some important variables.

In practice more complex grinding circuits are found. They generally have more of the same type of equipment for better efficiency and maximum throughput. For instance, there are often two cyclones in series to increase seperation efficiency. Their underflow streams would generally feed to another mill which grinds this material to required size and pours it out to the sump to be classified once more.

Table 3.1 lists some of the possible controlled and manipulated variables.

Table 3.1 Commonly used controlled and manipulated variables in a ball mill grinding circuit.

Controlled variable	Manipulated variable
(i) Mill throughput rate	(i) Fresh feed solids rate
(ii) Mill discharge density	(ii) Fresh water rate
(iii) Cyclone feed density	(iii) Sump water rate
(iv) Cyclone mass feed rate	(iv) Pumping rate
(v) Sump level	
(vi) Overflow product particle size	
(vii) Mill power	

Water to the mill is often coupled to the ore feed rate, and as a result is not often regarded as an independent variable. Pumping rate or the variable speed pump is often viewed as a variable that provides the conditions under which control objectives can be achieved rather than as a variable that actually achieves them. The sump is often on a level control which is a local control. This leaves only two manipulated variables; feed rate of the ore and sump water addition rate.

There are other important variables which affect the circuit performance but which cannot be controlled or modelled. These are:

- ore feed hardness,
- ore feed sizing analysis.

Ore feed rate and feed size disturbances can be countered by adjusting the water additions to the circuit, but ore hardness variations can cause drastic reduction in mill throughput. Hardness of the ore varies considerably, depending on the location from which it was mined (Rajamani and Herbst, 1991). Owing to ore hardness disturbances, the mass rate of recycled stream and the fineness of the circuit product tend to fluctuate and so the objective of the computer control strategy is to counteract these disturbances in an optimum sense.

To determine the dynamic effect of ore hardness and feed sizing analysis on circuit performance is difficult to carry out in an industrial environment (Lynch, 1977). A step change in ore hardness could be achieved by making a transfer from one ore bin to another which had been previously stocked with a different type of ore. However, the bin discharge characteristics would almost certainly produce some unplanned variation in feed sizing analysis making it difficult to separate the effects of changes in the two variables.

Grinding circuits are commonly controlled by a series of single loop PID-controllers. These controllers gradually vary the process variables until setpoint is reached. Disturbances to the process tend to persist for a long time due to large interactions that exist among the process variables. Advanced PID-control schemes have been implemented by workers

like Braae and Hulbert (1981) with reasonable success over the conventional single loop PID-controllers. This control scheme essentially tries to decouple the process variables through a suitable compensator, K(s), with single loop PID controllers designed for the decoupled system. Since K(s) is designed from the model of the process which is usually not a perfect representation of the actual process, in general some interactions still exist which degrade the quality of control. Since these control schemes view even a multivariable control problem as a series of single-loop control problems, manipulated variables have to be matched to dependent variables which they will have to control. Rajamani and Herbst (1991) list some of the commonly encountered pairings:

Type I

Mill throughput rate controlled by sump water rate, Product size controlled by fresh feed rate.

Type II

Mill throughput rate controlled by fresh feed rate, Product size controlled by sump water rate.

Depending on the control objective, a particular type of pairing is chosen. For the first listed control objective, only sump water addition rate can be manipulated, and the variations in the classifier overflow density and volumetric flowrate must be tolerated if there is a change in the process. In many applications, the second control objective is often wanted and either type I or II pairing can be used depending on the required speed of the response of the system. Details on the choice of pairing can be found from Wills (1985) and Lynch (1977).

However, DMC does not require pairing or decoupling of the system since the control problem is solved as a full multivariable problem. This makes the algorithm rather easy to implement.

In Chapters 4 and 5, the suitability of DMC for milling circuit control is investigated through simulation studies. In Chapter 4, a linear model

is used to represent a plant, whereas in Chapter 5, the plant is represented by a nonlinear model.

CHAPTER 4

CONTROL OF A LINEAR MILLING CIRCUIT MODEL

4.1 MODEL DESCRIPTION

The milling model used in this study is a representation of the number 1 milling circuit at East Driefontein Gold Mine given by Hulbert and Braae (1981).

The circuit has a rod mill and two parallel pebble mills in closed circuit, and there are two stages of classification by hydrocyclones. The mills discharge into a primary sump, from which slurry is pumped to a pair of primary cyclones. The underflows of these cyclones are fed to the pebble mills, and the overflows go to a secondary sump. Slurry is pumped from the secondary sump to a cyclone, where it is split into an overflow, which is the product of the circuit, and an underflow that returns to the primary sump. The simplified diagram of the plant is given in Figure 4.1 below.

Selection of Control Variables

Three inputs were selected for controlling the plant. These are:

(i) RMFD, feed rate of water to rod mill [kg/min],

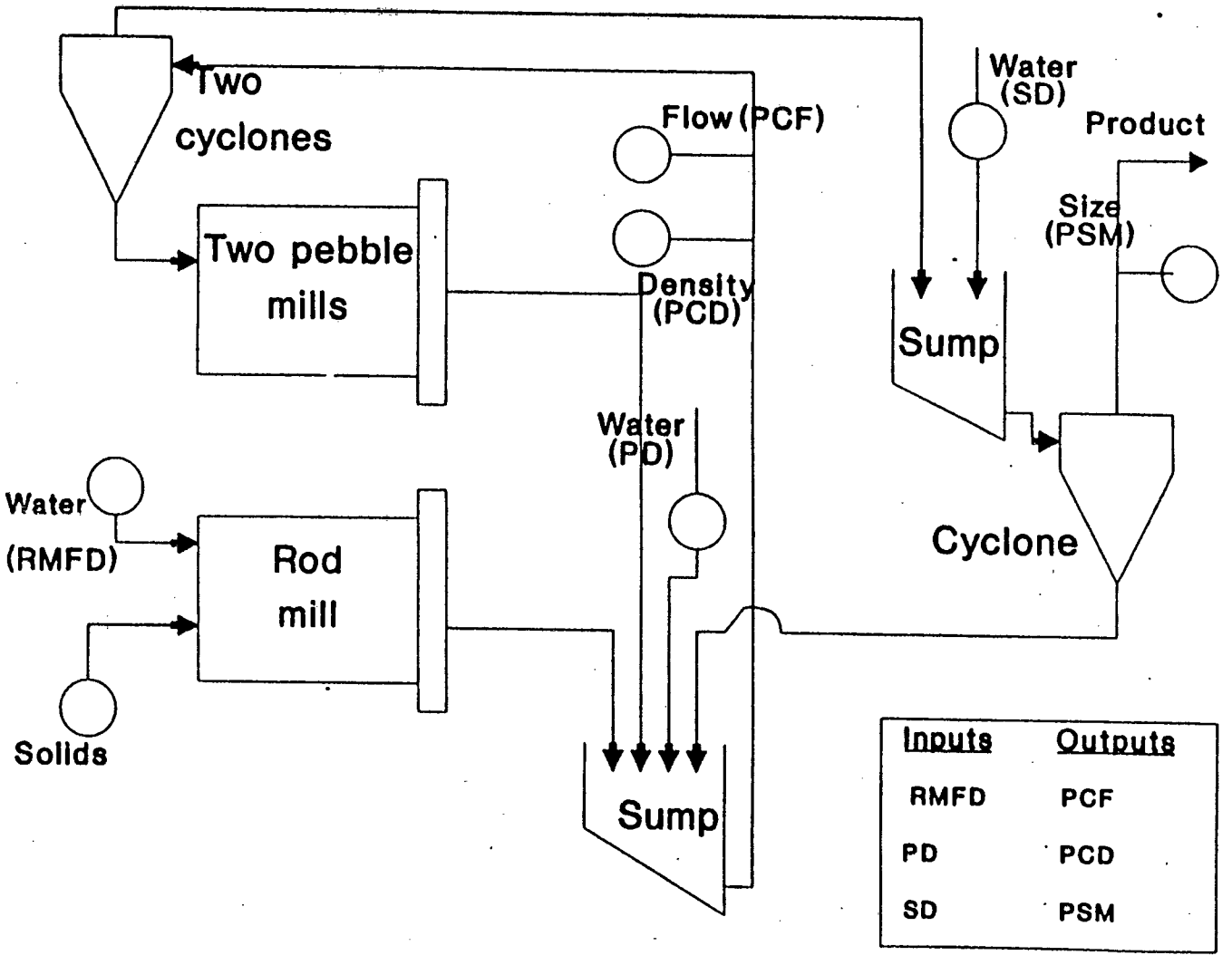


Figure 4.1 The number one East Driefontein Gold Mine milling circuit.

(ii) PD, the dilution water to the primary sump [kg/min],
(iii) SD, dilution water to the secondary sump [kg/min].

Three output variables could be selected for control since there are three inputs. Hulbert and Braae (1981) argue that controlled variables should determine the state of operation of the circuit; variables relating to circulating load, classification conditions, and the product stream were included.

The controlled plant outputs were then:

(i) PCF, flow of feed to primary cyclone [kg/min],
(ii) PCD, density of feed to primary cyclone [kg/m^3],
(iii) PSM, percentage particle size.

Dynamic Model Formulation

Step tests were done to get the dynamic response of the plant. The feed rates of ore and water to the circuit were controlled in fast-acting analogue-control loops. The proportion of solids to water in the feed to the rod mill was controlled automatically by a ratio controller.

The linear dynamic model of the plant (below) expressed in Laplace transforms shows the transfer function relating the inputs and outputs of the plant. Because the model is linear, the input and output variables are expressed as deviations from steady-state.

$$\begin{bmatrix} \Delta PCF \\ \Delta PCD \\ \Delta PSM \end{bmatrix} = \begin{bmatrix} \frac{6.72}{29.43s+1} & \frac{1.257}{1.022s+1} & \frac{0.1866}{9.55s+1} \\ \frac{80.9}{32.97s+1} & \frac{-3.61}{1.228s+1} & \frac{0.854}{10.90s+1} \\ \frac{-5.25e^{-15.7s}}{17.65s+1} & \frac{(2.95s+0.255)e^{-3.3s}}{5.81s^2+5.03s+1} & \frac{(2.04s+0.066)e^{-1.3s}}{7.31s^2+5.48s+1} \end{bmatrix}$$

$$X \begin{bmatrix} \overline{\Delta RMFD} \\ \overline{\Delta PD} \\ \overline{\Delta SD} \end{bmatrix} + \begin{bmatrix} \dfrac{1.006-10.133s}{1+114.17s+2166.67s^2} \\ \dfrac{4.57-133.167s}{1+62.50s+930.556s^2} \\ \dfrac{(-0.00406+12.22s)e^{-6.5s}}{1+38.67s+369.44s^2} \end{bmatrix} X \overline{\Delta PEB}$$

PEB represents pebble loading into the mill.

4.2 Program Description

The computer routines to integrate these models were written in standard Fortran-77. The main software development at this stage of study was to generalise the model subroutine to take any model of any size with and without time delays. The process model above is in the Laplace domain. Realization of each transfer function element i.e the transfer function that relates an input to an output, was performed in order to get the A, B, and C matrices of each element. The realization was done using the realization program in the CONSYD package. Each element could then be expressed in state-space representation of the form:

$$\frac{d\underline{x}}{dt} = A\underline{x} + Bu$$

$$y = C\underline{x}$$

where $\underline{x}$ represents a vector of states,
u represents the input and
y represents the output.

The differential equations were numerically integrated using the fourth-order Runge-Kutta method. Each individual input contribution to the output was then calculated by multiplying the new states by the C matrix.

Each output was obtained by adding the contributions of each of the inputs. Since there are time delays associated with some of the elements in the transfer function matrix, the corresponding input contribution to the output is suppressed by the number of time steps corresponding to the time delay associated with it. Figure 4.2 shows how the model routine fits into the simulation algorithm with the controller algorithm generated in Chapter 2. The computer programs used are listed in the Appendix.

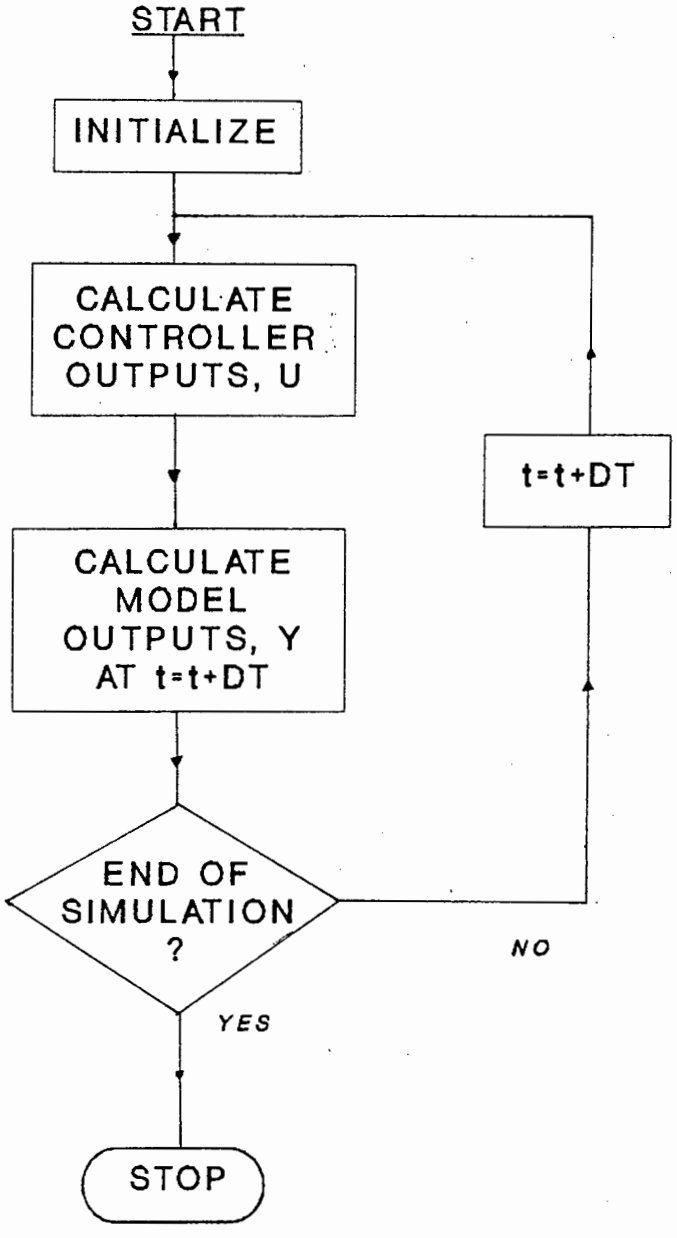


Figure 4.2 The flowsheet showing the main blocks of the simulation algorithm.

4.3 Results and Discussion

4.3.1 Open-loop Response of the Process

The open-loop responses of the process model to unit step changes in the inputs are shown in Figure 4.3. As can be seen, the process is characterised by a combination of fast and slow dynamics, time delays, and strong interactions between the variables. We would thus expect it to be a suitable candidate for DMC. In order to adequately represent the system, i.e. to capture even the fastest dynamics of the system, the sampling time (ΔT) was reduced to 0.3 min. Reducing the sampling rate even further showed insignificant improvements in control quality. The integration time step was taken to be 0.02 min. Further reduction of integration time step gave no improvement in the accuracy of the results and stability of the method used. The steady state appears to be at about 100 min.

4.3.2 Closed-loop Response of the Process

Based on the steady state of 100 min and ΔT of 0.3 min, the steady state horizon, M, is 330 data points. A control move horizon (N) of 80 and prediction horizon (P) of 410 were used in accordance with the rules of thumb stated in Chapter 2. Move suppression Λ was set at 0, and output weigthing matrix Q was set at equal weigthing corresponding to the identity matrix I.

A setpoint change of 5.0 kg/min in PCF was made at time 0.0, another of 10.0 kg/m^3 in PCD after 75.0 min, and finally a 2% change was made in PSM at 150.0 min. The simulation results with a DMC controller are shown in Figure 4.4. These results show good setpoint tracking with a reasonably low level of interactions. These preliminary results demonstrate the effectiveness of DMC for milling circuit control. Figure 4.5 show the corresponding input trajectories.

Control moves of 80 and prediction horizon of 410 results in large vectors and matrices to be stored and solved at each time step. This

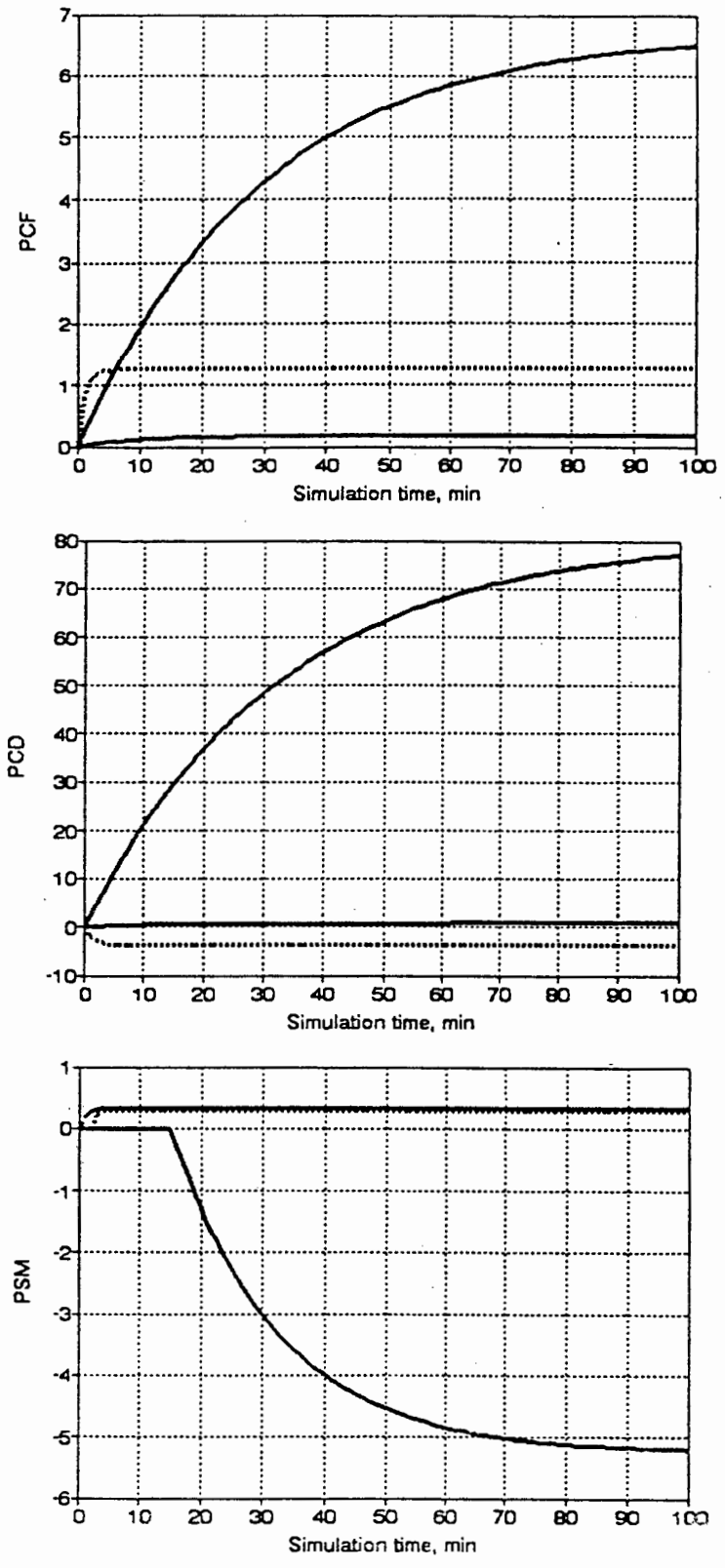


Figure 4.3 The open-loop response of the process to unit step changes in the inputs. Solid curves correspond to a unit step change in RMFD, dashed curves to PD, and dotted curves to SD.

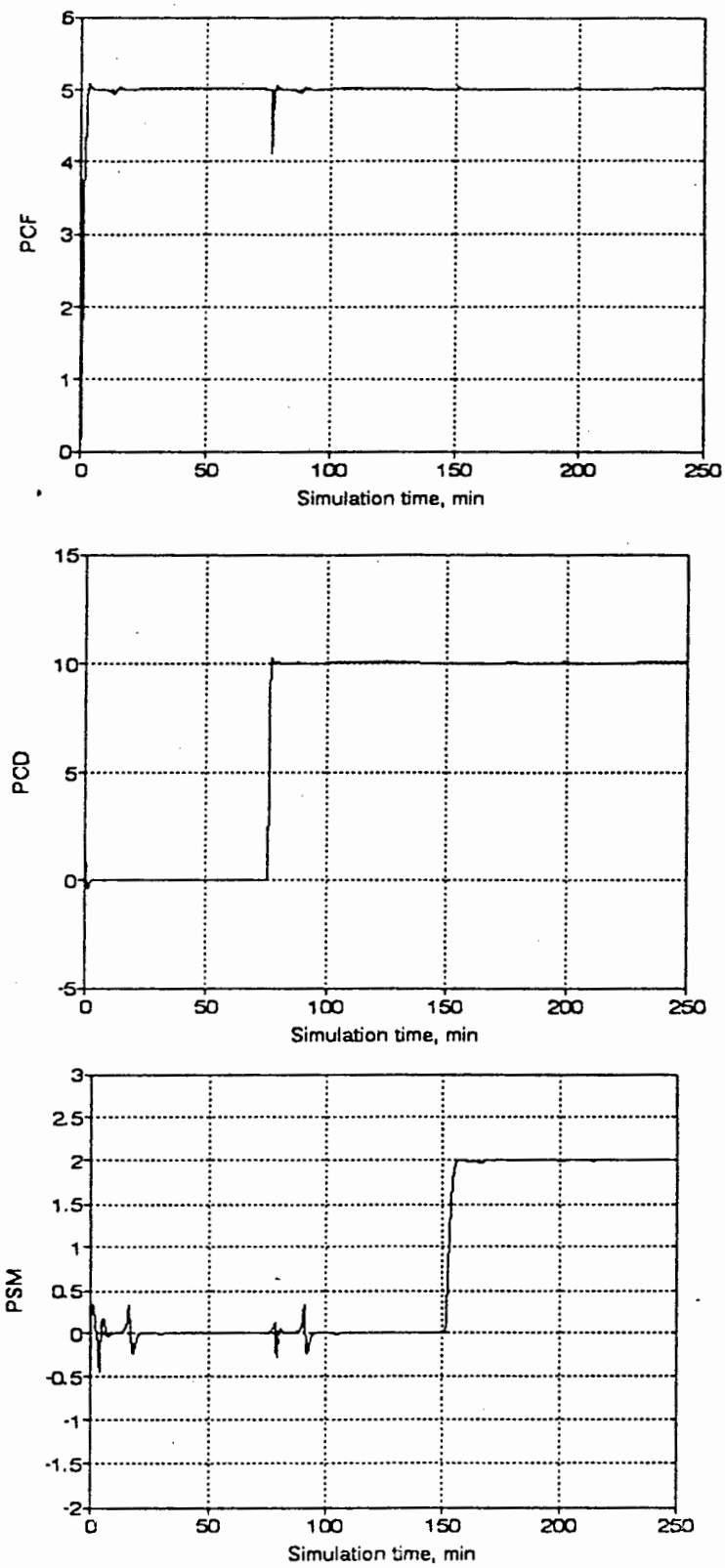


Figure 4.4 Process outputs of the unconstrained Dynamic Matrix Control of the milling circuit with M=330, N=80, P=410, Q=I and Λ=0.

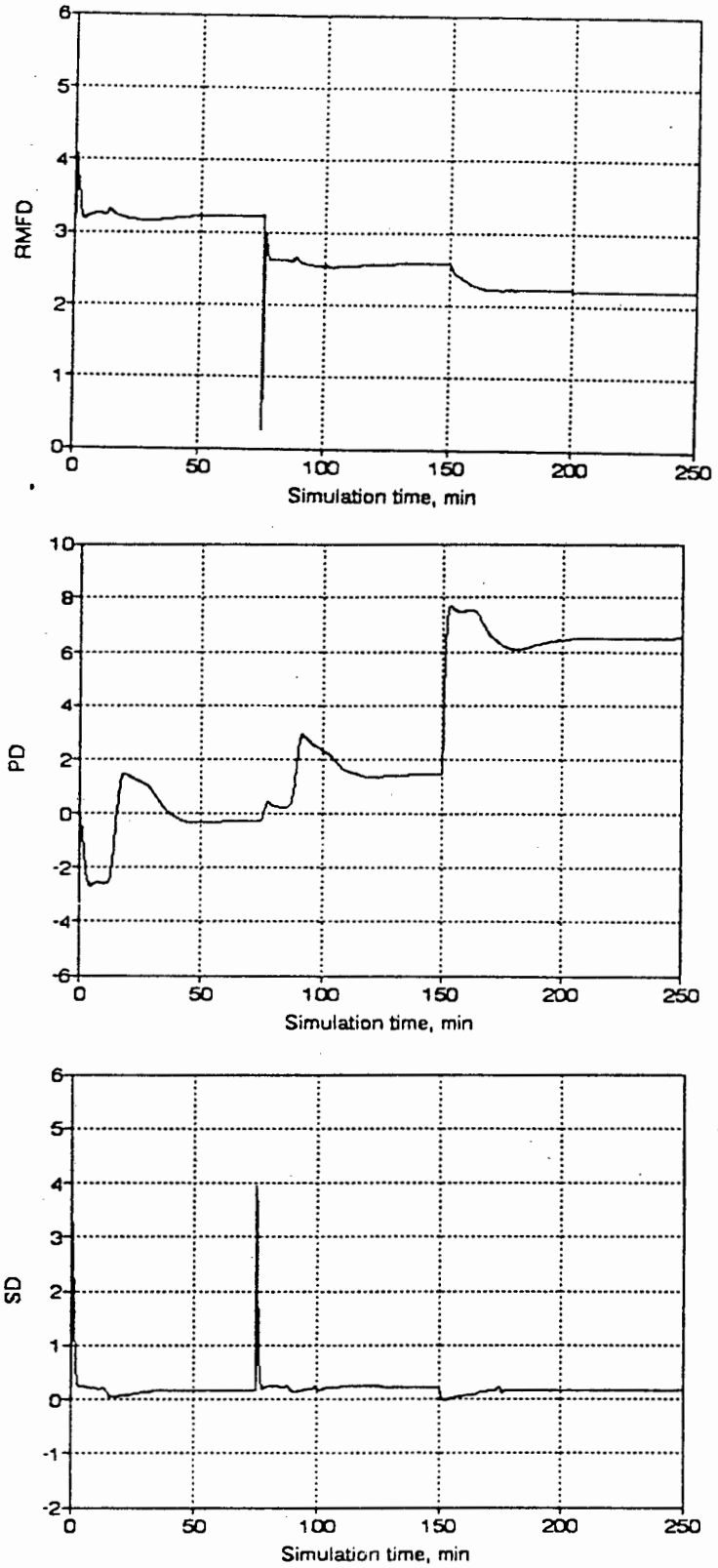


Figure 4.5 Process inputs of the unconstrained Dynamic Matrix Control of the milling circuit with M=330, N=80, P=410, Q=I and Λ=0.

required large memory space and resulted in very high computational load, increasing the CPU time for simulations to 17 min. This problem raises serious implications with real-time control using DMC, particularly for larger systems. Although the CPU time included calculating the inverse of the controller matrix which is normally solved once and stored, the solution was unconstrained and therefore with constraints the processing time might have been the same or even longer.

However, there are various ways to solve the problem. The first type of solution is to use a more powerful computer in terms of memory space and speed. A stand alone fast computer with large memory should suffice since one of the large calculations, the inverse of the controller matrix can be done off-line. As mentioned already, the controller matrix is solved once off-line and the inverse used for on-line control, unless one changes the tuning parameters in which case a new controller matrix will have to be solved.

The second type of improvement is to improve the DMC controller. A multi-rate sampling could be used to reduce vector and matrix sizes. Standard DMC assumes a global sampling rate which result in global M and P horizons. The M and the P used are for the slowest system to get to steady state. This means that one is compelled to collect extra unnecessary data for systems that fast reach steady state. For instance, the open-loop responses of this system shows that some responses reach steady state in as little as 10 min, but because of the global sampling rate, one is forced to store the next 90 min of repetitive data. Multi-rate sampling has a potential of reducing this problem. The method essentially proposes a smaller sampling rate for variables with fast dynamics and a larger sampling rate for variables with slow dynamics.

One other method is the programming style. Probably the largest devourers of memory are multi-dimensional arrays (matrices). Where possible uni-dimensional arrays must be used with pointers to mark the different data, at most there should be two-dimensional arrays. This programming style alone resulted in large savings in memory when applied in the programs.

The fourth type of method which again was used in this study, is to increase the sampling time. This is probably the first step to solving

computational load and memory problems; that is to keep the necessary data to a minimum. The sampling time was increased from 0.3 min to 1.0 min then to 2.0 min. Two minutes resulted in an unacceptable closed-loop response but 1.0 min showed a satisfactory closed-loop response. A sampling time of 1.0 min meant that M=100, and with N=20, P was made 120. This improvement resulted in a reduction in memory consumption and the CPU time dropped to three minutes. An even better closed-loop response was found with output weighting Q=(I10,I1,I10) and is shown in Figure 4.6. The response is not quite as good as that shown in Figure 4.4, but nevertheless this was found to be a practical solution in terms of CPU time. These parameters were then used throughout the simulation tests. Figures 4.7 show the corresponding inputs of the system.

From these results, it seems that it is not critical to obtain the full representation of the plant by basing the sampling rate on the fastest response. DMC seems robust enough to produce the required performance even with a relatively poor model. This aspect of DMC will be investigated in more detail in the next chapter.

4.3.3 DMC Disturbance Rejection Capabilities

The pebble loading disturbance in the above model was used to investigate disturbance rejection capabilities of DMC. The basic DMC algorithm assumes that unmeasured disturbances are step-like at the process outputs and takes the necessary steps to reject such disturbances as discussed in the previous chapter. The closed-loop system rejects step disturbances with no offset for as long as the system is stable.

Unfortunately, in practice most disturbances follow a ramp type of response. The open-loop response to the unit step change in disturbance (PEB) with the manipulated inputs held constant can be seen from Figure 4.8. It can be seen that some responses are more of a ramp type than a step and that the PSM response has also a time delay. One might therefore expect a poor DMC response to this disturbance. Figure 4.9 however, shows that a step estimate was sufficient to reject a step of 5.0 kg/min in disturbance applied at time zero. It took less than 45 min for the plant to settle back to setpoint.

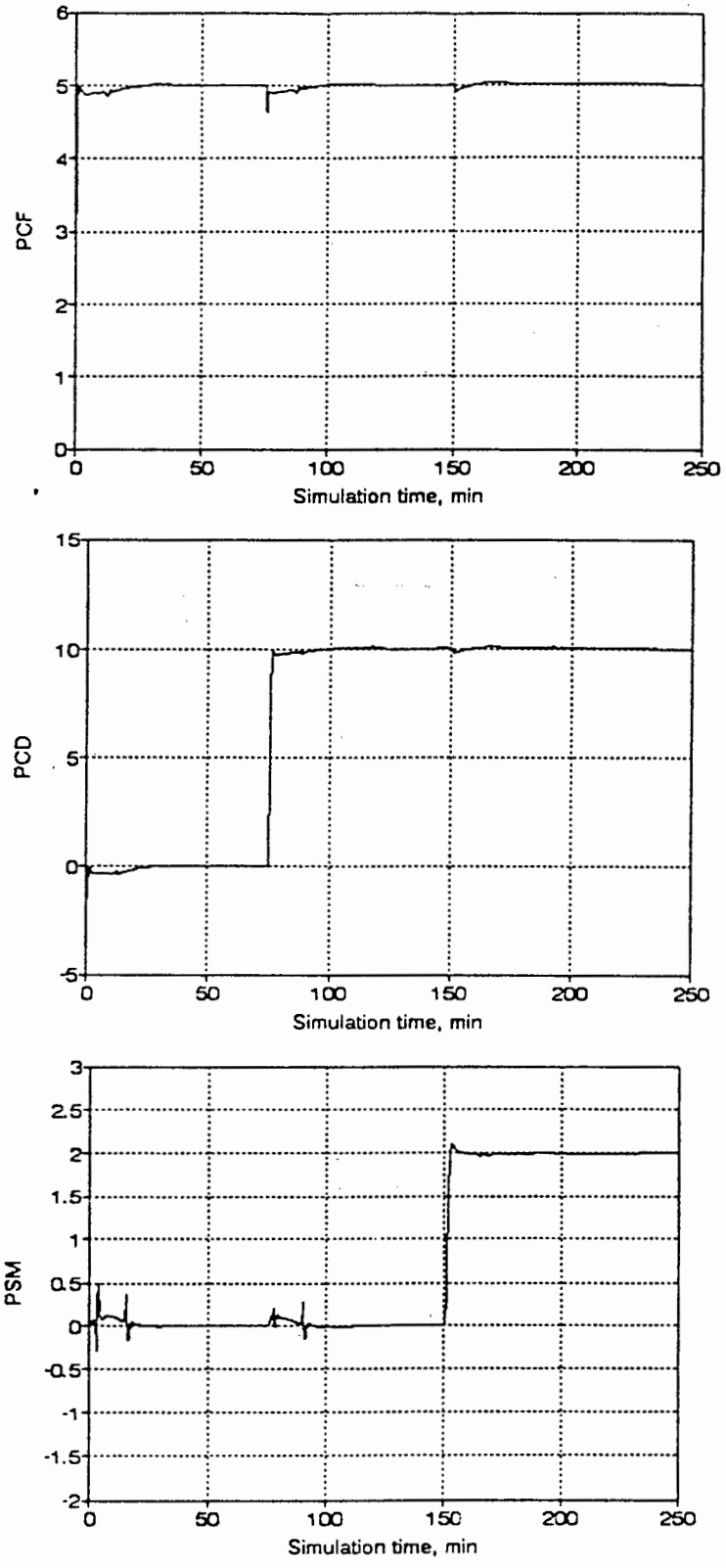


Figure 4.6 Process outputs of the unconstrained Dynamic Matrix Control of the milling circuit with M=100, N=20, P=120, Q=(10I,1I,10I) and Λ=0.

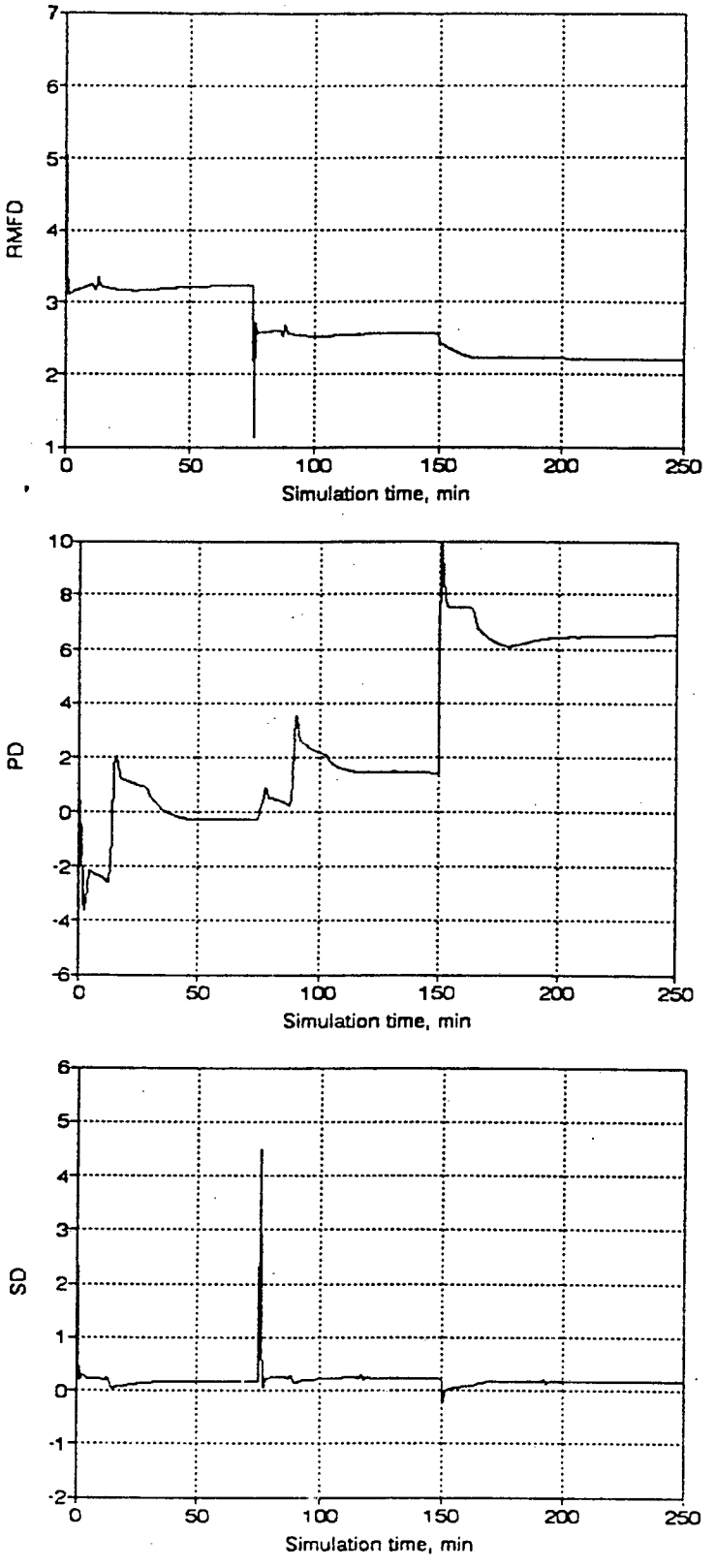


Figure 4.7 Process inputs of the unconstrained Dynamic Matrix Control of the milling circuit with M=100, N=20, P=120, Q=(10I,1I,10I) and Λ=0.

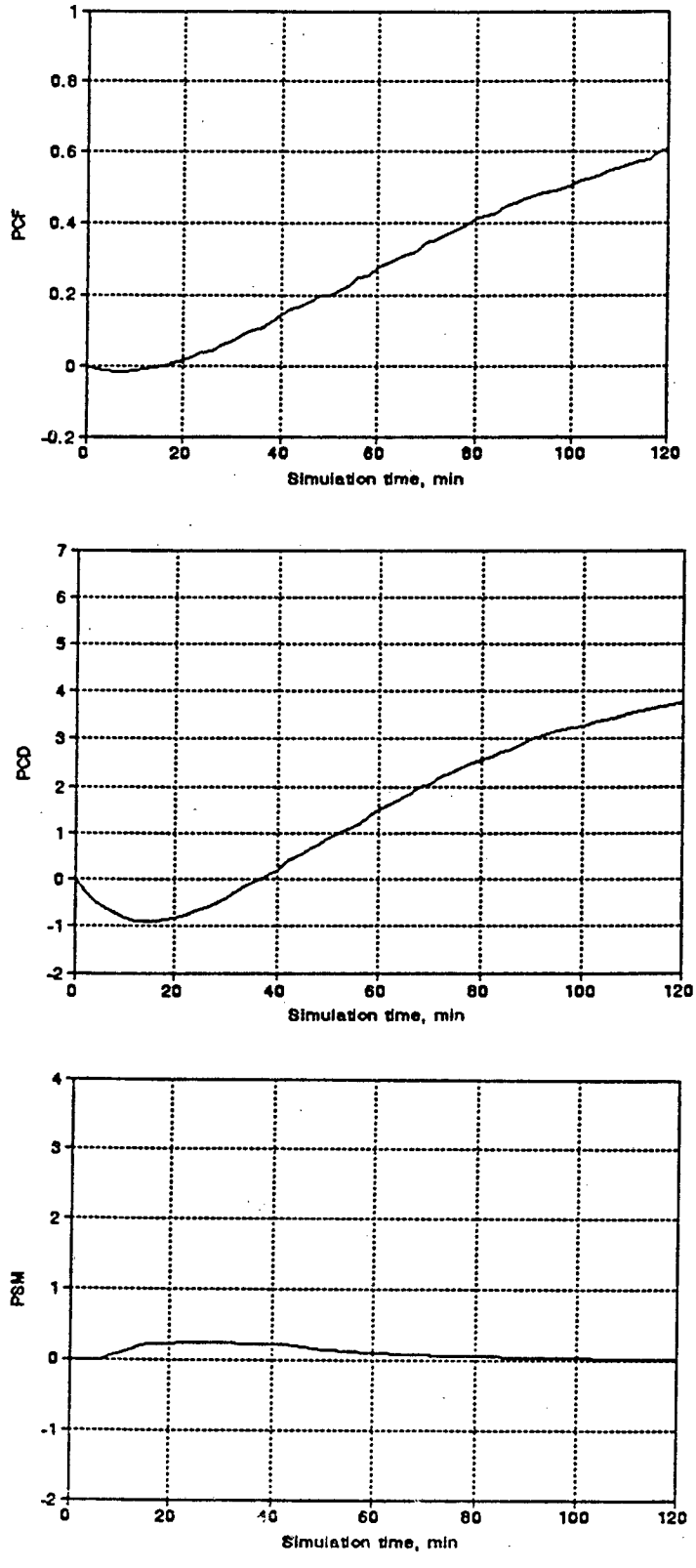


Figure 4.8 The open-loop response of the process to a unit step change in the disturbance, PEB.

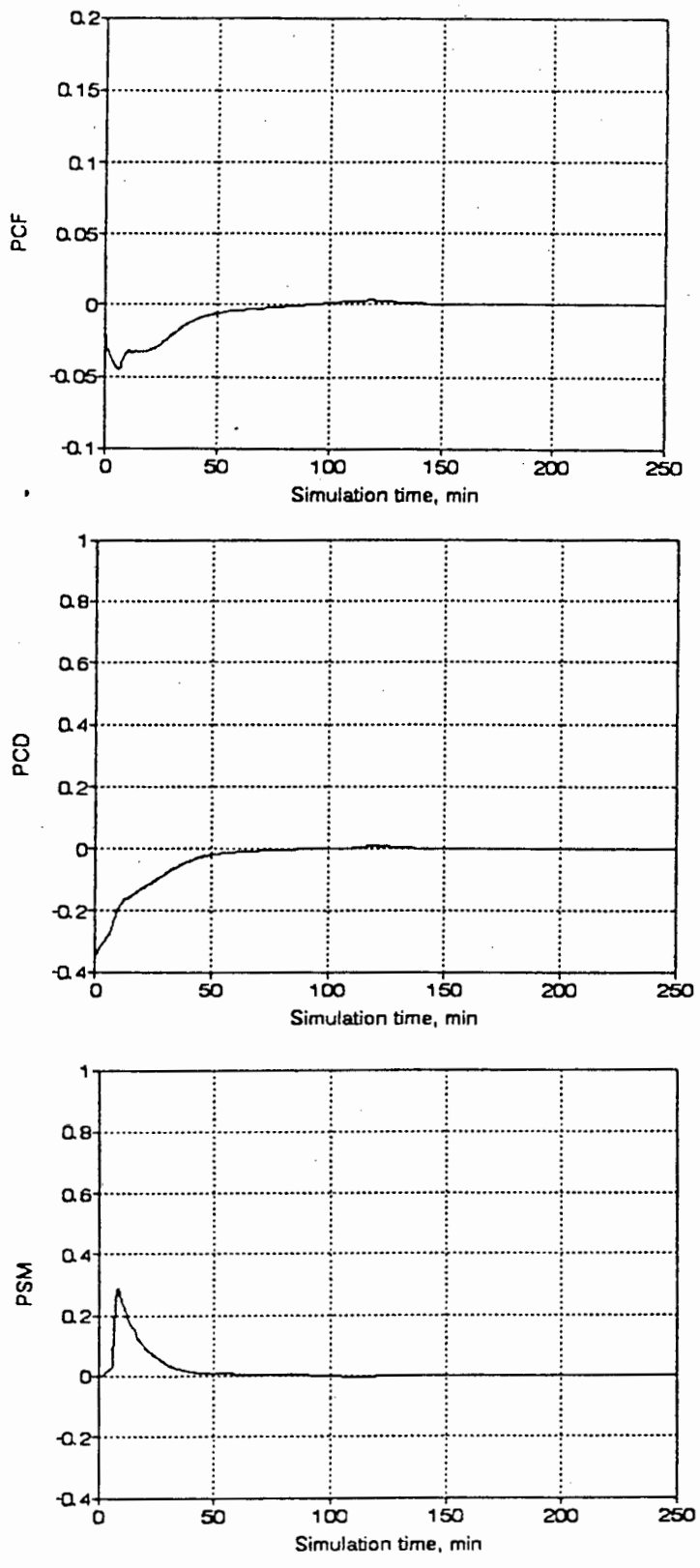


Figure 4.9 Process outputs of the unconstrained Dynamic Matrix Control of the milling circuit after a disturbance of 5kg/min applied at time zero. M=100, N=20, P=120, Q=(10I,1I,10I) and Λ=0.

Since there is full information about the disturbance, a feedfoward controller can be succesfully implemented (Cutler and Ramaker, 1979). Morari and Lee (1991) have proposed some improvements on disturbance estimation. They assume that disturbances are ramp like at the process output which is true when disturbances enter process output through slow dynamics as these do. They propose this assumption rather than the step assumption.

The feedfoward controller matrix was obtained from the open-loop disturbance responses in Figure 4.8, and was formulated in the same way as the dynamic matrix. The response with a feedforward controller can be seen from Figure 4.10. Even though the feedforward controller is exactly the disturbance model, the response shows some deviation from setpoint. This is because the disturbance and the process have different time delays. It should be noted that just like the process dynamic matrix, the disturbance matrix can represent any shape of response, so that it is unnecessary to make any assumptions about the type of disturbance to the process, as long as one gets the true dynamics of it.

4.3.4 Constrained DMC Solution

The unconstrained DMC solution is not always implementable or acceptable. It is also often required to place operational limits in order to protect the equipment, and also to prevent unsafe plant operation. Therefore a number of simulations were carried out with various constraints imposed, in order to demonstrate DMC capabilities in the face of process constraints.

Assuming that the primary dilution can only be allowed to increase by up to 7.0 kg/min or decrease by only 1.0 kg/min from its steady state value and likewise the secondary dilution allowed only to vary by 1.0 kg/min from its steady state value. Figure 4.11 shows the simulation results when these input constraints were imposed on the process:

$$-1.0 \leq PD \leq 7.0$$
$$-1.0 \leq SD \leq 1.0$$

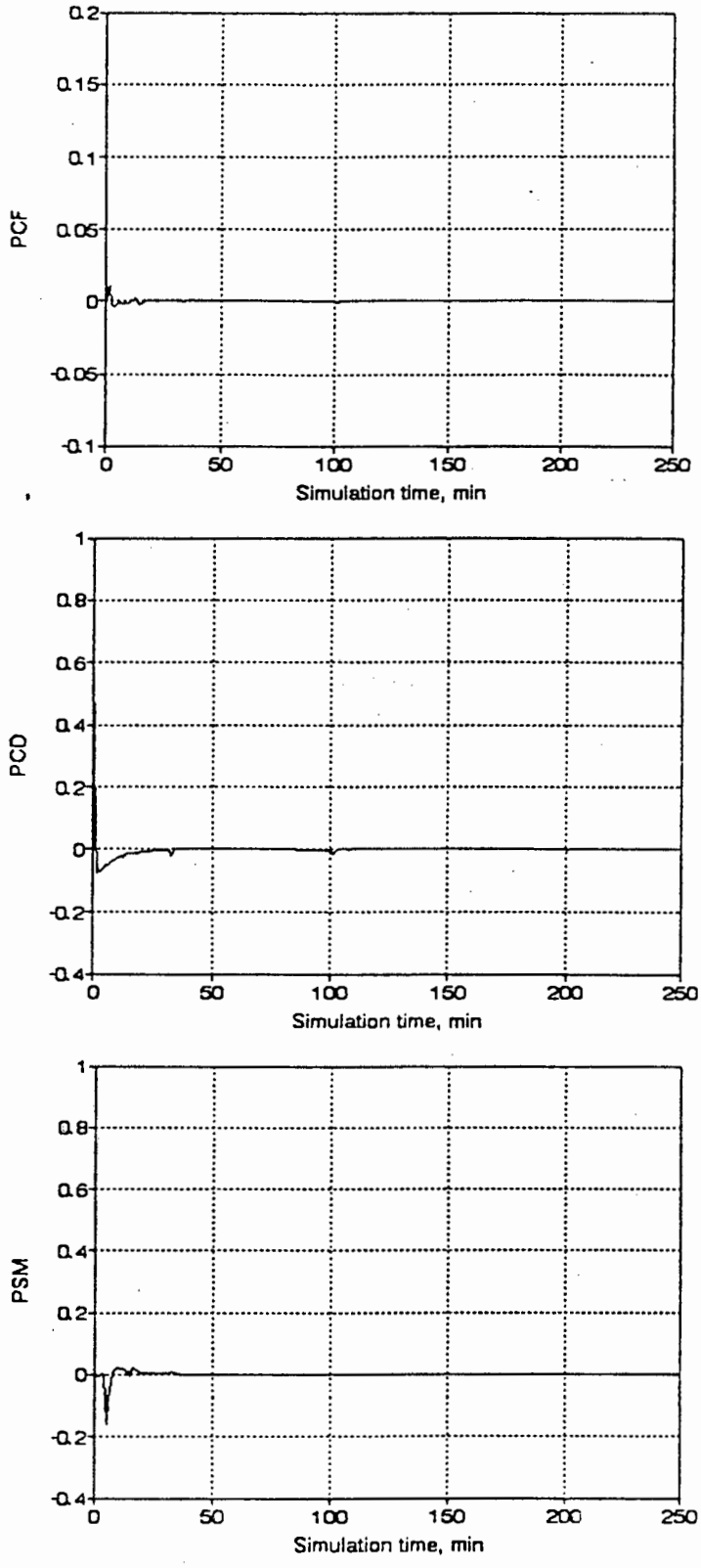


Figure 4.10 Process outputs of the unconstrained Dynamic Matrix Control of the milling circuit with a feedforward controller after a disturbance of 5kg/min applied at time zero. M=100, N=20, P=120, Q=(10I,1I,10I) and Λ=0.

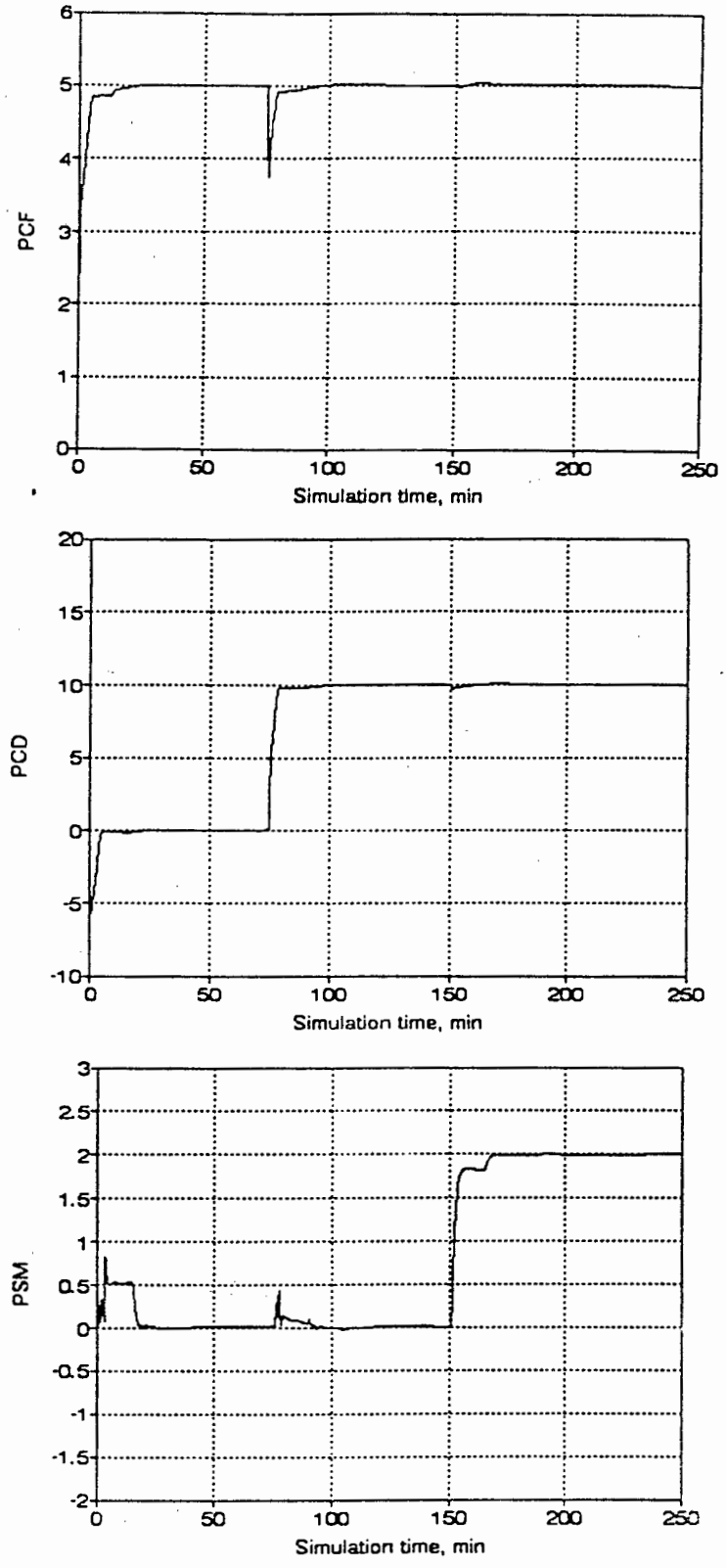


Figure 4.11 Process outputs of a constrained Dynamic Matrix Control of the milling circuit. Constraints are: ($-1.0 \leq PD \leq 7.0$) and ($-1.0 \leq SD \leq 1.0$). M=100, N=20, P=120, Q=(10I,1I,10I) and Λ=0.

Comparison with Figures 4.5 shows that the performance deteriorates since the inputs are limiting. Figure 4.12 show the corresponding inputs of the process.

Constraints were also imposed on the input changes to prevent violent unrealistic changes or changes which may upset plant operation. Figures 4.13 and 4.14 show the simulation results when the following constraints imposed on the primary dilution which was before making changes as large as 5.0 kg/min/min:

$$-1.0 \leq \Delta PD \leq 1.0$$

Again the response deteriorates since these constraints bound the changes in primary dilution to no more or less than 1.0 kg/min/min.

Constraints were also applied to the outputs. PCF was not allowed to deviate more or less than 0.01 [kg/min] from its steady state value. The disturbance was then implemented at time 0.0 without the feedforward controller. As can be seen from Figure 4.15, the process violated the lower constraint. Even though the predicted outputs satisfy the constraints, we observe that unknown disturbances may cause the actual outputs to violate the constraints. However, as was discussed above, if it is possible to identify possible disturbances to the plant, a feedforward controller can be readily and successfully applied on those disturbance inputs. Indeed, if one looks at simulation results with feedforward controller, the constraint would not have been violated since the unconstrained response does not reach this constraint anyway.

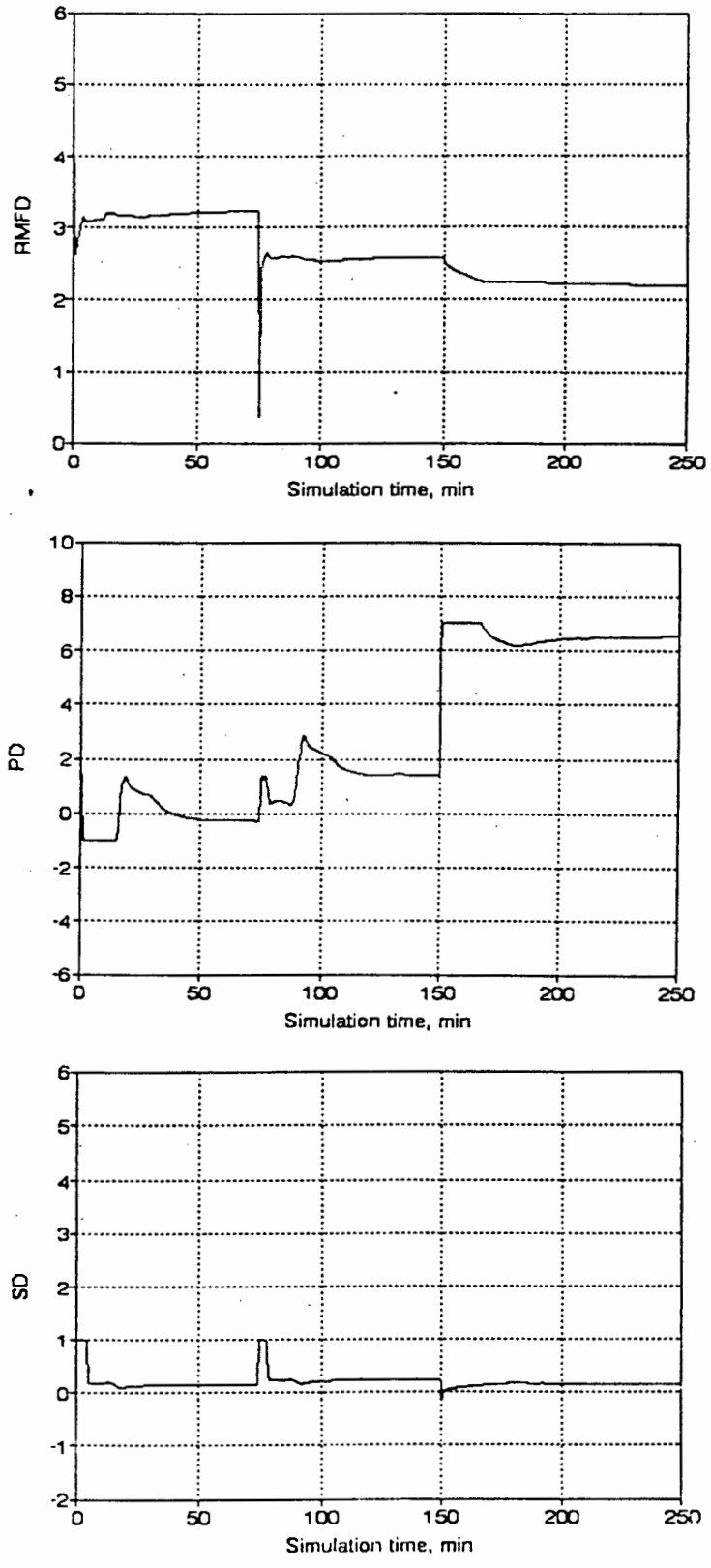


Figure 4.12 Process inputs of a constrained Dynamic Matrix Control of the milling circuit. Constraints are: ($-1.0 \leq PD \leq 7.0$) and ($-1.0 \leq SD \leq 1.0$). M=100, N=20, P=120, Q=(10I,1I,10I) and Λ=0.

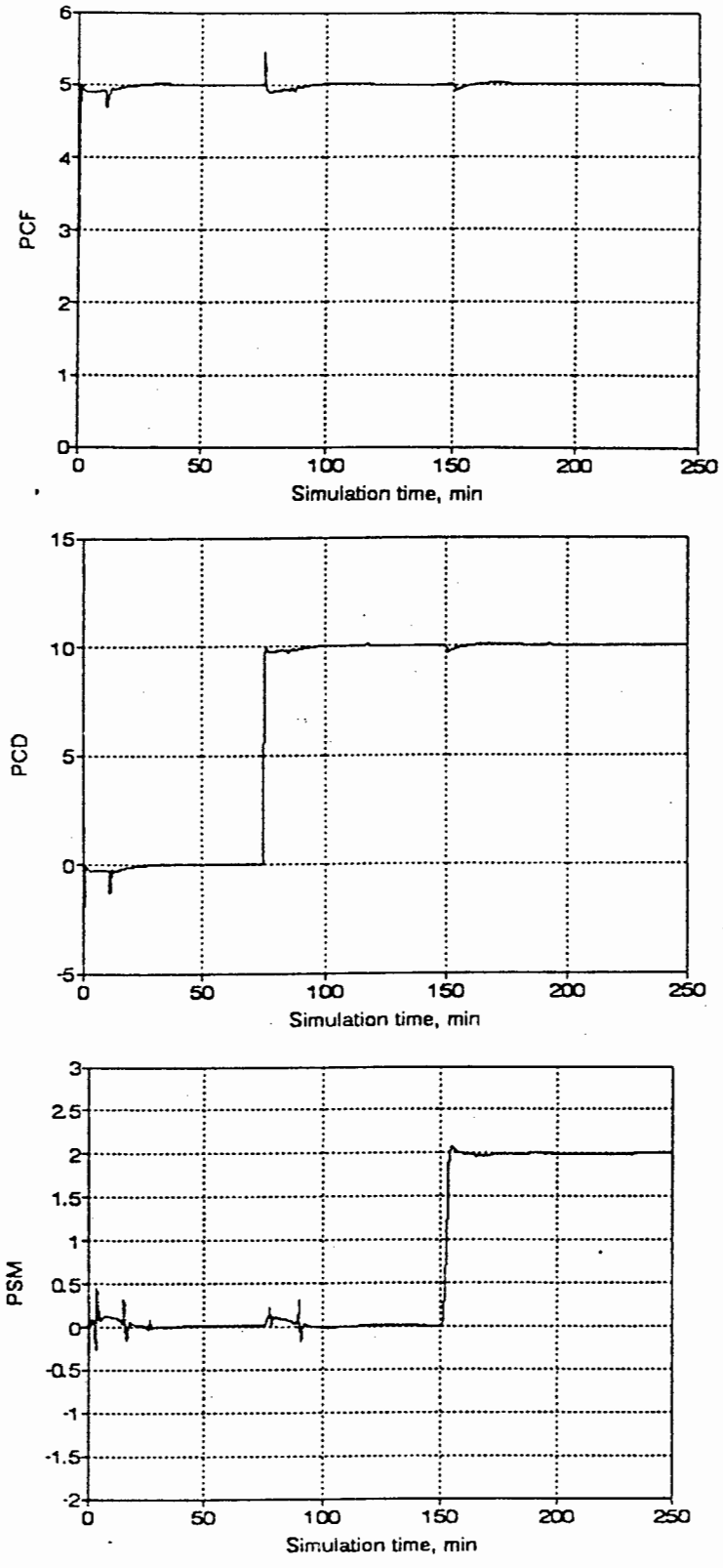


Figure 4.13 Process outputs of a constrained Dynamic Matrix Control of the milling circuit. Constraints are: ($-1.0 \leq \Delta PD \leq 1.0$). M=100, N=20, P=120, Q=(10I,1I,10I) and Λ=0.

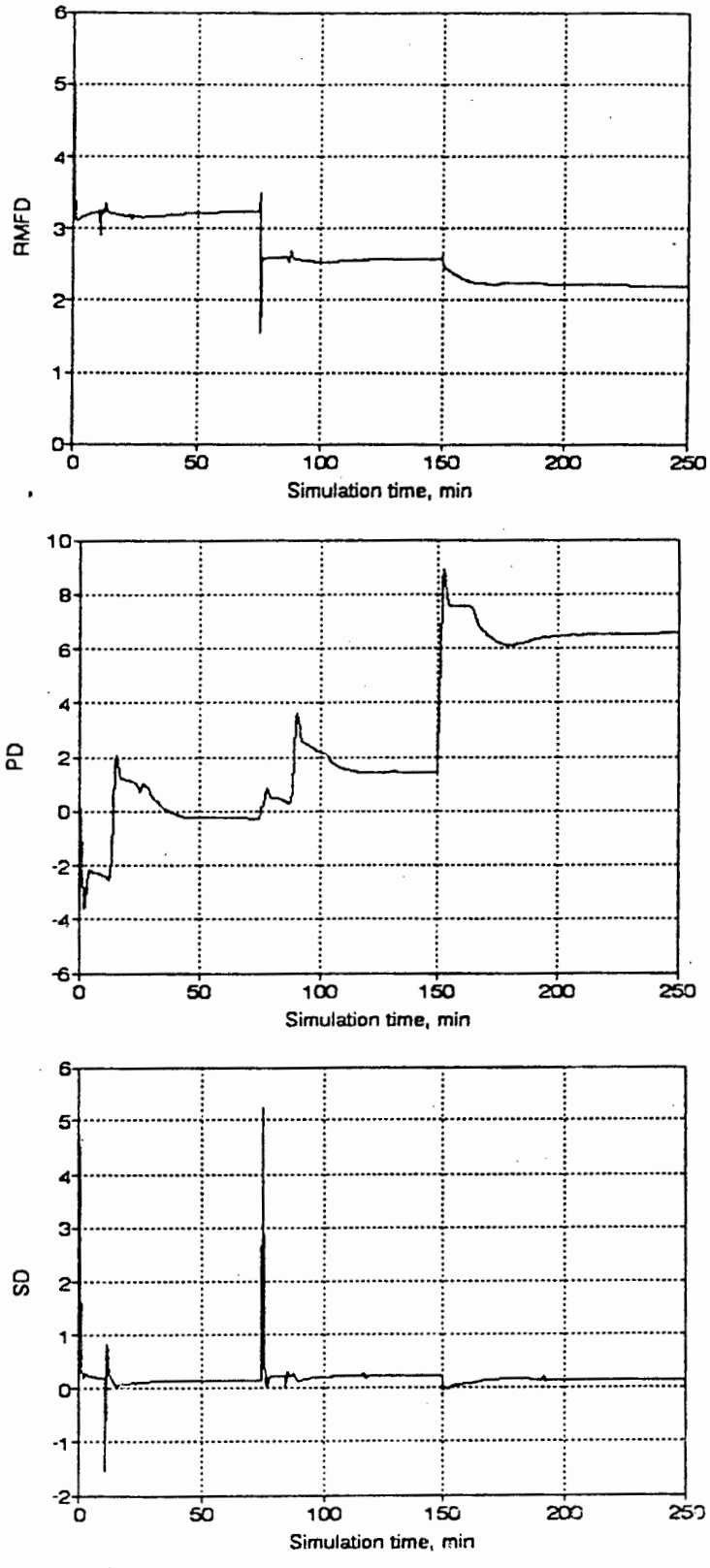


Figure 4.14 Process inputs of a constrained Dynamic Matrix Control of the milling circuit. Constraints are: ($-1.0 \leq \Delta PD \leq 1.0$). M=100, N=20, P=120, Q=(10I,1I,10I) and Λ=0.

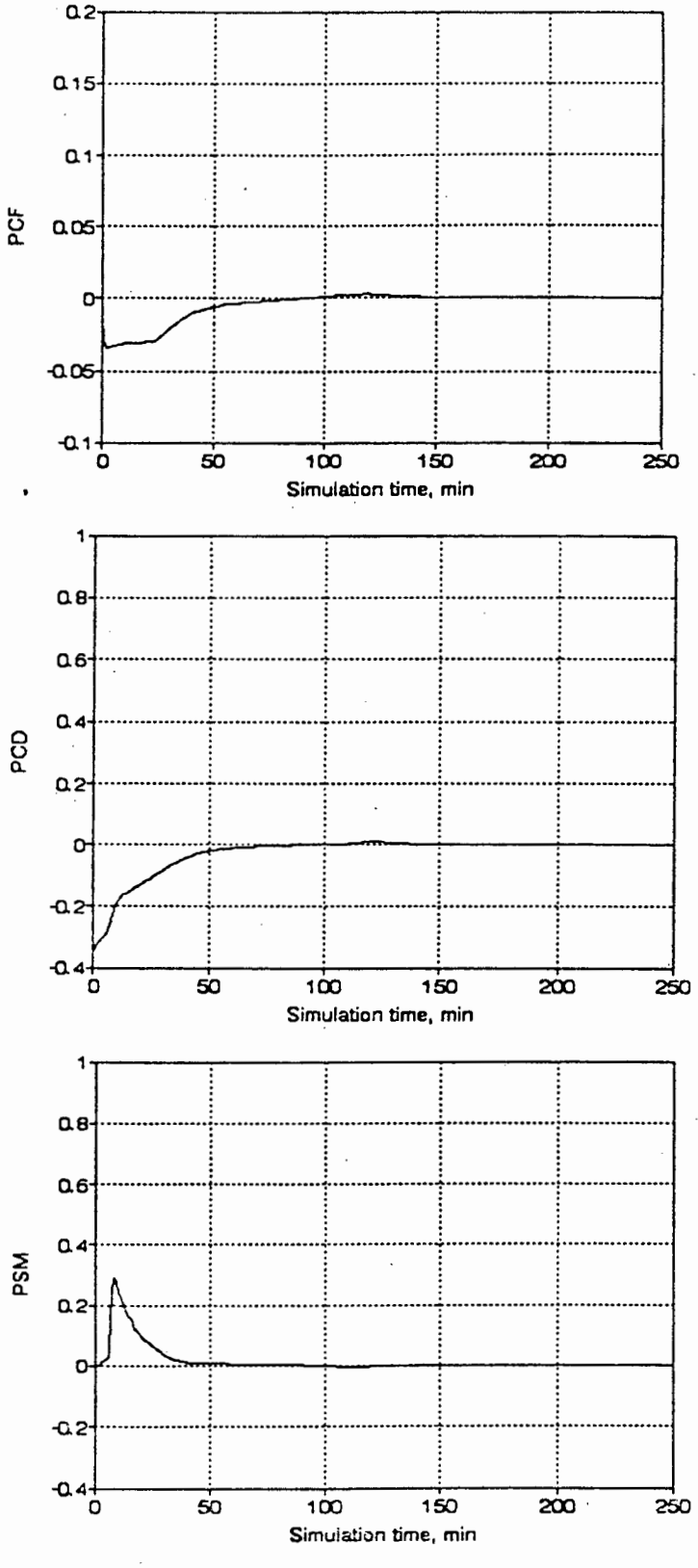


Figure 4.15 Process outputs of a constrained Dynamic Matrix Control of the milling circuit. Constraints are: ($-0.01 \leq PCF \leq 0.01$). M=100, N=20, P=120, Q=(10I,1I,10I) and Λ=0.

CHAPTER 5

CONTROL OF A NONLINEAR MILLING CIRCUIT MODEL

The purpose of this section of the study is to investigate the performance of DMC in the face of model uncertainties. Real plants are generally non-linear in nature and a linear controller like DMC, could deteriorate in performance as the model or dynamic matrix in the case of DMC, becomes less representative of the real process.

There are a number of sources of plant/model mismatch. This study looks specifically at mismatch due to plant nonlinearity.

Some reseachers (Bequette, 1991; Clarke, 1991; Morshedi, 1991; Georgiou et al, 1988) have extended basic linear DMC to a nonlinear DMC algorithm for nonlinear systems. This extension makes the resulting control problem complicated and simple linear systems theory on which linear DMC is based, can no longer be applied. The aim of this study is therefore to show whether or not the nonlinearity characteristic of milling circuits would significantly degrade the performance of linear DMC.

5.1 Model Description

The grinding circuit used in this study is an overflow ball mill-hydrocyclone circuit as shown in Figure 5.1 below. A simplified nonlinear

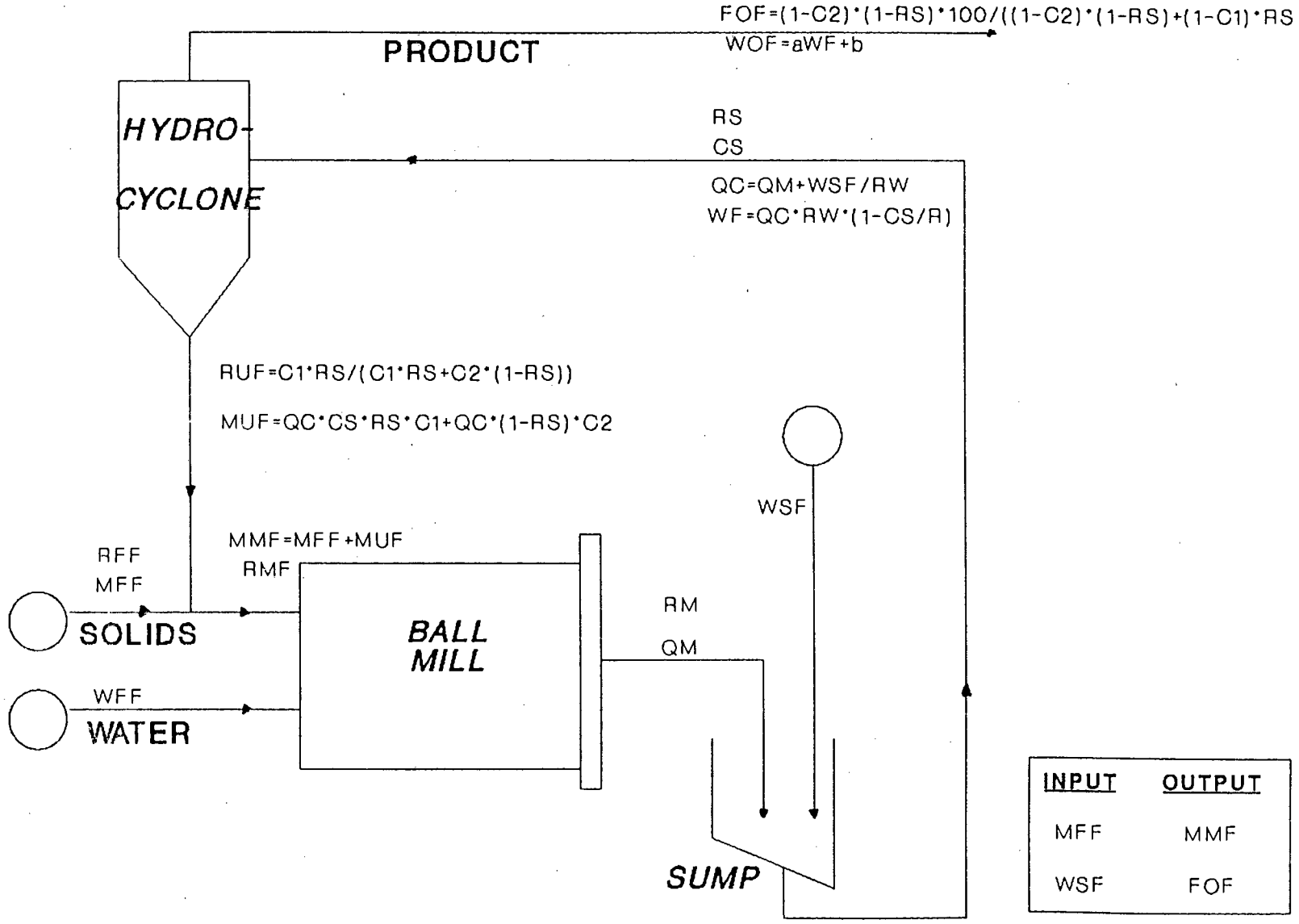


Figure 5.1 The ball mill circuit.

model for an experimental rig of this nature was derived by Rajamani and Herbst (1991) and has been reproduced below with modifications where indicated. Dynamic models for the ball mill, sump and hydrocyclone are given, after which their solution to describe the dynamic behavior of the milling circuit is discussed.

Ball mill model

A detailed ball mill model would include in the material balance all possible size fractions, which could result in hundreds of differential and algebraic equations. A simplified nonlinear model proposed by Rajamani and Herbst (1991) involving only two size fractions was considered to be adequate for the purpose of the present study:

$$H_M \frac{dR_M}{dt} = M_{FF}\, R_{FF} + M_{UF}\, R_{UF} - k_0\, H_M\, R_M - (M_{FF}+M_{UF})R_M \qquad (5.1)$$

where R_M is the fraction of material above 44μm in the mill,

R_{FF} is the fraction of material above 44μm in the mill fresh feed,

R_{UF} is the fraction of material above 44μm in the cyclone underflow,

H_M is mill hold-up mass,

k_0 is a kinetic rate parameter,

M is solids feed rate,

The subscript M refers to mill,
FF refers to fresh feed,
MF refers to mill feed and
UF refers to underflow.

Also associated with the mill is the fresh water (W_{FF}) which was assumed to be set at 40% of total fresh feed to the mill.

Sump model

The sump was considered to be perfectly mixed. The level control of the sump was assumed to involve a well-tuned local controller which manipulates the sump outflow to maintain the sump level steady at all times; thus the slurry volume in the sump was taken to be constant. Then, the following mass balance equations are applicable for the sump.

$$V_S \frac{dC_S R_S}{dt} = (M_{FF} + M_{UF})R_M - (Q_M + W_{SF}/\rho_W)C_S R_S \qquad (5.2)$$

$$V_S \frac{dC_S}{dt} = (M_{FF} + M_{UF}) - (Q_M + W_{SF}/\rho_W) C_S \qquad (5.3)$$

where V_S is the volume of slurry in the sump,

W_{SF} is the rate of addition of water to the sump,

ρ_W is water density,

Q_M is the total volume of slurry discharging from the mill,

ρ_S is solids density,

C_S is the concentration of solids in the sump expressed as mass of solids per unit volume of slurry,

R_S is the fraction of solids above 44μm in the sump.

Expanding the left hand side of (5.2) with substitution of (5.3) gives

$$V_S C_S \frac{dR_S}{dt} = (M_{FF} + M_{UF})(R_M - R_S) \qquad (5.4)$$

which together with (5.3) describes the dynamic behavior of the sump.

Hydrocyclone model

A dynamic model of the cyclone is unnecessary because the response is virtually instantaneous. The model equations given by Rajamani and Herbst (1991) are:

$$d_{50} = \exp(3.616 - 15.006*10^{-2}\ Q_C[\text{l/min}] + 2.3f_v) \tag{5.5}$$

$$C_1 = 0.0042M_{FF}\ d_{50} - 0.0154d_{50} - 0.0704M_{FF} + 1.3412 \tag{5.6}$$

$$C_2 = 0.0502M_{FF}\ d_{50} - 0.0660d_{50} - 2.9354M_{FF} + 4.642 \tag{5.7}$$

A constant M_{FF} (2.267 kg/min) was used in these equations in order to avoid unrealistic discontinuity in the cyclone classification to step changes in the feed rate.

$$W_{OF} = \begin{cases} 1.363W_F - 10.75 & \text{for } W_F < 21.4 \text{ kg/min} \\ 0.837W_F + 0.35 & \text{for } W_F > 21.4 \text{ kg/min} \end{cases} \tag{5.8}$$

where W_{OF} is water in the overflow,

W_F is the feed water rate to the cyclone,

d_{50} is the size at which 50% of the solids report to overflow and 50% to underflow. It was found that to get proper cylcone operation, the coefficient of Q_C had to be 1.8668 with Q_C in $[m^3/min]$,

Q_C is the volumetric feed rate to the cyclone,

f_v is the volume fraction of solids in the slurry feed, given by

$$f_v = C_S/\rho_S$$

C_1 is the fraction of plus 44μm material in the cyclone feed that

reports to underflow,

C_2 is the fraction of minus 44μm material in the cyclone feed that reports to underflow,

F_{OF} is the percentage of solids less than 44μm in the product, which may be shown through a mass balance to be given by

$$F_{OF} = \frac{(1-C_2)(1-R_S)100.0}{(1-C_2)(1-R_S) + (1-C_1)R_S} \tag{5.9}$$

The Complete Milling Circuit

RUF and MUF are needed in the ball mill model. A mass balance around hydrocyclone gives:

$$R_{UF} = \frac{C_1 R_S}{C_1 R_S + C_2(1.0 - R_S)}$$

and

$$M_{UF} = Q_C C_S R_S C_1 + Q_C (1-R_S) C_2 C_S$$

The volumetric flow rates used in (5.2), (5.3) and (5.5) are given by

$$Q_M = \frac{M_{MF}}{\rho_S} + \frac{W_{FF}}{\rho_W} + \frac{(W_F - W_{OF})}{\rho_W}$$

$$Q_C = Q_M + W_{SF}/\rho_W$$

with the mill feed rate M_{MF} given by

$$M_{MF} = M_{UF} + M_{FF}$$

The following parameter values were used in this study:

H_M = 48 kg

k_0 = 0.03465 min^{-1}

ρ_W = 1000 kg/m^3

V_S = 0.01414 m^3

ρ_S = 2700 kg/m^3 (assuming quartz density)

5.2 Program Description

Steady State Case

This nonlinear model routine was coded in standard Fortran-77 and implemented on a VAX. The programs used are listed in the Appendix. Firstly, a steady state had to be established given a set of inputs and initial conditions. The procedure to achieve steady state has been summarised in Figure 5.2 below. Since M_{UF} cannot be determined explicitly without knowing Q_C, an iterative procedure was used to determine M_{UF} at any given time for given values of the states R_S, C_S and R_M. Newton's method was used to achieve this convergence. Once convergence on M_{UF} had been achieved to within a reasonable error tolerance, all other algebraic equations could then be solved. Equations (5.1), (5.3) and (5.4) were solved with derivatives set to zero to obtain the steady states.

For input values:

M_{FF} = 2.267 kg/min

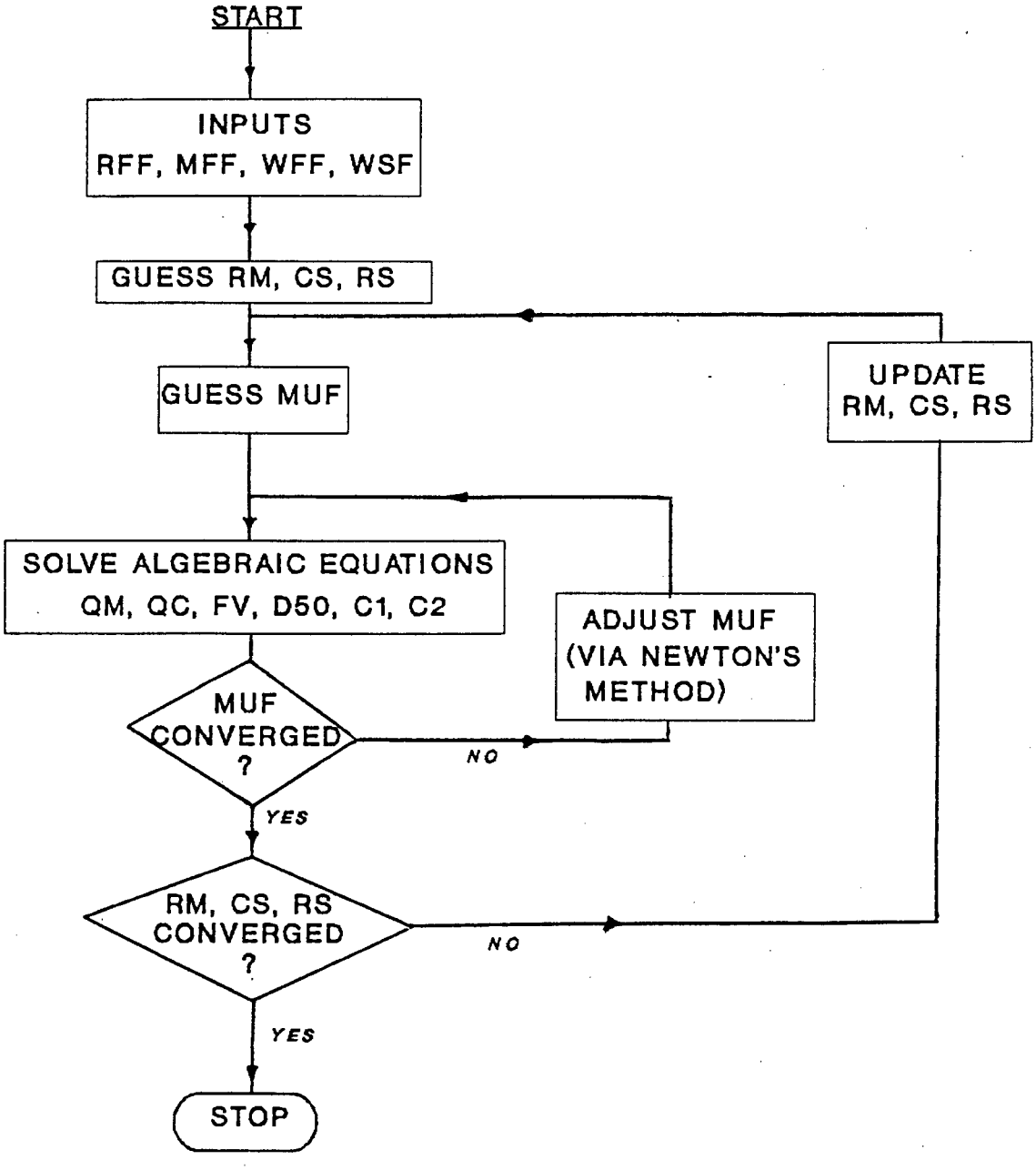


Figure 5.2 Flowsheet to determine steady state of the plant.

$R_{FF} = 0.85$

$W_{FF} = 1.533$ kg/min

$W_{SF} = 11.4$ kg/min

the following steady state conditions were found:

$M_{UF} = 6.716$ kg/min

$Q_M = 0.00929$ m^3

$R_M = 0.7625$

$R_S = 0.7625$

$C_S = 434.26$ kg/m^3

The outputs of interest were:

$M_{MF} = 8.984$ kg/min

$F_{OF} = 70.94\%$

Dynamic Case

The dynamic case implementation is similar to the steady state case except that the states start to move with time. The fourth-order Runge Kutta method was used to numerically integrate the state equations; (5.1), (5.3) and (5.4) above. Figure 4.2 in the previous chapter shows a similar layout used in the dynamic case with a DMC controller. The open-loop simulation is also similar except that there is no controller in the procedure. The computer programs are listed in the Appendix.

5.3 Results and Discussion

5.3.1 Open-loop Response of the Circuit

As discussed in earlier chapters, an open loop response is required to formulate the dynamic matrix. Step changes to the circuit were all done after 10 min of steady state operation. An equivalent of a unit step change was implemented in each case, with corresponding output responses corrected to a unit input change response in order to construct a standard dynamic matrix.

Response to change in ore feed rate

A step change from 2.267 kg/min to 2.0833 kg/min was implemented on the fresh feed rate while keeping the sump water addition rate at a constant value of 11.4 kg/min. The responses of the mill throughput and product size are a gradual decrease and increase, respectively, to new steady state since this is a negative step change. This can be seen from Figure 5.3 which plots mill throughput (M_{MF}) and percentage product size passing 44μm (F_{OF}).

Response to change in sump water rate

A step change in water rate from 11.4 kg/min to 15.9 kg/min with fresh feed rate at a constant 2.267 kg/min, resulted in a large instanteneous response in both the mill throughput and the product size. This was followed by a slow return to a steady state higher than the original steady state. The result is a consequence of the constant volume assumption for the mill and sump, which in practice is not strictly true. Figure 5.4 shows the mill throughput and the product size responses to this step change.

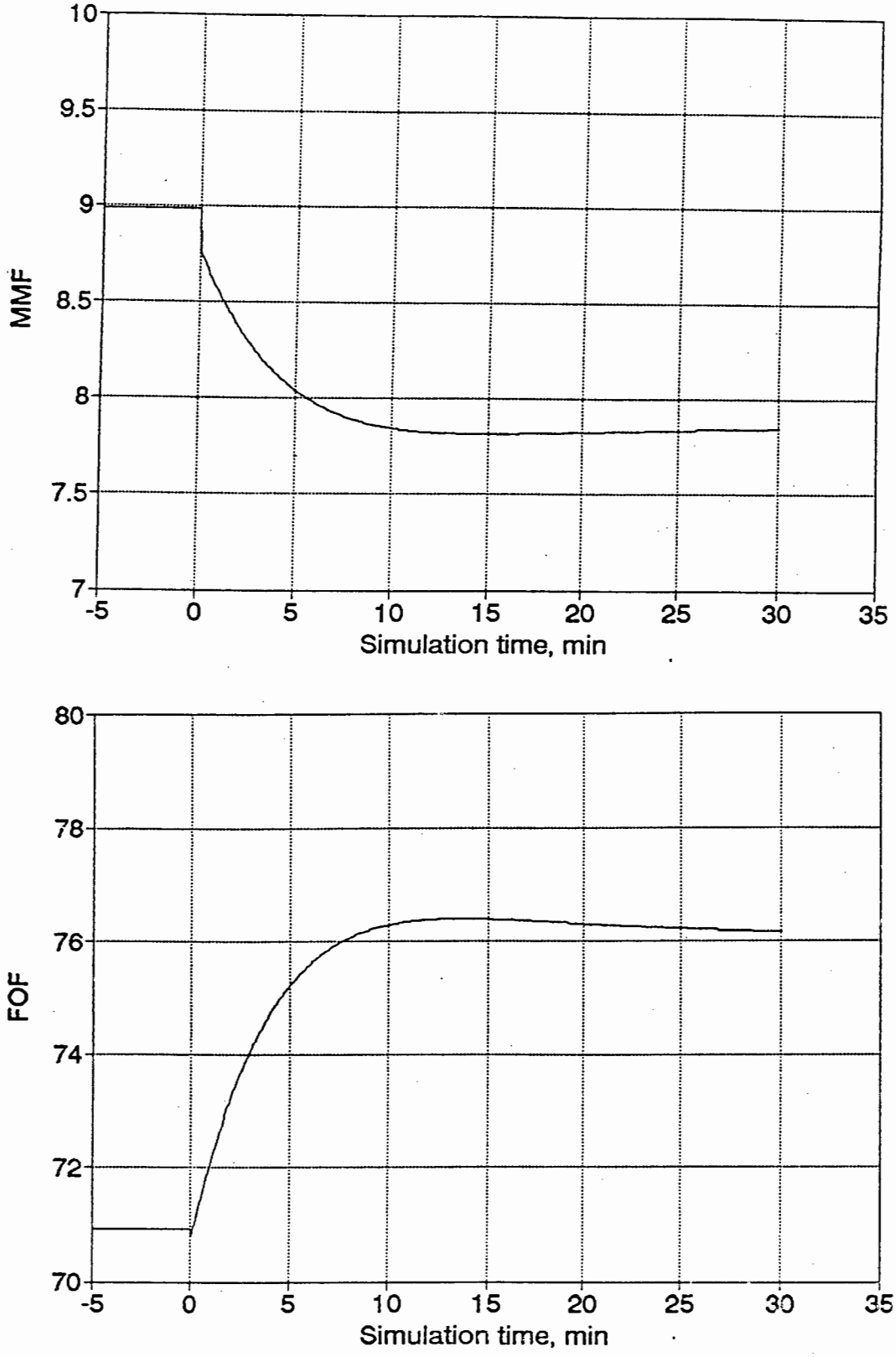


Figure 5.3 The open-loop response of the process to a step change from 2.2267 kg/min to 2.083 kg/min in ore fresh feed rate.

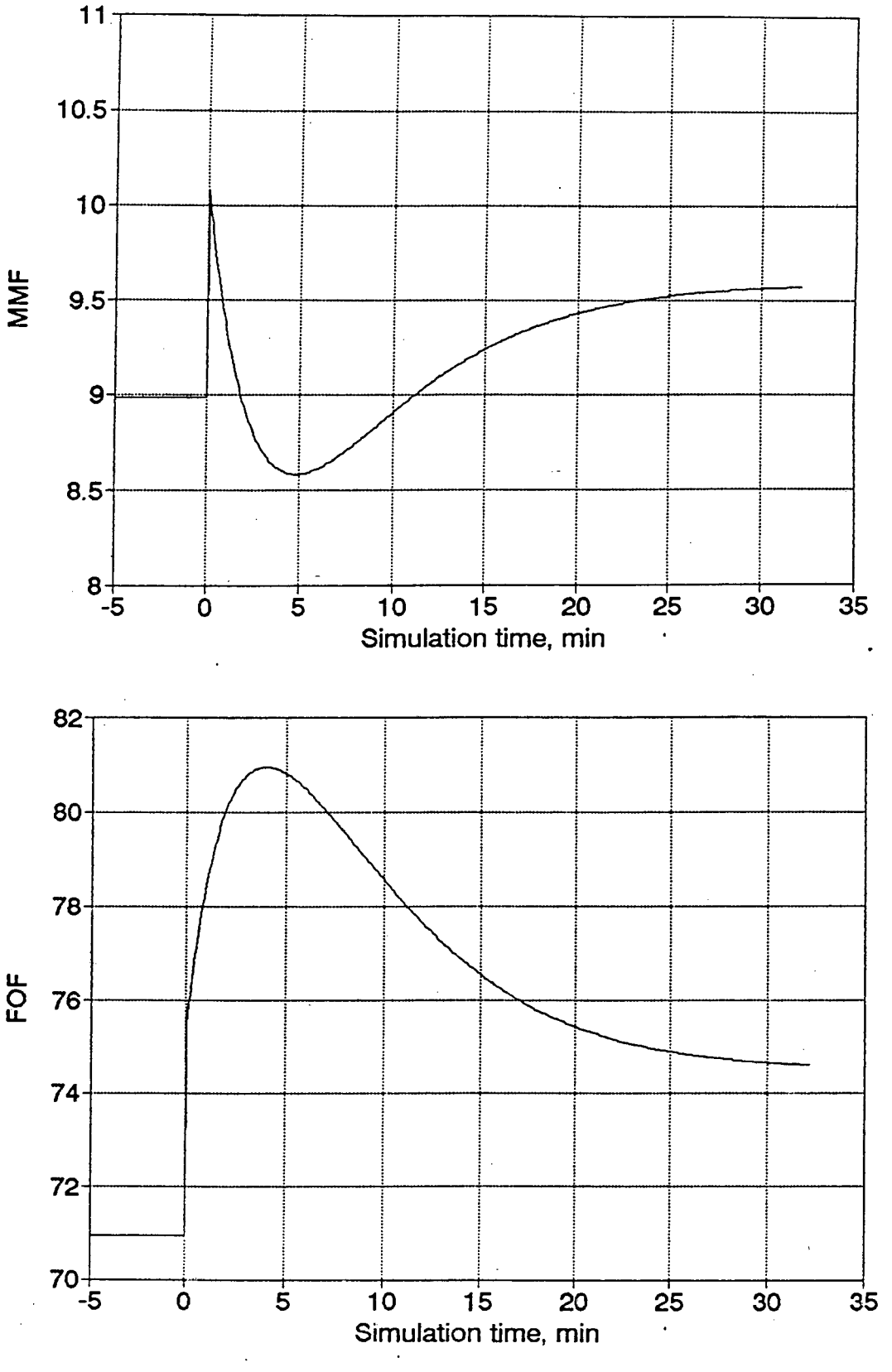


Figure 5.4 The open-loop response of the process to a step change from 11.4 kg/min to 15.9 kg/min in sump water addition rate.

5.3.2 Closed-loop Response of the Circuit

From the analysis of the open-loop responses mentioned above, it can be seen that steady state is reach after about 30 minutes. At a sampling time of 0.3 min, this result in a steady state horizon, M, of about 94 data points. A rule of thumb suggests that control moves (N) should be about one quarter of the steady state horizon (N=M/4). Using N=24, the prediction horizon (P) was set at N+M which equals 118. Both the weighting matrices Q and Λ were set at I for equal weighting and 0 for no input move suppression, respectively.

A setpoint change to 75% passing 44μm in product size was implemented after 10 min, and another on the mill throughput at 100 min to take it up to 9.5 kg/min. Figures 5.5 and 5.6 show the mill throughput and product size with corresponding inputs, respectively. While product size seems fairly stable to changes in mill throughput, mill throughput seems quite sensitive to setpoint changes with deviations of up to 0.4 kg/min. This can result in mill overload which can destabilize the circuit if this occurs around its point of overload. As expected, inputs for this case show drastic responses. In practice, these (if achievable), could cause stability problems.

Reducing the control move horizon result in less aggressive control action; hence reducing the stability risk (Garcia et al., 1989). The reduction in control move horizon also reduces computational time since the control problem becomes smaller. Reducing N to 6 moves resulted in P=100. The results can be seen from Figures 5.7 and 5.8. The mill throughput improves in terms of deviation from setpoint with a slightly larger overshoot in product size compared to the former simulation run, a price one can expect to pay. The inputs are indeed smoother reducing the stability risk. Because of the interactions inherent in the process, one cannot expect any greater improvement in one variable without adversely affecting the other.

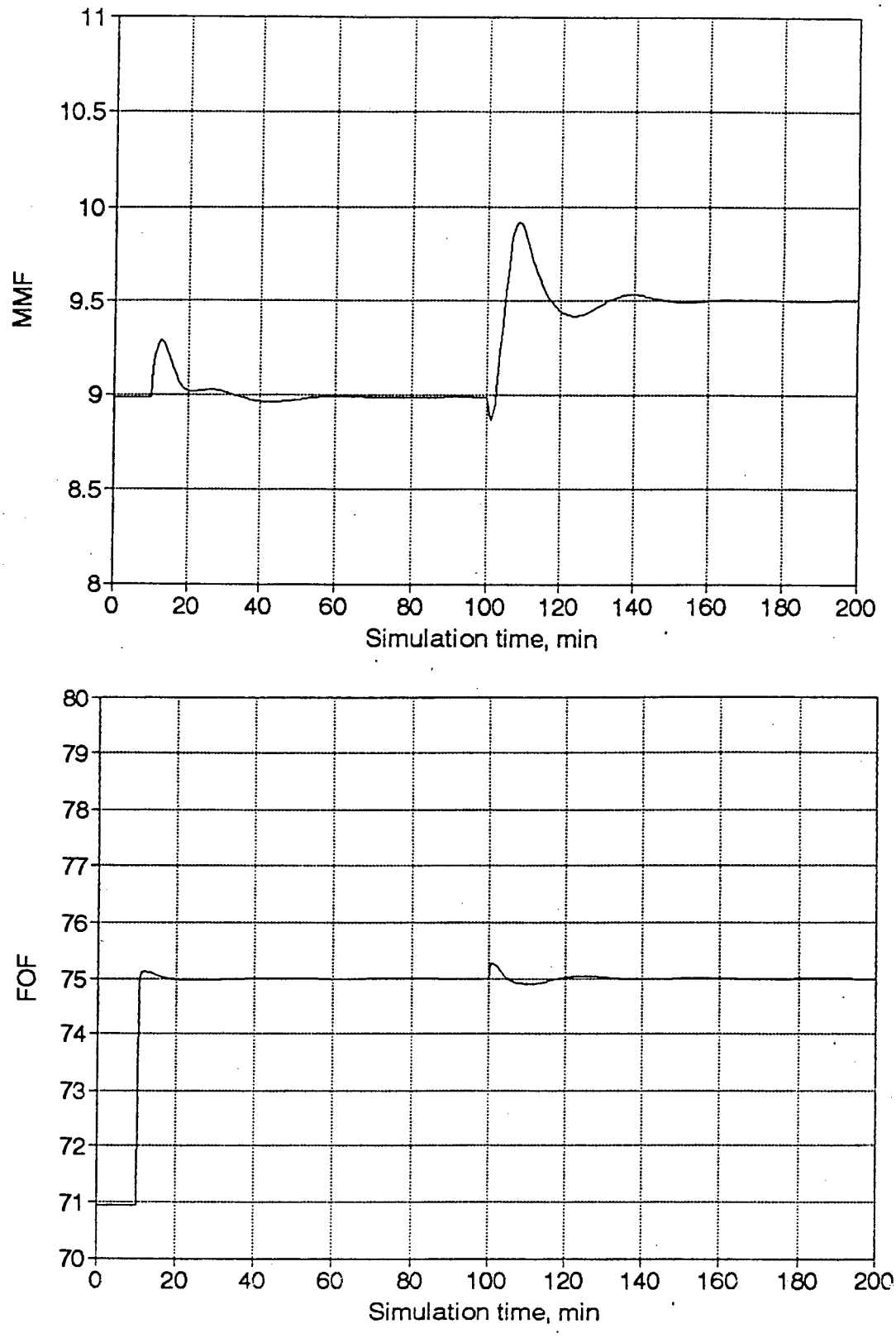


Figure 5.5 Process outputs of the unconstrained Dynamic Matrix Control of the ball mill circuit with M=94, N=24, P=118, Q=I and Λ=0.

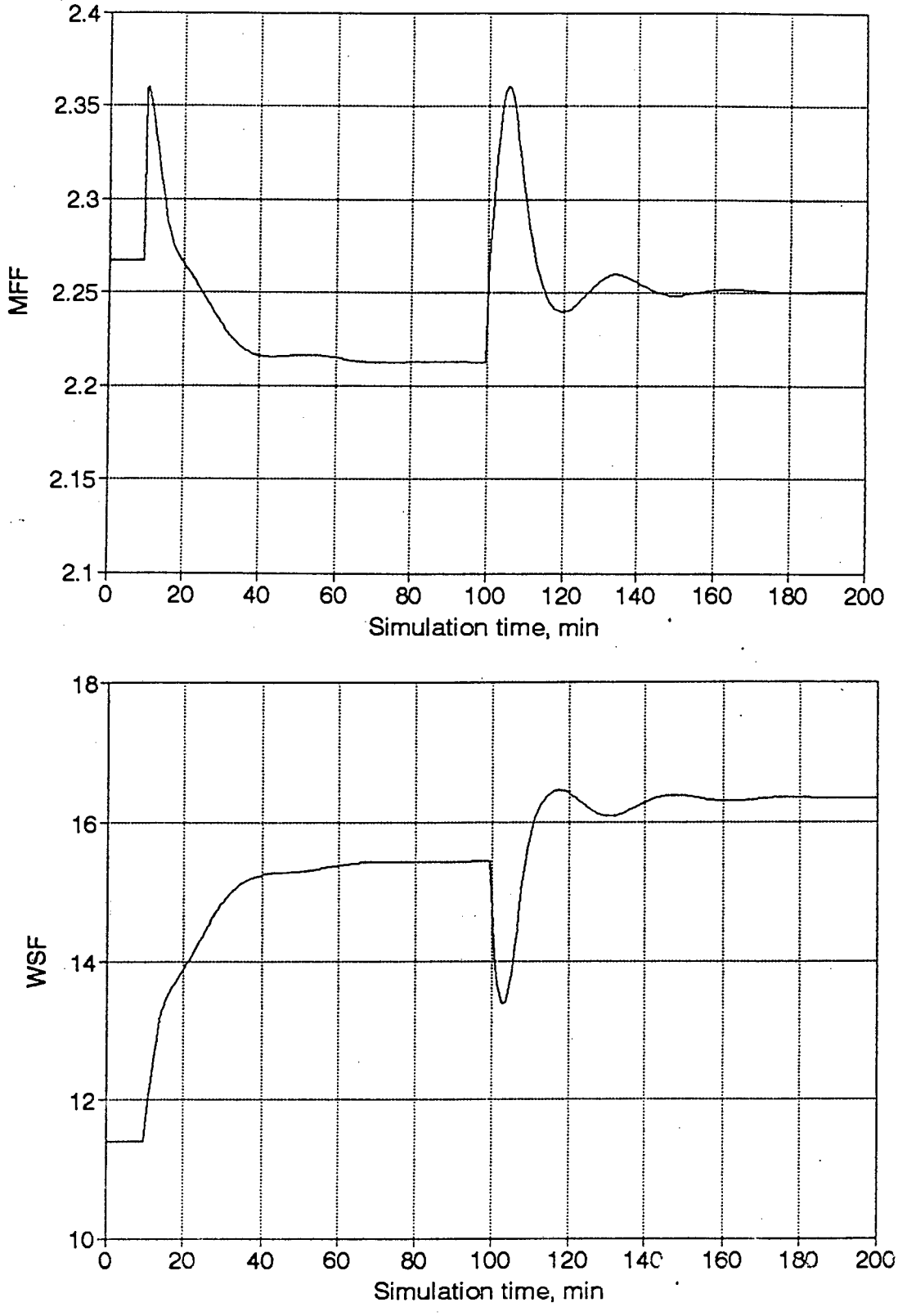


Figure 5.6 Process inputs of the unconstrained Dynamic Matrix Control of the ball mill circuit with M=94, N=24, P=118, Q=I and Λ=0.

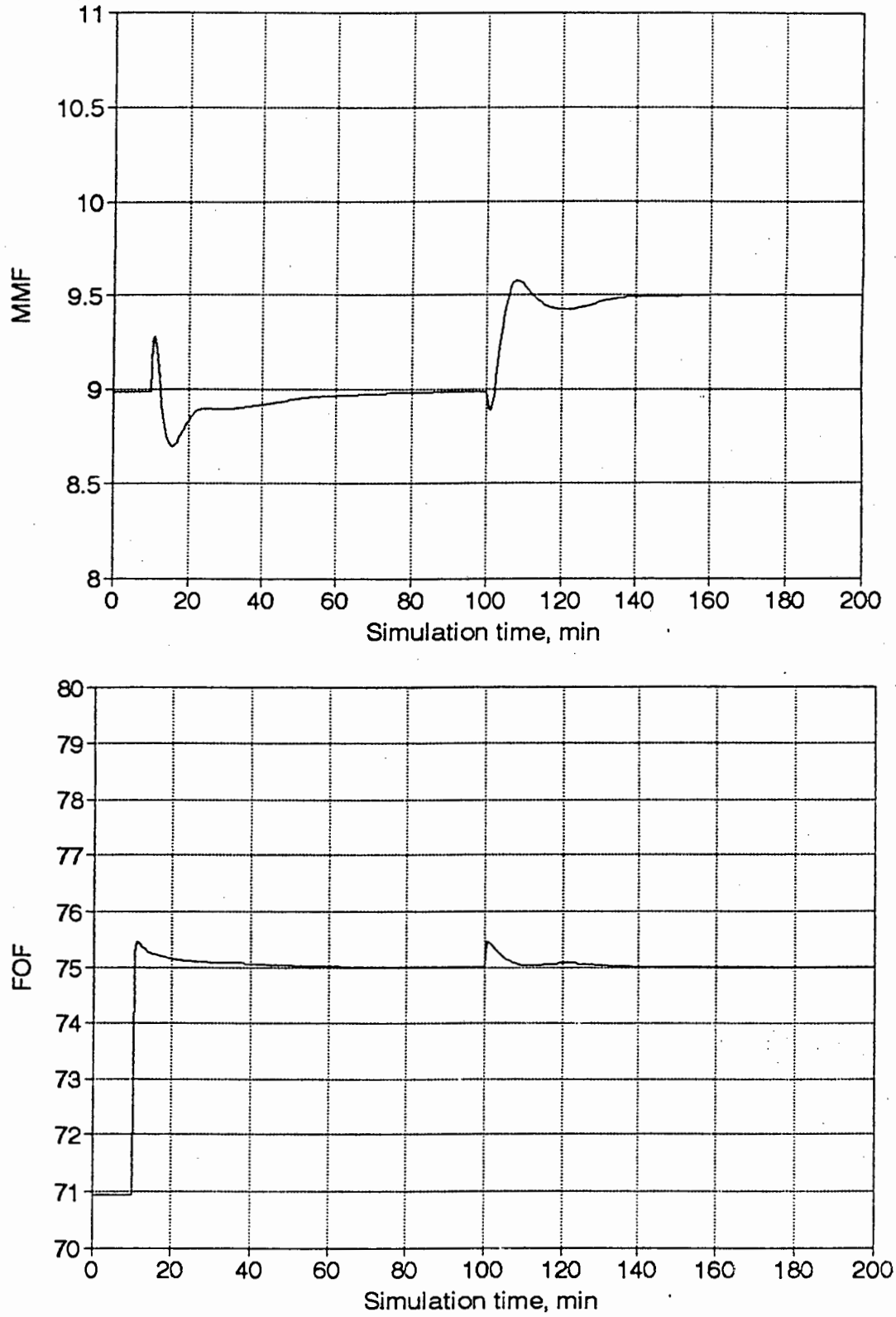


Figure 5.7 Process outputs of the unconstrained Dynamic Matrix Control of the ball mill circuit with M=94, N=6, P=100, Q=I and Λ=0.

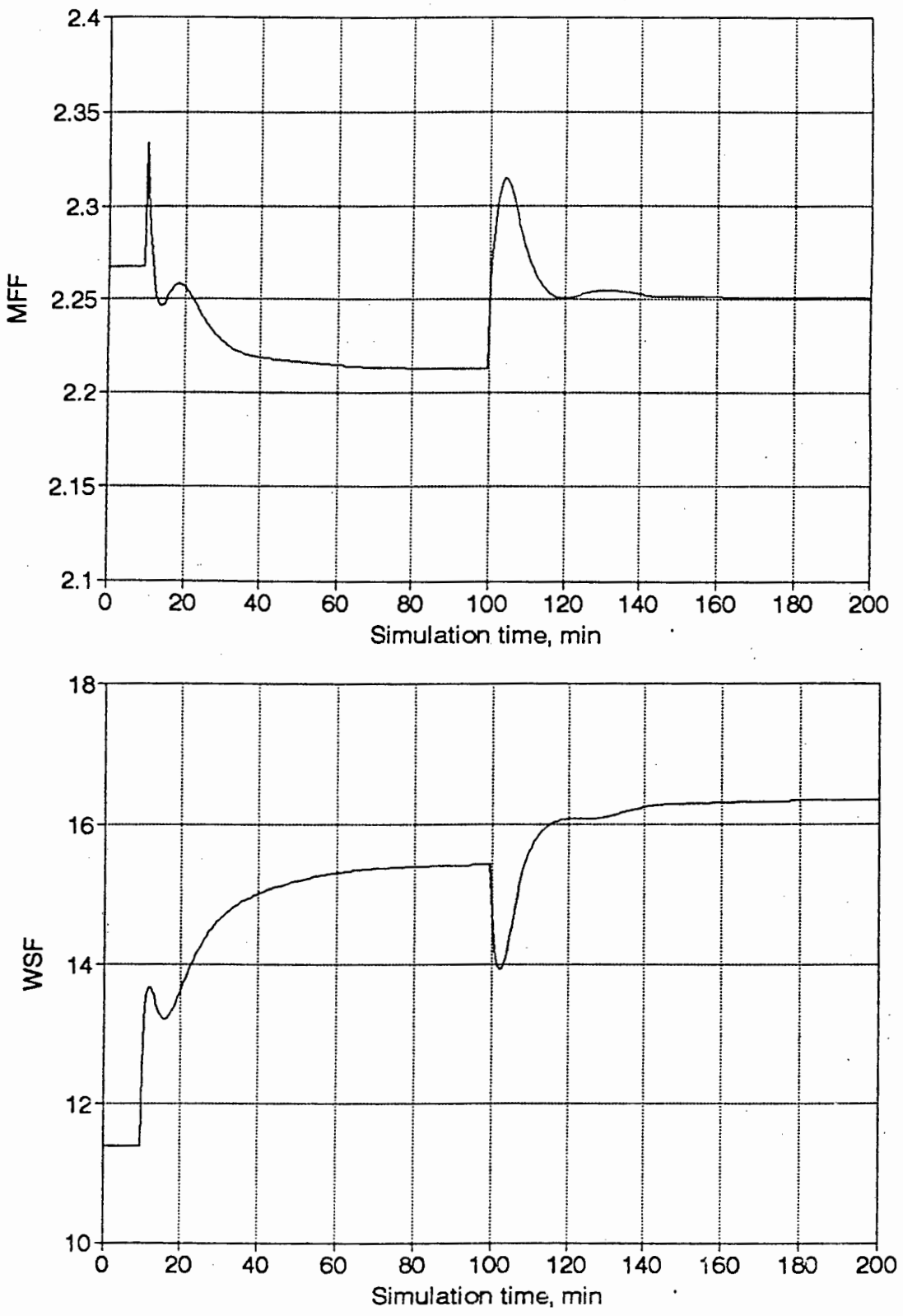


Figure 5.8 Process inputs of the unconstrained Dynamic Matrix Control of the ball mill circuit with M=94, N=6, P=100, Q=I and Λ=0.

5.3.3 Disturbances to the Circuit

In this study, disturbances refer to those disturbances that cannot be characterised or measured in the process. There are several types of these disturbances which may occur in an operating grinding circuit and which may cause irregularities or undesirable changes in the size distributions of the particles in the circuit product. Some of the important types are:

1. change in the characteristics of the ore entering the circuit, and in the hardness, size distribution, mineral composition;

2. change in the flow rate of water entering the circuit;

3. changes in the nature of the cyclone feed pulp, such as intermittent aeration of the pulp due to surging pump.

4. changes in the flow rates of pulp within a circuit due to mechanical reasons, such as blockage of a spigot in a hydrocyclone;

5. long-term changes in circuit performance due to mechanical wear of the equipment.

Unfortunately most of these cannot be tested by simulations, and require a real process to occur. However, two of these can be attained directly or indirectly.

Ore hardness can be indirectly changed by changing the fresh feed composition, R_{FF}. R_{FF} has been assumed to be constant at 0.85 in the simulations presented thus far. Increasing R_{FF} is characteristic of the change the increased ore hardness will have on the product from previous crushing units. The composition of the particles above a fixed size criterion like the 44μm, will increase with ore hardness. Figure 5.9 shows the effect of a step change of 0.05 in R_{FF} after 10 min to mill throughput and product size. The effect is slight due to size of the step

change. As expected this change causes a gradual increase in throughput due to increased recycled solids which do not meet the cut size. The product quality decreases slightly as this extra unexpected large size slips through the cyclone classification. Since DMC can counteract any step type of disturbance through its built in disturbance rejection capabilities, the process very soon stabilizes back to its steady state.

The other variable which can be changed readily is the fresh feed water to the circuit (W_{FF}). As mentioned ealier on, this variable is normally kept at 40% of total fresh feed to the circuit, hopefully by a well tuned ratio controller. Assuming that for some reason this controller breaks, allowing a step change in water rate of 20% total feed, which means that water will now account for 60% of total feed. Figure 5.10 shows that this change causes a rapid large change especially in product quality which soon damps out in about twenty minutes time. Again DMC shows satisfactory disturbance rejection capabilities over the step-type of disturbance.

5.3.4 Constrained Closed-loop Response of the Circuit

Control objectives for milling circuits are often conflicting. Many grinding circuits have additional limitations which exist due to local conditions. For instance, the motor on a variable speed pump may not be large enough to handle peaks in pulp flow entering the sump and there is a danger of burning out the motor if the upper limit does not exist. The importance of the limiting effect of mill throughput in particular is emphasised by the fact that the efficiency of closed grinding circuits in eliminating "oversize" from the circuit product tends to increase as the load circulating through the mill increases. The highest efficiency is at a maximum circulating load just before the mill overload point (Lynch, 1977).

Constraints were put on the controlled outputs and then the manipulated inputs to establish the effect these limitations might have on the circuit. In keeping with the objectives of the circuit, product quality was not allowed to exceed 75.5% and the throughput was not allowed to

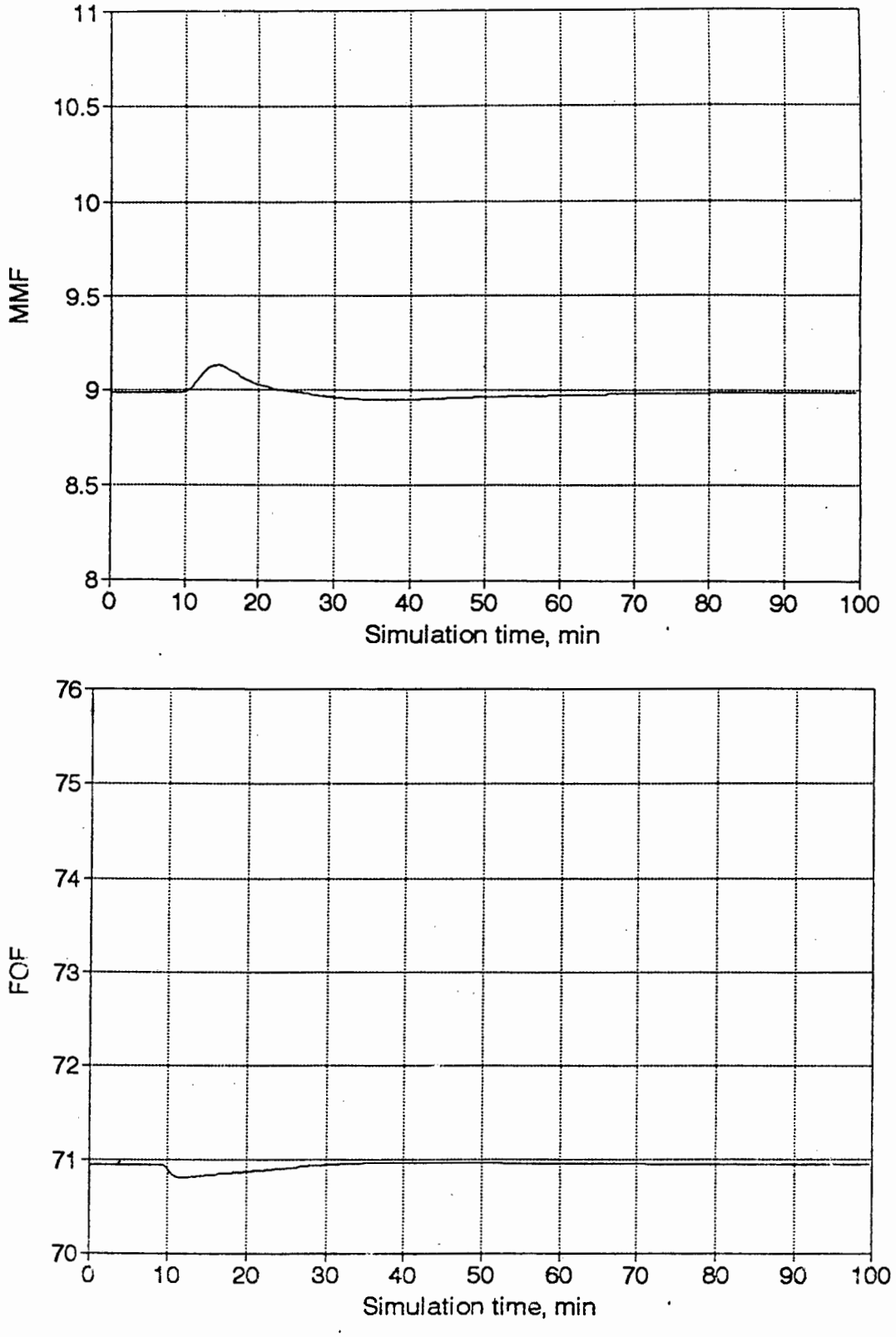


Figure 5.9 Process outputs of the unconstrained Dynamic Matrix Control of the ball mill circuit after a disturbance of 0.05 in RFF applied at 10 min. M=94, N=6, P=100, Q=I and Λ=0.

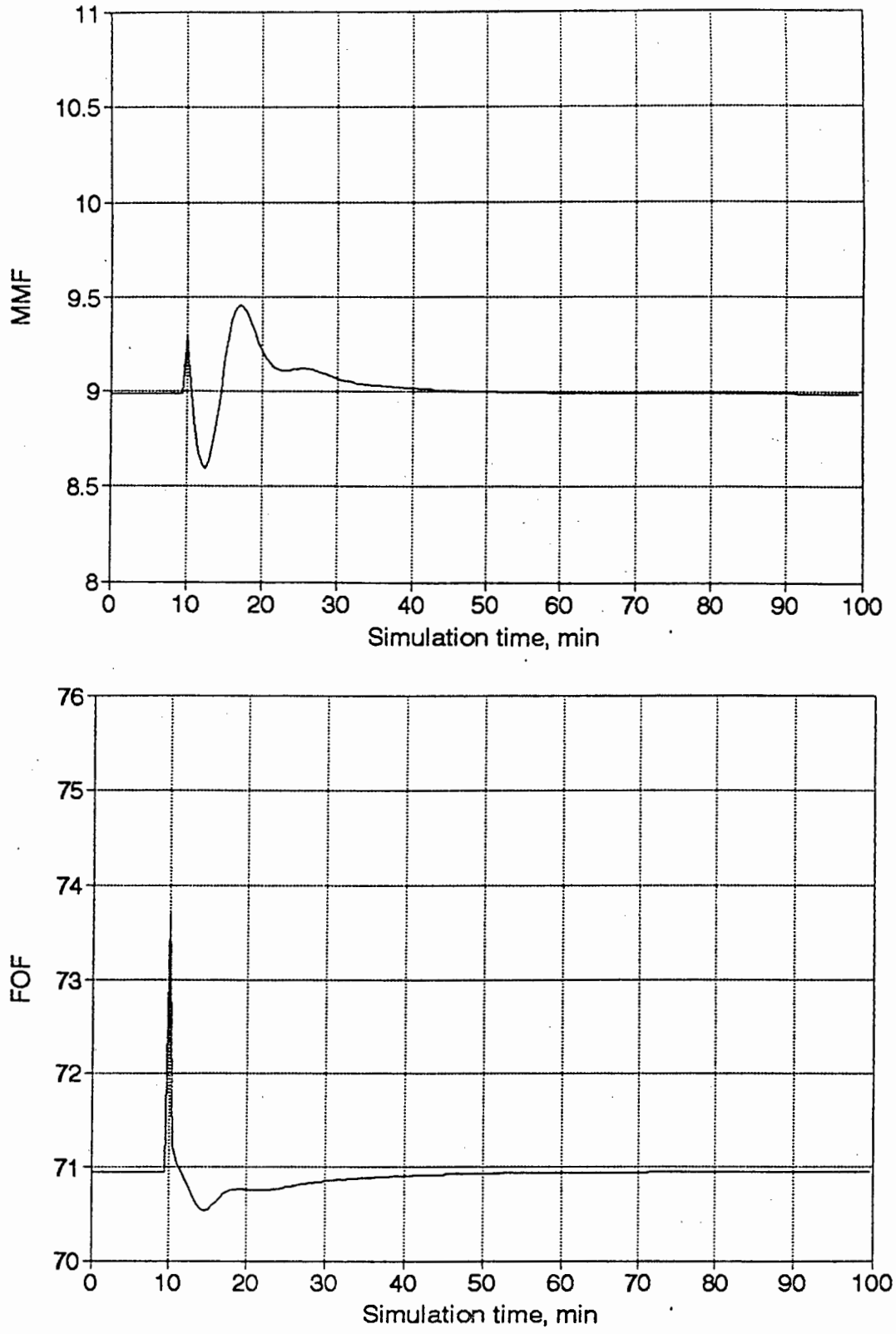
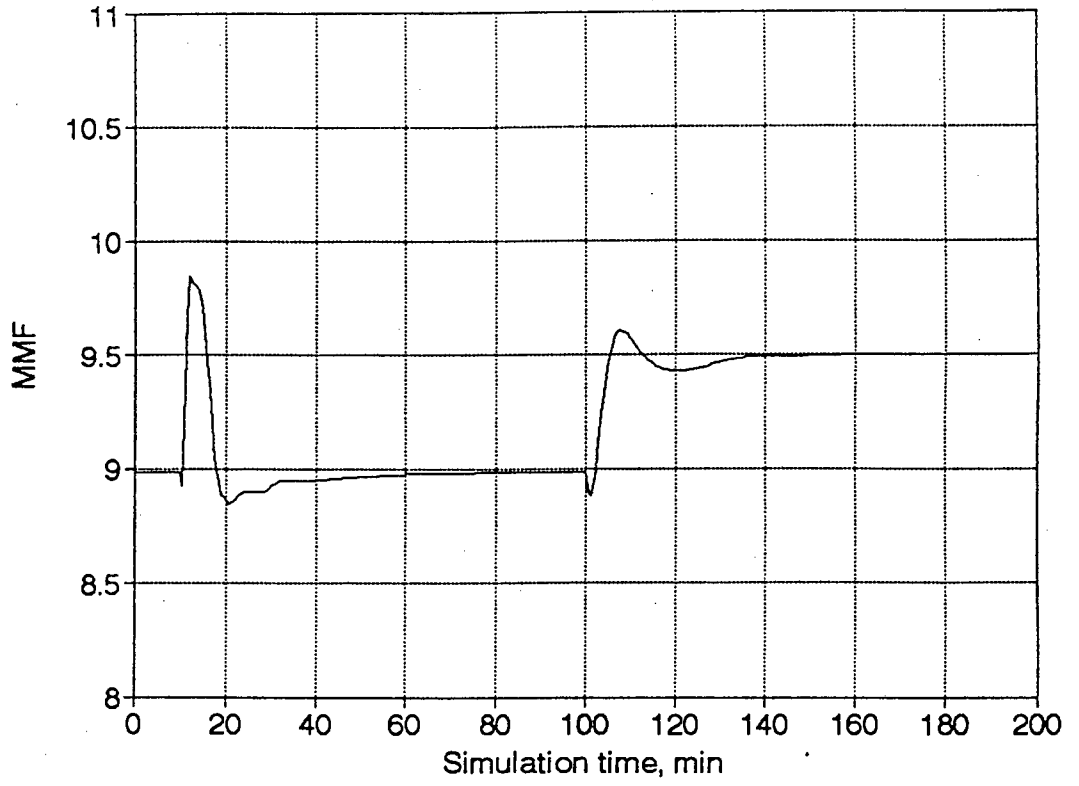


Figure 5.10 Process outputs of the unconstrained Dynamic Matrix Control of the ball mill circuit after a 20% disturbance in fresh feed water applied at 10 min. M=94, N=6, P=100, Q=I and Λ=0.

exceed 9.8 kg/min or fall below 8.9 kg/min. Overground product also presents downstream hydraulic problems and increases mill effort which is the cost factor one tries to reduce. As mentioned before, low mill throughput is costly. Figures 5.11 and 5.12 show the output responses to setpoint changes and the corresponding inputs, respectively. The results if compared with the results of the unconstrained case (Figures 5.7 and 5.8) show that product quality suffers if throughput is not allowed to drop to compensate for demand in quality.

Inputs were also constrained; sump water addition rate was not allowed below 11.0 kg/min and above 16.0 kg/min, and fresh feed was only allowed to move between 2.20 kg/min and 2.30 kg/min. The results can be seen from Figures 5.13 and 5.14. While DMC effectively prevents constraint violations, the outputs struggle to reach setpoint. It will appear that in the end the outputs fail to reach setpoints, certainly the mill throughput cannot reach 9.5 kg/min with the given inputs. This is a typical constraint one might expect in practice.

It is clear that linear DMC is robust enough to yield satisfactory performance even with nonlinear systems provided the plant is operated around the same steady state from which the model (or the dynamic matrix) of the plant was formulated. The further one moves away from this steady state, the poorer one might expect the performance to be until it gets to a point where it is no longer operable with the model. For instance, the FCCU full combustion mode model is not usable for partial combustion mode of operation. A different dynamic matrix is needed for the partial combustion mode. While this example may be a foreseen, deliberate change in the process, it is the unknown disturbances like the ones mentioned above that poses a real problem. It was seen that DMC can reject such disturbances.



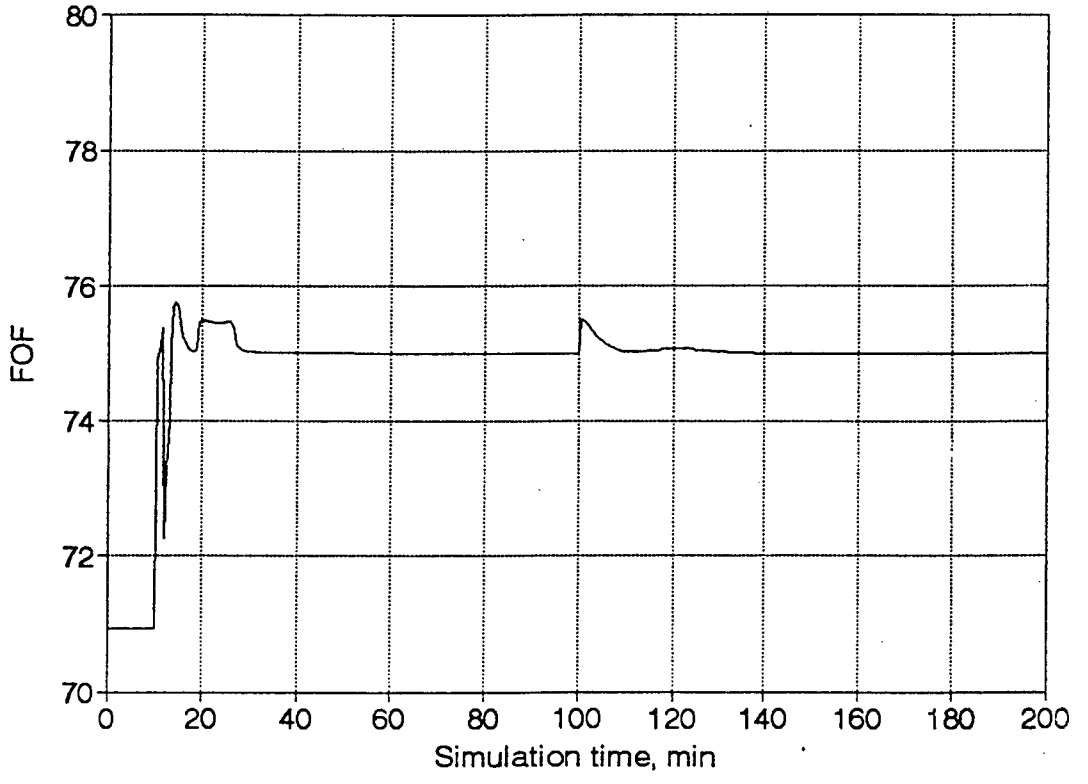
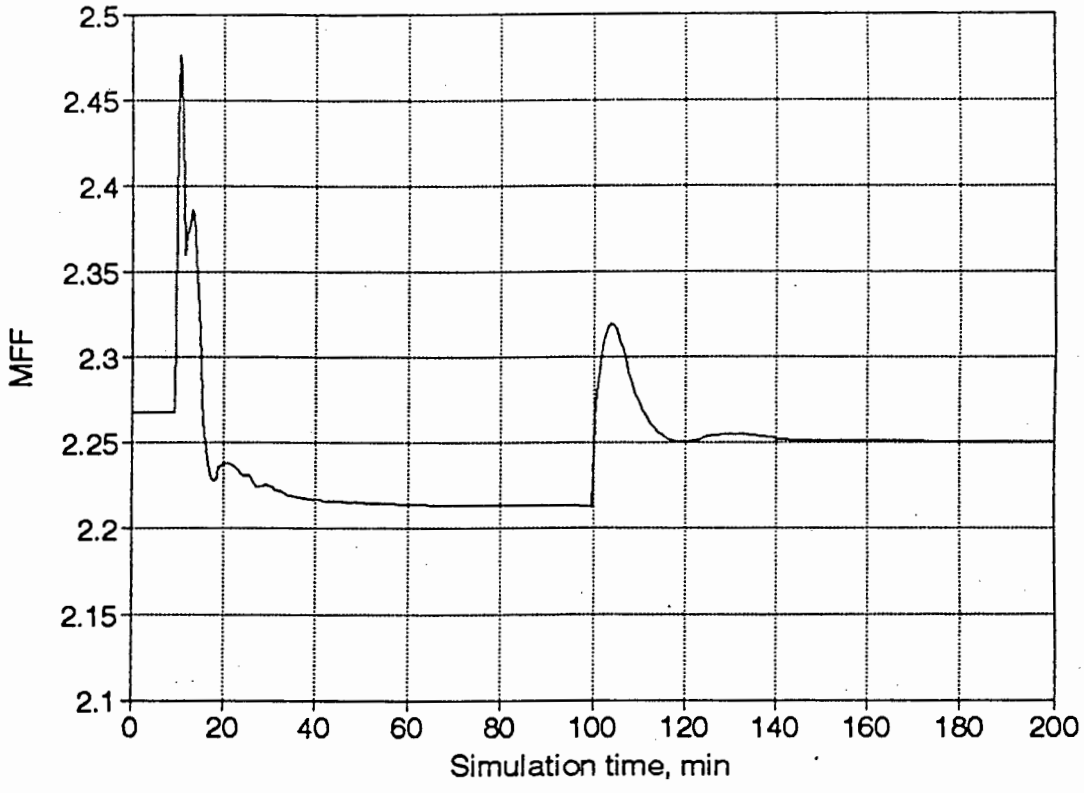


Figure 5.11 Process outputs of a constrained Dynamic Matrix Control of the milling circuit. Constraints are: (FOF $\leq$ 75.5) and (8.9 $\leq$ MMF $\leq$ 9.8). M=94, N=6, P=100, Q=I and Λ=0.



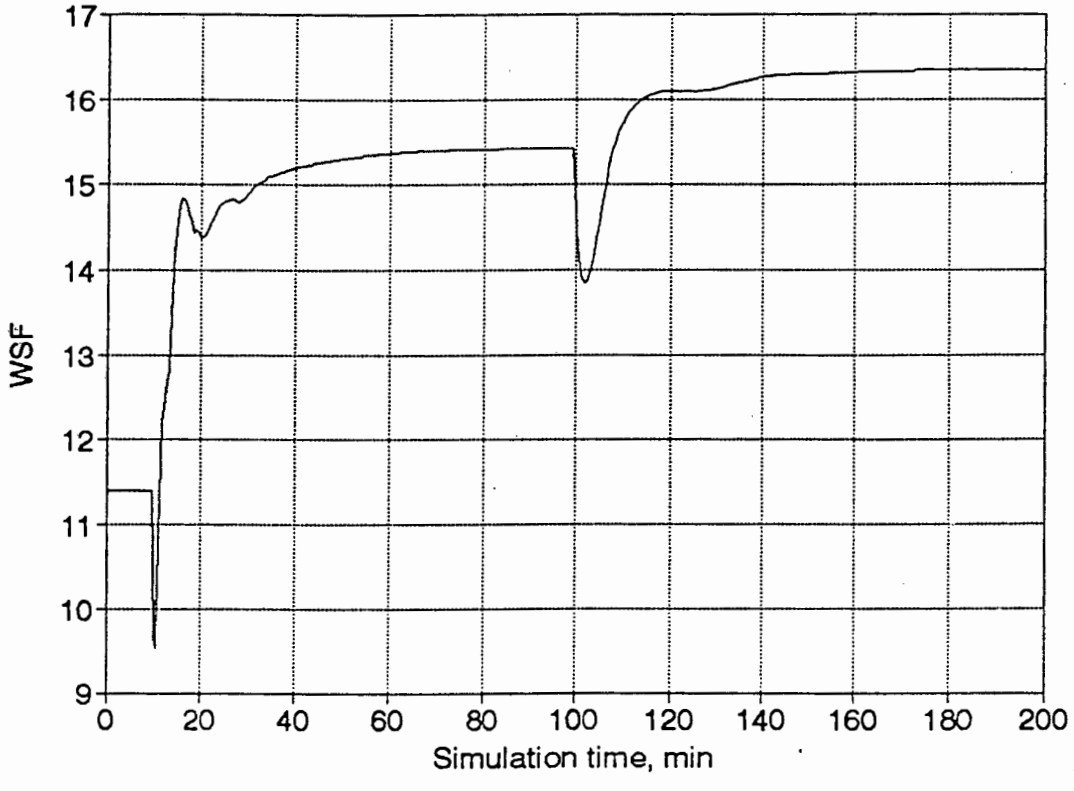
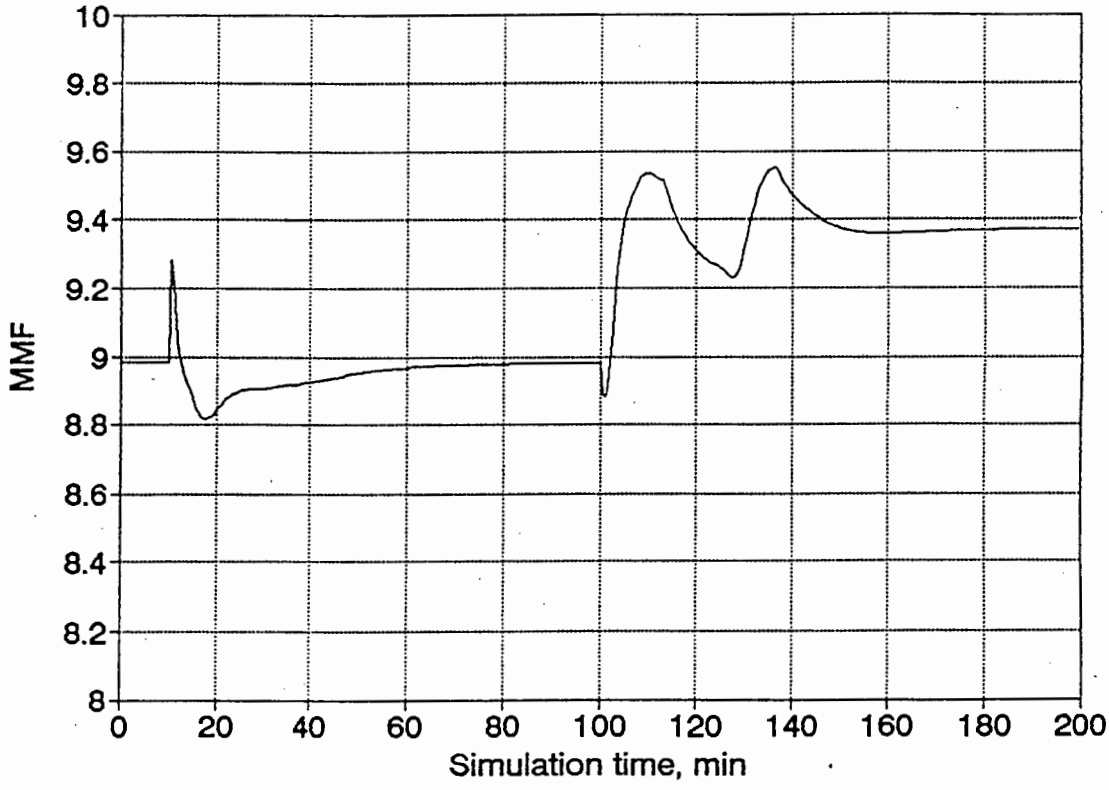


Figure 5.12 Process inputs of a constrained Dynamic Matrix Control of the milling circuit. Constraints are: (FOF $\leq$ 75.5) and (8.9 $\leq$ MMF $\leq$ 9.8). M=94, N=6, P=100, Q=I and Λ=0.



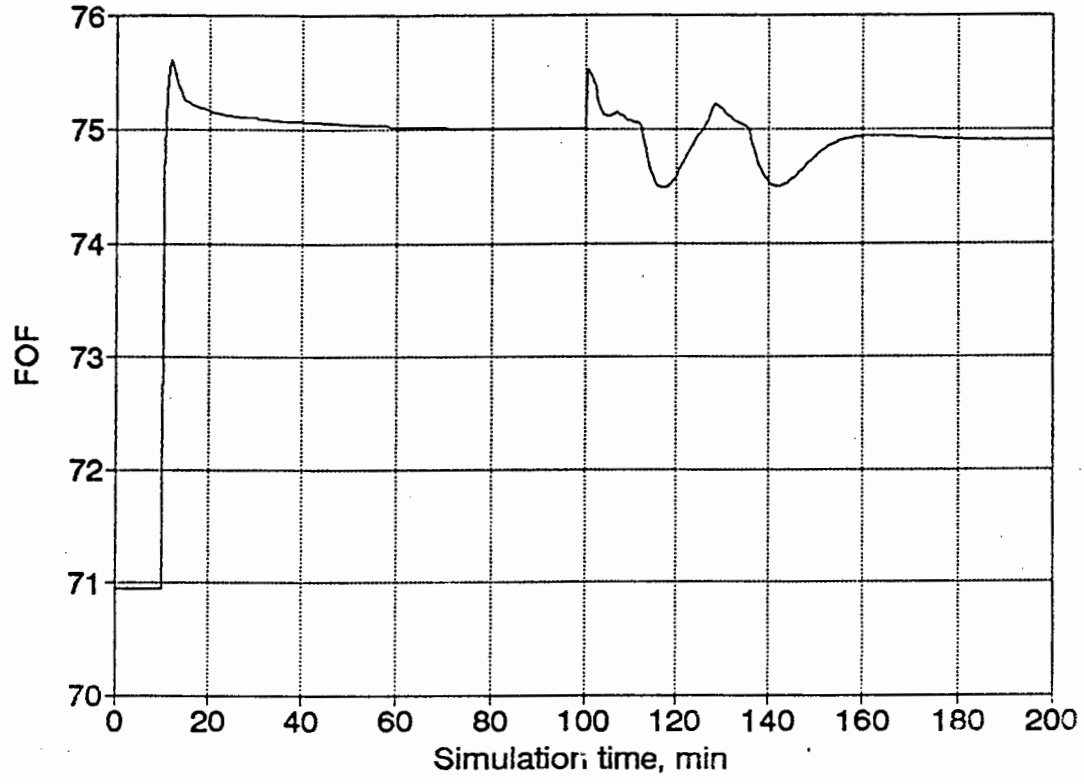
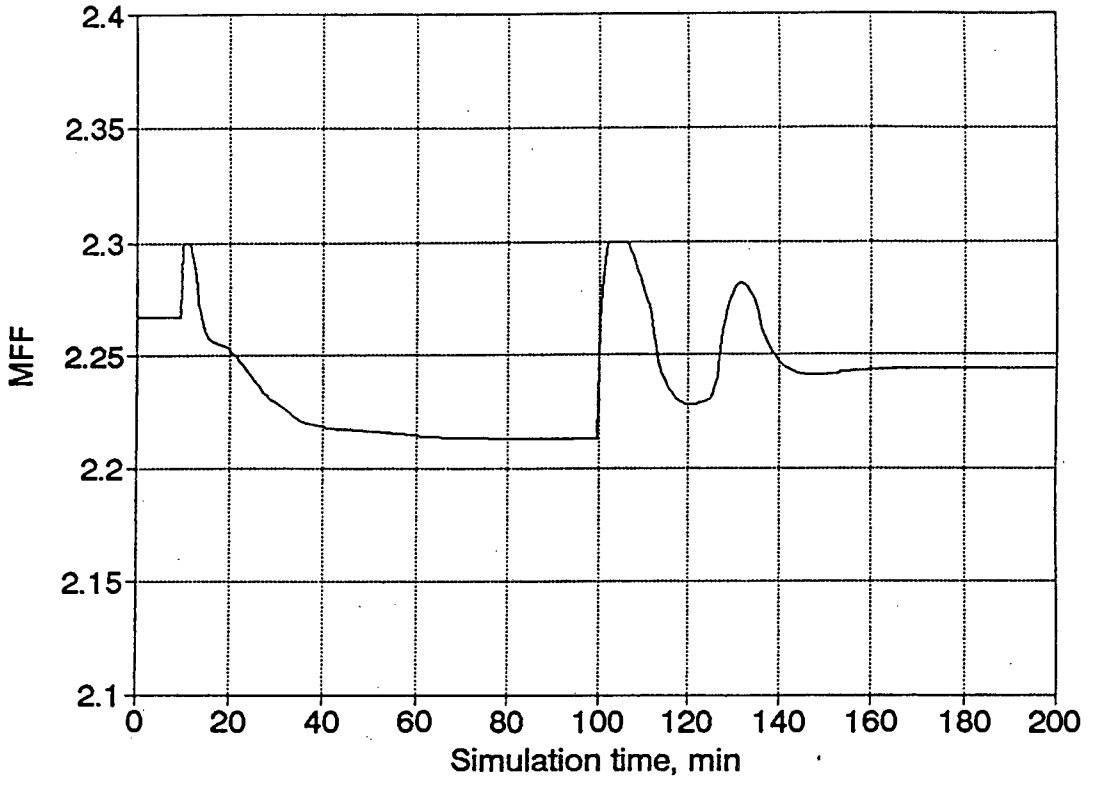


Figure 5.13 Process outputs of a constrained Dynamic Matrix Control of the milling circuit. Constraints are: ($11.0 \leq WSF \leq 16.0$) and ($2.2 \leq MFF \leq 2.3$). M=94, N=6, P=100, Q=I and Λ=0.



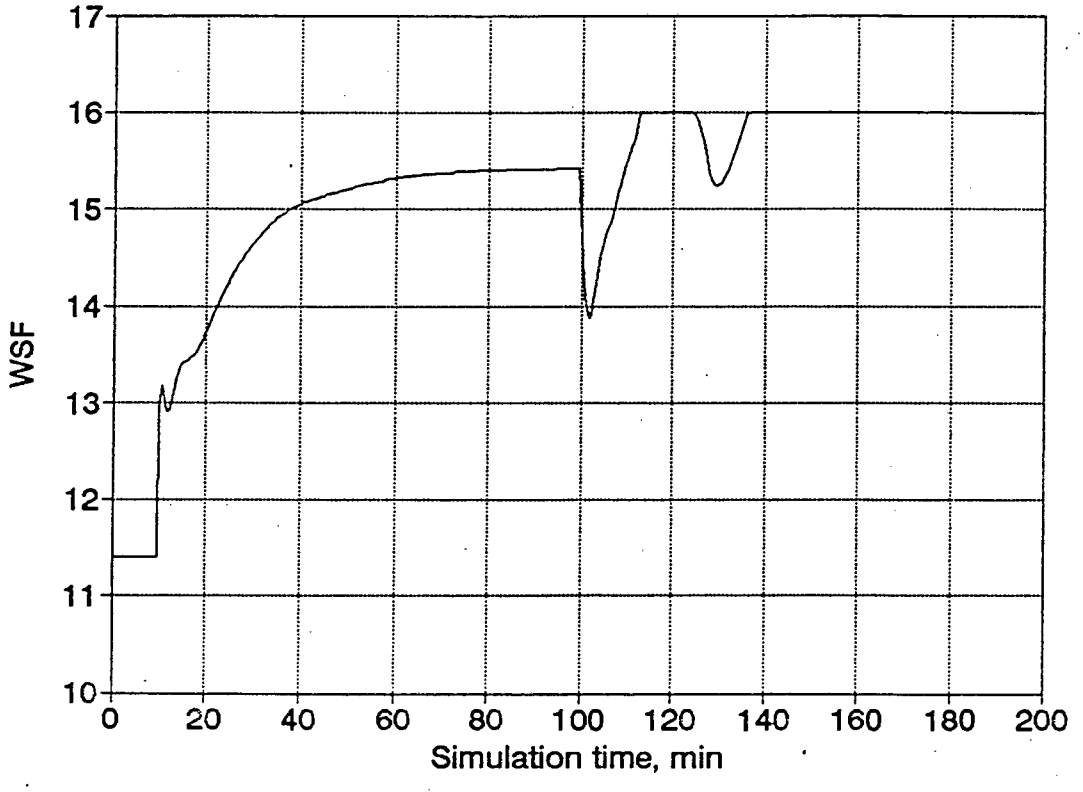


Figure 5.14 Process inputs of a constrained Dynamic Matrix Control of the milling circuit. Constraints are: ($11.0 \leq WSF \leq 16.0$) and ($2.2 \leq MFF \leq 2.3$). M=94, N=6, P=100, Q=I and Λ=0.

CHAPTER 6

CONCLUSION

The aim of the study was to investigate the suitability of Dynamic Matrix Control (DMC) for milling circuit control. The study was conducted through simulation studies using two different milling circuit models; one being linear and the other nonlinear. A generalised software package was developed for the implementation and evaluation of DMC.

Since milling circuits are multivariable in nature with strong interactions and time delays, and demand that they be operated close to constraints for maximum profit, DMC with its features appeared suitable for the control of these circuits.

This study has shown that DMC can be succesfully implemented in milling circuit control as in the other industries. Its performance has demonstrated its abilities to handle interactions and time delays inherent in the milling circuits. Since DMC is model-based control, it can be expected that a good model of the process will produce good control. However, it was observed that it is not too critical to obtain the full representation of the plant by basing the sampling rate on the fastest response. DMC performance was still reasonably good when the sampling rate was decreased. The advantage of decreasing the sampling rate is that the Dynamic Matrix becomes smaller producing a smaller control problem which reduces memory consumption as well as computational load. Nevertheless, for large systems, multi-rate sampling should be investigated which should reduce memory requirements.

DMC has been found to reject unknown disturbances without offset through its built-in step estimation even if they are not step type and have time delays. The linear model has a disturbance model associated with it. A feedforward controller was readily designed into the basic DMC algorithm. It was shown that a feedforward controller can effectively reject the disturbance.

It was also shown that constraints can be successfully imposed on control moves, inputs and outputs of the process. However, it was seen that outputs are not always guaranteed to remain within bounds in the presence of unmeasured disturbances. Output constraints should be carefully selected and tested before being implemented on-line. An in depth study on stability under constraints should be conducted.

DMC was implemented on a nonlinear model to investigate the effects of model errors associated with nonlinearity. DMC was found to be robust enough to achieve the control objectives in the face of model nonlinearities, even when the control move horizon was reduced. Nonlinearity is however, only one of the sources of model errors. In order to be able to reasonably quantify the gains of using DMC in milling circuit control, it is suggested that real-time control be investigated since it will avoid simplistic assumptions with regard to model errors and also avoid simplifying assumptions inherent in the generation of a plant model.

While the basic function of a control system is to stabilize the process performance at a desired level, by compensating for disturbances to the system, the ultimate objective is not only stabilization but also optimization of the process performance based on economic considerations. On-line optimization of milling circuits is therefore recommended.

REFERENCES

Arkun Y., Charos G. and Reeves D.E., "A Course in Model Predictive Control", Chem. Eng. Educ., 178, Fall (1988).

Avriel M., "Nonlinear Programming: Analysis and Methods", Prentice-Hall Inc., 1976.

Buequette B.W., "Nonlinear Predictive Control Using Multi-rate Sampling", The Canadian J. of Chem. Eng., 69, 136-146 (1991).

Caldwell J.M. and Dearwater J.G., "Model Predictive Control Applied to FCC Units", Fourth Int. Conf. on Chemical Process Control, South Padre Island, Texas, Feb. 1991.

Consyd - Integrated Software For Computer Aided Control System Design and Analysis., Comp. Chem. Eng., 11 (2), 187-203 (1987).

Cooper L. and Steinberg D., "Introduction to Methods of Optimization", W.B. Saunders Co., 1970.

Culter C.R., Haydel J.J. and Morshedi A.M., "An Industrial Perspective on Advanced Control", Paper presented at the AIChE Ann. Meeting, (1983).

Cutler C.E., and Ramaker B.L., "Dynamic Matrix Control - A Computer Control Algorithm", Paper presented at AIChE Natnl. Mtng. Houston, (1979).

Cutler C.R. and Hawkins R.B., "Application of a Large Predictive Multivariable Controller to a Hydrocracker Second Stage Reactor", American Control Conference, Minnepolis, (1987).

Edgar T.F. and Himmelblau D.M., "Optimization of Chemical Processes", McGraw-Hill, 1988.

Erickson K.T., and Otto R.E., " Development of a Multivariable Forward Modeling Controller", Ind. Eng. Chem. Res., Vol 30, 3, (1991).

Garcia C.E. and Morari M., "Internal Model Control. 1. A Unifying Review and Some New Results", Ind. Eng. Chem. Process Dev., 21 (2), 308 -323 (1982).

Garcia C.E., and Morshedi A.M., " Quadratic Programming Solution of Dynamic Matrix Control", Chem. Eng. Comm., 46, (1986).

Garcia C.E., and Prett D.M., "Advances in Industrial Model-Predictive Control", in Chemical Process Control - CPC III, Morari M. and McAvoy T.J. Eds., CACHE, Elsevier, (1986).

Garcia C.E., Prett D.M. and Morari M., "Model Predictive Control: Theory and Practice - A Survey", Automatica, 25 (3), 335-348 (1989).

Georgiou A., Georgakis C. and Luyben W.L., "Nonlinear Dynamic Matrix

Control for High-Purity Distillation Columns", 34 (8), 1287-1298 (1988).

Gill P.E., Murray W., and Wright M.H., "Practical Optimization", Academic Press, 1981.

Grimm W.M., Lee P.L and Callaghan P.J., "Practical Robust Predictive Control of a Heat Exchange Network, Chem. Eng. Comm., 81, 25-53 (1989).

Hulbert D.G., and Braae M., " Multivariable Control of a Milling Circuit at East Driefontein Gold Mine", NIM report no. 2113, (1981).

Jacobson D.H., " Extensions of Linear - Quadratic Control, Optimization and Matrix Theory", Academic Press, 133, (1977).

Li S., Lim K.Y. and Fisher D.G., "A State Space Formulation for Model Predictive Control", AIChE J., 35 (2), 241-249 (1989).

Lynch A.J., "Mineral Crushing and Grinding Circuits: Their Simulation, Optimisation, Design and Control. Elsevier, Amsterdam, 1977.

Maurath P.R., Laub A.J, Seborg D.E. and Mellichamp D.A., "Predictive Controller Design by Principal Components Analysis", Ind. Eng. Chem. Res., 27, 1204-1211 (1988).

Maurath P.R., Mellichamp D.A. and Seborg D.E., "Predictive Controller Design for Single-Input/Single-Output Systems", Ind. Eng. Chem. Res., 27, 956-963 (1988).

McDonald K. and McAvoy T.J., "Application of Dynamic Matrix Control to Moderate- and High-Purity Distillation Towers", Ind. Eng. Chem. Res., 26, 1011-1018 (1987).

Mehra R. and Rouhani R., "Thoretical Considerations on Model Algorithmic Control for Nonminimum Phase Systems, Proc. Joint Aut. Control Conf., San Fransisco, California, (1980).

McFarlane R.C. and Bacon D.W., "Adaptive Optimizing Control of Multivariable Constrained Chemical Processes. 1. Theoretical Development", Amer. Chem. Soc., 28, 1828-1834 (1989).

McFarlane R.C. and Bacon D.W., "Adaptive Optimizing Control of Multivariable Constrained Chemical Processes. 2. Application Studies", Amer. Chem. Soc., 28, 1834-1845 (1989).

Morari M., and Lee J.H.," Model Predictive Control: The Good, the Bad, and The Ugly", CPC IV, South Padre Island, Texas, Feb. 1991.

Morshedi A.M., Cutler C.R. and Skrovanek T.A., "Optimal Solution of Dynamic Matrix Control with Linear Programming Techniques", Amer. Contr. Conf., 1985.

Morshedi A.M., "Universal Dynamic Matrix Control", CPC III, CACHE, Elsevier, 1986.

Ogunnaike B.A. and Adewale K.E.P., "Dynamic Matrix Control for Process Systems with Time-Varying Parameters", Chem. Eng. Comm., 47, 295

-314 (1986).

Ogunnaike B.A., "Dynamic Matrix Control: A Nonstochastic Industrial Process Control Technique with Parallels in Applied Statistics", Ind. Eng. Chem. Fund., 25, 712-718 (1986).

Palazoglu A., Fruzzetti K.P., Romagnoli J.A. and McDonald K.A., "Robust Multivariable Predictive Control - A Linear Programming Approach", AIChE Annual Mtng., San Fransisco, 1989.

Park S., "The Application of an Optimized DMC Multivariable Controller to the PCC Catalytic Cracking Unit", 1982.

Prett D.M. and Garcia C.E., "Fundamental Process Control", Butterworths, Boston (1988).

Rajamani R.K. and Herbst J.A., "Optimal Control of a Ball Mill Grinding Circuit - I. Grinding Circuit Modeling and Dynamic Simulation. Chem. Eng. Sci., 46 (3), 861-870 (1991).

Rajamani R.K. and Herbst J.A., "Optimal Control of a Ball Mill Grinding Circuit - II. Feedback and Optimal Control., Chem. Eng. Sci., 46 (3), 871-879 (1991).

Ray W.H., "Advanced Process Control", McGraw-Hill, NY, 1981.

Ricker N.L., "Model-Predictive Control: State of the Art", CPC IV, South Padre Island, Texas, Feb. 1991.

Ricker N.L., Subrahmanian T. and Sim T., "Case Studies of Model Predictive Control in Pulp and Paper Production", Proceedings of the IFAC Workshop on Model Based Process Control, T.J. McAvoy, Arkun Y. and Zafiriou E, eds., Pergamon Press, Oxford, 1989.

Venugopal S. and Gupta Y.P., "Predictive Control of a Circulating Fluidized Bed Combustor", The Can. J. of Chem. Eng., 69, 130-134 (1991).

Wills B. A., "Mineral Processing Technology", Pergamon Press, Third ed., 1985.

Zafiriou E., "Robust Model Predictive Control of Processes with Hard Constraints", Comp. in Chem. Eng., 1989.

Zafiriou E. and Chiou H-W, "An Operator Control Theory Approach to the Design and Tuning of Constrained Model Predictive Controllers", AIChE Annual Mtng., San Fransisco, 1989.

PROGRAM LISTINGS

```
      PROGRAM MAIN
C
C*************************************************************
C                    DMC                                      *
C                    ---                                      *
C  Dynamic Matrix Control (DMC) is a multivariable control    *
C  algorithm. The method calculates a sequence of future      *
C  moves in the control inputs which minimizes the            *
C  projections of future errors between the process outputs   *
C  and their setpoints in the least-square sense. The         *
C  process model which DMC uses as a basis for making        *
C  predictions of future output behavior is given in the      *
C  form of step-response or impulse-response data.            *
C                                                            *
C  Basic tuning of the DMC controller is accomplished by      *
C  specifying how many future control moves to solve for as   *
C  well as how many sample intervals into the future to       *
C  predict the system response.                               *
C                                                            *
C  Further tuning can be performed by scaling of the input    *
C  and output variables relative to one another, by           *
C  assigning absolute penalties on changes in the             *
C  controller actions.                                        *
C                                                            *
C                         QDMC                               *
C                         ----                               *
C                                                            *
C  QDMC is a QP solution of the DMC equations. In practice    *
C  the computed soluiton in the unconstrained DMC may         *
C  not be implementable due to process operating limit        *
C  violations.                                                *
C                                                            *
C  Three types of process constraints are usually             *
C  encountered:                                               *
C                                                            *
C  Manipulated variable constraints: i.e valve saturation.    *
C  Controlled variable constraints: overshoots in the         *
C  controlled variables past allowable limits must be         *
C  avoided.                                                   *
C  Associated variables: key process variables which are      *
C  not directly controlled but that must be kept within       *
C  bounds.                                                    *
C                                                            *
C  The controller must be able to predict future violations   *
C  and prescribe moves that would keep these variables        *
C  within bounds.                                            *
C                                                            *
C  All tuning parameters for DMC still apply for              *
C  constrained case. However, in QDMC control quality is      *
C  additionally influenced by the selection of the            *
C  projection interval to be constrained. In practice, only   *
C  a subset of all P projections are constraints starting     *
C  with the 1th projection, where 1>1. This subset of         *
C  projections form a constraint window of future intervals   *
C  of time over which QDMC will prevent constraint            *
C  violations from occuring.                                  *
C                                                            *
C  In the presence of non-minimum phase behavior of           *
C  controlled and associated varaibles much improvement in    *
```

```
C  perfomance is achieved by moving the "constraint window"*
C  further in the horizon. The reason is that any projected*
C  violation inside the "constraint window" is handled  *
C  rigorously by the QP, not unlike a tightly tuned   *
C  controller. Therefore, if the QP is asked to correct for*
C  violations in the earlier projections, severe input *
C  moves might be required in the face of non-minimum phase*
C  characteristics.                                     *
C                                                       *
C*******************************************************
C
C     VARIABLE DECLARATION
C     --------------------
C
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DOUBLE PRECISION LAM,NSTATE
C
      INCLUDE 'PARAM.INC'
C
      INTEGER P
      DIMENSION YS(CCO),YSS(CCO),DISTUR(CCO),DY(CCO)
      DIMENSION Q(MAXIM),R(NMAXIM),DU(CCO),DUS(CCO)
      DIMENSION YM(CCO),U(CCO),USS(CCO)
      INTEGER ORDER,DORDER,DNUMIN
C
      DIMENSION STATE(IORDER,IORDER,IORDER),
     &          DIRSTA(IORDER,IORDER,IORDER),
     &          DSTATE(IORDER,IORDER,IORDER),
     &          DDIRSTA(IORDER,IORDER,IORDER),
     &          NSTATE(IORDER,IORDER),
     &          NDIRSTA(IORDER,IORDER)
C
      CHARACTER*1 ANS
C
      COMMON / CONTROL / ERROR(MAXIM),YI(MAXIM),
     &                   A(MAXIM,NMAXIM),LAM(NMAXIM,MAXIM),
     &                   DUM(NMAXIM,NMAXIM),FF(MAXIM),ANS5
      COMMON / DELAYS / TIMDEL(CCO,CCO), DTIMDEL(CCO,CCO)
C
C     CALL INITIALISING ROUTINE
C     -------------------------
C
      CALL INITIAL(DIRSTA,DISTUR,DSAMPL,DELTAT,YM,U,M,R,N,
     &             NUMIN,NUMOUT,ORDER,P,YS,YSS,STATE,DU,DUS,
     &             DY,TEST,TIME,TSIM,Q,PEB,DNUMIN,DORDER,
     &             USS,NSTATE,NDIRSTA)
C
      TIME = 0.0
C
C     FORM CONTROLLER MATRIX
C     ----------------------
C
      CALL MATRIX(R,N,NUMIN,NUMOUT,M,P,Q,DISTUR,DSAMPL,
     &            DELTAT,YM,U,ORDER,YS,YSS,DU,DUS,TIME,TSIM,
     &            DNUMIN,DORDER,USS)
C
C
C     SIMULATION BEGINS
C     -----------------
C
      CALL SIMUL(DISTUR,DSAMPL,DELTAT,YM,U,N,NUMIN,NUMOUT,
     &           ORDER,P,YS,YSS,DU,DUS,TIME,TSIM,DNUMIN,
     &           DORDER,USS)
C
```

```
C     END SIMULATION
C     --------------
C
      STOP
      END
C
C*****************************************************************
      SUBROUTINE INITIAL(DIRSTA,DISTUR,DSAMPL,DELTAT,YM,U,M,R,
     &                   N,NUMIN,NUMOUT,ORDER,P,YS,YSS,STATE,DU,
     &                   DUS,DY,TEST,TIME,TSIM,Q,PEB,DNUMIN,DORDER,
     &                   USS,NSTATE,NDIRSTA)
C*****************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DOUBLE PRECISION LAM,NSTATE
C
      INCLUDE 'PARAM.INC'
C
      INTEGER DNUMIN
      INTEGER P
      DIMENSION YS(CCO),YSS(CCO),DISTUR(CCO),DY(CCO)
      DIMENSION Q(MAXIM),R(NMAXIM),DU(CCO),DUS(CCO)
      DIMENSION YM(CCO),U(CCO),USS(CCO)
      INTEGER ORDER,DORDER
C
      DIMENSION STATE(IORDER,IORDER,IORDER),
     &          DIRSTA(IORDER,IORDER,IORDER),
     &          DSTATE(IORDER,IORDER,IORDER),
     &          DDIRSTA(IORDER,IORDER,IORDER),
     &          NSTATE(IORDER,IORDER),
     &          NDIRSTA(IORDER,IORDER)
C
      DOUBLE PRECISION NUMBER,HM,KO,MUF
C
      CHARACTER*1 ANS5
C
      COMMON / CONTROL / ERROR(MAXIM),YI(MAXIM),
     &                   A(MAXIM,NMAXIM),LAM(NMAXIM,MAXIM),
     &                   DUM(NMAXIM,NMAXIM),FF(MAXIM),ANS5
      COMMON / SPACE / ASTATE(IORDER,IORDER,IORDER,IORDER),
     &                 BSTATE(IORDER,IORDER,IORDER),
     &                 CSTATE(IORDER,IORDER,IORDER),
     &                 DASTATE(IORDER,IORDER,IORDER,IORDER),
     &                 DBSTATE(IORDER,IORDER,IORDER),
     &                 DCSTATE(IORDER,IORDER,IORDER)
C
      COMMON / SPACEN / HM,RUF,MUF,KO,VS,QM,RW,RS,RFF
      COMMON / STEADYN / SSTATE(IORDER,IORDER)
C
      COMMON / DELAYS / TIMDEL(CCO,CCO),DTIMDEL(CCO,CCO)
      COMMON / TEMPOUT / Y(CCO,CCO,TIMEDELAY),YD(CCO,CCO,TIMEDELAY)
C
C     READ DATA FORM FILE "DATA2.DAT'
C     -------------------------------
C
      INTEGER PRESS,J,K,L,I
C
C     INITIALISING VARIABLES
C     ----------------------
C
      TIME  = 0.0
      DO 10 I = 1 , NUMOUT
        DY(I)  = 0.0
10    CONTINUE
C
```

```
      TEST  = 0.0
      DELTAT = 0.0
C
C     INITIALISE CONTROLLER VARIABLES
C     -------------------------------
C
      N      = 0
      M      = 0
      P      = 0
      NUMIN = 0
      NUMOUT= 0
      DSAMPL= 0.0
      DNUMIN = 0
      PEB    = 0.0
C
      DO 20 I = 1 , CCO
            YS(I)  = 0.0
            YSS(I) = 0.0
            DISTUR(I) = 0.0
20    CONTINUE
C
      DO 30 I = 1 , MAXIM
         DO 40 J = 1 , NMAXIM
            A(I,J)   = 0.0
            LAM(J,I) = 0.0
            R(J)  = 0.0
            DUM(J,J) = 0.0
40       CONTINUE
C
      Q(I) = 0.0
      ERROR(I) = 0.0
      YI(I) = 0.0
30    CONTINUE
C
C     INITIALISING MODEL & DISUTRBANCE VARIABLES
C     ------------------------------------------
C
      DO 50 I = 1 , IORDER
          DO 60 J = 1 , IORDER
              DO 70 K = 1 ,IORDER
                 DO 80 L = 1 , IORDER
                    ASTATE(I,J,K,L)=0.0
                    DASTATE(I,J,K,L)=0.0
80               CONTINUE
                 BSTATE(I,J,K)=0.0
                 CSTATE(I,J,K)=0.0
                 DBSTATE(I,J,K)=0.0
                 DCSTATE(I,J,K)=0.0
                 STATE(I,J,K)=0.0
                 DIRSTA(I,J,K)=0.0
                 DSTATE(I,J,K)=0.0
                 DDIRSTA(I,J,K)=0.0
70            CONTINUE
              NSTATE(I,J)=0.0
              SSTATE(I,J)=0.0
              NDIRSTA(I,J)=0.0
60        CONTINUE
50    CONTINUE
      DO 90 I = 1 , CCO
         DU(I) = 0.0
         DUS(I) = 0.0
         USS(I) = 0.0
         YM(I) = 0.0
         DO 100 J = 1 , CCO
```

```
            TIMDEL(I,J) = 0.0
            DTIMDEL(I,J) = 0.0
100        CONTINUE
90     CONTINUE
C
7      FORMAT (I3,F11.6)
C
          OPEN(UNIT =11 , FILE = 'DATA.DAT',STATUS='OLD')
          READ(11,7) PRESS,NUMBER
C
110        IF ( PRESS .NE. 0 ) THEN
C
             IF (PRESS .EQ. 1) THEN
                TSIM = NUMBER
                READ(11,7) PRESS,NUMBER
             END IF
C
             IF ( PRESS .EQ. 2 ) THEN
                 DELTAT = NUMBER
                 READ(11,7) PRESS,NUMBER
             END IF
C
             IF ( PRESS .EQ. 3 ) THEN
                 DSAMPL = NUMBER
                 READ(11,7) PRESS,NUMBER
             END IF
C
             IF ( PRESS .EQ. 4 ) THEN
                NUMIN = NUMBER
                READ(11,7) PRESS,NUMBER
             END IF
C
             IF ( PRESS .EQ. 5 ) THEN
                NUMOUT = NUMBER
                READ(11,7) PRESS,NUMBER
             END IF
C
             IF ( PRESS .EQ. 6 ) THEN
                N = NUMBER
                READ(11,7) PRESS,NUMBER
             END IF
C
             IF ( PRESS .EQ. 7 ) THEN
                ORDER = NUMBER
                READ(11,7) PRESS,NUMBER
             END IF
C
             IF ( PRESS .EQ. 8 ) THEN
                DORDER = NUMBER
                READ(11,7) PRESS,NUMBER
             END IF
C
             IF ( PRESS .EQ. 9 ) THEN
                DNUMIN = NUMBER
             END IF
C
8            FORMAT(I3,1X,4I2,1X,F8.6)
             READ(11,8) PRESS,I,J,K,L,NUMBER
C
120          IF (PRESS .EQ. 10) THEN
                ASTATE(I,J,K,L) = NUMBER
                READ(11,8) PRESS,I,J,K,L,NUMBER
                GOTO 120
             END IF
```

```
C
130         IF (PRESS .EQ. 11) THEN
               BSTATE(I,J,K) = NUMBER
               READ(11,8) PRESS,I,J,K,L,NUMBER
               GOTO 130
            END IF
C
140         IF ( PRESS .EQ. 12) THEN
               CSTATE(I,J,K) = NUMBER
               READ(11,8) PRESS,I,J,K,L,NUMBER
               GOTO 140
            END IF
C
150         IF ( PRESS .EQ. 13) THEN
                TIMDEL(I,J) = NUMBER
                READ(11,8) PRESS,I,J,K,L,NUMBER
                GOTO 150
            END IF
C
160         IF (PRESS .EQ. 14) THEN
               DASTATE(I,J,K,L) = NUMBER
               READ(11,8) PRESS,I,J,K,L,NUMBER
               GOTO 160
            END IF
C
170         IF (PRESS .EQ. 15) THEN
               DBSTATE(I,J,K) = NUMBER
               READ(11,8) PRESS,I,J,K,L,NUMBER
               GOTO 170
            END IF
C
180         IF ( PRESS .EQ. 16) THEN
               DCSTATE(I,J,K) = NUMBER
               READ(11,8) PRESS,I,J,K,L,NUMBER
               GOTO 180
            END IF
C
190         IF ( PRESS .EQ. 17) THEN
                DTIMDEL(I,J) = NUMBER
                READ(11,8) PRESS,I,J,K,L,NUMBER
                GOTO 190
            END IF
C
C           READ NONLINEAR STEADY STATE PARAMETERS
C           --------------------------------------
C
            OPEN(UNIT =12 , FILE = 'DATA2.DAT',STATUS='OLD')
C
            READ(12,7) PRESS,NUMBER
            IF (PRESS .EQ. 1) THEN
               HM = NUMBER
               READ(12,7) PRESS,NUMBER
            END IF
C
            IF (PRESS .EQ. 2) THEN
               MUF = NUMBER
               READ(12,7) PRESS,NUMBER
            END IF
C
            IF (PRESS .EQ. 3) THEN
               KO = NUMBER
               READ(12,7) PRESS,NUMBER
            END IF
C
```

```
            IF (PRESS .EQ. 4) THEN
               VS = NUMBER
               READ(12,7) PRESS,NUMBER
            END IF
C
            IF (PRESS .EQ. 5) THEN
               QM = NUMBER
               READ(12,7) PRESS,NUMBER
            END IF
C
            IF (PRESS .EQ. 6) THEN
               RW = NUMBER
               READ(12,7) PRESS,NUMBER
            END IF
C
            IF (PRESS .EQ. 7) THEN
               RS = NUMBER
               READ(12,7) PRESS,NUMBER
            END IF
C
            IF (PRESS .EQ. 8) THEN
               RFF = NUMBER
            END IF
C
            READ(12,8) PRESS,I,J,K,L,NUMBER
200         IF ( PRESS .EQ. 9) THEN
                SSTATE(I,J) = NUMBER
                READ(12,8) PRESS,I,J,K,L,NUMBER
                GOTO 200
            END IF
C
210         IF ( PRESS .EQ. 10) THEN
                USS(J) = NUMBER
                READ(12,8) PRESS,I,J,K,L,NUMBER
                GOTO 210
            END IF
C
220         IF ( PRESS .EQ. 11) THEN
                YSS(I) = NUMBER
                READ(12,8) PRESS,I,J,K,L,NUMBER
                GOTO 220
            END IF
C
           GOTO 110
         END IF
        CLOSE(UNIT=11)
        CLOSE(UNIT=12)
C
C
C     READ A MATRIX FROM DISK
C     -----------------------
C
      OPEN ( UNIT = 13 , FILE = NAME//'.DIST',STATUS='UNKNOWN')
      READ(9,*) M,P
      DO 280  I = 1, NUMIN
        DO 290 J = 1, NUMOUT
          DO 300 K = 1, M
            READ(9,*) A(K+(J-1)*P,1+(I-1)*N)
300       CONTINUE
290     CONTINUE
280   CONTINUE
      CLOSE(UNIT=9)
C
C     READ FF MATRIX FROM DISK
```

```
C     ------------------------
C
        OPEN ( UNIT = 13 , FILE = NAME//'.DIST',STATUS='UNKNOWN')
        DO 430 J = 1, NUMOUT
          DO 440 K = 1, P
            READ(13,*) FF((J-1)*P+K)
440       CONTINUE
430     CONTINUE
      CLOSE(UNIT=13)
C
      RETURN
      END
C
C*****************************************************************
      SUBROUTINE DERIVS(DIRSTA,U,NUMIN,NUMOUT,ORDER,STATE)
C*****************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C
      INCLUDE 'PARAM.INC'
C
      DIMENSION U(CCO)
      DIMENSION STATE(IORDER,IORDER,IORDER),
     &          DIRSTA(IORDER,IORDER,IORDER)
C
      INTEGER NUMIN,NUMOUT,ORDER
C
      COMMON / SPACE / ASTATE(IORDER,IORDER,IORDER,IORDER),
     &                 BSTATE(IORDER,IORDER,IORDER),
     &                 CSTATE(IORDER,IORDER,IORDER),
     &                 DASTATE(IORDER,IORDER,IORDER,IORDER),
     &                 DBSTATE(IORDER,IORDER,IORDER),
     &                 DCSTATE(IORDER,IORDER,IORDER)
C
      COMMON / DELAYS / TIMDEL(CCO,CCO),DTIMDEL(CCO,CCO)
C
      DO 10 I = 1 , NUMOUT
         DO 20 J=1,NUMIN
            DO 30 K=1,ORDER
               DIRSTA(I,J,K) = 0.0
               DO 40 L = 1 , ORDER
                  DIRSTA(I,J,K) = DIRSTA(I,J,K) + ASTATE(I,J,K,L)
     &                            *STATE(I,J,K)
40             CONTINUE
30          CONTINUE
20       CONTINUE
10    CONTINUE
C
      DO 50 I=1,NUMOUT
         DO 60 J=1,NUMIN
            DO 70 K=1,ORDER
               DIRSTA(I,J,K) = DIRSTA(I,J,K) + BSTATE(I,J,K)*
     &                         U(J)
70          CONTINUE
60       CONTINUE
50    CONTINUE
C
      RETURN
      END
C
C*****************************************************************
      SUBROUTINE DDERIVS(DDIRSTA,PEB,DNUMIN,NUMOUT,DORDER,DSTATE)
C*****************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C
```

```
      INCLUDE 'PARAM.INC'
C
      DIMENSION DSTATE(IORDER,IORDER,IORDER),
     &          DDIRSTA(IORDER,IORDER,IORDER)
C
      INTEGER DNUMIN,NUMOUT,DORDER
C
      COMMON / SPACE / ASTATE(IORDER,IORDER,IORDER,IORDER),
     &                 BSTATE(IORDER,IORDER,IORDER),
     &                 CSTATE(IORDER,IORDER,IORDER),
     &                 DASTATE(IORDER,IORDER,IORDER,IORDER),
     &                 DBSTATE(IORDER,IORDER,IORDER),
     &                 DCSTATE(IORDER,IORDER,IORDER)
C
      COMMON / DELAYS / TIMDEL(CCO,CCO),DTIMDEL(CCO,CCO)
C
      DO 10 I = 1 , NUMOUT
         DO 20 J=1,DNUMIN
            DO 30 K=1,DORDER
               DDIRSTA(I,J,K) = 0.0
               DO 40 L = 1 , DORDER
                  DDIRSTA(I,J,K) = DDIRSTA(I,J,K) + DASTATE(I,J,K,L)
     &                             *DSTATE(I,J,K)
40             CONTINUE
30          CONTINUE
20       CONTINUE
10    CONTINUE
C
      DO 50 I=1,NUMOUT
         DO 60 J=1,DNUMIN
            DO 70 K=1,DORDER
               DDIRSTA(I,J,K) = DDIRSTA(I,J,K) + DBSTATE(I,J,K)*
     &                          PEB
70          CONTINUE
60       CONTINUE
50    CONTINUE
C
      RETURN
      END
C
C*********************************************************************
      SUBROUTINE NDERIVS(NDIRSTA,U,NSTATE)
C*********************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DOUBLE PRECISION NSTATE,NDIRSTA
      DOUBLE PRECISION HM,KO,MUF
C
      INCLUDE 'PARAM.INC'
C
      DIMENSION U(CCO)
      DIMENSION NSTATE(IORDER,IORDER),
     &          NDIRSTA(IORDER,IORDER)
C
      COMMON / SPACEN / HM,RUF,MUF,KO,VS,QM,RW,RS,RFF
C
      COMMON / PARAN / MFF,WFF,WSF,QC,C1,C2,RM,CS,ROS,WF
C
      NDIRSTA(1,1) = (U(1)*RFF + MUF*RUF - KO*HM*NSTATE(1,1) -
     &                (U(1)+MUF)*NSTATE(1,1))/HM
C
      NDIRSTA(2,2) = (MUF+U(1) - (QM+(U(2)/RW))*NSTATE(2,2))/VS
C
      NDIRSTA(3,3) = ((MUF+U(1))*(NSTATE(1,1)-NSTATE(3,3)))/
     &                (VS*NSTATE(2,2))
```

```
C
      RETURN
      END
C
C*****************************************************************
      SUBROUTINE INTEG(DELTAT,NSTATE,U,NUMIN,NUMOUT,TIME,YM)
C*****************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DOUBLE PRECISION NSTATE
      DOUBLE PRECISION HM,KO,MUF,MMF
C
      INCLUDE 'PARAM.INC'
C
      DIMENSION U(CCO),YM(CCO)
      DIMENSION YT(IORDER,IORDER),
     &          DYT(IORDER,IORDER),
     &          DYM(IORDER,IORDER),
     &          DYDX(IORDER,IORDER)
C
      DIMENSION NSTATE(IORDER,IORDER)
C
      INTEGER NUMIN,NUMOUT
C
      COMMON / SPACEN / HM,RUF,MUF,KO,VS,QM,RW,RS,RFF
C
      COMMON / PARAN / MFF,WFF,WSF,QC,C1,C2,RM,CS,ROS,WF
C
      HH=DELTAT*0.5
      H6=DELTAT/6.0
      XH=DELTAT+HH
C
C     PERFORMING THE RUNGE-KUTTA ON THE PROCESS STATES
C     ------------------------------------------------
C
      CALL NDERIVS(DYDX,U,NSTATE)
C
      DO 10 I=1,NUMOUT+1
         DO 20 J=1,NUMIN+1
           YT(I,J)=NSTATE(I,J)+HH*DYDX(I,J)
20       CONTINUE
10    CONTINUE
C
      CALL NDERIVS(DYT,U,YT)
C
      DO 30 I=1,NUMOUT+1
         DO 40 J=1,NUMIN+1
           YT(I,J)=NSTATE(I,J)+HH*DYT(I,J)
40       CONTINUE
30    CONTINUE
C
      CALL NDERIVS(DYM,U,YT)
C
      DO 50 I=1,NUMOUT+1
         DO 60 J=1,NUMIN+1
           YT(I,J)=NSTATE(I,J)+DELTAT*DYM(I,J)
           DYM(I,J)=DYM(I,J)+DYT(I,J)
60       CONTINUE
50    CONTINUE
C
      CALL NDERIVS(DYT,U,YT)
C
      DO 70 I=1,NUMOUT+1
         DO 80 J=1,NUMIN+1
           NSTATE(I,J)=NSTATE(I,J)+H6*(DYDX(I,J)+
```

```
     &                    DYT(I,J)+2*DYM(I,J))
80        CONTINUE
70     CONTINUE
C
C      PARAMETERS
C      ----------
C
       A1=1.363
       A2=-10.75
       WFF=U(1)*0.4/0.6
C
C      CALC ALGEBRAIC EQUATIONS
C      ------------------------
C
100    QM = ((U(1)+MUF)/RS + WFF/RW - (A2)/RW +
     &      ((1.0-A1)*(1.0-(NSTATE(2,2)/RS))*U(2))/RW)/
     &      (1.0 - (1.0-(NSTATE(2,2)/RS))*(1.0-A1))
C
       WF = (QM+U(2)/RW)*(1-(NSTATE(2,2)/RS))*RW
       IF( WF .GT. 21.4) THEN
         A1 = 0.837
         A2 = 0.35
         GOTO 100
       END IF
C
       QC = QM + U(2)/RW
C
       FV = NSTATE(2,2)/RS
C
       D50 = EXP(3.5433 - 1.6895E-01*QC*60 + 3.2449*FV)
C
       C1 = 0.0042*U(1)*D50 - 0.0154*D50 - 0.0704*U(1) + 1.3412
C
       C2 = 0.0502*U(1)*D50 - 0.0660*D50 - 2.9354*U(1) + 4.642
C
       RUF = C1*NSTATE(3,3)/(C1*NSTATE(3,3)+C2*(1-NSTATE(3,3)))
C
C      CALCULATE OUTPUTS
C      -----------------
C
       FOF = (1-C2)*(1-NSTATE(3,3))*100/((1-C2)*(1-NSTATE(3,3))
     &       +(1-C1)*NSTATE(3,3))
       YM(2)=FOF
C
       MMF = U(1) + MUF
       YM(1) = MMF
C
       TIME = TIME + DELTAT
C
       RETURN
       END
C
C*****************************************************************
       SUBROUTINE NONMODEL(DELTAT,NSTATE,U,NUMIN,NUMOUT,TIME,YM)
C*****************************************************************
       IMPLICIT DOUBLE PRECISION (A-H, O-Z)
       INCLUDE 'PARAM.INC'
       DOUBLE PRECISION KO,MFF,MUF,MMF,NSTATE(IORDER,IORDER)
       DIMENSION U(CCO),YM(CCO),YMOLD(CCO),DYM(CCO)
       EXTERNAL FSUB,DFSUB
C
C      DECLARE PARAMETERS
C      ------------------
C
```

```
      COMMON / SPACEN / HM,RUF,MUF,KO,VS,QM,RW,RS,RFF
C
      COMMON / PARAN / MFF,WFF,WSF,QC,C1,C2,RM,CS,ROS,WF
C
      DIMENSION X(1),DWORK(50),IWORK(50),F(1)
C
C     DECLARE SUB DNEWTON PARAMETERS
C     ------------------------------
C
      N = 1
      IPRINT = 0
      ITMAX = 50
      MAXDMP = 50
      ATOL = 1.0E-06
      METH = 1
      LDWORK = 50
      LIWORK = 50
C
      MFF = U(1)
      WSF = U(2)
C
      RM=NSTATE(1,1)
      CS=NSTATE(2,2)
      ROS=NSTATE(3,3)
C
      X(1) = MUF
      WFF = 0.4*MFF/0.6
C
C     CALL SUB NEWTON TO GET CORRECT X VALUE
C     --------------------------------------
C
      CALL DNEWTON(N,X,IERR,FSUB,DFSUB,IPRINT,ITMAX,MAXDMP,
     &             ATOL,METH,DWORK,LDWORK,IWORK,LIWORK)
C
      IF( IERR .NE. 0 ) THEN
        STOP
      END IF
C
      MUF = X(1)
C      WRITE(7,*) 'MUF',MUF
      NSTATE(1,1)=RM
      NSTATE(2,2)=CS
      NSTATE(3,3)=ROS
      U(1) = MFF
      U(2) = WSF
C
C     CALCULATE DERIVS & ALGEBRAIC EQUATIONS
C     --------------------------------------
C
      CALL INTEG(DELTAT,NSTATE,U,NUMIN,NUMOUT,TIME,YM)
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE MODEL(DELTAT,DSAMPL,STATE,U,NUMIN,NUMOUT,
     &                 ORDER,TIME,PEB,DNUMIN,DSTATE,DORDER)
C*****************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C
      INCLUDE 'PARAM.INC'
C
      DIMENSION U(CCO)
      DIMENSION YT(IORDER,IORDER,IORDER),
     &          DYT(IORDER,IORDER,IORDER),
```

```
     &           DYM(IORDER,IORDER,IORDER),
     &           DYDX(IORDER,IORDER,IORDER)
C
      DIMENSION YDT(IORDER,IORDER,IORDER),
     &           DYDT(IORDER,IORDER,IORDER),
     &           DYDM(IORDER,IORDER,IORDER),
     &           DYDDX(IORDER,IORDER,IORDER)
C
      DIMENSION STATE(IORDER,IORDER,IORDER),
     &           DSTATE(IORDER,IORDER,IORDER)
C
      INTEGER NUMIN,NUMOUT,ORDER,DORDER,DNUMIN
C
      COMMON / SPACE / ASTATE(IORDER,IORDER,IORDER,IORDER),
     &                 BSTATE(IORDER,IORDER,IORDER),
     &                 CSTATE(IORDER,IORDER,IORDER),
     &                 DASTATE(IORDER,IORDER,IORDER,IORDER),
     &                 DBSTATE(IORDER,IORDER,IORDER),
     &                 DCSTATE(IORDER,IORDER,IORDER)
C
      COMMON / DELAYS / TIMDEL(CCO,CCO),DTIMDEL(CCO,CCO)
      COMMON / TEMPOUT / Y(CCO,CCO,TIMEDELAY),YD(CCO,CCO,TIMEDELAY)
C
      HH=DELTAT*0.5
      H6=DELTAT/6.0
      XH=DELTAT+HH
C
C     PERFORMING THE RUNGE-KUTTA ON THE PROCESS STATES
C     ------------------------------------------------
C
      CALL DERIVS(DYDX,U,NUMIN,NUMOUT,ORDER,STATE)
C
      DO 10 I=1,NUMOUT
         DO 20 J=1,NUMIN
            DO 30 K=1,ORDER
               YT(I,J,K)=STATE(I,J,K)+HH*DYDX(I,J,K)
30          CONTINUE
20       CONTINUE
10    CONTINUE
C
      CALL DERIVS(DYT,U,NUMIN,NUMOUT,ORDER,YT)
C
      DO 40 I=1,NUMOUT
         DO 50 J=1,NUMIN
            DO 60 K=1,ORDER
               YT(I,J,K)=STATE(I,J,K)+HH*DYT(I,J,K)
60          CONTINUE
50       CONTINUE
40    CONTINUE
C
      CALL DERIVS(DYM,U,NUMIN,NUMOUT,ORDER,YT)
C
      DO 70 I=1,NUMOUT
         DO 80 J=1,NUMIN
            DO 90 K=1,ORDER
               YT(I,J,K)=STATE(I,J,K)+DELTAT*DYM(I,J,K)
               DYM(I,J,K)=DYM(I,J,K)+DYT(I,J,K)
90          CONTINUE
80       CONTINUE
70    CONTINUE
C
      CALL DERIVS(DYT,U,NUMIN,NUMOUT,ORDER,YT)
C
      DO 100 I=1,NUMOUT
```

```
          DO 110 J=1,NUMIN
             DO 120 K=1,ORDER
                STATE(I,J,K)=STATE(I,J,K)+H6*(DYDX(I,J,K)+
     &                       DYT(I,J,K)+2*DYM(I,J,K))
120          CONTINUE
110       CONTINUE
100    CONTINUE
C
C      IS THERE DISTURBANCE
C      --------------------
C
       IF(PEB .NE. 0.0) THEN
C
C      PERFORMING THE RUNGE-KUTTA ON THE DSTATES
C      -----------------------------------------
C
       CALL DDERIVS(DYDDX,PEB,DNUMIN,NUMOUT,DORDER,DSTATE)
C
       DO 130 I=1,NUMOUT
          DO 140 J=1,DNUMIN
             DO 150 K=1,DORDER
                YDT(I,J,K)=DSTATE(I,J,K)+HH*DYDDX(I,J,K)
150          CONTINUE
140       CONTINUE
130    CONTINUE
C
       CALL DDERIVS(DYDT,PEB,DNUMIN,NUMOUT,DORDER,YDT)
C
       DO 160 I=1,NUMOUT
          DO 170 J=1,DNUMIN
             DO 180 K=1,DORDER
                YDT(I,J,K)=DSTATE(I,J,K)+HH*DYDT(I,J,K)
180          CONTINUE
170       CONTINUE
160    CONTINUE
C
       CALL DDERIVS(DYDM,PEB,DNUMIN,NUMOUT,DORDER,YDT)
C
       DO 190 I=1,NUMOUT
          DO 200 J=1,DNUMIN
             DO 210 K=1,DORDER
                YDT(I,J,K)=DSTATE(I,J,K)+DELTAT*DYDM(I,J,K)
                DYDM(I,J,K)=DYDM(I,J,K)+DYDT(I,J,K)
210          CONTINUE
200       CONTINUE
190    CONTINUE
C
       CALL DDERIVS(DYDT,PEB,DNUMIN,NUMOUT,DORDER,YDT)
C
       DO 220 I=1,NUMOUT
          DO 230 J=1,DNUMIN
             DO 240 K=1,DORDER
                DSTATE(I,J,K)=DSTATE(I,J,K)+H6*(DYDDX(I,J,K)+
     &                       DYDT(I,J,K)+2*DYDM(I,J,K))
240          CONTINUE
230       CONTINUE
220    CONTINUE
C
       ENDIF
C
       TIME = TIME + DELTAT
C
       RETURN
       END
```

```
C
C*****************************************************************
      SUBROUTINE OUT(OUTPUT,STATE,NUMIN,NUMOUT,ORDER,DSAMPL,DELTAT,
     &               DOUTPUT,DSTATE,DNUMIN,DORDER,PEB)
C*****************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C
      INCLUDE 'PARAM.INC'
C
      DIMENSION OUTPUT(CCO),DOUTPUT(CCO)
      DIMENSION STATE(IORDER,IORDER,IORDER),
     &          DSTATE(IORDER,IORDER,IORDER)
C
      INTEGER NUMIN,NUMOUT,ORDER,DORDER,DNUMIN
C
      COMMON / SPACE / ASTATE(IORDER,IORDER,IORDER,IORDER),
     &                 BSTATE(IORDER,IORDER,IORDER),
     &                 CSTATE(IORDER,IORDER,IORDER),
     &                 DASTATE(IORDER,IORDER,IORDER,IORDER),
     &                 DBSTATE(IORDER,IORDER,IORDER),
     &                 DCSTATE(IORDER,IORDER,IORDER)
C
      COMMON / DELAYS / TIMDEL(CCO,CCO),DTIMDEL(CCO,CCO)
      COMMON / TEMPOUT / Y(CCO,CCO,TIMEDELAY),YD(CCO,CCO,TIMEDELAY)
C
C     ADJUSTING THE TIME DELAY MATRICES
C     ---------------------------------
C
         DO 130 I=1,NUMOUT
            DO 140 J=1,NUMIN
               IF (TIMDEL(I,J) .NE. 0.0) THEN
                  DO 150 K=1,(ABS(NINT(TIMDEL(I,J)/DSAMPL))-1)
                     Y(I,J,K) = Y(I,J,K+1)
150                  CONTINUE
C
C     CALCULATING THE FUTURE INDIVIDUAL OUTPUTS
C     -----------------------------------------
C
                  Y(I,J,ABS(NINT(TIMDEL(I,J)/DSAMPL))) = 0.0
                  DO 160 L=1,ORDER
                     Y(I,J,ABS(NINT(TIMDEL(I,J)/DSAMPL))) =
     &               Y(I,J,ABS(NINT(TIMDEL(I,J)/DSAMPL))) +
     &               CSTATE(I,J,L)*STATE(I,J,L)
160               CONTINUE
               ELSE
                  Y(I,J,1) = 0.0
                  DO 170 L = 1,ORDER
                     Y(I,J,1) = Y(I,J,1) + CSTATE(I,J,L)*STATE(I,J,L)
170                  CONTINUE
               END IF
140         CONTINUE
130      CONTINUE
C
C     SUMMING THE DELAYED INDIVIDUAL OUTPUTS
C     --------------------------------------
C
         DO 180 I=1,NUMOUT
            OUTPUT(I) = 0.0
            DO 190 J=1,NUMIN
               OUTPUT(I) = OUTPUT(I) + Y(I,J,1)
190         CONTINUE
180      CONTINUE
C
C     IS THERE DISTURBANCE
```

```
C     -------------------
C
      IF(PEB .NE. 0.0) THEN
C
C     ADJUSTING THE TIME DELAY MATRICES
C     ---------------------------------
C
         DO 200 I=1,NUMOUT
            DO 210 J=1,DNUMIN
               IF (DTIMDEL(I,J) .NE. 0.0) THEN
                  DO 220 K=1,(ABS(NINT(DTIMDEL(I,J)/DSAMPL))-1)
                     YD(I,J,K) = YD(I,J,K+1)
220               CONTINUE
C
C     CALCULATING THE FUTURE INDIVIDUAL OUTPUTS
C     -----------------------------------------
C
                  YD(I,J,ABS(NINT(DTIMDEL(I,J)/DSAMPL))) = 0.0
                  DO 230 L=1,DORDER
                     YD(I,J,ABS(NINT(DTIMDEL(I,J)/DSAMPL))) =
     &               YD(I,J,ABS(NINT(DTIMDEL(I,J)/DSAMPL))) +
     &               DCSTATE(I,J,L)*DSTATE(I,J,L)
230               CONTINUE
               ELSE
                  YD(I,J,1) = 0.0
                  DO 240 L = 1,DORDER
                     YD(I,J,1) = YD(I,J,1) + DCSTATE(I,J,L)*
     &                                        DSTATE(I,J,L)
240               CONTINUE
               END IF
210         CONTINUE
200      CONTINUE
C
C     SUMMING THE DELAYED INDIVIDUAL OUTPUTS
C     --------------------------------------
C
         DO 250 I=1,NUMOUT
            DOUTPUT(I) = 0.0
            DO 260 J=1,DNUMIN
               DOUTPUT(I) = DOUTPUT(I) + YD(I,J,1)
260         CONTINUE
250      CONTINUE
C
C        ADD DISTURBANCE TO THE PROCESS
C        ------------------------------
C
         DO 270 I = 1,NUMOUT
            OUTPUT(I) = OUTPUT(I) + DOUTPUT(I)
270      CONTINUE
C
         END IF
C
      RETURN
      END
C
C*********************************************************************
      SUBROUTINE MATRIX(R,N,NUMIN,NUMOUT,M,P,Q,DISTUR,DSAMPL,
     &                  DELTAT,YM,U,ORDER,YS,YSS,DU,DUS,TIME,
     &                  TSIM,DNUMIN,DORDER,USS)
C*********************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DOUBLE PRECISION LAM,R
C
      INCLUDE 'PARAM.INC'
```

```
C
      INTEGER COUNT
      INTEGER P
      DIMENSION YS(CCO),YSS(CCO),DISTUR(CCO),DY(CCO)
      DIMENSION Q(MAXIM),R(NMAXIM),DU(CCO),DUS(CCO),USS(CCO)
C
      COMMON / CONTROL / ERROR(MAXIM),YI(MAXIM),
     &                   A(MAXIM,NMAXIM),LAM(NMAXIM,MAXIM),
     &                   DUM(NMAXIM,NMAXIM),FF(MAXIM),ANS5
C
      DIMENSION YM(CCO),U(CCO)
      INTEGER ORDER
C
      INTEGER C
      INTEGER IPVT(NMAXIM)
      INTEGER JOB
      DIMENSION WORK(NMAXIM),DET(2)
C
      CHARACTER*1 ANS1,ANS2,ANS3,ANS5
      CHARACTER*10 NAME
C
C     FORMAT STATEMENTS
C     -----------------
C
5     FORMAT (10X,' ENTER WEIGHTING FACTOR FOR OUTPUT',I,':')
15    FORMAT (10X,' ENTER MOVE SUPPRESSION FACTOR FOR INPUT',I,':')
25    FORMAT (A1)
35    FORMAT (10X,' QP PROBLEM ? Y/N')
45    FORMAT (10X,' SAVE DUM/HESS ? Y/N')
55    FORMAT (10X,' NOTE Q & R MUST BE SAME FOR NEXT RUNS.')
65    FORMAT (A10)
75    FORMAT (10X,' READ DUM/HESS FROM FILE ? Y/N')
85    FORMAT (10X,' WHAT FILE NAME ?')
95    FORMAT (10(/))
C
C     SCALE OUTPUTS OR THE A MATRIX IF REQUIRED
C     -----------------------------------------
C
      DO 210 I = 1, NUMIN
        IF(DUS(I) .NE. 1.0) THEN
          DO 220 K = 1,NUMOUT
            DO 230 J = 1,M
              A((K-1)*P+J,(I-1)*N+1) = (A((K-1)*P+J,(I-1)*N+1)
     &                                  - YSS(K))/DUS(I)
230         CONTINUE
220       CONTINUE
        END IF
210   CONTINUE
C
C     SET UP DYNAMIC MATRIX
C     ---------------------
C
      IF( P .GT. M) THEN
        DO 10 I = 1,NUMIN
          DO 20 J = 1,NUMOUT
            DO 30 K = M+1,P
              A(K+(J-1)*P,1+(I-1)*N) = A(M+(J-1)*P,1+(I-1)*N)
30          CONTINUE
20        CONTINUE
10      CONTINUE
      END IF
C
      IF(N .GT. 1) THEN
        DO 40 I = 0 , NUMIN-1
```

```
          DO 50 J = 2 , N
            DO 60 K = 0 , NUMOUT-1
              DO 70 L = 2 , P
                A(K*P+L,I*N+J) = A(K*P+L-1,I*N+J-1)
70            CONTINUE
60          CONTINUE
50        CONTINUE
40      CONTINUE
        END IF
C
        WRITE(*,95)
        WRITE(*,75)
        READ(*,25) ANS1
        IF(ANS1 .EQ. 'Y' .OR. ANS1 .EQ. 'y') THEN
          WRITE(*,85)
          READ(*,65) NAME
          OPEN (UNIT=13, FILE = NAME//'.DMC',STATUS='UNKNOWN')
          DO 80 I = 1 , NUMIN*N
            DO 90 J = 1 , NUMIN*N
              READ(13,*) DUM(I,J)
90          CONTINUE
80        CONTINUE
          CLOSE(UNIT=13)
          GOTO 1000
        END IF
C

C       READ IN WEIGHTS AND MOVE SUPPRESSION FACTOR
C       -------------------------------------------
C
        DO 100 I = 1 , NUMOUT
          WRITE(*,5) I
          READ(*,*) DUMMY
          DO 110 J = 1 , P
            Q((I-1)*P+J) = DUMMY
110       CONTINUE
100     CONTINUE
C
        DO 120 I = 1 , NUMIN
          WRITE(*,15) I
          READ(*,*) DUMMY
          DO 130 J = 1 , N
            R((I-1)*N+J) = DUMMY
130       CONTINUE
120     CONTINUE
C
C       CALCULATE ATRANS*Q*A+R
C       ----------------------
C
        DO 140 I = 1 , NUMIN*N
          DO 150 J = 1 , NUMIN*N
            SUM = 0.0
            DO 150 K = 1 , NUMOUT*P
              SUM = SUM + A(K,I)*Q(K)*A(K,J)
160         CONTINUE
            DUM(I,J) = SUM
150       CONTINUE
          DUM(I,I) = DUM(I,I) + R(I)
140     CONTINUE
C
C       DO YOU WANT TO SAVE DUM OR HESS, NOTE Q & R MUST NOT CHANGE
C       -----------------------------------------------------------
C
        WRITE(*,45)
```

```
      WRITE(*,55)
      READ(*,25) ANS2
      IF(ANS2 .EQ. 'Y' .OR. ANS2 .EQ. 'y') THEN
        WRITE(*,85)
        READ(*,65) NAME
        OPEN (UNIT=12, FILE = NAME//'.DMC',STATUS='UNKNOWN')
        DO 170 I = 1 , NUMIN*N
          DO 180 J = 1 , NUMIN*N
            WRITE(12,*) DUM(I,J)
180       CONTINUE
170     CONTINUE
        CLOSE(UNIT=12)
      END IF
C
C     QP OR LP PROBLEM
C     ----------------
C
1000  WRITE(*,95)
      WRITE(*,35)
      READ(*,25) ANS3
      IF(ANS3 .EQ. 'Y' .OR. ANS3 .EQ. 'y') THEN
        CALL QSIMUL(DISTUR,DSAMPL,DELTAT,YM,U,N,NUMIN,
     &              NUMOUT,ORDER,P,YS,YSS,DU,DUS,TIME,TSIM,
     &              DNUMIN,DORDER,Q,USS)
        STOP
      END IF
C
C     CALCULATE INVERSE OF (ATRANS*Q*A+R)
C     -----------------------------------
C
      JOB = 1
      C = NUMIN * N
C
      CALL SGEFA(DUM,NMAXIM,C,IPVT,INFO)
      IF(INFO .NE. 0) THEN
         WRITE(*,*) 'INFO=',INFO
         STOP
      END IF
C
      CALL SGEDI(DUM,NMAXIM,C,IPVT,DET,WORK,JOB)
C
C     CALCULATE INVERSE OF (ATRANS*Q*A+R)*ATRANS*Q
C     --------------------------------------------
C
      DO 190 I = 1 , N*NUMIN
        DO 200 K = 1 , NUMOUT*P
          SUM = 0.0
          DO 210 J = 1 , NUMIN*N
            SUM = SUM + DUM(I,J)*A(K,J)*Q(K)
210       CONTINUE
          LAM(I,K) = SUM
200     CONTINUE
190   CONTINUE
C
      RETURN
      END
C
C*****************************************************************
      SUBROUTINE SGEFA(A,LDA,N,IPVT,INFO)
C*****************************************************************
      INTEGER LDA,N,IPVT(1),INFO
      DOUBLE PRECISION A(LDA,1)
C
C     SGEFA FACTORS A REAL MATRIX BY GAUSSIAN ELIMINATION.
```

```
C
C     SGEFA IS USUALLY CALLED BY SGECO, BUT IT CAN BE CALLED
C     DIRECTLY WITH A SAVING IN TIME IF  RCOND  IS NOT NEEDED.
C     (TIME FOR SGECO) = (1 + 9/N)*(TIME FOR SGEFA) .
C
C     ON ENTRY
C
C        A        REAL(LDA, N)
C                 THE MATRIX TO BE FACTORED.
C
C        LDA      INTEGER
C                 THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N        INTEGER
C                 THE ORDER OF THE MATRIX  A .
C
C     ON RETURN
C
C        A        AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
C                 WHICH WERE USED TO OBTAIN IT.
C                 THE FACTORIZATION CAN BE WRITTEN  A = L*U  WHERE
C                 L  IS A PRODUCT OF PERMUTATION AND UNIT LOWER
C                 TRIANGULAR MATRICES AND  U  IS UPPER TRIANGULAR.
C
C        IPVT     INTEGER(N)
C                 AN INTEGER VECTOR OF PIVOT INDICES.
C
C        INFO     INTEGER
C                 = 0  NORMAL VALUE.
C                 = K  IF  U(K,K) .EQ. 0.0 .  THIS IS NOT AN ERROR
C                      CONDITION FOR THIS SUBROUTINE, BUT IT DOES
C                      INDICATE THAT SGESL OR SGEDI WILL DIVIDE BY ZERO
C                      IF CALLED.  USE  RCOND  IN SGECO FOR A RELIABLE
C                      INDICATION OF SINGULARITY.
C
C     LINPACK. THIS VERSION DATED 08/14/78 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LAB.
C
C     SUBROUTINES AND FUNCTIONS
C
C     BLAS SAXPY,SSCAL,ISAMAX
C
C     INTERNAL VARIABLES
C
      DOUBLE PRECISION T
      INTEGER ISAMAX,J,K,KP1,L,NM1
C
C
C     GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
C
      INFO = 0
      NM1 = N - 1
      IF (NM1 .LT. 1) GO TO 70
      DO 60 K = 1, NM1
         KP1 = K + 1
C
C        FIND L = PIVOT INDEX
C
         L = ISAMAX(N-K+1,A(K,K),1) + K - 1
         IPVT(K) = L
C
C        ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED
C
         IF (A(L,K) .EQ. 0.0E0) GO TO 40
```

```
C
C           INTERCHANGE IF NECESSARY
C
            IF (L .EQ. K) GO TO 10
               T = A(L,K)
               A(L,K) = A(K,K)
               A(K,K) = T
   10       CONTINUE
C
C           COMPUTE MULTIPLIERS
C
            T = -1.0E0/A(K,K)
            CALL SSCAL(N-K,T,A(K+1,K),1)
C
C           ROW ELIMINATION WITH COLUMN INDEXING
C
            DO 30 J = KP1, N
               T = A(L,J)
               IF (L .EQ. K) GO TO 20
                  A(L,J) = A(K,J)
                  A(K,J) = T
   20          CONTINUE
               CALL SAXPY(N-K,T,A(K+1,K),1,A(K+1,J),1)
   30       CONTINUE
         GO TO 50
   40    CONTINUE
            INFO = K
   50    CONTINUE
   60 CONTINUE
   70 CONTINUE
      IPVT(N) = N
      IF (A(N,N) .EQ. 0.0E0) INFO = N
      RETURN
      END
C
C*****************************************************************
      SUBROUTINE SGEDI(A,LDA,N,IPVT,DET,WORK,JOB)
C*****************************************************************
      INTEGER LDA,N,IPVT(1),JOB
      DOUBLE PRECISION A(LDA,1),DET(2),WORK(1)
C
C     SGEDI COMPUTES THE DETERMINANT AND INVERSE OF A MATRIX
C     USING THE FACTORS COMPUTED BY SGECO OR SGEFA.
C
C     ON ENTRY
C
C        A       REAL(LDA, N)
C                THE OUTPUT FROM SGECO OR SGEFA.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A .
C
C        IPVT    INTEGER(N)
C                THE PIVOT VECTOR FROM SGECO OR SGEFA.
C
C        WORK    REAL(N)
C                WORK VECTOR.  CONTENTS DESTROYED.
C
C        JOB     INTEGER
C                = 11   BOTH DETERMINANT AND INVERSE.
C                = 01   INVERSE ONLY.
```

```
C                 = 10    DETERMINANT ONLY.
C
C     ON RETURN
C
C        A        INVERSE OF ORIGINAL MATRIX IF REQUESTED.
C                 OTHERWISE UNCHANGED.
C
C        DET      REAL(2)
C                 DETERMINANT OF ORIGINAL MATRIX IF REQUESTED.
C                 OTHERWISE NOT REFERENCED.
C                 DETERMINANT = DET(1) * 10.0**DET(2)
C                 WITH  1.0 .LE. ABS(DET(1)) .LT. 10.0
C                 OR  DET(1) .EQ. 0.0 .
C
C     ERROR CONDITION
C
C        A DIVISION BY ZERO WILL OCCUR IF THE INPUT FACTOR CONTAINS
C        A ZERO ON THE DIAGONAL AND THE INVERSE IS REQUESTED.
C        IT WILL NOT OCCUR IF THE SUBROUTINES ARE CALLED CORRECTLY
C        AND IF SGECO HAS SET RCOND .GT. 0.0 OR SGEFA HAS SET
C        INFO .EQ. 0 .
C
C     LINPACK. THIS VERSION DATED 08/14/78 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LAB.
C
C     SUBROUTINES AND FUNCTIONS
C
C     BLAS SAXPY,SSCAL,SSWAP
C     FORTRAN ABS,MOD
C
C     INTERNAL VARIABLES
C
      DOUBLE PRECISION T
      DOUBLE PRECISION TEN
      INTEGER I,J,K,KB,KP1,L,NM1
C
C
C     COMPUTE DETERMINANT
C
      IF (JOB/10 .EQ. 0) GO TO 70
         DET(1) = 1.0E0
         DET(2) = 0.0E0
         TEN = 10.0E0
         DO 50 I = 1, N
            IF (IPVT(I) .NE. I) DET(1) = -DET(1)
            DET(1) = A(I,I)*DET(1)
C           ...EXIT
            IF (DET(1) .EQ. 0.0E0) GO TO 60
   10       IF (ABS(DET(1)) .GE. 1.0E0) GO TO 20
               DET(1) = TEN*DET(1)
               DET(2) = DET(2) - 1.0E0
            GO TO 10
   20       CONTINUE
   30       IF (ABS(DET(1)) .LT. TEN) GO TO 40
               DET(1) = DET(1)/TEN
               DET(2) = DET(2) + 1.0E0
            GO TO 30
   40       CONTINUE
   50    CONTINUE
   60    CONTINUE
   70 CONTINUE
C
C     COMPUTE INVERSE(U)
C
```

```
      IF (MOD(JOB,10) .EQ. 0) GO TO 150
         DO 100 K = 1, N
            A(K,K) = 1.0E0/A(K,K)
            T = -A(K,K)
            CALL SSCAL(K-1,T,A(1,K),1)
            KP1 = K + 1
            IF (N .LT. KP1) GO TO 90
            DO 80 J = KP1, N
               T = A(K,J)
               A(K,J) = 0.0E0
               CALL SAXPY(K,T,A(1,K),1,A(1,J),1)
   80       CONTINUE
   90       CONTINUE
  100    CONTINUE
C
C        FORM INVERSE(U)*INVERSE(L)
C
         NM1 = N - 1
         IF (NM1 .LT. 1) GO TO 140
         DO 130 KB = 1, NM1
            K = N - KB
            KP1 = K + 1
            DO 110 I = KP1, N
               WORK(I) = A(I,K)
               A(I,K) = 0.0E0
  110       CONTINUE
            DO 120 J = KP1, N
               T = WORK(J)
               CALL SAXPY(N,T,A(1,J),1,A(1,K),1)
  120       CONTINUE
            L = IPVT(K)
            IF (L .NE. K) CALL SSWAP(N,A(1,K),1,A(1,L),1)
  130    CONTINUE
  140    CONTINUE
  150 CONTINUE
      RETURN
      END
C
C*====================================================================*
      SUBROUTINE SAXPY(N,SA,SX,INCX,SY,INCY)
C*====================================================================*

C
C     CONSTANT TIMES A VECTOR PLUS A VECTOR.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION SX(1),SY(1),SA
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      IF(N.LE.0)RETURN
      IF (SA .EQ. 0.0E0) RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        SY(IY) = SY(IY) + SA*SX(IX)
        IX = IX + INCX
```

```
          IY = IY + INCY
       10 CONTINUE
          RETURN
C
C           CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C           CLEAN-UP LOOP
C
       20 M = MOD(N,4)
          IF( M .EQ. 0 ) GO TO 40
          DO 30 I = 1,M
            SY(I) = SY(I) + SA*SX(I)
       30 CONTINUE
          IF( N .LT. 4 ) RETURN
       40 MP1 = M + 1
          DO 50 I = MP1,N,4
            SY(I) = SY(I) + SA*SX(I)
            SY(I + 1) = SY(I + 1) + SA*SX(I + 1)
            SY(I + 2) = SY(I + 2) + SA*SX(I + 2)
            SY(I + 3) = SY(I + 3) + SA*SX(I + 3)
       50 CONTINUE
          RETURN
          END
C
C*********************************************************************
          SUBROUTINE  SSCAL(N,SA,SX,INCX)
C*********************************************************************
C
C     SCALES A VECTOR BY A CONSTANT.
C     USES UNROLLED LOOPS FOR INCREMENT EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
          DOUBLE PRECISION SA,SX(1)
          INTEGER I,INCX,M,MP1,N,NINCX
C
          IF(N.LE.0)RETURN
          IF(INCX.EQ.1)GO TO 20
C
C           CODE FOR INCREMENT NOT EQUAL TO 1
C
          NINCX = N*INCX
          DO 10 I = 1,NINCX,INCX
            SX(I) = SA*SX(I)
       10 CONTINUE
          RETURN
C
C           CODE FOR INCREMENT EQUAL TO 1
C
C
C           CLEAN-UP LOOP
C
       20 M = MOD(N,5)
          IF( M .EQ. 0 ) GO TO 40
          DO 30 I = 1,M
            SX(I) = SA*SX(I)
       30 CONTINUE
          IF( N .LT. 5 ) RETURN
       40 MP1 = M + 1
          DO 50 I = MP1,N,5
            SX(I) = SA*SX(I)
            SX(I + 1) = SA*SX(I + 1)
            SX(I + 2) = SA*SX(I + 2)
            SX(I + 3) = SA*SX(I + 3)
```

```
        SX(I + 4) = SA*SX(I + 4)
   50 CONTINUE
      RETURN
      END
C
C*************************************************************************
      SUBROUTINE SSWAP (N,SX,INCX,SY,INCY)
C*************************************************************************
C
C     INTERCHANGES TWO VECTORS.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO 1.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION SX(1),SY(1),STEMP
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C       CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS NOT EQUAL
C         TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        STEMP = SX(IX)
        SX(IX) = SY(IY)
        SY(IY) = STEMP
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C       CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C       CLEAN-UP LOOP
C
   20 M = MOD(N,3)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        STEMP = SX(I)
        SX(I) = SY(I)
        SY(I) = STEMP
   30 CONTINUE
      IF( N .LT. 3 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,3
        STEMP = SX(I)
        SX(I) = SY(I)
        SY(I) = STEMP
        STEMP = SX(I + 1)
        SX(I + 1) = SY(I + 1)
        SY(I + 1) = STEMP
        STEMP = SX(I + 2)
        SX(I + 2) = SY(I + 2)
        SY(I + 2) = STEMP
   50 CONTINUE
      RETURN
      END
C
C*************************************************************************
```

```
      INTEGER FUNCTION ISAMAX(N,SX,INCX)
C*****************************************************************
C
C     FINDS THE INDEX OF ELEMENT HAVING MAX. ABSOLUTE VALUE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION SX(1),SMAX
      INTEGER I,INCX,IX,N
C
      ISAMAX = 0
      IF( N .LT. 1 ) RETURN
      ISAMAX = 1
      IF(N.EQ.1)RETURN
      IF(INCX.EQ.1)GO TO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      IX = 1
      SMAX = ABS(SX(1))
      IX = IX + INCX
      DO 10 I = 2,N
         IF(ABS(SX(IX)).LE.SMAX) GO TO 5
         ISAMAX = I
         SMAX = ABS(SX(IX))
    5    IX = IX + INCX
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
   20 SMAX = ABS(SX(1))
      DO 30 I = 2,N
         IF(ABS(SX(I)).LE.SMAX) GO TO 30
         ISAMAX = I
         SMAX = ABS(SX(I))
   30 CONTINUE
      RETURN
      END
C
C*********************************************************************
      SUBROUTINE SIMUL(DISTUR,DSAMPL,DELTAT,YM,U,N,NUMIN,
     &                 NUMOUT,ORDER,P,YS,YSS,DU,DUS,TIME,TSIM,
     &                 DNUMIN,DORDER,USS)
C*********************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DOUBLE PRECISIQN LAM,NSTATE,HM,KO,MUF
C
      INCLUDE 'PARAM.INC'
C
      DIMENSION YM(CCO),U(CCO),USS(CCO)
      INTEGER ORDER,DORDER,DNUMIN
      INTEGER P
      DIMENSION YS(CCO),YSS(CCO),DISTUR(CCO),DY(CCO),DOUTPUT(CCO)
      DIMENSION Q(MAXIM),R(NMAXIM),DU(CCO),DUS(CCO)
C
      COMMON / CONTROL / ERROR(MAXIM),YI(MAXIM),
     &                   A(MAXIM,NMAXIM),LAM(NMAXIM,MAXIM),
     &                   DUM(NMAXIM,NMAXIM),FF(MAXIM),ANS5
      COMMON / TEMPOUT / Y(CCO,CCO,TIMEDELAY),YD(CCO,CCO,TIMEDELAY)
      COMMON / SPACEN / HM,RUF,MUF,KO,VS,QM,RW,RS,RFF
      COMMON / STEADYN / SSTATE(IORDER,IORDER)
C
      CHARACTER*1 ANS5
      CHARACTER*10,NAME
```

```
      DIMENSION STATE(IORDER,IORDER,IORDER),
     &          DSTATE(IORDER,IORDER,IORDER),
     &          NSTATE(IORDER,IORDER)
C
      INTEGER NC,NCOUNT
      DIMENSION DYS(NUM),DTYS(NUM)
C
C     FORMAT STATEMENTS
C     -----------------
C
5     FORMAT (10X,'INPUT THE SET POINT CHANGE FOR OUTPUT ',I3,':')
15    FORMAT (10X,'UNDER WHAT FILENAME MUST THE DATA BE STORED :')
25    FORMAT (A10)
35    FORMAT (10X,'ENTER TIME FOR SETPOINT CHANGE OF OUTPUT',I3,':')
45    FORMAT (2X,F6.2,1X,F15.12,1X,F15.12,1X,F15.12)
C     &      ,1X,F15.12,1X,F15.12,1X,F15.12)
55    FORMAT (10X,'INPUT THE DISTURBANCE CHANGE')
65    FORMAT (10X,'INPUT THE TIME OF DISTURBANCE CHANGE')
75    FORMAT (10X,'SETPOINT CHANGE TOO LATE,/10X,
     &            SIMULATION TIME=')
215   FORMAT (23(/))
225   FORMAT (27X,26('*'))
235   FORMAT (27X,'Computation in progress !')
245   FORMAT (27X,26('*'))
255   FORMAT (11(/))
C
C     INITIALISE VARIABLES
C     --------------------
C
      DO 10 I = 1 , NUMOUT
        YM(I) = 0.0
        DO 20 J = 1 , P
          YI((I-1)*P+J) = 0.0
20      CONTINUE
10    CONTINUE
C
      PEB = 0.0
      DPEB = 0.0
      TDIST = 0.0
      TIME = 0.0
      COUNT = 0
      NC = NINT(DSAMPL/DELTAT)
C
      DO 30 I = 1,NUMOUT
        YS(I) = YSS(I)
        DYS(I) = 0.0
        DTYS(I) = 0.0
30    CONTINUE
C
      DO 40 I = 1,NUMOUT+1
        DO 50 J = 1,NUMIN+1
          DO 60 K = 1,IORDER
            STATE(I,J,K)=0.0
            DSTATE(I,J,K)=0.0
60        CONTINUE
          DO 70 L = 1,TIMEDELAY
            Y(I,J,L) = 0.0
            YD(I,J,L) = 0.0
70        CONTINUE
C
C         IF(ANS5 .EQ. 'N' .OR. ANS5 .EQ. 'n') THEN
          U(J) = USS(J)
          NSTATE(J,I) = SSTATE(J,I)
50      CONTINUE
```

```
40    CONTINUE
C
C     READ IN VARIABLES
C     -----------------
C
      DO 80 I = 1 , NUMOUT
90      WRITE(*,5) I
        READ(*,*) DYS(I)
        WRITE(*,35) I
        READ(*,*) DTYS(I)
        IF( DTYS(I) .GE. TSIM) THEN
          WRITE(*,75) TSIM
          GOTO 90
        END IF
80    CONTINUE
C
      WRITE(*,55)
      READ(*,*) DPEB
      WRITE(*,65)
      READ(*,*) TDIST
C
C     READ IN NAME OF OUTPUT FILE
C     ---------------------------
C
      WRITE(*,15)
      READ(*,25) NAME
      OPEN (UNIT = 12 , FILE = NAME//'.PLT',STATUS='UNKNOWN')
      OPEN (UNIT=13, NAME='DIST.PLT',STATUS='UNKNOWN')
      OPEN (UNIT=14, NAME='U.PLT',STATUS='UNKNOWN')
      OPEN (UNIT=15, NAME='DU.PLT',STATUS='UNKNOWN')
C
C     START NEW PAGE
C     --------------
C
      WRITE(*,215)
      WRITE(*,225)
      WRITE(*,235)
      WRITE(*,245)
      WRITE(*,255)
C
C      WRITE(12,45) TIME,YM(1),YM(2),YM(3)
C      WRITE(14,45) TIME,U(1),U(2),U(3)
C
100    IF ( TIME .LE. TSIM ) THEN
C
C     RESET SETPOINT AND DISTURBANCE
C     ------------------------------
C
        DO 110 I = 1,NUMOUT
          IF(TIME .GE. DTYS(I)) THEN
            YS(I) = YS(I) + DYS(I)
            DYS(I) = 0.0
          END IF
110     CONTINUE
C
        IF( TIME .GE. TDIST) THEN
          PEB = PEB + DPEB
          DPEB = 0.0
        END IF
C
        IF(ANS5 .EQ. 'N' .OR. ANS5 .EQ. 'n') THEN
          CALL NONMODEL(DELTAT,NSTATE,U,NUMIN,NUMOUT,
     &                  TIME,YM)
        ELSE
```

```
      CALL MODEL(DELTAT,DSAMPL,STATE,U,NUMIN,NUMOUT,ORDER,
     &           TIME,PEB,DNUMIN,DSTATE,DORDER)
      END IF
       NCOUNT = NCOUNT + 1
       COUNT = COUNT + 1
C
       IF ( COUNT*DELTAT .GE. DSAMPL ) THEN
C
         IF(ANS5 .EQ. 'N' .OR. ANS5 .EQ. 'n') THEN
C
         ELSE
          CALL OUT(YM,STATE,NUMIN,NUMOUT,ORDER,DSAMPL,DELTAT,
     &             DOUTPUT,DSTATE,DNUMIN,DORDER,PEB)
         END IF
          CALL UNCONTR(DISTUR,YM,U,N,NUMIN,NUMOUT,P,YS,DU,DUS,
     &                 DOUTPUT,PEB,TIME)
          COUNT = 0
       END IF
C
       IF ( NCOUNT .GE. NC ) THEN
           WRITE(12,*) (TIME-DSAMPL),YM(1),YM(2)
           WRITE(14,*) (TIME-DSAMPL),U(1),U(2)
           WRITE(15,*) (TIME-DSAMPL),DU(1),DU(2)
           WRITE(13,45) (TIME-DSAMPL),DOUTPUT(1),DOUTPUT(2),
     &                                DOUTPUT(3)
           NCOUNT = 0
       END IF
C
       GOTO 100
      END IF
C
      CLOSE (15)
      CLOSE (12)
      CLOSE (13)
      CLOSE (14)
C
      RETURN
      END
C
C*************************************************************
      SUBROUTINE UNCONTR(DISTUR,YM,U,N,NUMIN,NUMOUT,P,
     &                   YS,DU,DUS,DOUTPUT,PEB,TIME)
C*************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DOUBLE PRECISION LAM
C
      INCLUDE 'PARAM.INC'
C
      INTEGER N
      INTEGER P
      DIMENSION YS(CCO),DISTUR(CCO),DY(CCO),DOUTPUT(CCO)
      DIMENSION Q(MAXIM),DU(CCO),DUS(CCO)
C
      COMMON / CONTROL / ERROR(MAXIM),YI(MAXIM),
     &                   A(MAXIM,NMAXIM),LAM(NMAXIM,MAXIM),
     &                   DUM(NMAXIM,NMAXIM),FF(MAXIM),ANS5
      COMMON / PREDICTIONS / PRED
C
      DIMENSION YM(CCO),U(CCO)
      INTEGER ORDER
      CHARACTER*1 ANS5,PRED
C
C     FEED FOWARD CONTROL
C     -------------------
```

```
C
      TIMER = TIME + 1
      DO 10 I = 1,NUMOUT
        IF(TIMER .GT. P) THEN
          TIMER = P
        END IF
          YM(I) = YM(I) - FF((I-1)*P+TIMER)*PEB
10    CONTINUE
C
C     STEP INPUT/OUTPUT DISTURBANCE ESTIMATE
C     --------------------------------------
C
      DO 20 I = 1 , NUMOUT
         C = 0.0
         DO 30 J = 1 , NUMIN
            C = C + A((I-1)*P+1,N*(J-1)+1) * DU(J)
30       CONTINUE
         YI((I-1)*P+1) = YI((I-1)*P+1) + C
         DISTUR(I) = YM(I) - YI((I-1)*P+1)
20    CONTINUE
C
C     CALC B VECTOR
C     -------------
C
      DO 40 I = 1 , NUMOUT
         DO 50 J = 1 , P-1
            B = 0.0
            DO 60 K = 1 , NUMIN
               B = B + A((I-1)*P+J+1,(K-1)*N+1)*DU(K)
60          CONTINUE
            YI((I-1)*P+J) = YI((I-1)*P+J+1) + B
50       CONTINUE
C
         DO 70 K = 1 , NUMIN
            YI(I*P) = YI(I*P) + A(I*P,(K-1)*N+1)
     &                      * DU(K)
70       CONTINUE
C
40    CONTINUE
C
C     CALC ERROR VECTOR
C     -----------------
C
      DO 80 I = 1 , NUMOUT
         DO 90 J = 1 , P
            ERROR((I-1)*P+J) = YS(I) - YI((I-1)*P+J)
     &                               - DISTUR(I)
90       CONTINUE
80    CONTINUE
C
C     CALC NEW INPUT CHANGES
C     ----------------------
C
      DO 100 I = 1 , NUMIN
         DU(I) = 0
         DO 110 J = 1 , NUMOUT*P
            DU(I) = DU(I) + LAM((I-1)*N+1,J)*ERROR(J)
110      CONTINUE
100   CONTINUE
C
C     CALC FIRST INPUT
C     ----------------
C
      DO 130 I = 1 , NUMIN
```

```
         U(I) = U(I) + DU(I)
130    CONTINUE
C
C      CALCULATE PREDICTIONS IF REQUIRED
C      ---------------------------------
C
       IF(PRED .EQ. 'Y' .OR. PRED .EQ. 'y') THEN
         IDENT = 0
         CALL PREDIC(YS,P,M,N,NUMIN,NUMOUT,TIME,IDENT)
       END IF
C
       RETURN
       END
C
C*****************************************************************
       SUBROUTINE PREDIC(YS,P,M,N,NUMIN,NUMOUT,TIME,IDENT)
C*****************************************************************
       IMPLICIT DOUBLE PRECISION (A-H, O-Z)
       DOUBLE PRECISION LAM
C
       INCLUDE 'PARAM.INC'
C
       COMMON / CONTROL / ERROR(MAXIM),YI(MAXIM),
     &                    A(MAXIM,NMAXIM),LAM(NMAXIM,MAXIM),
     &                    HESS(NMAXIM,NMAXIM),FF(MAXIM),ANS5
       COMMON / QPRED / X(NMAXIM)
C
       DIMENSION Y(MAXIM),YS(CCO),DU(NMAXIM),SUM(CCO)
       INTEGER P
C
       IF (IDENT EQ. 0) THEN
C
C        CALCULATE ALL DU'S
C        ------------------
C
         DO 10 I = 1 , NUMIN*N
           DU(I) = 0
           DO 20 J = 1 , NUMOUT*P
             DU(I) = DU(I) + LAM(I,J)*ERROR(J)
20         CONTINUE
10       CONTINUE
C
       ELSE IF (IDENT .EQ. 1) THEN
         DO 25 J = 1, NUMIN*N
           DU(J) = X(J)
25       CONTINUE
       END IF
C
C      CALCULATE PREDICTED Y'S
C      -----------------------
C
       DO 30 I = 1, NUMOUT*P
         Y(I) = YI(I)
         DO 40 K = 1, NUMIN*N
           Y(I) = Y(I) + A(I,K)*DU(K)
40       CONTINUE
30     CONTINUE
C
C      CALCULATE ERROR L2-NORM
C      -----------------------
C
       DO 50 I = 1, NUMOUT
         SUM(I) = 0.0
         DO 60 J = 1,P
```

```
          SUM(I) = SUM(I) + (YS(I) - Y((I-1)*P+J))**2
60      CONTINUE
50    CONTINUE
C
C     PRINT PREDICTIONS TO A FILE
C     ---------------------------
C
      OPEN(UNIT=6, FILE='PRED.PLT', STATUS='UNKNOWN')
        DO 70 I = 1, NUMOUT
          DO 80 J = 1,P
            WRITE(6,*) TIME*J,Y((I-1)*P+J),YS(I)
80        CONTINUE
70      CONTINUE
      CLOSE(6)
C
      RETURN
      END
C
C*********************************************************************
      SUBROUTINE QSIMUL(DISTUR,DSAMPL,DELTAT,YM,U,N,NUMIN,
     &                  NUMOUT,ORDER,P,YS,YSS,DU,DUS,TIME,TSIM,
     &                  DNUMIN,DORDER,Q,USS)
C*********************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DOUBLE PRECISION LAM,NSTATE,HM,KO,MUF
C
      INCLUDE 'PARAM.INC'
C
      LOGICAL COLD,LP,MINSUM
C
      DIMENSION YM(CCO),U(CCO),USS(CCO)
      INTEGER ORDER,DORDER,DNUMIN
      INTEGER P
      DIMENSION YS(CCO),YSS(CCO),DISTUR(CCO),DY(CCO),DOUTPUT(CCO)
      DIMENSION DU(CCO),DUS(CCO),Q(MAXIM)
C
      COMMON / CONTROL / ERROR(MAXIM),YI(MAXIM),
     &                   A(MAXIM,NMAXIM),LAM(NMAXIM,MAXIM),
     &                   HESS(NMAXIM,NMAXIM),FF(MAXIM),ANS5
      COMMON / TEMPOUT / Y(CCO,CCO,TIMEDELAY),YD(CCO,CCO,TIMEDELAY)
      COMMON / QP / LWORK,LIWORK,COLD,LP,MINSUM,NROWA,NROWH,NCOLH,
     &              NCLIN,NCOUT,ITMAX,MSGLVL,NCTOTL,BIGBND,OBJ,
     &              FEATOL,ANS1(CCO),ANS2(CCO),ANS3(CCO),
     &              NN,X(NMAXIM)
      COMMON / BOUNDS / DUMIN(CCO),DUMAX(CCO),UMIN(CCO),UMAX(CCO),
     &                  YMIN(CCO),YMAX(CCO),
     &                  DELAY(CCO),AA(NMAXIM+MAXIM,NMAXIM+MAXIM)
      COMMON / DELAYS / TIMDEL(CCO,CCO), DTIMDEL(CCO,CCO)
      COMMON / SPACEN / HM,RUF,MUF,KO,VS,QM,RW,RS,RFF
      COMMON / STEADYN / SSTATE(IORDER,IORDER)
C
      CHARACTER*1 ANS1,ANS2,ANS3,ANS5
      CHARACTER*10,NAME
C
      DIMENSION STATE(IORDER,IORDER,IORDER),
     &          DSTATE(IORDER,IORDER,IORDER),
     &          NSTATE(IORDER,IORDER)
C
      INTEGER NC,NCOUNT
      DIMENSION DYS(NUM),DTYS(NUM)
C
C     FORMAT STATEMENTS
C     -----------------
C
```

```
5     FORMAT (10X,'INPUT THE SET POINT CHANGE FOR OUTPUT ',I3,':')
15    FORMAT (10X,'UNDER WHAT FILENAME MUST THE DATA BE STORED :')
25    FORMAT (A10)
35    FORMAT (10X,'ENTER TIME FOR SETPOINT CHANGE OF OUTPUT',I3,':')
45    FORMAT (2X,F6.2,1X,F15.12,1X,F15.12,1X,F15.12)
55    FORMAT (10X,'INPUT THE DISTURBANCE CHANGE')
65    FORMAT (10X,'INPUT THE TIME OF DISTURBANCE CHANGE')
75    FORMAT (A1)
85    FORMAT (10X,'ENTER DUmin(',I3,') AND DUmax(',I3,')')
95    FORMAT (10X,'ENTER Umin(',I3,') AND Umax(',I3,')')
105   FORMAT (10X,'ENTER Ymin(',I3,') AND Ymax(',I3,')')
115   FORMAT (10X,'SETPOINT CHANGE TOO LATE,/10X,
     &            SIMULATION TIME=')
125   FORMAT (10X,'IS THERE INPUT MOVE CONSTRAINT   ',I3,'  Y/N ?')
135   FORMAT (10X,'IS THERE ABSOLUTE INPUT CONSTRAINT ',I3,' Y/N ?')
145   FORMAT (10X,'IS THERE OUTPUT CONSTRAINT    ',I3,'  Y/N ?')
215   FORMAT (23(/))
225   FORMAT (27X,26('*'))
235   FORMAT (27X,'Computation in progress !')
245   FORMAT (27X,26('*'))
255   FORMAT (11(/))
C
C     INITIALISE VARIABLES
C     --------------------
C
      DO 10 I = 1 , NUMOUT
        YM(I) = 0.0
        DO 20 J = 1 , P
          YI((I-1)*P+J) = 0.0
20      CONTINUE
10    CONTINUE
C
      PEB = 0.0
      DPEB = 0.0
      TDIST = 0.0
      TIME = 0.0
      COUNT = 0
      NC = NINT(DSAMPL/DELTAT)
C
      DO 30 I = 1,NUMOUT
        YS(I) = YSS(I)
        DYS(I) = 0.0
        DTYS(I) = 0.0
30    CONTINUE
C
      DO 40 I = 1,NUMOUT+1
        DO 50 J = 1,NUMIN+1
          DO 60 K = 1,IORDER
            STATE(I,J,K)=0.0
            DSTATE(I,J,K)=0.0
60        CONTINUE
          DO 70 L = 1,TIMEDELAY
            Y(I,J,L) = 0.0
            YD(I,J,L) = 0.0
70        CONTINUE
C
C         IF(ANS5 .EQ. 'N' .OR. ANS5 .EQ. 'n') THEN
          U(J) = USS(J)
          NSTATE(J,I) = SSTATE(J,I)
50      CONTINUE
40    CONTINUE
C
      DO 80 I = 1,NUMIN
        DO 90 J = 1,N
```

```
          X((I-1)*N+J) = 0.0
90        CONTINUE
80      CONTINUE
C
C       READ IN VARIABLES
C       -----------------
C
        DO 100 I = 1 , NUMOUT
110       WRITE(*,5) I
          READ(*,*) DYS(I)
          WRITE(*,35) I
          READ(*,*) DTYS(I)
          IF( DTYS(I) .GE. TSIM) THEN
            WRITE(*,115) TSIM
            GOTO 110
          END IF
100     CONTINUE
C
        WRITE(*,55)
        READ(*,*) DPEB
        WRITE(*,65)
        READ(*,*) TDIST
C
C       READ IN NAME OF OUTPUT FILE
C       ---------------------------
C
        WRITE(*,15)
        READ(*,25) NAME
        OPEN (UNIT=12, FILE = NAME//'.PLT',STATUS='UNKNOWN')
        OPEN (UNIT=13, NAME='DIST.PLT',STATUS='UNKNOWN')
        OPEN (UNIT=14, NAME='U.PLT',STATUS='UNKNOWN')
        OPEN (UNIT=15, NAME='DU.PLT',STATUS='UNKNOWN')
C
C       INITIALISE QP VARIABLES
C       -----------------------
C
        NN = N*NUMIN
        NROWA = NMAXIM+MAXIM
        NROWH = NMAXIM
        NCOLH = NN
        NCOUT = P*NUMOUT
        ITMAX = 850
C        MSGLVL = 4
        MSGLVL = 0
C
        BIGBND = 1.0E+10
        FEATOL = 1.0D-08
        COLD = .TRUE.
        LP = .FALSE.
        MINSUM = .TRUE.
C
        DO 120 J = 1,NUMOUT
          DELAY(J) = ABS(NINT(TIMDEL(J,J)/DSAMPL))
120     CONTINUE
C
C       ARE THERE CONTROL MOVES CONSTRAINTS ? Y/N
C       -----------------------------------------
C
        DO 130 I = 1,NUMIN
          WRITE(*,125) I
          READ(*,75) ANS1(I)
          IF( ANS1(I) .EQ. 'Y' .OR. ANS1(I) .EQ. 'y') THEN
            WRITE(*,85) I,I
            READ(*,*) DUMIN(I),DUMAX(I)
```

```
          ELSE
            DUMIN(I) = -BIGBND
            DUMAX(I) = BIGBND
          END IF
C
C       ARE THERE ABSOLUTE INPUT CONSTRAINTS ? Y/N
C       ------------------------------------------
C
          WRITE(*,135) I
          READ(*,75) ANS2(I)
          IF( ANS2(I) .EQ. 'Y' .OR. ANS2(I) .EQ. 'y') THEN
            WRITE(*,95) I,I
            READ(*,*) UMIN(I),UMAX(I)
          ELSE
            UMIN(I) = -BIGBND
            UMAX(I) = BIGBND
          END IF
C

C         CREATE A LOWER TRIANG MATRIX OF 1's TO BOUND ABS U's
C         ----------------------------------------------------
C
            DO 140 L = 1,N
              DO 150 J = 1,L
                AA((I-1)*N+L,(I-1)*N+J) = 1.0
150           CONTINUE
140         CONTINUE
130     CONTINUE
C
C       ARE THERE OUTPUT CONSTRAINTS ? Y/N
C       ----------------------------------
C
        DO 160 I = 1,NUMOUT
          WRITE(*,145) I
          READ(*,75) ANS3(I)
          IF( ANS3(I) .EQ. 'Y' .OR. ANS3(I) .EQ. 'y') THEN
            WRITE(*,105) I,I
            READ(*,*) YMIN(I),YMAX(I)
          ELSE
            NCOUT = 0
            YMIN(I) = -BIGBND
            YMAX(I) = BIGBND
          END IF
160     CONTINUE
C
C       ADD DYNAMIC MATRIX ON TO AA
C       ---------------------------
C
        DO 170 K = 1, P*NUMOUT
          DO 180 J = 1, N*NUMIN
            AA((N*NUMIN)+K,J) = A(K,J)
180       CONTINUE
170     CONTINUE
C
C       CALCULATE DIMENSIONS
C       --------------------
C
        NCLIN = (NN + NCOUT)
        LIWORK = (NN + 2*NN + NCLIN)
        LWORK =  (2*NN**2 + 6*NN + 3*NCLIN)
        NCTOTL = NN + NCLIN
C
C       START NEW PAGE
C       --------------
```

```
C
      WRITE(*,215)
      WRITE(*,225)
      WRITE(*,235)
      WRITE(*,245)
      WRITE(*,255)
C
C      WRITE(12,45) TIME,YM(1),YM(2),YM(3)
C      WRITE(14,45) TIME,U(1),U(2),U(3)
C
190   IF ( TIME .LE. TSIM ) THEN
C
C       RESET SETPOINT AND DISTURBANCE
C       ------------------------------
C
        DO 200 I = 1,NUMOUT
          IF(TIME .GE. DTYS(I)) THEN
            YS(I) = YS(I) + DYS(I)
            DYS(I) = 0.0
          END IF
200     CONTINUE
C
        IF( TIME .GE. TDIST) THEN
          PEB = PEB + DPEB
          DPEB = 0.0
        END IF
C
        IF(ANS5 .EQ. 'N' .OR. ANS5 .EQ. 'n') THEN
          CALL NONMODEL(DELTAT,NSTATE,U,NUMIN,NUMOUT,
     &                  TIME,YM)
        ELSE
        CALL MODEL(DELTAT,DSAMPL,STATE,U,NUMIN,NUMOUT,ORDER,
     &             TIME,PEB,DNUMIN,DSTATE,DORDER)
        END IF
         NCOUNT = NCOUNT + 1
         COUNT = COUNT + 1
C
         IF ( COUNT*DELTAT .GE. DSAMPL ) THEN
           IF(ANS5 .EQ. 'N' .OR. ANS5 .EQ. 'n') THEN
C
           ELSE
            CALL OUT(YM,STATE,NUMIN,NUMOUT,ORDER,DSAMPL,DELTAT,
     &               DOUTPUT,DSTATE,DNUMIN,DORDER,PEB)
           END IF
            CALL QCONTR(DISTUR,YM,U,N,NUMIN,NUMOUT,P,YS,DU,DUS,
     &                  DOUTPUT,Q,PEB,TIME)
            COUNT = 0
         END IF
C
         IF ( NCOUNT .GE. NC ) THEN
             WRITE(12,45) (TIME-DSAMPL),YM(1),YM(2),YM(3)
             WRITE(14,45) (TIME-DSAMPL),U(1),U(2),U(3)
             WRITE(15,45) (TIME-DSAMPL),DU(1),DU(2),DU(3)
             WRITE(13,45) (TIME-DSAMPL),DOUTPUT(1),DOUTPUT(2),
     &                                  DOUTPUT(3)
             NCOUNT = 0
         END IF
C
         GOTO 190
       END IF
C
       CLOSE (15)
       CLOSE (12)
       CLOSE (13)
```

```
      CLOSE (14)
C
      RETURN
      END
C
C*************************************************************
      SUBROUTINE QCONTR(DISTUR,YM,U,N,NUMIN,NUMOUT,P,
     &                  YS,DU,DUS,DOUTPUT,Q,PEB,TIME)
C*************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DOUBLE PRECISION LAM
C
      INCLUDE 'PARAM.INC'
C
      DIMENSION BL(MAXIM+(2*NMAXIM)),BU(MAXIM+(2*NMAXIM)),
     &          CLAMDA(MAXIM+(2*NMAXIM)),CVEC(NMAXIM),
     &          WORK(ICMAXIM),
     &          BLD(MAXIM+TIMEDELAY),BUD(MAXIM+TIMEDELAY)
      DIMENSION ISTATE(MAXIM+(2*NMAXIM)),IWORK(MAXIM+(2*NMAXIM))
      LOGICAL COLD,LP,MINSUM
      EXTERNAL QPHES2
      CHARACTER*1 ANS1,ANS2,ANS3,PRED
C
      INTEGER N
      INTEGER P
      DIMENSION YS(CCO),DISTUR(CCO),DY(CCO),DOUTPUT(CCO)
      DIMENSION DU(CCO),DUS(CCO),Q(MAXIM)
C
      COMMON / CONTROL / ERROR(MAXIM),YI(MAXIM),
     &                   A(MAXIM,NMAXIM),LAM(NMAXIM,MAXIM),
     &                   HESS(NMAXIM,NMAXIM),FF(MAXIM),ANS5
      COMMON / QP / LWORK,LIWORK,COLD,LP,MINSUM,NROWA,NROWH,NCOLH,
     &              NCLIN,NCOUT,ITMAX,MSGLVL,NCTOTL,BIGBND,OBJ,
     &              FEATOL,ANS1(CCO),ANS2(CCO),ANS3(CCO),
     &              NN,X(NMAXIM)
      COMMON / BOUNDS / DUMIN(CCO),DUMAX(CCO),UMIN(CCO),UMAX(CCO),
     &                  YMIN(CCO),YMAX(CCO),
     &                  DELAY(CCO),AA(NMAXIM+MAXIM,NMAXIM+MAXIM)
      COMMON / DELAYS / TIMDEL(CCO,CCO), DTIMDEL(CCO,CCO)
      COMMON / QPRED / IDENT
C
      DIMENSION YM(CCO),U(CCO)
      INTEGER ORDER
      CHARACTER*1 ANS5
C
C     FORMAT STATEMENTS
C     -----------------
C
105   FORMAT(1HO,' CVEC')
115   FORMAT(1HO,' HESS')
125   FORMAT(1HO,' A MATRIX')
135   FORMAT(1H0,' LOWER BOUNDS')
145   FORMAT(1HO,' UPPER BOUNDS')
155   FORMAT(1HO,' INITIAL X')
165   FORMAT(1H ,1X,5(F10.2))
175   FORMAT(1H ,1X,5(E10.2))
C
C     FEED FOWARD CONTROL
C     -------------------
C
      TIMER = TIME + 1
      DO 10 I = 1,NUMOUT
        IF(TIMER .GT. P) THEN
          TIMER = P
```

```
        END IF
          YM(I) = YM(I) - FF((I-1)*P+TIMER)*PEB
10    CONTINUE
C
C     STEP INPUT/OUTPUT DISTURBANCE ESTIMATE
C     --------------------------------------
C
      DO 20 I = 1 , NUMOUT
         C = 0.0
         DO 30 J = 1 , NUMIN
            C = C + A((I-1)*P+1,N*(J-1)+1) * DU(J)
30       CONTINUE
         YI((I-1)*P+1) = YI((I-1)*P+1) + C
         DISTUR(I) = YM(I) - YI((I-1)*P+1)
20    CONTINUE
C
      DO 40 I = 1 , NUMOUT
         DO 50 J = 1 , P-1
            B = 0.0
            DO 60 K = 1 , NUMIN
               B = B + A((I-1)*P+J+1,(K-1)*N+1)*DU(K)
60          CONTINUE
            YI((I-1)*P+J) = YI((I-1)*P+J+1) + B
50       CONTINUE
C
         DO 70 K = 1 , NUMIN
            YI(I*P) = YI(I*P) + A(I*P,(K-1)*N+1)
     &                      * DU(K)
70       CONTINUE
C
40    CONTINUE
C
      DO 80 I = 1 , NUMOUT
        DO 90 J = 1 , P
          ERROR((I-1)*P+J) = YS(I) - YI((I-1)*P+J)
     &                           - DISTUR(I)
90       CONTINUE
80    CONTINUE
C
C     UPDATE GRADIENT VECTOR CVEC (ATRANS*Q*ERROR)
C     --------------------------------------------
C
      DO 100 I = 1 , N*NUMIN
        SUM = 0.0
        DO 110 K = 1 , NUMOUT*P
          SUM = SUM + A(K,I)*Q(K)*ERROR(K)
110     CONTINUE
        CVEC(I) = (-1)*SUM
100   CONTINUE
C
C     CALCULATE CONSTRAINTS BL & BU
C     -----------------------------
C
C     INPUT CONSTRAINTS
C     -----------------
C
      DO 120 J=1,NUMIN
C
C       INPUT CHANGES BOUNDS
C       --------------------
C
        IF( ANS1(J) .EQ. 'Y' .OR. ANS1(J) .EQ. 'y') THEN
          DO 130 I=1,N
           BL((J-1)*N+I) = DUMIN(J)
```

```
            BU((J-1)*N+I) = DUMAX(J)
130       CONTINUE
          ELSE
            DO 140 I=1,N
              BL((J-1)*N+I) = -BIGBND
              BU((J-1)*N+I) = BIGBND
140       CONTINUE
          END IF
C
C         ABSOLUTE INPUT BOUNDS
C         ---------------------
C
          IF (ANS2(J) .EQ. 'Y' .OR. ANS2(J) .EQ. 'y') THEN
            DO 150 I = 1,N
              BL((N*NUMIN)+(J-1)*N+I) = UMIN(J) - U(J)
              BU((N*NUMIN)+(J-1)*N+I) = UMAX(J) - U(J)
150       CONTINUE
          ELSE
            DO 160 I = 1,N
              BL((N*NUMIN)+(J-1)*N+I) = -BIGBND
              BU((N*NUMIN)+(J-1)*N+I) = BIGBND
160       CONTINUE
          END IF
C
C         MULTIPLY ABS BOUNDS BY INVERSE OF LOWER TRIANG MATRIX
C         -----------------------------------------------------
C
c         IF( N .GT. 1) THEN
c           DO 170 I = N,2,-1
c             BL((N*NUMIN)+(J-1)*N+I) = BL((N*NUMIN)+(J-1)*N+I)
C     &                               - BL((N*NUMIN)+(J-1)*N+(I-1))
c             BU((N*NUMIN)+(J-1)*N+I) = BU((N*NUMIN)+(J-1)*N+I)
C     &                               - BU((N*NUMIN)+(J-1)*N+(I-1))
c170        CONTINUE
c          END IF
C
120   CONTINUE
C
C         OUTPUT CONSTRAINTS
C         ------------------
C
          DO 180 J = 1,NUMOUT
            IF(ANS3(J) .EQ. 'Y' .OR. ANS3(J) .EQ. 'y') THEN
C
C             ADJUST WINDOW FOR NON-MINIMUM PHASE
C             -----------------------------------
C
              IF(TIME .LE. DELAY(J)) THEN
                DO 190 K = 1, DELAY(J)
                  BLD((J-1)*P+K) = - BIGBND
                  BUD((J-1)*P+K) = BIGBND
190             CONTINUE
                DO 200 K = DELAY(J)+1, P
                  BLD((J-1)*P+K) = YMIN(J) - YS(J) +
     &                             ERROR((J-1)*P+(K-DELAY(J)))
                  BUD((J-1)*P+K) = YMAX(J) - YS(J) +
     &                             ERROR((J-1)*P+(K-DELAY(J)))
200             CONTINUE
C
C               ASSIGN DELAYED BOUNDS
C               ---------------------
C
                DO 210 I= 1, P
                  BL((J-1)*P+(2*(N*NUMIN))+I) = BLD((J-1)*P+I)
```

```
                BU((J-1)*P+(2*(N*NUMIN))+I) = BUD((J-1)*P+I)
210           CONTINUE
C
C             SHIFT THE DELAY UP
C             ------------------
C
              DELAY(J) = DELAY(J) - 1
            ELSE
              DO 220 I= 1,P
                BL((J-1)*P+(2*(N*NUMIN))+I) = YMIN(J) - YS(J) +
     &                                        ERROR((J-1)*P+I)
                BU((J-1)*P+(2*(N*NUMIN))+I) = YMAX(J) - YS(J) +
     &                                        ERROR((J-1)*P+I)
220           CONTINUE
            END IF
C
C           UNCONSTRAINED OUTPUTS
C           ---------------------
C
          ELSE
            DO 230 I= 1,P
              BL((J-1)*P+(2*(N*NUMIN))+I) = -BIGBND
              BU((J-1)*P+(2*(N*NUMIN))+I) = BIGBND
230         CONTINUE
          END IF
180     CONTINUE
C
        CALL QPSOL(ITMAX,MSGLVL,NN,
       &           NCLIN,NCTOTL,NROWA,NROWH,NCOLH,
       &           BIGBND,FEATOL,AA,BL,BU,CVEC,HESS,QPHES2,
       &           COLD,LP,MINSUM,ISTATE,X,
       &           INFORM,ITER,OBJ,CLAMDA,
       &           IWORK,LIWORK,WORK,LWORK)
C
C       CALCULATE FIRST MOVE WHICH YOU IMPLEMENT
C       ----------------------------------------
C
        DO 270 I = 1 , NUMIN
          U(I) = U(I) + X((I-1)*N+1)
          DU(I) = X((I-1)*N+1)
270     CONTINUE
C
C       CALCULATE PREDICTIONS IF REQUIRED
C       ---------------------------------
C
        IF(PRED .EQ. 'Y' .OR. PRED .EQ. 'y') THEN
          IDENT = 1
          CALL PREDIC(YS,P,M,N,NUMIN,NUMOUT,TIME,IDENT)
        END IF
C
        RETURN
        END
```

```
C*****************************************************************
      SUBROUTINE DNEWTON(N,X,IERR,FSUB,DFSUB,IPRINT,ITMAX,MAXDMP,
     &                   ATOL,METH,DWORK,LDWORK,IWORK,LIWORK)
C*****************************************************************
C
C       PROGRAM DNEWTON
C       DAMPED NEWTON'S METHOD  ( DOUBLE PRECISION VERSION ) .
C       NEWTON STEP-SIZE HALVED UNTIL REDUCTION IN RESIDUAL EUCLIDIAN
C          NORM ACHIEVED .  REF. :
C          CONTE, S. D. AND C. DE BOOR, "ELEMENTARY NUMERICAL
C          ANALYSIS" (1980) .
C
C       C.L.E. SWARTZ.  LATEST UPDATE: OCTOBER 1991
C
C       UPON ENTRY :
C       N      =  NUMBER OF EQNS AND VBLS.
C
C       X      =  VECTOR OF INITIAL VALUES.
C
C       IPRINT = 0  ===>  STREAMLINED PRINTOUT
C              = 1  ===>  DETAILED PRINTOUT
C              =-1  ===>  NO PRINTING
C
C       ITMAX  =  MAXIMUM ALLOWABLE NO. OF NEWTON ITERATIONS.
C                 REPLACED UPON RETURN BY ACTUAL NUMBER OF ITERATIONS
C                 TAKEN.
C
C       MAXDMP =  MAXIMUM ALLOWABLE NO. OF DAMPING ITERATIONS.
C
C       ATOL   =  TOLERANCE.  CONVERGED WHEN RESIDUAL MAX NORM < ATOL.
C
C       METH   = 0  ===>  USER-SUPPLIED JACOBIAN.
C              = 1  ===>  JACOBIAN APPROXIMATED BY FORWARD DIFFERENCING.
C              = 2  ===>  JACOBIAN APPROXIMATED BY CENTRAL DIFFERENCING.
C
C       DWORK  =  DOUBLE PRECISION WORK ARRAY OF DIMENSION >= N*(N+5).
C
C       LDWORK    =  LENGTH OF DWORK ARRAY.
C
C       IWORK  =  INTEGER WORK ARRAY OF DIMENSION >= N.
C
C       LIWORK    =  LENGTH OF IWORK ARRAY.
C
C       FSUB   =  USER-SUPPLIED SUBROUTINE FOR CALCULATION OF SET OF
C                 FUNCTION VALUES, F.  CALLED IN THE FORM :
C                   "  CALL FSUB(N,X,F)  "
C
C       DFSUB  =  USER-SUPPLIED SUBROUTINE FOR CALCULATION OF JACOBIAN,
C                 DF.  ( DUMMY IF METH=1,2 )  CALLED IN THE FORM :
C                   "  CALL DFSUB(N,X,DF).
C                 LEADING DIMENSION OF DF SHOULD BE N.
C
C       UPON RETURN :
C       X      =  SOLUTION VECTOR  ( PROVIDED THAT IERR = 0 ! )
C
C       IERR   = 0  ===>  NO ERRORS DETECTED IN SOLUTION PROCEDURE.
C              = 1  ===>  INSUFFICIENT WORKSPACE
C              = 2  ===>  SINGULAR JACOBIAN
C              = 3  ===>  MAX. NO. OF DAMPING ITERATIONS REACHED
C              = 4  ===>  MAX. NO. OF NEWTON ITERATIONS REACHED
C
C
       IMPLICIT DOUBLE PRECISION (A-H,O-Z)
       DIMENSION X(1),DWORK(1),IWORK(1)
```

```
      COMMON /MACH/ EPS
      EXTERNAL FSUB,DFSUB
C
C     MACHINE-DEPENDENT PARAMETER :
C
      EPS=2.776D-17
C
C     A SIMPLE-MINDED CHECK ON DECLARED WORK ARRAY DIMENSIONS
C
      LREQ=N*(N+5)
      IF(LDWORK.LT.LREQ.OR.LIWORK.LT.N) THEN
         IERR = 1
         IF( IPRINT .GT. -1 ) WRITE(6,600) LREQ,N
         RETURN
      END IF
600   FORMAT(1H ,//' INSUFFICIENT WORKSPACE.  REQUIRE : '
     &          /' LDWORK >=',I6,5X,'LIWORK >=',I3)
C
C     WORKSPACE ALLOCATION
C
      IDW1=1
      IDW2=IDW1+N
      IDW3=IDW2+N
      IDW4=IDW3+N
      IDW5=IDW4+N*N
      IDW6=IDW5+N
      IIW1=1
C
      IERR=0
      CALL METHOD(N,X,IERR,FSUB,DFSUB,IPRINT,ITMAX,MAXDMP,
     &          ATOL,METH,DWORK(IDW1),DWORK(IDW2),DWORK(IDW3),
     &          DWORK(IDW4),DWORK(IDW5),DWORK(IDW6),IWORK(IIW1))
C
      RETURN
      END
C
C
      SUBROUTINE METHOD(N,X,IERR,FSUB,DFSUB,IPRINT,ITMAX,MAXDMP,
     &          ATOL,METH,DX,XNEW,F,DF,FPLUS,FMINUS,IPVT)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION X(1),DX(1),XNEW(1),F(1),DF(N,1),FPLUS(1),FMINUS(1),
     &          IPVT(1)
      EXTERNAL FSUB,DFSUB
C
C     COMPUTE RESIDUAL VECTOR AND ITS EUCLIDEAN NORM
C
      CALL FSUB(N,X,F)
      CALL NORM(N,F,FNORM)
      ITER=0
      IF(IPRINT.GT.0) THEN
         WRITE(6,600) ITER
         WRITE(6,610) FNORM
      END IF
      MAXDMP=MAXDMP+1
C
C..... INITIAL TEST FOR CONVERGENCE
C
      BIG=DABS(F(1))
      DO 51 J=1,N
         TEST=DABS(F(J))
         IF(TEST.GT.BIG) BIG=TEST
51    CONTINUE
      IF(BIG.LT.ATOL) GO TO 60
C
```

```
C        NEWTON ITERATION LOOP
C
         DO 10 I=1,ITMAX
            ITER=I
            IF(IPRINT.GT.0) WRITE(6,600) I
C
C        COMPUTING NEWTON STEP
C
            CALL DELTAX(N,X,FSUB,DFSUB,F,DF,DX,FPLUS,FMINUS,IPVT,
     &           METH,IFLAG)
            IF(IFLAG.NE.0) THEN
               IERR=2
               IF( IPRINT .GT. -1 ) WRITE(6,620)
               RETURN
            END IF
C
C        DAMPING ITERATION LOOP
C
            IF(IPRINT.GT.0) WRITE(6,630)
            DO 20 K=1,MAXDMP
               DO 25 J=1,N
25                XNEW(J)=X(J)+DX(J)/2.D0**(K-1)
C
C        NON-NEGATIVITY CHECK
C
C                SMALL=1.D0
C                DO 22 J=1,N
C22                 IF(XNEW(J).LT.SMALL) SMALL=XNEW(J)
C                IF(SMALL.LT.1.D-08) GO TO 20
C
                 CALL FSUB(N,XNEW,F)
                 CALL NORM(N,F,FNORM2)
                 IF(IPRINT.GT.0) THEN
                    WRITE(6,640) K-1,FNORM2
                 END IF
                 IF(FNORM2.LT.FNORM) GO TO 30
20          CONTINUE
            IERR=3
            IF ( IPRINT .GT. -1 ) WRITE(6,650)
            RETURN
C
C        UPDATE X AND FNORM
C
30          DO 40 J=1,N
40             X(J)=XNEW(J)
            FNORM=FNORM2
C
C        TEST FOR CONVERGENCE
C
            BIG=DABS(F(1))
            DO 50 J=1,N
               TEST=DABS(F(J))
               IF(TEST.GT.BIG) BIG=TEST
50          CONTINUE
            IF(BIG.LT.ATOL) GO TO 60
10       CONTINUE
         IERR=4
         IF ( IPRINT .GT. -1) WRITE(6,660)
         RETURN
C
60       IF(IPRINT.GT.-1) WRITE(6,670) ITER,FNORM
         ITMAX = ITER
C
C        FORMAT STATEMENTS
```

```
C
600     FORMAT(1H ,/' NEWTON ITERATION # ',I2)
610     FORMAT(1H ,'          RESIDUAL NORM = ',1PD15.8)
620     FORMAT(1H ,/' SINGULAR JACOBIAN ')
630     FORMAT(1H ,'DAMPING  :    ITERATION     RESIDUAL NORM')
640     FORMAT(1H ,16X,I3,7X,1PD15.8)
650     FORMAT(1H ,/' MAXIMUM DAMPING ITERATION LIMIT REACHED ')
660     FORMAT(1H ,/' MAXIMUM NEWTON ITERATION LIMIT REACHED ')
670     FORMAT(1H ,/' CONVERGENCE AFTER',I3,' NEWTON ITERATIONS'
     &          /' RESIDUAL MAX-NORM = ',1PD15.8)
C
        RETURN
        END
C
C
        SUBROUTINE NORM(N,F,FNORM)
C
C       CALCULATION OF EUCLIDEAN NORM OF F
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        DIMENSION F(1)
C
        SUM=0.D0
        DO 10 I=1,N
10         SUM=SUM+F(I)*F(I)
        FNORM=DSQRT(SUM)
C
        RETURN
        END
C
C
        SUBROUTINE DELTAX(N,X,FSUB,DFSUB,F,DF,DX,FPLUS,FMINUS,IPVT,
     &          METH,IFLAG)
C
C       NUMERICAL DIFFERENTIATION ( IF REQUIRED ), AND
C       SOLUTION OF LINEAR SYSTEM  (DF)(DX) = -F
C       "OPTIMAL" DIFFERENCING STEP-SIZES AS IN :
C       MURTAGH, B. A. AND M. A. SAUNDERS, "MINOS 5.0 USER'S MAMUAL"
C           (1983)
C
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        DIMENSION X(1),F(1),DF(N,1),DX(1),FPLUS(1),FMINUS(1),IPVT(1)
        COMMON /MACH/ EPS
        EXTERNAL FSUB,DFSUB
        DATA ONE/1.D0/
C
        IFLAG=0
        GO TO (10,20,30), METH+1
C
C       USER-SUPPLIED JACOBIAN
C
10      CALL DFSUB(N,X,DF)
        GO TO 100
C
C       FORWARD-DIFFERENCE APPROXIMATION
C
20      DELTA=DSQRT(EPS)
        DO 40 I=1,N
           XI=X(I)
           XSTEP=DELTA*(ONE+DABS(XI))
           X(I)=XI+XSTEP
           CALL FSUB(N,X,FPLUS)
           X(I)=XI
           DO 40 J=1,N
              DF(J,I)=(FPLUS(J)-F(J))/XSTEP
```

```
40      CONTINUE
        GO TO 100
C
C       CENTRAL-DIFFERENCING
C
30      DELTA=EPS**0.333333
        DO 50 I=1,N
           XI=X(I)
           XSTEP=DELTA*(ONE+DABS(XI))
           X(I)=XI+XSTEP
           CALL FSUB(N,X,FPLUS)
           X(I)=XI-XSTEP
           CALL FSUB(N,X,FMINUS)
           X(I)=XI
           DO 50 J=1,N
              DF(J,I)=0.5D0*(FPLUS(J)-FMINUS(J))/XSTEP
50      CONTINUE
C
C       SOLUTION OF LINEAR SYSTEM
C
100     CALL DGEFA(DF,N,N,IPVT,INFO)
        IF(INFO.NE.0) THEN
           IFLAG=1
           RETURN
        END IF
        DO 110 I=1,N
110        DX(I)=-F(I)
        CALL DGESL(DF,N,N,IPVT,DX,0)
C
        RETURN
        END
```

```
C*****************************************************************
      SUBROUTINE DGEFA(A,LDA,N,IPVT,INFO)
C*****************************************************************
      INTEGER LDA,N,IPVT(1),INFO
      DOUBLE PRECISION A(LDA,1)
C
C     SGEFA FACTORS A REAL MATRIX BY GAUSSIAN ELIMINATION.
C
C     SGEFA IS USUALLY CALLED BY SGECO, BUT IT CAN BE CALLED
C     DIRECTLY WITH A SAVING IN TIME IF  RCOND  IS NOT NEEDED.
C     (TIME FOR SGECO) = (1 + 9/N)*(TIME FOR SGEFA) .
C
C     ON ENTRY
C
C        A        REAL(LDA, N)
C                 THE MATRIX TO BE FACTORED.
C
C        LDA      INTEGER
C                 THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N        INTEGER
C                 THE ORDER OF THE MATRIX  A .
C
C     ON RETURN
C
C        A        AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
C                 WHICH WERE USED TO OBTAIN IT.
C                 THE FACTORIZATION CAN BE WRITTEN  A = L*U  WHERE
C                 L  IS A PRODUCT OF PERMUTATION AND UNIT LOWER
C                 TRIANGULAR MATRICES AND  U  IS UPPER TRIANGULAR.
C
C        IPVT     INTEGER(N)
C                 AN INTEGER VECTOR OF PIVOT INDICES.
C
C        INFO     INTEGER
C                 = 0  NORMAL VALUE.
C                 = K  IF  U(K,K) .EQ. 0.0 .  THIS IS NOT AN ERROR
C                      CONDITION FOR THIS SUBROUTINE, BUT IT DOES
C                      INDICATE THAT SGESL OR SGEDI WILL DIVIDE BY ZERO
C                      IF CALLED.  USE  RCOND  IN SGECO FOR A RELIABLE
C                      INDICATION OF SINGULARITY.
C
C     LINPACK. THIS VERSION DATED 08/14/78 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LAB.
C
C     SUBROUTINES AND FUNCTIONS
C
C     BLAS FAXPY,FSCAL,IFAMAX
C
C     INTERNAL VARIABLES
C
      DOUBLE PRECISION T
      INTEGER IFAMAX,J,K,KP1,L,NM1
C
C
C     GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
C
      INFO = 0
      NM1 = N - 1
      IF (NM1 .LT. 1) GO TO 70
      DO 60 K = 1, NM1
         KP1 = K + 1
C
C        FIND L = PIVOT INDEX
```

```
C
      L = IFAMAX(N-K+1,A(K,K),1) + K - 1
      IPVT(K) = L
C
C     ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED
C
      IF (A(L,K) .EQ. 0.0E0) GO TO 40
C
C        INTERCHANGE IF NECESSARY
C
         IF (L .EQ. K) GO TO 10
            T = A(L,K)
            A(L,K) = A(K,K)
            A(K,K) = T
   10    CONTINUE
C
C        COMPUTE MULTIPLIERS
C
         T = -1.0E0/A(K,K)
         CALL FSCAL(N-K,T,A(K+1,K),1)
C
C        ROW ELIMINATION WITH COLUMN INDEXING
C
         DO 30 J = KP1, N
            T = A(L,J)
            IF (L .EQ. K) GO TO 20
               A(L,J) = A(K,J)
               A(K,J) = T
   20       CONTINUE
            CALL FAXPY(N-K,T,A(K+1,K),1,A(K+1,J),1)
   30    CONTINUE
      GO TO 50
   40 CONTINUE
         INFO = K
   50 CONTINUE
   60 CONTINUE
   70 CONTINUE
      IPVT(N) = N
      IF (A(N,N) .EQ. 0.0E0) INFO = N
      RETURN
      END
C
C*****************************************************************
      SUBROUTINE DGESL(A,LDA,N,IPVT,B,JOB)
C*****************************************************************
      INTEGER LDA,N,IPVT(1),JOB
      DOUBLE PRECISION A(LDA,1),B(1)
C
C     SGESL SOLVES THE REAL SYSTEM
C     A * X = B  OR  TRANS(A) * X = B
C     USING THE FACTORS COMPUTED BY SGECO OR SGEFA.
C
C     ON ENTRY
C
C        A       REAL(LDA, N)
C                THE OUTPUT FROM SGECO OR SGEFA.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A .
C
C        IPVT    INTEGER(N)
```

```
C                THE PIVOT VECTOR FROM SGECO OR SGEFA.
C
C        B       REAL(N)
C                THE RIGHT HAND SIDE VECTOR.
C
C        JOB     INTEGER
C                = 0          TO SOLVE  A*X = B ,
C                = NONZERO    TO SOLVE  TRANS(A)*X = B  WHERE
C                             TRANS(A)  IS THE TRANSPOSE.
C
C     ON RETURN
C
C        B       THE SOLUTION VECTOR  X .
C
C     ERROR CONDITION
C
C        A DIVISION BY ZERO WILL OCCUR IF THE INPUT FACTOR CONTAINS A
C        ZERO ON THE DIAGONAL.  TECHNICALLY THIS INDICATES SINGULARITY
C        BUT IT IS OFTEN CAUSED BY IMPROPER ARGUMENTS OR IMPROPER
C        SETTING OF LDA .  IT WILL NOT OCCUR IF THE SUBROUTINES ARE
C        CALLED CORRECTLY AND IF SGECO HAS SET RCOND .GT. 0.0
C        OR SGEFA HAS SET INFO .EQ. 0 .
C
C     TO COMPUTE  INVERSE(A) * C  WHERE  C  IS A MATRIX
C     WITH  P  COLUMNS
C           CALL SGECO(A,LDA,N,IPVT,RCOND,Z)
C           IF (RCOND IS TOO SMALL) GO TO ...
C           DO 10 J = 1, P
C              CALL SGESL(A,LDA,N,IPVT,C(1,J),0)
C        10 CONTINUE
C
C     LINPACK. THIS VERSION DATED 08/14/78 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LAB.
C
C     SUBROUTINES AND FUNCTIONS
C
C     BLAS FAXPY,SDOT
C
C     INTERNAL VARIABLES
C
      DOUBLE PRECISION SDOT,T
      INTEGER K,KB,L,NM1
C
      NM1 = N - 1
      IF (JOB .NE. 0) GO TO 50
C
C        JOB = 0 , SOLVE  A * X = B
C        FIRST SOLVE  L*Y = B
C
         IF (NM1 .LT. 1) GO TO 30
         DO 20 K = 1, NM1
            L = IPVT(K)
            T = B(L)
            IF (L .EQ. K) GO TO 10
               B(L) = B(K)
               B(K) = T
   10       CONTINUE
            CALL FAXPY(N-K,T,A(K+1,K),1,B(K+1),1)
   20    CONTINUE
   30    CONTINUE
C
C        NOW SOLVE  U*X = Y
C
         DO 40 KB = 1, N
```

```
            K = N + 1 - KB
            B(K) = B(K)/A(K,K)
            T = -B(K)
            CALL FAXPY(K-1,T,A(1,K),1,B(1),1)
   40     CONTINUE
      GO TO 100
   50 CONTINUE
C
C        JOB = NONZERO, SOLVE  TRANS(A) * X = B
C        FIRST SOLVE  TRANS(U)*Y = B
C
         DO 60 K = 1, N
            T = SDOT(K-1,A(1,K),1,B(1),1)
            B(K) = (B(K) - T)/A(K,K)
   60    CONTINUE
C
C        NOW SOLVE TRANS(L)*X = Y
C
         IF (NM1 .LT. 1) GO TO 90
         DO 80 KB = 1, NM1
            K = N - KB
            B(K) = B(K) + SDOT(N-K,A(K+1,K),1,B(K+1),1)
            L = IPVT(K)
            IF (L .EQ. K) GO TO 70
               T = B(L)
               B(L) = B(K)
               B(K) = T
   70       CONTINUE
   80    CONTINUE
   90    CONTINUE
  100 CONTINUE
      RETURN
      END
      SUBROUTINE FAXPY(N,SA,SX,INCX,SY,INCY)
C
C     CONSTANT TIMES A VECTOR PLUS A VECTOR.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION SX(1),SY(1),SA
      INTEGER I,INCX,INCY,IXIY,M,MP1,N
C
      IF(N.LE.0)RETURN
      IF (SA .EQ. 0.0E0) RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        SY(IY) = SY(IY) + SA*SX(IX)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP
```

```
C
   20 M = MOD(N,4)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        SY(I) = SY(I) + SA*SX(I)
   30 CONTINUE
      IF( N .LT. 4 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,4
        SY(I) = SY(I) + SA*SX(I)
        SY(I + 1) = SY(I + 1) + SA*SX(I + 1)
        SY(I + 2) = SY(I + 2) + SA*SX(I + 2)
        SY(I + 3) = SY(I + 3) + SA*SX(I + 3)
   50 CONTINUE
      RETURN
      END
      SUBROUTINE  FSCAL(N,DA,DX,INCX)
C
C     SCALES A VECTOR BY A CONSTANT.
C     USES UNROLLED LOOPS FOR INCREMENT EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION DA,DX(1)
      INTEGER I,INCX,M,MP1,N,NINCX
C
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GO TO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        DX(I) = DA*DX(I)
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,5)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DX(I) = DA*DX(I)
   30 CONTINUE
      IF( N .LT. 5 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,5
        DX(I) = DA*DX(I)
        DX(I + 1) = DA*DX(I + 1)
        DX(I + 2) = DA*DX(I + 2)
        DX(I + 3) = DA*DX(I + 3)
        DX(I + 4) = DA*DX(I + 4)
   50 CONTINUE
      RETURN
      END
      INTEGER FUNCTION IFAMAX(N,SX,INCX)
C
C     FINDS THE INDEX OF ELEMENT HAVING MAX. ABSOLUTE VALUE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION SX(1),SMAX
      INTEGER I,INCX,IX,N
```

```
C
      IFAMAX = 0
      IF( N .LT. 1 ) RETURN
      IFAMAX = 1
      IF(N.EQ.1)RETURN
      IF(INCX.EQ.1)GO TO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      IX = 1
      SMAX = DABS(SX(1))
      IX = IX + INCX
      DO 10 I = 2,N
         IF(DABS(SX(IX)).LE.SMAX) GO TO 5
         IFAMAX = I
         SMAX = DABS(SX(IX))
    5    IX = IX + INCX
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
   20 SMAX = DABS(SX(1))
      DO 30 I = 2,N
         IF(DABS(SX(I)).LE.SMAX) GO TO 30
         IFAMAX = I
         SMAX = DABS(SX(I))
   30 CONTINUE
      RETURN
      END
      DOUBLE PRECISION FUNCTION SDOT(N,SX,INCX,SY,INCY)
C     FORMS THE DOT PRODUCT OF TWO VECTORS.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION SX(1),SY(1),TEMP
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      SDOT = 0.0E0
      TEMP = 0.0E0
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        TEMP = TEMP + SX(IX)*SY(IY)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      SDOT = TEMP
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,5)
```

```
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        TEMP = TEMP + SX(I)*SY(I)
30    CONTINUE
      IF( N .LT. 5 ) GO TO 60
40    MP1 = M + 1
      DO 50 I = MP1,N,5
        TEMP = TEMP + SX(I)*SY(I) + SX(I + 1)*SY(I + 1) +
     *    SX(I + 2)*SY(I + 2) + SX(I + 3)*SY(I + 3) + SX(I + 4)*SY(I + 4)
50    CONTINUE
60    SDOT = TEMP
      RETURN
      END
```

```
C*************************************************************
      SUBROUTINE FSUB(N,X,F)
C*************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C
      DIMENSION X(1),F(1)
C
      COMMON / SPACEN / HM,RUF,QMUF,QKO,VS,QM,RW,RS,RFF
C
      COMMON / PARAN / QMFF,WFF,WSF,QC,C1,C2,RM,CS,ROS,WF
C
C     PARAMETERS
C     ----------
C
      A1=1.363
      A2=-10.75
C
100   QM = ((QMFF+X(1))/RS + WFF/RW - (A2)/RW +
     &      ((1.0-A1)*(1.0-(CS/RS))*WSF)/RW)/
     &      (1.0 - (1.0-(CS/RS))*(1.0-A1))
C
      WF = (QM+WSF/RW)*(1-(CS/RS))*RW
      IF( WF .GT. 21.4) THEN
        A1 = 0.837
        A2 = 0.35
        GOTO 100
      END IF
C
      QC = QM + WSF/RW
C
      FV = CS/RS
C
      D50 = EXP(3.5433 - 1.6895E-01*QC*60 + 3.2449*FV)
C
      C1 = 0.0042*QMFF*D50 - 0.0154*D50 - 0.0704*QMFF + 1.3412
C
      C2 = 0.0502*QMFF*D50 - 0.0660*D50 - 2.9354*QMFF + 4.642
C
      RUF = C1*ROS/(C1*ROS+C2*(1-ROS))
C
      F(1) = X(1) - QC*CS*ROS*C1 - QC*CS*(1-ROS)*C2
C
      RETURN
      END
```

```
C*************************************************************
      SUBROUTINE FSUB(N,X,F)
C*************************************************************
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C
      DIMENSION X(4),F(4)
      COMMON /NEWPARA/ RFF,MFF,WFF,WSF,HM,KO,VS,RW,RS,RUF,
     &                 QC,C1,C2,QM,WF,FV,D50
      DOUBLE PRECISION KO,MFF
C
      A1=1.363
      A2=-10.75
C
100   QM = ((MFF+X(3))/RS + WFF/RW - (A2)/RW +
     &     ((1.0-A1)*(1.0-(X(2)/RS))*WSF)/RW)/
     &     (1.0 - (1.0-(X(2)/RS))*(1.0-A1))
C
      WF = (QM+WSF/RW)*(1-(X(2)/RS))*RW
      IF( WF .GT. 21.4) THEN
        A1 = 0.837
        A2 = 0.35
        GOTO 100
      END IF
C
      QC = QM + WSF/RW
C
      FV = X(2)/RS
C
      D50 = EXP(3.5433 - 1.6895E-01*QC*60 + 3.2449*FV)
C
      C1 = 0.0042*MFF*D50 - 0.0154*D50 - 0.0704*MFF + 1.3412
C
      C2 = 0.0502*MFF*D50 - 0.0660*D50 - 2.9354*MFF + 4.642
C
      RUF = C1*X(4)/(C1*X(4)+C2*(1-X(4)))
C
      F(1) = MFF*RFF + X(3)*RUF - KO*HM*X(1) - (MFF+X(3))*X(1)
      F(2) = (MFF+X(3)) - QC*X(2)
      F(3) = X(3) - QC*X(2)*X(4)*C1 - QC*X(2)*(1-X(4))*C2
      F(4) = (MFF+X(3))*(X(1)-X(4))
C
      RETURN
      END
```

```
C
C      CONTAINS PARAMETERS FOR VECTORS
C
       PARAMETER(IORDER=4, MAXIM=1500, NMAXIM=90, TIMEDELAY=100,
      &          VECTOR=25, NUM=3, CCO=10, ICMAXIM=11500)
```