

https://dspace.org/

Simulation of ore dressing plants

Ford, Merrill Anthony

2015-01-21
http://hdl.handle.net/10539/16662

Downloaded from a DSpace repository. https://dspace.org/

SIMULATION OF C[illegible] DRESSING PLANTS

Merrill Anthony Ford

A thesis submitted to the Faculty of Engineering, University of the Witwatersrand, for the degree of Doctor of Philosophy.

Johannesburg 1979.

SIMULATION OF ORE DRESSING PLANTS

MERRILL ANTHONY [illegible]

Johannesburg, 19[illegible]3

ABSTRACT

The mass balance of any process plant can be described by a set of simultaneous equations. In a chemical plant the stream variables can be described in terms of the concentrations of several chemical species whereas, in an ore dressing plant, the stream variables must be used to describe a large number of unique solid particles. This implies that, in general, ore dressing plants will require a far greater number of simultaneous equations to describe same and consequently the method of solution of these equations should be efficient and automated.

This thesis demonstrates how particles can be categorized and the mass balance for complex recycle systems rapidly achieved. The simulator that has been developed uses unit model modules which are linked together by an executive computer program. The executive, by treating each module in turn, obtains a set of simultaneous equations which are solved iteratively. The sequence in which the modules are treated is determined by partitioning and tearing algorithms which select tear streams so rendering the system acyclic. The algorithms objective is to choose that tear set (as there are usually several) which will allow convergence in the fewest iterations, based on a criterion which can be extended to apply to all convergence methods considered.

A simultaneous convergence technique is developed which is applicable specifically to linear ore dressing plants, but can be applied to non-linear systems. This technique is called the reduced Newton method and is shown to exhibit local convergence under reasonable conditions and is generally superior to both direct substitution and the bounded Wegstein methods.

Mathematical models of all types of ore dressing equipment are reviewed and some, after modification, selected for programming as unit modules. These modules include ones for crushing, grinding, flotation and cyclones while a general purpose partition curve module has been written to model gravity, magnetic and electrostatic separators, screens, classifiers and sorters. A very simple module is available as a model of thickeners and filters.

The executive program is divided into phases. The phases allow variation in problem size without wasted storage capacity plus the overlaying of storage. The input phase uses words with the data to make it readable and at the same time eliminating the need for dummy data. The next phase

orders the calculation according to a user selected criterion and sets up the data files for the calculation phase. The preprocessing of data in previous phases allows the calculation phase to execute efficiently and to obtain a mass balance using a chosen convergence method. The output of the first two phases lets the user find any faults in his data before the potentially expensive calculation phase is run, while the output from this last phase is very flexible and the user can take as much or as little detail as he likes.

The four test examples demonstrate the usage and flexibility of the simulator while giving an indication of its scope.

ACKNOWLEDGEMENTS

I wish to record my thanks to those without whom this thesis may not have been produced;

Professor R.P. King, B.Sc (Eng.), M.Sc (Eng.) (Rand), PhD. (Manc.), M.I. Chem. E., M.S.A.I. Chem. E. (Fellow), for his help, criticism and guidance;

The members of staff of the Extraction Metallurgy Division, of the Atomic Energy Board, particularly Mr H.E. James and Mr H.A. Simonsen for their encouragement and Mrs A. Gerber for the typing of the thesis;

The Atomic Energy Board for financial assistance in the form of a bursary;

My parents for their support and the opportunities they have given me.

DECLARATION

I hereby declare that the work contained in this thesis is my own, except where indicated, and has not been submitted for any other degree.



M.A. Ford

TABLE OF CONTENTS

		PAGE NUMBER
PART I - BACKGROUND AND THEORY		
1	INTRODUCTION	2
	1.1 History of Ore Dressing Simulators	2
	1.2 History of Chemical Plant Simulators	3
	1.3 Value and Use of a Simulator	4
	1.4 Benefits and Cost of a Simulator	6
2	STRATEGY OF SIMULATION	7
	2.1 Simulation Mode versus Design Mode	7
	2.2 Modular Approach versus Equation Solving Approach	8
	2.3 Selection of Strategy	9
3	SOLUTION TECHNIQUES	10
	3.1 Simultaneous Solution of Recycle Systems	10
	3.2 Sequential Solution of Recycle Systems	11
	3.2.1 Direct Substitution	12
	3.2.2 Accelerated Direct Substitution	12
	3.2.3 Newton and Quasi-Newton	14
	3.3 Discussion of Solution Techniques	15
4	DIFFERENCES BETWEEN ORE DRESSING AND CHEMICAL PLANT SIMULATORS	18
	4.1 Representation of Streams	18
	4.2 Transformation Units	21
	4.3 Physical Properties	22
5	SOLUTION OF LINEAR SYSTEMS	23
	5.1 Non-Triviality of Linear Systems	24
	5.2 Mathematical Representation of Linear Systems	25
	5.3 Simultaneous Solution of Linear Systems	27
	5.3.1 Grouping	28
	5.3.2 Solution within the Groups	30
	5.3.3 Solution within the D-Classes	31
	5.3.4 Overall Simultaneous Solution	32
	5.4 Iterative Solution of Linear Systems	33
	5.4.1 Condition for Global Convergence of Linear Systems	33
	5.4.2 Proof of Convergence of Linear Systems when using Direct Substitution	33

		PAGE NUMBER
6	SOLUTION OF NON-LINEAR SYSTEMS	38
6.1	Iterative Solution Schema for Non-linear Systems	38
6.1.1	Conditions for Convergence of a Non-Linear Iterative Scheme	39
6.1.2	Rate of Convergence of Non-Linear Iterative Scheme	39
6.2	F-Differentiability of G(X')	41
6.3	Convergence of the Method of Direct Substitution	41
6.4	Convergence of Newton-like Methods	45
6.5	The Reduced Newton Method	46
6.5.1	Convergence of the Reduced Newton Method	46
6.5.2	Determination of Q for the Reduced Newton Method	47
6.5.3	Grouping Non-Linear Systems into Subplants	48
6.5.4	Effect of Problem Size on the Reduced Newton Method	51
6.6	Comparison of the Reduced Newton Method with Other Convergence Methods	56
7	PLANT TOPOLOGY	59
7.1	Representation of a System of Process Units	59
7.2	Conversion to Numerical Form	61
7.2.1	Relationships Between Streams of Units	61
7.2.2	Relationships Between Streams and Units	62
7.2.3	Relationships Between Unit Number and Unit Type	64
7.2.4	Selection of Numerical Form	64
8	PARTITIONING	66
8.1	Definitions of: Partitioning, Groups, Nodes, Maximal Nets	66
8.2	History of Partitioning Algorithms	66
8.3	Selection of the Partitioning Algorithm	67
9	DECOMPOSITION	69
9.1	History of Decomposition Procedures	69
9.2	Choice of Tearing Criterion	71
9.2.1	Effect of $\rho(\Phi')$ on $\rho(G'(X'))$	72
9.2.2	'No Double Tear' Criterion	74
9.2.3	Implementation of Tearing Criteria	75
9.3	Cycle Finding Algorithms	76
9.3.1	Methods Using the Adjacency Matrix	76
9.3.2	Methods Using Path Searching	77
9.3.3	Selection of Cycle Finding Algorithm	78

PAGE NUMBER

9 DECOMPOSITION (Continued)
9.4 Tearing Algorithms 78
9.4.1 Selection of the Tearing Algorithm 79
9.4.2 Description of the Tearing Algorithm 80

10 SUMMARY OF PART I 83

LIST OF TABLES

1 Summary of Existing Simulators 16
2 Separation Criteria for Ore Dressing Equipment 18
3 Results of Flotation Plant Simulations 53
4 Comparison of Convergence Methods 56
5 Time and Space Bounds of Cycle Finding Algorithms 77

LIST OF FIGURES

1 Flow Diagram of Flotation Plant 52
2 Process Flow Diagram 59
3 Information Flow Diagram 59
4 Digraphs and their Associated Adjacency Matrices 61
5 Arc Operator Matrix (For Signal Flow Graph) 62
6 Stream Vector and its Index 62
7 Incidence Matrix 63
8 Stream Connection Matrix 63
9 Process Matrix 64

LIST OF GRAPHS

1 Effect of Problem Size - Changing Number of Variables 55
2 Effect of Problem Size - Changing Number of Cells 55
3 Comparison of Convergence Methods - Execution Time 57
4 Comparison of Convergence Methods - Number of Iterations 57

Page Number

PART II - PROGRAMMING

11 INTRODUCTION TO PROGRAMMING 89

12 OVERALL CONDITIONS 89

12.1 Programming Language 89

12.2 Structure of the Program 90

12.3 Precompilation of Programs 91

12.4 Information Storage 91

12.5 Information Transmission 92

12.6 Calling Unit Model Subroutines 93

12.7 Changing Stream Variables - 'Converter' 94

13 PROGRAM DESCRIPTION 96

13.1 Phase One - Overall Dimensions 96

13.2 Phase Two - Input 96

13.2.1 System Data 97

13.2.2 Plant Data 97

13.2.3 Run Data 98

13.2.4 Programming 99

13.3 Phase Three - Ordering 99

13.3.1 ALGORITHM 1 - Enumerate all Cycles in each Plant 101

13.3.2 ALGORITHM 2 - Group Cycles into Maximal Nets 103

13.3.3 ALGORITHM 3 - Find Inputs and Outputs for all Nodes 104

13.3.4 ALGORITHM 4 - [illegible] Calculation Order 105

13.3.5 ALGORITHM 5 - [illegible]ver Streams and Cycles in Complex Node 107

13.3.6 ALGORITHM 6 - Find Tear Streams of Complex Nodes 109

13.3.7 ALGORITHM 7 - Find Paths between Tears in each Complex Node 112

13.3.8 ALGORITHM 8 - Set up Vector String of Paths affected by each Stream 114

13.4 Phase Four - Calculation 118

		Page Number
14	USAGE OF SIMULATOR	121
	14.1 Conversion of Process Flowsheet to Schematic Diagram	121
	14.2 Seperation of System into Plants	121
	14.3 Numbering of Units and Streams	121
	14.4 Creating the Input Data File	121
	14.4.1 System Data	122
	14.4.2 Plant Data	125
	14.4.3 Run Data	127
	14.4.4 End the Data File	130
	14.5 Writing of Subroutines	130
	14.6 Features and Limitations of the Converter	131
	14.7 Running the Simulator	133
	14.7.1 Running on O.	133
	14.7.2 Running on WITS	135

LIST OF TABLES

6	System Keywords	97
7	Plant Keywords	97
8	Run Keywords	98
9	Definition of Verbs Used in Logic Flow Diagrams	101
10	Possible States for Combination of Opened Cycles	110
11	Stream Weightings	110
12	Summary of Simulators Files and their Functions for Running on OS	134
13	Summary of Simulators Files and their Functions for Running on WITS	136

LIST OF FIGURES

10	Example Information Flow Diagram	100
11	Stream Connection Matrix Corresponding to Example in Figure 10	100
12	Logic Flow Diagram - ALGORITHM 1	102
13	Logic Flow Diagram - ALGORITHM 2	103
14	Logic Flow Diagram - ALGORITHM 3	104
15	Logic Flow Diagram - ALGORITHM 4	106
16	Logic Flow Diagram - ALGORITHM 5	107
17	Logic Flow Diagram - ALGORITHM 6	109
18	State Diagram Illustrating Application of ALGORITHM 6	111
19	Logic Flow Diagram - ALGORITHM 7	113
20	Logic Flow Diagram - ALGORITHM 8	116
21	Logic Flow Diagram - Calculation Phase	119

Page Number

PART III - UNIT MODELS

15 INTRODUCTION TO UNIT MODELLING 139

16 GENERAL PURPOSE MODELS OF UNITS 141
16.1 Partition (Tromp) Curve Models 141
16.1.1 Background 141
16.1.2 Subroutines TRMP - Partition Curve Library Module 144
16.2 Empirical Models 147
16.3 Subroutine MIXR - Mixer Library Module 148
16.4 Subroutine SPLT - Splitter Library Module 149

17 SPECIFIC MODELS OF UNITS 150
17.1 Modelling of Comminution 150
17.1.1 Time Discontinuous Models 150
17.1.2 Time Continuous Models 152
17.1.3 Subroutine CRSH - Crusher Library Module 155
17.1.4 Subroutine MILL - Milling Library Module 156
17.2 Modelling of Size Separators 157
17.2.1 Screen Modelling 157
17.2.2 Classifier Modelling 159
17.2.3 Cyclone Modelling 160
17.2.4 Subroutine CYCL - Cyclone Library Module 166
17.3 Modelling of Gravity Separators 167
17.4 Modelling of Sorters and Electrostatic and Magnetic Separators 168
17.5 Modelling of Flotation 169
17.5.1 Subroutine FLTN - Flotation Library Module (Sutherland) 175
17.5.2 Subroutine FLTK - Flotation Library Module (King) 176
17.6 Modelling of Solid/Liquid Separators 176

LIST OF TABLES

14 Imperfection Ranges for Several Equipment Types 143
15 Corrected Partition Curve Equations with Codes 147

LIST OF FIGURES

22 Actual Partition Curve 141
23 Corrected Partition Curve 141
24 Partition Curve for Secondary Size Separation 144

		Page Number
PART IV - TEST EXAMPLES		
18	TEST EXAMPLE I	180
	18.1 Flow Diagram	180
	18.2 Data	180
	18.3 Input Data Echo	182
	18.4 Results of ORDERING Subroutine	183
	18.5 Results of CALCULATION Phase	183
19	TEST EXAMPLE II	186
20	TEST EXAMPLE III	188
21	TEST EXAMPLE IV	190
LIST OF TABLES		
16	Results of Simulation - Example IV	191
LIST OF FIGURES		
25	Flow Diagram - Test Example I	181
26	Flow Diagram - Test Example II	186
27	Flow Diagram - Test Example III	188
28	Flow Diagram - Conventional Milling Circuit	190
29	Flow Diagram - Recommended Milling Circuit	190
LIST OF GRAPHS		
5 Percent Passing Versus Size (Test Example IV)		192
BIBLIOGRAPHY		
LIST OF REFERENCES FOR PART I AND PART II		194
LIST OF REFERENCES FOR PART III AND PART IV		206
APPENDIX A - CONVERGENCE OF THE REDUCED NEWTON METHOD		220
APPENDIX B - THE SPECTRAL RADIUS OF A LOWER BLOCK TRIANGULAR MATRIX IS THE MAXIMUM OF THE SPECTRAL RADII OF EACH OF THE DIAGONAL BLOCKS		230
APPENDIX C - WEIGHTING FACTORS		231
COMPUTER DISPLAYS FOR TEST EXAMPLE I		
(i)	INPUT DATA	234
(ii)	OUTPUT FROM INPUT PHASE	239
(iii)	OUTPUT FROM ORDERING PHASE	253
(iv)	OUTPUT FROM CALCULATION PHASE	255

PART I

BACKGROUND AND THEORY

1 INTRODUCTION

As the standard of living increases throughout the world, the demand for minerals would appear to deplete the known supply. To meet this growing [illegible] continuing process of new mineral deposit recovery is required. Deposits must not only be found but they must also be economically processed in order that full use is made of these resources.

Mineral technology is the link in the chain of supply between the finding and winning of an ore and its conversion into a metal or primary raw chemical. The function of a mineral technologist is to devise cheap methods of extracting the useful constituents from available deposits - a task with a wide range of complexity related to, inter alia, the nature and location of the ore and the current world demand for the material (Jones et al 1969).

There has been a constant advance in mineral technology for hundreds of years, due to changes in equipment and ores. Processes that were suitable fifty years ago are now barely adequate or outmoded, and processes considered highly successful today may be replaced by new technology in the future. Basic to every process is an understanding and implied analysis of the system. In the past this has been achieved by careful appraisal of the ore, pilot plant tests and laborious hand calculations.

Some of the principles of ore dressing operations have been understood for many decades but it is only recently that quantitative models have begun to appear in the literature. Coupled with this has been the advent of digital computers. The combination of these two facts makes the simulation of ore dressing plants possible.

1.1 History of Ore Dressing Simulators

As adequate models of comminution units, flotation cells and cyclones are the only ones to have been developed, (see Part III) and these only recently, it is not surprising that the history of ore dressing simulators is brief.

Crushing plant simulators which include screen models have been published by Whiten (1972) and Gorun (1972). The simulation of grinding circuits is quite advanced and the papers by Luckie and Austin (1972) and Austin, Luckie and Wightman (1975) give the basic theory and an example of a cement milling circuit, respectively.

Scrimgeour, Hamilton and Toong (1970) have reviewed and presented models of grinding and flotation circuits. While Davis (1964), Loveday and Merchant (1972), King (1972) and Sutherland (1976) have described flotation models and simulation of these circuits.

Pederson and Gurun (1970) and King (1975) have discussed the techniques of ore dressing plant simulation although not in detail. Overall it appears that, although much effort has been expended in developing mathematical models of ore dressing operations, simulators which treat either milling, or flotation are the only ones available. These simulators have the drawback that they are very limited in the number of possible configurations that can be handled.

Clearly there is a need for a general purpose ore dressing simulator which can utilize existing mathematical models of unit operation and at the same time allow for the development of new ones, by allowing them to be combined in any configuration for purposes of simulation.

As existing ore dressing simulators only allow a limited variation in plant configuration and use solution methods which are specific to that plant it will be useful to examine chemical plant simulators.

1.2 History of Chemical Plant Simulators

Process simulation has been used by chemical engineers since the 1950's but the advances have depended both on the improvement of computer technology and on the development of mathematical models for the various unit processes. The development and acceptance of process simulation by the chemical industry has been a continuous process (see Motard, Shacham and Rosen (1975)).

The period 1955-1959 was characterized by the development of computer programs for individual unit operations. The developer of such programs was required to be an expert in chemical engineering, mathematics and computers, including machine lang[illegible] with the result that the developer was usually the only one who co[illegible]hese programs for solution of actual problems. Successes wer[illegible]ely few, but when there were successes they generated huge savi[illegible]. This encouraged management to fund limited process simulator development projects in the period 1960-1964. The simulators which have survived from this era and still enjoy wide acceptance had the following developmental guidelines:-

(a) Coding in a high level language (e.g. FORTRAN) and highly modular.

(b) Physical property correlations as rigorous and as accurate as possible.

(c) System easy to use with little or no intervention by an expert.

The period 1965-1969 was characterized by the refinement of industrial simulation programs. As simulators were released within various companies for general use, serious problems were often discovered related to their reliability and generality. Many first users became disenchanted with the concept of process simulation and would not attempt an application for several years following a bad experience. However, by the end of this period, the concept was largely established and accepted. The reliability of process simulators had increased as had the speed and reliability of computers.

Since 1970 the simulator has become a well used tool in the process engineers kit.

Comparing the state of simulation in the minerals industry with that in the chemical industry it is apparent that the minerals industry is fifteen to twenty years behind. Fortunately, in the development of an easy to use general purpose ore dressing simulator, it is possible to draw on the experience of the chemical industry thus obviating much time consuming development work.

1.3 Value and Use of a Simulator

At almost every stage in the development of a treatment plant there is a need for simulation programs with different levels of complexity.

(a) Research and Development Stage

A simple simulation which requires the minimum of data may be used for checking the economic feasibility of different processes.

(b) Critical Examination Stage

Once a financially attractive process has been found, different alternatives involving plant size, layout and operating conditions should be tested for optimality.

(c) Pilot Plant Stage

The use of the process simulator may help to obtain estimates of the operating conditions in the full scale plant from relatively few results on the pilot plant.

(d) Design Stage

The process simulator has available all the process data required for the detailed design of various pieces of equipment. In this way safety factors due to uncertainty in equipment design may be reduced.

(e) Simulation of Existing Plants

Simulation of an existing plant may be very useful when there is a need to change the operating conditions. A simulation is of use in finding the best strategy for raising production, to enhance the efficiency of the process, to adapt the plant to a different raw material or to different product composition requirements.

The advantages of using a simulator during the above stages include:-

(a) An engineer need not be expert in mathematics or computers to set up the simulator, before using it as a tool, thus releasing the engineer from routine calculations for more creative work.

(b) By using the building blocks of a simulator the specialized knowledge of an expert on a particular aspect can be tapped by the engineer.

(c) It is easy to use more complex models as the project progresses.

(d) It can help in the understanding of complex processes.

(e) Each trial plant does not become a research project as it is easy to rearrange the plant units and the operating data.

(f) All available data can be explored fully before going on to the next stage of the project.

(g) By testing each piece of equipment over its full range of operating conditions a lot more information is obtained than is possible when testing an integrated pilot plant whose range of operating conditions does not necessarily include the optimum operating point of the full scale plant

(h) It can help in making a decision on how flexible the design should be and what degree of overdesign is required.

(i) No amount of detailed design on individual units can be fully effective unless the interaction between the units in the plant can be taken into account.

(j) The simulator can easily be used in obtaining the plant sensitivity to changes in input and design variables.

(k) The simulator can be used in comparing operating conditions with specifications.

(l) When modifications have to be made to an operating plant it may be impossible to disrupt production for test runs. Instrumentation is expensive and difficult to install. It is difficult to run a meaningful controlled experiment on an operating plant because of process disturbances. Safety may prevent running the plant at off-normal conditions. In complicated processes it is impossible to determine the relationship between variables from experimental data and mathematical data is necessary.

(m) It can be used in the design of a control system.

1.4 Benefits and Cost of a Simulator

A major benefit of process simulation has been that it produces better process designs with lower capital and operating costs in less time than hand methods alone or hand methods in combination with stand alone computer programs. The Monsanto company estimates the saving obtained by using the FLOWTRAN simulation program at twenty-eight man months in one project, and a potential yearly net gain of $750,000 in increased production in another one (Motard et al 1975).

However there are several other intangible, yet highly important, benefits that are perhaps less well understood. These are the transferability of process simulations and an improvement in communications via the use of process simulation. The different engineering groups which deal with the same plant, in different stages, may use the simulator as a basis for faster and improved communications.

Briddel (1974 a,b) summarizes the most severe pitfalls in the use of simulation. Among these are carrying out a simulation for its own sake, not having clear objectives, being overly rigorous, not knowing how much to assume and not working with the people who have the problem. For proper use of simulation, Briddel (1974) proposes to judge the necessity of computer simulation and the degree of sophistication that must be employed against a realistic cost estimate.

To these pitfalls must be added the costs of producing a simulator. The development, documentation and maintenance of a commercial simulation program requires a large investment and a large engineering team (Seider 1972). Thus usually only large companies can afford the initial development of commercial programs. PACER 245 contains about 40,000 source cards. The cost of development of a good computer library is about $15 to $20 per source card. So that the cost of a program like PACER 245 may be $600,000 to $800,000. Reproducing FLOWTRAN might cost twice as much (Motard et al 1975).

2 STRATEGY OF SIMULATION

Process simulation is the representation of a process by a mathematical model which is then solved to obtain information about the performance of the process. The mathematical model is usually a computer program which can be written in different ways depending on its required end use.

2.1 *Simulation Mode versus Design Mode*

Simulators should be able to both simulate existing plants and design new plants. It is characteristic of the simulation/performance/rating mode of calculation that all system inputs and design parameters are specified and the information flow in the program is in the same direction as the heat and material flows in the plant. Ideally, in the design mode of calculation the system inputs and/or design parameters are calculated from specified outputs.

The simulator can only be written to operate in one of these modes. As the *simulation mode* is generally more stable numerically (Motard et al (1975), Rinard and Ripps (1969)), (in other words a solution is more readily obtained in the simulation mode) the usual answer is to perform design calculations by iterated simulation.

There are several advantages in using this *approach*:-

(a) Computer time can become overwhelming in design mode.

(b) Design variables are selected consistently.

(c) In general the engineer knows the feed streams and has good estimates of the equipment parameters.

(d) Information transfer parallels material and energy transfer. This makes it conceptually easy for the engineer to assemble the unit modules which have been written so that physical outputs are calculated from physical inputs.

However fixed design variables *are not always an advantage*. Lee, Christensen and Rudd (1966) have demonstrated that the selection of *design variables* influences the rate of solution considerably. Frequently the selection of design variables in accordance with the simulation approach does not minimize the computation time. This is especially the case when it is possible to avoid iterative solution of a recycle process by choice of another set of design variables.

2.2 Modular Approach versus Equation Solving Approach

Kehat and Shacham (1973 a) classify process simulation programs according to their structure and possible uses. A simulation program may be specific and designed to simulate a particular process with a fixed plant configuration. Such a plant may be solved efficiently using numerical techniques, since simulation may be treated as a mathematical problem with fully defined equations and constraints.

A more widely accepted type of program, however, is one which is constructed using a modular approach. In this approach each process unit is represented as a separate mathematical model called a 'unit module'. The unit modules are connected by data sets which represent the streams of material and energy flowing between units of the plant. An executive program supervises the information flow between the unit modules.

Although the equation solving approach is computationally more efficient the modular approach has been used in almost all simulation programs. There are good reasons for this.

(a) Flexibility

Any portion of the program can be used independently or combined with other modules to represent any kind of flowsheet. In other words just a few building blocks are required and it is possible to assemble processes of any complexity.

(b) Efficient Use of Manpower

Computer costs are dropping while man-hour costs are rising so the simulator must be quick and easy to use. With the modular approach very complicated unit models can be written by experts so saving the engineer the problems involved in analysing a particular process, finding a method of calculation and then programming it. In this way several engineers could each work on different aspects of the same process and then use the simulator to combine their efforts.

(c) Maintenance of the Program

Once a simulator exists it is easy to add new building blocks and remove old ones if the simulator uses the modular approach.

(d) Suitable for Programming

Fortran is structured to use subroutines as building blocks consequently the modular approach is suitable for programming.

Recently attempts have been made to combine the advantages of the *modular and equation solving approach*. *Mah and Rafal* (1971) describe a symbolic compiler which accepts data in the form of mathematical equations and produces automatically a working digital computer program. In this program three languages, PL/I, FORTRAN and FORMAC are used.

Solymar and Sieder (1973) tried to make this approach more practical by considering the structure of equations and sorting degrees of non-linearity (this is done with a heuristic algorithm). The order in which the equations are to be solved, and the variable for which each equation is to be solved (the output set), is found using Steward's (1965) algorithm.

2.3 Selection of Strategy

Based on the previous discussion it was decided to write the simulator in the simulation mode and to structure it using the modular approach.

3 SOLUTION TECHNIQUES

In the case of simple processes (those with no recycles), process simulation poses no problem, except perhaps those associated with the modelling of the unit modules. In this case starting with the feed streams, it is possible to calculate the outputs from the first unit, which in turn become the feeds to the next units and so on until the last unit. However most processes are not of this type and have recycle streams. It then becomes impossible to proceed with the calculation as above, because no unit in the recycle loop will have all its inputs known.

There are two basically different approaches to the computation of recycle processes: simultaneous and sequential.

3.1 Simultaneous Solution of Recycle Systems

In the simultaneous approach the units are linearized around an assumed steady state and the resulting equations solved simultaneously. This gives new values to the process stream variables which are in turn used as the basis for a revised linearization. Such a process is repeated until the values at the end of each successive iteration changes by less than a predetermined number, or as it is usually called, tolerance limit. The calculation is said to have converged in such a case. On the other hand it is quite possible for the iterations to diverge, i.e. the values after successive iterations change significantly and show no sign of approaching a limiting value.

One of the first expositions of this approach is by Naglov (1957). He introduced the concept of 'split fractions' to linearize the process units. The 'split fraction' is the mass fraction of a feed component to a unit which leaves the unit in each of the output streams. The mass and energy balances are then written in terms of split fractions for each process unit. Such equations are linear and can thus be solved simultaneously.

Vola (1961) uses a similar approach on systems in which the units are represented by very simple input/output models. Rosen (1962) describes the general mass balance problem as one of solving simultaneously a large set of non-linear equations and suggests an approach based on that of Naglov (1957). Rosen is concerned with the case where the split fractions are complex functions of the process conditions and of the quantity and nature of the other components, e.g. non-conservative units like reactors and units which involve more than one phase. Rosen's method is a systematic way of setting up the interlinking equations between

units and can be described as follows:-

(a) Guess split fractions for each unit.

(b) Solve the set of simultaneous linear equations for flows.

(c) Using these calculated flows to each unit determine the actual split fractions for this duty from the unit model.

(d) Compare the previous iterations split fractions with the current ones and then predict a new set.

(e) If the change in split fractions in successive iterations is greater than the tolerance limit return to step (b).

Naphtali (1964) presented a method of solution that is in essence the simultaneous solution of the heat and material balance equations, but the number of equations that have to be solved simultaneously are reduced by inspection. Ravier and Norman (1964) also reduce the number of equations that have to be solved simultaneously but their method is automatic.

Both Kometani (1963) and Nishimura (1960) recommend the use of linear models and show that by careful selection of design variables many types of unit operations can be linearized. They have described linear models of flash separators, absorbers and strippers, distillation columns, heat exchangers and reactors. An example of a linearized reactor is one in which a first order reaction occurs and the residence time is given as a parameter (because the extent of reaction is independent of the feed).

More recently Hutchison and Leckey (1973) and Hutchison (1974) have described the advantages of having linear unit models, particularly for the early stages of design. They have produced a general purpose flowsheeting package called SYFOOL which solves linear systems.

3.2 Sequential Solution of Recycle Systems

In the sequential approach the values of variables in certain recycle streams, referred to as torn streams, are assumed. Stream variables are the mass flows of each component in that stream and are discussed in detail in the next chapter. The set of torn streams must render the system acyclic so that the values of stream variables in all the other streams can be calculated serially. Having calculated a new set of torn variables (variables in the torn streams) these can be compared with

the assumed values and a new estimate made for the next iteration. This procedure is repeated until the successive corrections are less than the tolerance limit.

If the recycle streams were unique, there would be only one possible set of torn streams, however structural properties of flowsheets can allow several different sets, therefore the problem of selection of a set of tear streams arises. The set of torn streams is called a decomposition. In general different decompositions exhibit different convergence properties. For a particular system using a particular convergence method the 'best' decomposition will be the one that requires the least number of iterations for convergence from given starting values.

3.2.1 Direct Substitution

The simplest method of correcting the torn variables is direct substitution. In this method the newly calculated torn variables are substituted as the new starting estimate of torn variables for the next iteration.

If X is the input vector of torn variables and Y the output vector of torn variables then:-

$$Y^k = \phi(X^k) \qquad k \text{ is the iteration index}$$

and direct substitution is:-

$$X^{k+1} = Y^k \qquad \text{(3.1)}$$

The convergence or divergence and rate of convergence of this method is linked to the maximum eigenvalue of the matrix of partial derivatives for the acyclic system, $\frac{\partial Y}{\partial X}$, at the solution. The matrix $\frac{\partial Y}{\partial X}$ is called the Jacobian of the acyclic system. If y_i are the elements of vector Y and x_j the elements of vector X then $\frac{\partial Y}{\partial X}$ is the two dimensional matrix $\left(\frac{\partial y_i}{\partial x_j}\right)$. The modulus of the maximum eigenvalue of a matrix Z is called the spectral radius and symbolized $\rho(Z)$. Details of the link are discussed in chapter 5.

3.2.2 Accelerated Direct Substitution

Although direct substitution is simple to apply it can be very slow to converge. Kehat and Shacham (1973 c) have reviewed ways of accelerating convergence. There appear to be only two which are currently in use: the bounded Wegstein (Wegstein (1958), modified by Kliesch in a PhD thesis (1967)) and the dominant eigenvalue method (Orbach and Crowe (1971)).

In both these methods it is assumed that the guessed, x_j^i, and the calculated value, y_j^i, (where i is the iteration index) are related by:-

$$y_j^i = a_j.x_j^i + b_j$$

and $$y_j^{i-1} = a_j.x_j^{i-1} + b_j$$

So that $$a_j = \frac{y_j^i - y_j^{i-1}}{x_j^i - x_j^{i-1}} \quad \text{and} \quad b_j = y_j^i - a_j.x_j^i$$

and then x_j^{i+1} is estimated from:-

$$x_j^{i+1} = \frac{b_j}{1 - a_j} = (1-q_j).y_j^i + q_j.x_j^i \qquad \text{---------- (3.2)}$$

In the Wegstein method $q_j = \frac{a_j}{a_j - 1}$ and in the bounded Wegstein method ($q_j - 1$) has the limits (-10.0,0). (Worley and Motard (1972)).

In the dominant eigenvalue method $q_j = \frac{\lambda}{\lambda - 1}$ for all j

where $\lambda = ||Y^i - Y^{i-1}||/||X^i - X^{i-1}||$

and $||Z||$ is the l_2 norm of vector Z and equals $(Z.Z^T)^{\frac{1}{2}}$

Shacham and Motard (1974) have demonstrated that there is an optimal value for the acceleration parameter but there is as yet no convenient method for computing this optimal value. They also claim that in the Wegstein method the acceleration parameter will be the same for all variables after a few direct substitution iterations so that the two methods become the same. (An exception to this is when the system Jacobian is diagonal).

Recently Crowe and Nishio (1975) have proposed a method called the 'General Dominant Eigenvalue Method'. This method is an extension of the previous one and offers flexibility of use and more effective acceleration steps according to the authors. This method does not appear to have been used on a full scale simulator.

The Wegstein and dominant eigenvalue methods can become unstable and the use of the bounded Wegstein method is one way of overcoming this. Orbach and Crowe (1971) suggest multiplying the acceleration parameter

In both these methods it is assumed that the guessed, x_j^i, and the calculated value, y_j^i, (where i is the iteration index) are related by:-

$$y_j^i = a_j . x_j^i + b_j$$

and $$y_j^{i-1} = a_j . x_j^{i-1} + b_j$$

So that $$a_j = \frac{y_j^i - y_j^{i-1}}{x_j^i - x_j^{i-1}}$$ and $$b_j = y_j^i - a_j . x_j^i$$

and then x_j^{i+1} is estimated from:-

$$x_j^{i+1} = \frac{b_j}{1 - a_j} = (1-q_j).y_j^i + q_j.x_j^i \qquad \text{---------- (3.2)}$$

In the Wegstein method $q_j = \frac{a_j}{a_j - 1}$ and in the bounded Wegstein method (q_j - 1) has the limits (-10.0.0). (Worley and Motard (1972)).

In the dominant eigenvalue method $q_j = \frac{\lambda}{\lambda - 1}$ for all j

where $\lambda = ||Y^i - Y^{i-1}||/||X^i - X^{i-1}||$

and $||Z||$ is the l_2 norm of vector Z and equals $(Z.Z^T)^{\frac{1}{2}}$

Shacham and Motard (1974) have demonstrated that there is an optimal value for the acceleration parameter but there is as yet no convenient method for computing this optimal value. They also claim that in the Wegstein method the acceleration parameter will be the same for all variables after a few direct substitution iterations so that the two methods become the same. (An exception to this is when the system Jacobian is diagonal).

Recently Crowe and Nishio (1975) have proposed a method called the 'General Dominant Eigenvalue Method'. This method is an extension of the previous one and offers flexibility of use and more effective acceleration steps according to the authors. This method does not appear to have been used on a full scale simulator.

The Wegstein and dominant eigenvalue methods can become unstable and the use of the bounded Wegstein method is one way of overcoming this. Orbach and Crowe (1971) suggest multiplying the acceleration parameter

by a relaxtion factor between zero and one.

A reason for the instability is the fact that the interaction between term variables are not considered, however this can be done by Newton-like methods.

3.2.3 Newton and Quasi-Newton

The Newton method is given by:-

$$X^{k+1} = X^k - F'(X^k)^{-1}.F(X^k) \qquad \text{---------- (3.3)}$$

where $F(X) = X - \phi(X)$ and k is the iteration index

and $F'(X)$ the matrix of partial derivatives of $F(X)$ w.r.t. X. In a real system $F'(X)$ will not initially be available so that it must be approximated by $J(X)$. $J(X)$ is estimated using secants in each direction and this is called the secant method. If X has n elements then the general secant method requires n+1 evaluations of $F(X)$ at each stage to determine $J(X)$. However if $k>n$ then the previous n evaluations of $F(X)$ plus the current one can be used to determine a new $J(X)$ for each new evaluation of $F(X)$. This is called a (n+1)-point sequential secant method.

Further computational savings can be made by estimating $J(X)^{-1}$ at every stage instead of $J(X)$. This saves having to invert the nxn matrix at every stage, ie:-

$$J(X^{k+1})^{-1} = J(X^k)^{-1} - \frac{J(X^k)^{-1}.F(X^{k+1}).(V^k)^T.J(X^k)^{-1}}{(V^k)^T.J(X^k)^{-1}.p^k}$$

where $p^k = F(X^{k+1}) - F(X^k)$ and V^k can be arbitrarily selected.

(This equation is derived using the Sherman-Morrison formula in Ortega and Rheinboldt (1970) p 207)).

Barnes (1965) chose $(V^k)^T$ in a direction orthogonal to n-1 previous iterations steps, i.e. $(V^k)^T.(X^{j+1} - X^j) = 0$ for n-1 selections of j from $0 \le j \le k-1$. Both Barnes (1965) and Ortega and Rheinboldt (1970 p 208) show that this must reduce to the (n+1)-point sequential secant method after at most n steps, *provided* the n previous directions are linearly independent. This means that the only difference is that for Barnes method $J(X^0)^{-1}$ can be chosen arbitrarily, while for the former method $J(X^0)$ is determined by the secants about X^0, and then inverted.

The most popular choice of (V^k) for chemical engineers is that of Broyden. He chose $V^k = X^{k+1} - X^k$. Rosen (1960) has compared these two methods and concluded that both show considerable promise. He suggests that by averaging V^k, component by component, from both methods even better results are obtained.

Broyden (1965) finds the first approximation $J(X^0)^{-1}$ by numerical differentiation and inversion. Another possibility is to start with the identity matrix and Rosen (1968) states that this gives efficient results in mass balance calculations.

Although a distinction has been made between simultaneous and sequential approaches, it is clear that the secant type methods involve local linearization that sets up a set of simultaneous linear equations about the current point and these are solved for the next point. So that the Newton and quasi-Newton methods can be considered as a combined simultaneous-sequential technique.

3.3 Discussion of Solution Techniques

Table 1 is extracted from Flower and Whitehead (1973). The abreviations used for column headings are summarized below.

U = Number of Units
S = Number of Streams
C = Number of Components
PM = Process Matrix
SCM = Stream Connection Matrix

R = Recoding Facility
G = Grouping Facility
C = Checking of user input for feasibility

DS = Direct Substitution
BW = Bounded Wegstein
L = Linearization
NR = Newton Raphson
S = Secant
US = User Supplied

From table 1 it can be seen that all the simulation packages offer direct substitution and most offer the bounded Wegstein as methods of convergence. This is because of their simplicity.

Kesler and Griffiths (1963) were the first to publish results on the practical application of this method and concluded from their examples that in certain circumstances direct substitution converges in fewer iterations than the method of Naglov and Rosen. They further expect that direct substitution will always converge from any starting point under the conditions considered in physical systems. Other like Naphtali (1964), Rinard and Ripps (1965) and Shacham and Motard (1974) also claim that direct substitution will always converge for any physically realizable system.

TABLE 1 SUMMARY OF EXISTING SIMULATORS

NAME	SOURCE	MAXIMUM SIZE				PLANT TOPOLOGY	ORDERING	CONVERGENCE METHOD					
		U	S	C	SxC			DG	BW	L	NR	S	US
PACER	Purdue University	20	30	10		PM	Simple R	X					
GEMCS	McMaster University	25	50	25		PM	No	X					
SPEED-UP	Imperial College					SCM	Comprehensive R,G,C						
CHESS	Houston University	100	50	20		PM	No	X	X				
NETWORK67	ICI	50			500	SCM	Simple G	X			X		
FLOWPACK	ICI	100		25	2000	SCM	Comprehensive R,G	X	X			X	
CONCEPT	Cambridge University	50	150	20		SCM	Comprehensive	X		X			
PACER245	Digital Systems Corporation					PM	Comprehensive						X
PROVE	Diamond Shamrock	20	30	20		PM	No	X					
GFS	Kellogg			48		PM	No						
FLOWTRAN	Monsanto					PM	No	X	X				

Cavett (1963) has demonstrated that Wegstein's method is usually faster converging than direct substitution and this is supported by Batstone and Prince (1970) who find that Wegstein is competitive with direct substitution in every application.

The Newton and quasi-Newton methods can be very efficient for the solution of recycle systems (Genna and Motard (1975), Batstone and Prince (1970)) but may sometimes diverge. Kohet and Shacham (1973 c) claim that the classical Newton method will converge in a finite number of iterations if the initial guess of the values of vector X^0 is sufficiently close to the solution and $F'(X)$ at the solution is non-singular. The limitations of these methods are:-

(a) No convergence if the initial guess of X^0 is poor

(b) No convergence if $F'(X)$ is *singular or near* singular at the solution

(c) The excessive computation effort required to compute and invert $F'(X)$.

Various techniques are used to try and get around these difficulties and are discussed by Kahet and Shacham (1973 c). However for all Newton and *quasi-Newton methods* the main drawback is the large storage requirements for matrix $F'(X)$ or $F'(X)^{-1}$. Much effort is needed to determine $F'(X)$ so that these methods involve far more overhead computation than *direct substitution* or *accelerated* direct substitution.

4 DIFFERENCES BETWEEN ORE DRESSING AND CHEMICAL PLANT SIMULATORS

4.1 Representation of Streams

In a chemical plant the process streams are generally liquid and gas and can be represented in terms of total mass flow plus the concentration of all chemical species in each phase. However in ore dressing plants the process streams contain solids and liquid. The liquid is usually water and dissolved chemicals are not considered. The solids are particles with distributions in size and mineral content. This fact gives each particle a unique set of physical properties.

In any physical separation or transformation unit each particle will behave according to its own physical properties, and those of the particles surrounding it. It is impossible to create a simulator based on the behaviour of each individual particle because of their large number, but it is this that allows the possibility of grouping together particles whose properties fall within certain limits. This is a discretized distribution, but if the limits are infinitely narrow then the distribution will be continuous and it may be possible to represent it by a mathematical function. A mathematical function can describe the complete distribution by means of a few parameters, however an accurate representation is difficult if not impossible to achieve. For example a size distribution will often be log-normal over most of the size range but deviate from this at larger sizes. Even in the case where the system feed stream can be adequately represented by a simple functional form it is unlikely that recycle or product streams will retain the same functional form (particles are selectively removed from certain sections of the distribution causing severe kinks). In any event it has been customary in the mineral processing industry to use discretized distributions, like size distributions, because this facilitates measurement. Consequently most of the ore dressing models developed to date are based on discretized rather than continuous distributions, and this simulator will follow suit. The three main advantages of this approach are summarized below -

(a) Most existing unit models are formulated in this manner

(b) Data measurement can generally only be obtained for discrete groups of particles

(c) The mass balance can be exact as selective removal of certain categories of particles will not cause a deterioration in the accuracy of representation of a stream

Having decided to use discretized distributions the choice of discrete categories is determined largely by the unit model's requirements.

TABLE 2 SEPARATION CRITERIA FOR ORE DRESSING EQUIPMENT

EQUIPMENT TYPE	PHYSICAL PROPERTY USED AS SEPARATION CRITERION						
	SIZE	GRADE			SURFACE		
		S.G.	MAG	FRI	FLT	RFL	CDT
Dense Media	T	P					
Gravity	S	P					
Magnetic	T		P				
Electrostatic	T						P
Sorter	T					P	
Screen	P	T					
Cyclone	P(S)	S(P)					
Classifier	P	S					
Thickener	P	T					
Filter	P						
Comminution	S			P			
Conditioner					P		
Flotation	S	T			P		

S.G. - specific gravity MAG - magnetic susceptibility
FLT - floatability FRI - friability
RFL - reflectivity CDT - conductivity

The P, S and T represent the Primary, Secondary and Tertiary separation criteria for the particular equipment type.

Table 2 is a comprehensive list of ore dressing equipment and the physical properties which are exploited by them. From this table it

can be seen that the physical properties are split into three fundamental classes:-

(a) Particle size

(b) Grade - Mass or volume fraction of each mineral species which is a component of the particle

(c) Surface - The surface area occupied by each mineral component of the particle

There is a fourth fundamental class. This is the shape of the particle. However there is no convenient way of describing particle shape and none of the existing models use shape so it will not be considered specifically.

The surface area distribution of components in a large unbiased sample will equal the volume fraction distribution. But if the separation is being done on the basis of a surface property then the sample will become biased in the product streams and the direct connection between grade and surface area will no longer apply. Therefore in a separation of this kind an independent surface distribution must be used.

The size and grade category of a particle must be given if it is to be completely specified. The surface area distribution is also necessary if a surface property is to be used for separation. If friability, or another property not directly related to size, grade or surface is to be used, this property must also be expressed independently. In terms of discretized distributions this means that a n-dimensional grid of variables (where n is the number of independent properties) must be used to adequately describe the solid particles in the stream. This can be very expensive in terms of computer storage and computation time. For example if each of the continuous distributions is broken into five discrete classes and there are four independent properties required then the number of variables per stream will be $5^4 = 625$.

However most plants will be of a particular type and therefore require only size, grade and one other distribution to describe the stream. In this simulator a three-dimensional grid will be allowed, catering for D-classes, G-classes and S-classes. The D and G-classes refer to size and grade distributions. The S-classes can be any third independent physical property depending on the situation. The value carried as the variable in the DGS-class is the actual mass of solids

in that category.

When reference is made to 'similar variables' in different streams, this means these variables belong to the same D, G and S-classes in the different streams.

4.2 Transformation Units

In chemical plants chemical species are transformed to other chemical species by reactions in reactors. In ore dressing plants there are no reactors but transformation from one stream variable to another does occur.

Definition: A unit operation which transforms particles in the input stream belonging to a certain DGS-class to another DGS-class in the output stream is called a 'transformation unit'.

In terms of this definition there are only two ore dressing units which can be classed as transformation units; comminution units and conditioners.

The purpose of a comminution unit is to break particles into smaller ones and this clearly transforms the particles from one D-class to another.

A conditioner may or may not be a transformation unit depending on the way in which it is used. In a flotation circuit the S-classes can correspond to the distribution of the flotation rate constant (k) for each DG-class. If the conditioner is used to modify the feed variables by creating the k-classes then it is not acting as a transformation unit but simply expanding the DG-classes into their component k-classes. If on the other hand the conditioner is contained within a recycle loop then changing of mass from one k-class to another faster or slower floating k-class is a transformation and the conditioner will be acting as a transformation unit.

It will be seen in the next chapter that the fewer the number of transformations, within a recycle loop, the faster is the convergence. Therefore it is preferable to use the conditioner as a 'non-transformation' unit where possible.

4.3. Physical Properties

The success or failure of a chemical plant simulator often depends on the accuracy and size of its physical property package. A large portion of this package is connected to the variation of properties with temperature and pressure. Ore dressing plants usually operate at ambient conditions and if there is any variation in temperature or pressure the effect will at worst be slight.

Another advantage of ore dressing plants in this respect is that relatively few physical properties are used (see table 2), and when these few properties are known for a pure mineral species many can be related to mixtures of these mineral very simply e.g. specific gravity. On the other hand properties like flotability are distributed even for pure minerals and predicting the distribution for mixed particles is difficult. Therefore in this case laboratory test work must be done for the particular ore and a physical property package will be of little use.

In an ore dressing simulator there appear to be two kinds of physical properties, those which are readily available and those which must be determined for the specific ore. In both these cases a comprehensive physical property package is not of much value, and as such none will be developed for this simulator. However it is not impossible for good methods of correlating data and predicting properties of mixed particles to evolve in the future. The simulator is written in such a way that interfacing a physical property package can be readily achieved.

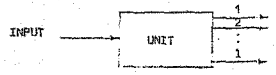
5 SOLUTION OF LINEAR SYSTEMS

Definition: A *non negative* matrix, for purposes of this thesis, is one which has all its elements greater than or equal to zero.

Definition: A *linear model* of a unit is one in which the transformation matrix from input to output stream is constant, i.e. independent of the input stream.

Definition: A *linear system* is one in which the models of the systems unit operations are linear.

For example if a unit has one input stream and l output streams each containing n variables



then $W_j = T_j . V \qquad j = 1,l$

where W_j is the vector of variables in output stream j (n elements)

V is the vector of input variables (n elements)

and T_j the fixed (nxn) transformation matrix for each j.

For a linear model the transformation matrices are fixed, and as the stream variables are mass flows, element i,k of T_j is effectively the mass fraction of input variable k which reports as variable i in output stream j.

For a separator the transformation matrices will be diagonal but for a comminution unit it will be lower triangular as particles are only transformed from larger to smaller sizes. It should be noted that the sum of the sums of the elements in the same columns of all the transformation matrices of a particular unit must be unity for every set of columns. Also all the elements of the transformation matrices are zero or mass fractions which must be positive. The transformation matrices therefore are non-negative.

5.1 Non-Triviality of Linear Systems

At first glance a linear model may seem to be a poor reflection of any real system. However as discussed in chapter 3.1 a linear approach has been used by several authors and recommended by Kometsu (1968), Nishimura (1968), Hutchison and Leesley (1973) and Sheoham and Motard (1975) for chemical plants. Briddel (1974) even warns that being overly rigorous is one of the pitfalls of simulation.

The units of ore dressing plants can be split into three types; mixers, transformation units and separation units.

Mixers simply sum similar variables in the input streams to give the output stream. From the definition of a linear model it is clear that a mixer is a linear model and that the transformation matrix from each input stream to the output stream is the same diagonal matrix which has all the diagonal elements equal to unity.

Transformation units have been discussed in sub-chapter 4.2 and although they can be very complicated the most commonly used comminution model is based on fixed selection and breakage functions (Austin (1971)) and in this formulation, if the mill residence time is set as a parameter the transformation matrix, is constant and the model linear. Conditioner models are very undeveloped and the only one that has been found is by Scrimgeour et al (1970). This model calculates the flotation rate constant k for a pure mineral from an expression which depends only on reagent addition. The flotation rate constants for mixed particles are obtained by summing the flotation rate constants of the components pure minerals in proportion to their mass fractions. If reagent addition is set as a parameter to this model then it becomes linear. Scrimgeour et al (1970) used this conditioner component externally to any recycle loop therefore it can have no influence on the convergence of the loop. The general purpose simulator is structured to allow the mass flow in each k-class to be carried as a stream variable and this implies that, with Scrimgeour et al's (1970) model, no transformation will take place when the conditioner is internal to the recycle loop, (as each k-class has its own flotation rate constant which is not changed as long as the reagent addition is fixed). Other flotation circuit simulators like King (1972) and Sutherland (1976) do not use conditioners at all and simply rely on the k-classes directly.

Separation units split the input streams into two or more output streams. The mass flow in the various particle categories can interact with each other but are chosen to minimize this interaction. Therefore it is expected that these interactions will be small. This is demonstrated by the fact that many separation units are modelled using a Tromp curve.

If the Tromp curve for the unit is independent of the input stream then the transformation matrices for this unit, which are diagonal, are constant and hence the model is linear.

Therefore almost all types of ore dressing operations can be represented by linear models and hence many non-trivial ore dressing plants can be simulated as linear systems. This is particularly true when the range of operation of the equipment is small.

5.2 Mathematical Representation of Linear Systems

If in a linear system, the units are simply sequential, then there is no need for a convergence algorithm. When recycle streams do exist the system is essentially a set of simultaneous linear equations. It may be possible to solve these equations simultaneously (see sub-chapter 3.1) for very small systems but even in the medium size plant, using 20 streams with 20 variables in each, this implies solving 400 simultaneous equations.

A more usual method of setting up and solving such a system (Shacham and Motard (1975)) is:-

(a) Decomposition to an acyclic system (see chapter 9)

(b) Estimating a set of starting values for the torn variables

(c) Iterative refinement of the assumed values, by calculating the outputs from each unit when the inputs to this unit are available.

This sequential solution technique is discussed in sub-chapter 3.2 and implies that instead of trying to solve for all the stream variables in all the streams, simultaneously the variables in the torn streams are found and then the variables in all the other streams (including product streams) are generated by sequential calculation.

To discover whether this method of solution, called direct substitution (see sub-chapter 3.2.1) will converge for linear systems, the system must be defined in mathematical terms.

To every tear stream there are paths from some or all of the other tear streams, itself and the feed streams along which particles in all categories flow. The mass fraction of particles in each category reaching a tear stream along a single path from some stream is given by the product of transformation matrices along this path. There may be several paths between the same two streams so that the total mass fraction of particles reaching the second stream from the first is the sum of the product of transformation matrices along each of these paths.

Starting with a set of feed variables and assumed values for the torn variables it is possible, using the sums of products of transformation matrices from feed and tear streams to every tear stream, to calculate the mass flows in the calculated torn variables, as follows:-

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix} = \begin{bmatrix} B_{11} B_{12} & \cdots & B_{1m} \\ \vdots & & \vdots \\ B_{m1} & \cdots & B_{mm} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix} + \begin{bmatrix} A_{11} & \cdots & A_{1k} \\ \vdots & & \vdots \\ A_{m1} & \cdots & A_{mk} \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_k \end{bmatrix} \qquad \text{------- (5.1a)}$$

when the system has m tear streams, k feed streams and n variables per stream

where X_i is the vector of guessed values for tear stream i (n elements)

Y_i is the vector of calculated values for tear stream i (n elements)

F_i is the vector of variables in feed stream i (n elements)

B_{ij} is the sum of products of transformation matrices from tear stream j to tear stream i, $B_{ij} \varepsilon R^{nxn}$

and A_{ij} is the sum of products of transformation matrices from feed stream j to tear stream i, $A_{ij} \varepsilon R^{nxn}$

Similarly for product streams:-

$$\begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_p \end{bmatrix} = \begin{bmatrix} BB_{11} & \cdots & BB_{1m} \\ \vdots & & \vdots \\ BB_{p1} & \cdots & BB_{pm} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix} + \begin{bmatrix} AA_{11} & \cdots & AA_{1k} \\ \vdots & & \vdots \\ AA_{p1} & \cdots & AA_{pk} \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_k \end{bmatrix} \qquad \text{------ (5.1b)}$$

when the system has p product streams

where P_i is the vector of variables in product stream i (n elements)

BB_{ij} is the sum of products of transformation matrices from tear stream j to product stream i, $BB_{ij} \varepsilon R^{nxn}$

and AA_{ij} is the sum of products of transformation matrices from feed stream j to product stream i, $AA_{ij} \varepsilon R^{nxn}$

Equations 5.1a and 5.1b can also be written as:-

$$Y = B.X + A.F \qquad \text{---------- (5.2a)}$$

and $$P = BB.X + AA.F \qquad \text{---------- (5.2b)}$$

As the matrices A, B, AA and BB are constructed from the sums of products of constant non-negative matrices (T_j) they must also be constant and non-negative.

Equation 5.2 can be considered to be the model of an acyclic linear system and 5.2a must be solved so that $Y = X = X^*$

5.3 Simultaneous Solution of Linear Systems

It is possible to solve equation 5.2a for X^* simultaneously by substituting $Y=X=X^*$ to give:-

$$X^* = B.X^* + A.F \qquad \text{---------- (5.3)}$$

$$\therefore\ X^* = (I-B)^{-1}.A.F \qquad \text{---------- (5.4)}$$

The existance of the fixed point X^* is assured by the Neumann Lemma, as $B \geq 0$ and $\rho(B)<1$ from equation 5.2b ($B \geq 0$ means each element of matrix B is non-negative), (Ortega and Rheinboldt (1970) p45).

Obtaining X^* using equation 5.4 can be difficult for two reasons. Firstly the elements of B and A are not readily available and secondly the inversion of (I-B) can be expensive if the dimensions of B are large (due to many variables per stream or many tear streams or both). Nevertheless this approach is attractive as an exact solution can be obtained immediately.

The magnitude of the problem has already been reduced by solving simultaneously only for torn variables and then generating the remaining variables sequentially, as opposed to solving for all the variables in all the streams simultaneously. Consequently a matrix of size $(n.m \times n.m)$ has to be inverted instead of one of size $(n_s.n \times n_s.n)$, (where n_s is the number of streams in the system, n the number of variables per stream and m the number of tear streams). In the following sub-chapters it will be shown how, by taking advantage of the nature of ore dressing plants, the problem can be reduced to very manageable proportions.

5.3.1 Grouping

Many plants can be divided into sub-plants in such a way that there is only a feed forward of mass from one sub-plant to the next. Such sub-plants will be called 'groups'.

Consider a plant consisting of N groups which are numbered so that there is mass flow only from smaller numbered groups to larger numbered groups. B is the sum of products of transformation matrices between tears in group i and I_{ij} the sum of products of transformation matrices from tears in group j to tears in group i, then B can be arranged by row and column interchanges so that:-

$$B = \begin{bmatrix} B_{G1} & I_{12} & \cdots & I_{1N} \\ I_{21} & B_{G2} & & \\ \vdots & & \ddots & \\ I_{N1} & \cdots & & B_{GN} \end{bmatrix}$$

If m_{Gi} is the number of tear streams in group i and n_{Gi} the number of variables per stream in group i then B_{Gi} is a $(n_{Gi} \cdot m_{Gi} \times n_{Gi} \cdot m_{Gi})$ matrix, and I_{ij} is a $(n_{Gi} \cdot m_{Gi} \times n_{Gj} \cdot m_{Gj})$ matrix, (rows x columns)

By definition there is no mass flow from group j to group i if j>i, therefore $I_{ij}=0$ for j>i. Therefore:-

$$B = \begin{bmatrix} B_{G1} & \cdots & 0 \\ I_{21} & B_{G2} & \\ \vdots & \ddots & \\ I_{N1} & \cdots & B_{GN} \end{bmatrix}$$

and by similarly rearranging equation 5.1a it can be written as:-

$$\begin{bmatrix} Y_{G1} \\ Y_{G2} \\ \vdots \\ Y_{GN} \end{bmatrix} = \begin{bmatrix} B_{G1} & 0 & \cdots & 0 \\ I_{21} & B_{G2} & & \\ \vdots & & \ddots & \\ I_{N1} & \cdots & & B_{GN} \end{bmatrix} \begin{bmatrix} X_{G1} \\ X_{G2} \\ \vdots \\ X_{GN} \end{bmatrix} + \begin{bmatrix} e_{G1} \\ e_{G2} \\ \vdots \\ e_{GN} \end{bmatrix} \qquad \text{---------- (5.5)}$$

where Y_{Gi} and X_{Gi} are the vectors of calculated and estimated torn variables for group i, and the a_{Gi} are the corresponding vectors of the rearranged A.F for group i.

By equation 5.5:-

$$
\begin{aligned}
Y_{G1} &= B_{G1}X_{G1} + a_{G1} \\
Y_{G2} &= I_{21}X_{G2} + B_{G2}X_{G2} + a_{G2} \\
&\vdots \\
Y_{Gi} &= \sum_{j=1}^{i-1} I_{ij}X_{Gj} + B_{Gi}X_{Gi} + a_{Gi} \qquad \text{---------- (5.7)} \\
&\vdots \\
Y_{GN} &= \sum_{j=1}^{N-1} I_{Nj}X_{Gj} + B_{GN}X_{GN} + a_{GN}
\end{aligned}
$$

therefore when $Y_{Gi} = X_{Gi} = X^{*}_{Gi}$

then

$$
\begin{aligned}
X^{*}_{G1} &= (I-B_{G1})^{-1}.a_{G1} \\
X^{*}_{G2} &= (I-B_{G2})^{-1}.(I_{21}X^{*}_{G1} + a_{G2}) \\
&\vdots \\
X^{*}_{Gi} &= (I-B_{Gi})^{-1}.\left(\sum_{j=1}^{i-1} I_{ij}X^{*}_{Gj} + a_{Gi}\right) \qquad \text{---------- (5.8)} \\
&\vdots \\
X^{*}_{GN} &= (I-B_{GN})^{-1}.\left(\sum_{j=1}^{N-1} I_{Nj}X^{*}_{Gj} + a_{GN}\right)
\end{aligned}
$$

Now

$$\sum_{j=1}^{i-1} I_{ij}X_{Gj} + a_{Gi} = Y_{Gi} - B_{Gi}X_{Gi}$$ by equation 5.7

therefore if B_{Gi} is *determined directly* during an iteration of group i for some X_{Gi} and Y_{Gi} then $\sum_{j=1}^{i-1} I_{ij}X_{Gj} + a_{Gi}$ can be calculated. Further if $X^{*}_{Gj}, j=1,i-1$ is known then the term $\sum_{j=1}^{i-1} I_{ij}X^{*}_{Gj} + a_{Gi}$ will be known and hence it is possible to calculate $X^{*}_{G1}, X^{*}_{G2} \ldots X^{*}_{GN}$ sequentially until the whole system is solved in N stages.

5.3.2 Solution within the Groups

It has been shown in sub-chapter 5.3.1 that it is possible to consider each group independently as long as this is done in sequence. In this sub-chapter the solution of one particular group will be treated.

Let the matrix containing the sum of products of transformation matrices for this group be B_G. Then if this group has M, D-classes and B_G is rearranged by row and column interchanges so that variables of the same D-class are collected together and stacked in decreasing size order from 1 to M then:-

$$B_G = \begin{bmatrix} B_{D1} & C_{12} & \cdots & C_{1M} \\ C_{21} & B_{D2} & & \vdots \\ \vdots & & \ddots & \\ C_{M1} & \cdots & & B_{DM} \end{bmatrix} \qquad B_G \text{ is a } [n_G.m_G \times n_G.m_G] \text{ matrix}$$

where n_G is the number of variables per stream in this group

n_D is the number of variables in each D-class

m_G is the number of torn streams in this group.

B_{Di} is the $(m_G.n_D \times m_G.n_D)$ matrix containing sum of products of transformation matrices for variables belonging to D-class i and C_{ij} are the $(m_G.n_D \times m_G.n_D)$ matrices of sums of products of transformation matrices from torn variables of D-class j to torn variables of D-class i.

In ore dressing plants particles are broken and so can only become smaller, therefore there is no mass flow from torn variables of D-class j to those of D-class i when j>i, so that $C_{ij} = 0$ for j>i.

Just as the groups were solved sequentially so can the variables in the D-classes. If X_{Di} and Y_{Di} i = 1,M are the estimated and calculated torn variables in this group belonging to each D-class, respectively, and a_{Di} the elements of a a_G corresponding to D-class i then:-

$$X^*_{Di} = (I-B_{Di})^{-1}.\left(\sum_{j=1}^{i-1} C_{ij}X^*_{Dj} + a_{Di}\right) \qquad \text{---------- (5.9)}$$

This implies that each group can be solved in as many stages as there are D-classes, and to find the solution only M matrices, viz $(I-B_{Di}), i=1,M$ of size $(m_G.n_D \times m_G.n_D)$ need be inverted.

5.3.3 Solution within the D-Classes

If group G has M, D-classes and only one G-class and one S-class then equation 5.9 is the final solution. But when there is more than one G or S-class then further processing is possible.

Sub-chapter 5.3.2 showed that variables in each D-class can be solved independently of the rest as long as variables in all previous D-classes (larger particles) have already been found. Choosing some D-class, corresponding to a particular $B_{Di} = B_D$, allows rearrangement, by row and column interchanges, so that similar variables from each tear are collected together, then:-

$$B_D = \begin{bmatrix} B_{T1} & E_{12} & \cdots & E_{1L} \\ E_{21} & B_{T2} & & \vdots \\ \vdots & & \ddots & \vdots \\ E_{L1} & \cdots & \cdots & B_{TL} \end{bmatrix} \qquad L = \text{number of G-classes} \times \text{number of S-classes}$$

where B_{Ti} are the $(m_G \times m_G)$ matrices containing the sums of products of transformation matrices for the same G or S class in each tear belonging to D-class D

and E_{ij} are the $(m_G \times m_G)$ matrices containing the sums of products of transformation matrices from variable j to variable i in each tear belonging to D-class D.

If a comminution unit is the only transformation unit then there can be no mass flow between G or S-classes belonging to the same D-class, as the only possible transformation is caused by breakage. Even when liberation occurs, a grade transformation, the parent particle must first be broken. Therefore for this case the $E_{ij} = 0$ for all i and j.

If Y_{Ti} and X_{Ti} are the calculated and estimated torn variables respectively in group G, D-class D and the same G and S-class, corresponding to variable i, and B_{Ti} the elements of B_D corresponding to the same torn variable in each torn stream then:-

$$X^*_{Ti} = (I - B_{Ti})^{-1} . a_{Ti} \qquad i = 1, L \qquad \text{---------- (5.10)}$$

In this case the a_{Ti} are independent of X^*_{Tj} $j \neq i$.

Therefore X^*_{Ti} $i = 1, L$ can be found in a single step. This involves the inversion of L $(m_G \times m_G)$ matrices corresponding to $(I - B_{Ti})$, $i = 1, L$.

The only other transformation unit, the conditioner has been discussed in sub-chapter 5.1 and it was found that if k-classes were used then in all existing simulators no transformation actually takes place. In the hypothetical case of a conditioner being modelled to act within a recycle loop and actually causing a transformation then as long as the transformations are in one direction, (i.e. some fraction of the material in a particular k-class is only distributed into k-classes with a greater (or lesser) flotation rate constant) it is possible, just as the group was rearranged into collections of D-classes, to rearrange each κ-class into collections of k-classes, and the D-class solved in n_k steges (n_k is the number of k-classes).

If the k-classes within a size fraction transform in both directions (i.e. to k-classes with flotation rate constants distributed to other k-classes with both larger and smaller rate constants) then all the variables in each D-class must be solved simultaneously. This possibility is very unlikely.

5.3.4 *Overall Simultaneous Solution*

In the previous sub-chapters the problem of finding the elements and inverting a (n.m x n.m) matrix has been reduced to finding the elements and inverting n matrices of size (m_G x m_G) for N groups (each group may have a different number of tears so the size of the matrices to be inverted (m_G x m_G) is group dependent). Within each group this is a major reduction as instead of having to find $n^2.m_G^2$ elements and then invert a (n.m_G x n.m_G) matrix only n.m_G^2 elements need be found and n (m_G x m_G) matrices inverted. Generally n >> 1 and the computation time for inversion of a matrix increases with the cube of its size so that this is a tremendous reduction in computation effort.

The Y_{Ti} and X_{Ti} can be found for each iteration, and the B_{Ti} determined as the sum of products of transformation matrices, (mass fractions). Then by applying equation 5.10 to each i (term variables) for M iterations to each group in turn, X^* will be found (note M is group dependent) and equation 5.4 completely solved.

5.4 Iterative Solution of Linear Systems

To allow equation 5.2a to be solved by an iterative scheme, so that $Y = X = X^*$, it can be written as:-

$$x^{i+1} = B.X^i + A.F \qquad i \text{ is the iteration index} \qquad \text{---------- (5.11)}$$

Then starting with some X^0, equation 5.11 is used repeatedly until:-

$$\lim_{k \to \infty} X^k = X^* \qquad \text{---------- (5.12)}$$

5.4.1 Condition for Global Convergence of Linear Systems

The iteration scheme defined by equation 5.11 may or may not converge to X^* as required by 5.12. Whether convergence is achieved or not depends on the value of the spectral radius of B, where the spectral radius of B is defined as:-

$$\rho(B) = |\lambda_{max}|$$

and λ_{max} is the largest eigenvalue of B.

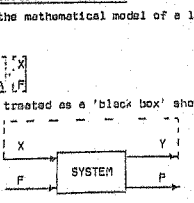
Ortega and Rheinboldt (1970) p 303 show that $\rho(B)<1$ is both necessary and sufficient for the convergence of equation 5.11 from any $X^0 \epsilon R^{n_T}$. Therefore if it can be shown for any linear system that $\rho(B)<1$ then it can be concluded that a linear system will always converge from any starting point when the iterative scheme is direct substitution (equation 5.11). (n_T = n.m is the total number of term variables).

5.4.2 Proof of Convergence of Linear Systems when using Direct Substitution

Equation 5.2 is the mathematical model of a linear system and can be written as:-

$$\begin{bmatrix} Y \\ P \end{bmatrix} = \begin{bmatrix} B & A \\ BB & AA \end{bmatrix} \begin{bmatrix} X \\ F \end{bmatrix} \qquad \text{---------- (5.13)}$$

If the system is treated as a 'black box' shown below:-



then $\begin{bmatrix} X \\ F \end{bmatrix}$ is the system input variables, $\begin{bmatrix} Y \\ P \end{bmatrix}$ the system output variables

and $\begin{bmatrix} B & A \\ BB & AA \end{bmatrix}$ the system transformation matrix. Now just as the column sums of unit transformation matrices must equal unity so must the column sums of $\begin{bmatrix} B & A \\ BB & AA \end{bmatrix}$ equal unity. For example some variable in vector X, x_j, is distributed amongst the output variables Y and P, according to the values of the elements in column J of $\begin{bmatrix} B & A \\ BB & AA \end{bmatrix}$. But the total mass of variable x_j into the system must equal the total mass of variable x_j in the form of Y and P variables out of the system, therefore the sum of the elements in column J of $\begin{bmatrix} B & A \\ BB & AA \end{bmatrix}$ must equal one.

It has been established that for a linear system the elements of the matrix $\begin{bmatrix} B & A \\ BB & AA \end{bmatrix}$ are non-negative and that the column sums of this matrix equal one.

From Varga (1962) p21 it is known that when $D = (d_{ij})$ is an irreducible nxn matrix, and

$$\sum_{j=1}^{n} |d_{ij}| \leq v \qquad \text{for all } 1 \leq i \leq n \qquad \text{---------- (5.14)}$$

with strict inequality for at least one i then $\rho(D) < v$

Similarly when

$$\sum_{i=1}^{n} |d_{ij}| \leq v' \qquad \text{for all } 1 \leq j \leq n \qquad \text{---------- (5.15)}$$

with strict inequality for at least one j then $\rho(D) < v'$.

Consequently $\rho(D) < \min(v, v')$ ---------- (5.16)

Irreducibility is defined by Varga (1962) p18:-

Definition: For $n \geq 2$ an nxn complex matrix D is *reducible* if there exists an nxn permutation matrix P_M such that

$$P_M A P_M^T = \begin{bmatrix} D_{11} & D_{12} \\ 0 & D_{22} \end{bmatrix}$$

where D_{11} is an rxr submatrix and D_{22} an (n-r)x(n-r) submatrix where $1 \leq r \leq n$. If no such permutation exists then D is *irreducible*.

Therefore if B, in equation 5.13, is irreducible and at least one element of BB is positive then $\rho(B) < 1$. Sub-chapter 5.3 shows that B is reducible, however it has been shown in Appendix B that for a lower block triangular matrix, D, with D_k the diagonal blocks of the per-

titioned matrix:-

$$\rho(D) = \max_k (\rho(D_k)) \qquad \text{---------- (5.17)}$$

Therefore if the diagonal blocks of B in its reduced form are irreducible, and if it can be shown that at least one of the column sums of each of the diagonal blocks are less than one then $\rho(B)<1$.

Rearranging the elements in 5.13, so that B is lower blocktriangular and has only irreducible diagonal block matrices, can be done by reducing each reducible diagonal block matrix, by row and column interchanges. The resulting matrix need not necessarily have diagonal block matrices of the same size.

The rearranged equation 5.13 can be written:-

$$\left[\begin{array}{c|ccc|c} Y_{R1} & B_{R1} & 0 \cdots & 0 & A_{R1} \\ Y_{R2} & B_{R21} & B_{R2} & \ddots & A_{R2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ Y_{RL} & B_{RL1} & \cdots & B_{RL} & A_{RL} \\ \hline P & BB_1 & \cdots & BB_L & AA \end{array}\right] \left[\begin{array}{c} X_{R1} \\ X_{R2} \\ \vdots \\ X_{RL} \\ \hline F \end{array}\right] \qquad \text{---------- (5.18)}$$

where B_{Ri} are the irreducible block diagonal matrices of the rearranged B, these matrices are square but not necessarily the same size.

B_{Rij} the matrices corresponding to the interaction of B_{Ri} on B_{Rj}

BB_i are the matrices formed from BB after the required column interchanges have been made and BB partitioned to correspond with the columns of the B_{Ri}.

X_{Ri} and Y_{Ri} are the elements of vectors X and Y respectively which correspond to the rows of the B_{Ri}.

Hence

$$Y_{Ri} = \sum_{k=1}^{i-1} B_{Rik}X_{Rk} + B_{Ri}X_{Ri} + A_{Ri}F \qquad i = 1,L \text{ ---------- (5.19)}$$

At the solution $Y_{Ri} = X_{Ri} = X^*_{Ri}$ $\qquad i = 1,L$

$$\text{So } X^*_{Ri} = \sum_{k=1}^{i-1} B_{Rik}X^*_{Rk} + B_{Ri}X^*_{Ri} + A_{Ri}F \qquad i = 1,L \text{ ---------- (5.20)}$$

Summing the elements of the vectors on both sides of equation 5.20 gives:-

$$\Sigma\{X^{*}_{Ri}\} = \Sigma\{\sum_{k=1}^{i-1} B_{Rik}X^{*}_{Rk}\} + \Sigma\{B_{Ri}X^{*}_{Ri}\} + \Sigma\{A_{Ri}F\} \qquad i=1,L \quad \text{----(5.21)}$$

where the $\Sigma\{\cdot\}$ sign simply indicates the sum of all the elements of the vector in the brackets $\{\cdot\}$.

As the sum of the elements of each column of $\begin{bmatrix} B & A \\ BB & AA \end{bmatrix}$ equals one, it can be concluded that:-

$$\Sigma\left\{\begin{bmatrix} B_{Ri} \\ B_{Ri+1,i} \\ \vdots \\ B_{RL,i} \\ BB_i \end{bmatrix} \cdot \{X_{Ri}\}\right\} = \Sigma\{X_{Ri}\} \qquad i=1,L$$

$$\text{or } \Sigma\{B_{Ri}X_{Ri}\} + \Sigma\{\sum_{j=i+1}^{L} B_{Rj,i}X_{Ri}\} + \Sigma\{BB_iX_{Ri}\} = \Sigma\{X_{Ri}\} \qquad i=1,L \text{ --- (5.22)}$$

Rearranging equation 5.22 and setting $X = X^{*}$ gives:-

$$\Sigma\{X^{*}_{Ri}\} - \Sigma\{B_{Ri}X^{*}_{Ri}\} = \Sigma\{\sum_{j=i+1}^{L} B_{Rji}X^{*}_{Ri}\} + \Sigma\{BB_iX^{*}_{Ri}\} \qquad i=1,L \text{ --- (5.23)}$$

Rearranging equation 5.21 gives:-

$$\Sigma\{X^{*}_{Ri}\} - \Sigma\{B_{Ri}X^{*}_{Ri}\} = \Sigma\{\sum_{k=1}^{i-1} B_{Rik}X^{*}_{Rk}\} + \Sigma\{A_{Ri}F\} \qquad i=1,L \text{ --- (5.24)}$$

Comparing equations 5.23 and 5.24 leads to:-

$$\Sigma\{\sum_{k=1}^{i-1} B_{Rik}X^{*}_{Rk}\} + \Sigma\{A_{Ri}F\} = \Sigma\{\sum_{j=i+1}^{L} B_{Rji}X^{*}_{Ri}\} + \Sigma\{BB_iX^{*}_{Ri}\}$$

$$i=1,L \text{ --- (5.25)}$$

If the left hand side of equation 5.25 equals zero for some particular i, say i = I, then this means that no material is becoming any of these variables, either from feed or tear streams. Therefore the only possible real solution, in this case, is for X^{*}_{RI} to be a null vector (i.e. each element zero). When $\rho(B_{RI}) = 1$, this solution will only be reached by an iterative scheme if the initial guess, X^{o}_{RI}, is the null vector. Conversely when $\rho(B_{RI}) = 1$ and $X^{o}_{RI} = 0$, then the true solution $X^{*}_{RI} = 0$ will be found immediately and will not effect convergence of other

variables. So if the initial guess used, is the null vector, then the possible problem of convergence of variables, for which the left hand side of equation 5.25 equals zero, will never arise.

The left hand side of equation 5.25 cannot be negative (as the elements of the matrices and vectors in the linear case are non-negative) and the problem of it being zero has been eliminated. Therefore for all variables of interest the left hand side of equation 5.25 must be positive and consequently the right hand side must also be positive.

For the right hand side of equation 5.25 to be positive when $i = I$, at least one element of B_{RjI} or BB_I, $j = I+1,L$ must be positive, and hence at least one column sum of B_{RI} is less than one. B_{RI} is irreducible and has column sums equal to or less than one with strict inequality for at least one column, therefore $\rho(B_{RI}) < 1$ (by equation 5.18).

This is true for $I = 1,L$

i.e. $\rho(B_{Ri})<1 \qquad i=1,L$

therefore by equation 5.17

$$\rho(B)<1 \qquad \text{---------- (5.26)}$$

and the convergence of linear systems by the iterative scheme of 'direct substitution' is assured.

6 SOLUTION OF NON-LINEAR SYSTEMS

Definition: A *non-linear system* is one in which some or all of the models of the unit operations are non-linear.

Definition: A *non-linear model* is one in which the unit transformation matrix varies as a function of the unit feed conditions.

Definition: The *unit transformation matrix* of a non-linear model is the matrix of partial derivatives of the unit output variables with respect to the unit input variables.

Definition: The *system and group Jacobians* are the matrices of partial derivatives of calculated torn variables with respect to the estimated torn variables in the system and group respectively.

From these definitions the system and group Jacobians are made up from the sum of products of unit transformation matrices along the paths between tear streams.

6.1 Iterative Solution Scheme for Non-Linear Systems

The mass balance of a non-linear system can be found by solving the large set of simultaneous non-linear equations. To do this an iterative solution scheme must be used. A typical iterative procedure is split into two parts:-

(a) The decomposition of the plant network to an acyclic system
(b) The convergence algorithm which makes the iteration converge to the solution

As in chapter 3.2

X the vector of input variables (or estimated torn variables) to the decomposed system (n_T elements = n.m, n = number of variables per stream and m = number of tear streams)

$\phi(X)$ the vector of output variables (or calculated torn variables) from the decomposed system (n_T=n.m elements)

Then an iterative scheme uses X and $\phi(X)$ plus possibly previous values of these vectors to find a new estimate of X, which is hopefully closer to the solution X^* of the equation $X = \phi(X)$.

The iterative scheme can be written as:-

$$X^{k+1} = G(X^k) \qquad k = 0,1,\ldots \text{ is the iteration index} \qquad \text{---------- (6.1)}$$

where $X \in R^{n_T}$, $G: R^{n_T} \to R^{n_T}$, $\Phi: R^{n_T} \to R^{n_T}$

So X is an n_T dimensional vector and G and Φ operators on X.

6.1.1 Conditions for Convergence of a Non-Linear Iterative Scheme

Sufficient conditions for local convergence of the iterative scheme defined by equation 6.1, to the solution X^* are given by the OSTROWSKI THEOREM (Ortega and Rheinboldt (1970) p300) as:-

(a) G(X) must be F-differentiable at X^*

and (b) $\rho(G'(X^*))<1$

$G'(X^*)$ is the matrix of partial derivatives of G(X) with respect to X at $X = X^*$.

6.1.2 Rate of Convergence of a Non-Linear Iterative Scheme

If the conditions of 6.1.1 apply then according to the LINEAR CONVERGENCE THEOREM (Ortega and Rheinboldt (1970) p301)

$$R_1(I,X^*) = \rho(G'(X^*)) \qquad \text{---------- (6.2)}$$

where $R_1(I,X^*) = \sup \{R_1(\{X^k\}) | \{X^k\} \in C(I,X^*)\}$

and $C(I,X^*)$ is the set of all sequences generated by I which converge to X^*, and I refers to the iterative process 6.1

and $R_1(\{X^k\}) = \lim_{k\to\infty} \sup ||X^k - X^*||^{1/k}$ ---------- (6.3)

is called the '*root convergence factor*' or R-factor of the sequence.

If I_1 and I_2 are two iterative processes with limit X^* then I_1 is '*R-faster*' than I_2 at X^* if $R_1(I_1,X^*) < R_1(I_2,X^*)$. (Ortega and Rheinboldt (1970) p289).

The utility of this measure of rate of convergence will be shown if $R_1(I_1,X^*) < R_1(I_2,X^*)$ implies that

$$||X^k - X^*|| < ||Z^k - X^*|| \qquad \text{for } k > K$$

where X^k is a member of the sequence generated by I_1 and Z^k a member of the sequence generated by I_2. k is the iteration index, and $||.||$ any suitable norm.

If a sequence $\{a^k\} \subset R^1$ converges then

$$\lim_{k\to\infty} \sup \{a^k\} = \lim_{k\to\infty} \{a^k\} \qquad \text{(Rudin(1964))}$$

So that

$$R_1(I_1, X^*) = \lim_{k\to\infty} ||X^k - X^*||^{1/k} \qquad \text{---------- (6.4)}$$

and $$R_1(I_2, X^*) = \lim_{k\to\infty} ||Z^k - X^*||^{1/k} \qquad \text{---------- (6.5)}$$

Therefore for some $k > K_1$

$$| \, ||X^k - X^*||^{1/k} - R_1(I_1, X^*)| < \varepsilon \qquad \varepsilon > 0$$

$$\text{or } R_1(I_1, X^*) - \varepsilon < ||X^k - X^*||^{1/k} < R_1(I_1, X^*) + \varepsilon \qquad \text{---------- (6.6)}$$

Similarly for some $k > K_2$

$$R_1(I_2, X^*) - \varepsilon < ||Z^k - X^*||^{1/k} < R_1(I_2, X^*) + \varepsilon \qquad \text{---------- (6.7)}$$

When $R_1(I_1, X^*) < R_1(I_2, X^*)$ then some $\varepsilon > 0$ exists so that:-

$$R_1(I_1, X^*) + 2\varepsilon \leq R_1(I_2, X^*)$$

$$\text{or} \quad R_1(I_1, X^*) + \varepsilon \leq R_1(I_2, X^*) - \varepsilon \qquad \text{---------- (6.8)}$$

So that for $k > K = \max\{K_1, K_2\}$ equation 6.6, 6.7 and 6.8 can be compared to give:-

$$||X^k - X^*||^{1/k} < ||Z^k - X^*||^{1/k}$$

Consequently:-

$$||X^k - X^*|| < ||Z^k - X^*|| \text{ for } k > K \text{ when } R_1(I_1, X^0) < R_1(I_2, X^*) \text{ ---- (6.9)}$$

Therefore, by equation 6.2 and 6.9, $\rho(G'(X^*))$ can be used as a measure of the rate of convergence of an iterative scheme, and the smaller this value the faster will be the convergence. Or in other words the iterative scheme with the smallest $\rho(G'(X^*))$ will be closest to the solution after a fixed number of iterations provided that this number is larger than some minimum.

6.2 F-Differentiability of $G(X^*)$

One of the conditions for local convergence in sub-chapter 6.1.1 is that $G(X)$ be F-differentiable at $X = X^*$. This is very difficult to show directly, however Apostel (1974) p357 gives sufficient conditions. These are that one of the partial derivatives of $G(X)$ exists and the remaining $(n.m - 1)$ partial derivatives exist in the vicinity of X^* and are continuous at $X = X^*$.

The system is made up of models of units of ore dressing equipment, and the mathematical model of the acyclic system is made up from these equipment's unit transformation matrices. Therefore if each of the unit models fulfill the above existence and continuity conditions then the system must also fulfill the conditions.

Linear models definitely have partial derivatives which are continuous (in fact constant over the full range of input variables). Non-linear models, which have been extracted from the literature, (see Part III), and are included as library module subroutines, also have continuous derivatives.

The conclusion is therefore that although F- differentiality cannot be proved for every possible case it can be ensured by careful selection of models, that this condition is met.

6.3 Convergence of the Method of Direct Substitution

substitution is the method of convergence when the iteration scheme by equation 6.1 has

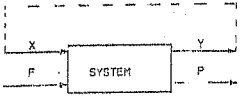
$$G(X) = \Phi(X)$$

Convergence of this scheme is ensured if (by sub-chapter 6.1.1)

$$\rho(\Phi'(X^*)) < 1$$

where $\Phi'(X^*)$ is the matrix of partial derivatives of $\Phi(X)$ with respect to X at $X = X^*$.

Consider the system shown below:-



where $Y = \Phi(X)$ and F and P are the vectors of all the system feed and product variables respectively.

At steady state $Y = X = X^*$

and $\sum_{i=1}^{n_F} f_i^* = \sum_{j=1}^{n_P} p_j^*$

where n_F and n_P are the number of feed and product variables respectively and $n_T = nxm$ is the total number of torn variables. If at this steady state a small increase occurs to one of the variables of X, say x_j, then by mass balance :-

$$\sum_{i=1}^{n_T} x_i^* + \Delta x_j + \sum_{k=1}^{n_F} f_k^* = \sum_{i=1}^{n_T} (y_i^* + \Delta y_i) + \sum_{l=1}^{n_P} (p_l^* + \Delta p_l)$$

hence $\Delta x_j = \sum_{i=1}^{n_T} \Delta y_i + \sum_{l=1}^{n_P} \Delta p_l$

$$\therefore \quad \Delta x_j = \Delta\Sigma^{n_T}_{i=1} y_i + \Delta\Sigma^{n_P}_{l=1} p_l \qquad \text{---------- (6.10)}$$

Using equation 6.10 for $\Delta x_j > 0$, there are six possible consequences for $\Delta\Sigma^{n_T}_{i=1} y_i$ and $\Delta\Sigma^{n_P}_{l=1} p_l$:-

(Let $\Sigma y \equiv \sum_{i=1}^{n_T} y_i$ and $\Sigma p \equiv \sum_{l=1}^{n_P} p_l$)

(a) If $\Delta\Sigma y = 0$ then $\Delta\Sigma p = \Delta x_j$ and $\frac{\partial \Sigma p}{\partial x_j} > 0$

(b) If $\Delta\Sigma y > 0$ and $\Delta\Sigma p > 0$ then $\frac{\partial \Sigma p}{\partial x_j} > 0$

(c) If $\Delta\Sigma y > 0$ and $\Delta\Sigma p < 0$ then $\frac{\partial \Sigma p}{\partial x_j} < 0$

but if this happened $\Delta\Sigma y > \Delta x_j$, and this implies that if a small increase is made in x_j, from the steady state value, then a larger increase occurs in Σy. This means that instead of the disturbance being damped it becomes increasingly larger and the system moves further and further from the steady state.

(d) If $\Delta\Sigma y < 0$ and $\Delta\Sigma p > 0$ then $\frac{\partial \Sigma p}{\partial x_j} > 0$

(e) If $\Delta\Sigma y < 0$ and $\Delta\Sigma p < 0$, impossible as $\Delta x_j > 0$

(f) If $\Delta\Sigma p = 0$ then $\Delta\Sigma y = \Delta x_j$, this situation implies that for any change in x_j exactly the same change occurs to Σy and hence

$$\sum_{i=1}^{n_T} x_i^* + \Delta x_j = \sum_{i=1}^{n_T} y_i^* + \Delta\Sigma y$$

so that there are infinitely many possible steady states, which is impossible in normal circumstances.

Therefore $\frac{\partial\Sigma p}{\partial x_j} > 0$, except for the very unusual situation when a plant is controlled to operate at an unstable point. For purposes of this steady state simulator it is possible, without imposing any important restriction on the user, to ignore these strange cases.

A general mass balance on the system gives:-

$$\sum_{i=1}^{n_T} x_i + \sum_{k=1}^{n_F} f_k = \sum_{i=1}^{n_T} y_i + \sum_{l=1}^{n_P} p_l \qquad \text{---------- (6.11)}$$

Partially differentiating equation 6.11 with respect to x_j gives:-

$$1 + 0 = \frac{\partial\Sigma y}{\partial x_j} + \frac{\partial\Sigma p}{\partial x_j} \qquad \text{---------- (6.12)}$$

but $\frac{\partial\Sigma p}{\partial x_j} > 0$

therefore $\frac{\partial\Sigma y}{\partial x_j} < 1$ ---------- (6.13)

Now as x_j was chosen arbitrarily:-

$$\frac{\partial\Sigma y}{\partial x_j} < 1 \text{ for } 1 \leq j \leq n_T$$

hence $\sum_{i=1}^{n_T} \frac{\partial y_i}{\partial x_j} < 1$ for $1 \leq j \leq n_T$ ---------- (6.14)

From the definition of the system Jacobian, $\Phi'(X^*)$.

$$\Phi'(X^*) = \left[\begin{array}{ccc} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \frac{\partial y_1}{\partial x_{n_T}} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_{n_T}}{\partial x_1} & \cdots & \frac{\partial y_{n_T}}{\partial x_{n_T}} \end{array}\right]_{X = X^*} \qquad \text{---------- (6.15)}$$

$$||\Phi'(X^*)||_1 = \max_{1 \le j \le n_T} \sum_{i=1}^{n_T} \left|\frac{\partial y_i}{\partial x_j}\right| \quad \text{(} l_1 \text{ norm for matrices, Ortega and Rheinboldt (1970) p41)}$$

and if $\frac{\partial y_i}{\partial x_j} \ge 0$ for $1 \le i, j \le n_T$ ---------- (6.16)

then $||\Phi'(X^*)||_1 = \max_{1 \le j \le n_T} \sum_{i=1}^{n_T} \frac{\partial y_i}{\partial x_j}$

so $||\Phi'(X^*)||_1 < 1$ ---------- (6.17)

(by equation 6.14)

and $\rho(\Phi'(X^*)) < 1$ for $\frac{\partial y}{\partial x} \ge 0$ ---------- (6.18)

(Varga(1962) p8)

The restriction imposed by equation 6.16, that the elements of the Jacobian be non-negative, is not severe. For a linear system it must be true (see sub-chapter 5.3.2) and for a system that is slightly non-linear it is expected that the positive mass flows will dominate any negative non-linearitives.

It has been stated by Brauer (1964) p27; that as the characteristic roots of a matrix change continuously with the elements of the matrix; a matrix, which has mostly positive elements and a few negative elements of relatively small absolute value, will have similar properties to positive matrices. Therefore as decreasing the elements of a non-negative matrix reduces its eigenvalues, (Tausky (1964)p127), and hence its spectral radius, having a few negative elements of relatively small absolute value implies:-

$$\rho(\Phi'(X^*) \text{ few negative}) < \rho(\Phi'(X^*) \text{ non negative})$$

Hence

$$\rho(\Phi'(X^*)) < 1 \quad \text{in normal circumstances} \qquad \text{---------- (6.19)}$$

Several authors (see sub-chapter 3.3) have claimed that direct substitution will always converge and this supports the conclusion represented by equation 6.19.

6.4 Convergence of Newton-like Methods

Direct substitution is very simple to apply but unfortunately can be slow to converge (sub-chapter 3.2 and 3.3).

To speed up the rate of convergence Newton-like methods have been applied, and equation 3.3 in terms of equation 6.1 gives:-

$$G(X) = X - (I - \phi'(X))^{-1}(X - \phi(X)) \qquad \text{---------- (6.20)}$$

As the elements of $\phi'(X)$ are not usually readily available (see sub-chapter 3.2.3) $\phi'(X)$ is approximated. If $Q(X)$ is chosen to approximate $\phi'(X)$ in equation 6.20 then a Newton-like method has been created, and:-

$$G(X) = X - (I - Q(X))^{-1}(X - \phi(X)) \qquad \text{---------- (6.21)}$$

Partially differentiating $G(X)$ in equation 6.21 with respect to X gives the matrix $G'(X^*)$ and at $X = X^*$:-

$$G'(X^*) = I - (I - Q(X^*))^{-1}(I - \phi'(X^*)) \qquad \text{as } X^* = \phi(X^*)$$

$$\therefore\ G'(X^*) = (I - Q(X^*))^{-1}(\phi'(X^*) - Q(X^*)) \qquad \text{---------- (6.22)}$$

For the local convergence of these Newton-like methods it is sufficient for $\rho(G'(X^*)) < 1$ (by sub-chapter 6.1.1).

The choice of $Q(X)$ clearly has a significant effect on the rate of convergence. For example:-

(a) $Q(X) = \phi'(X)$. This is Newton's method as given by equation 6.20. From equation 6.22, $G'(X^*) = 0$ and hence $\rho(G'(X^*)) = 0$.

(b) $Q(X) = 0$. This is direct substitution, as from equation 6.21, $G(X) = \phi(X)$. From equation 6.22, $\rho(G'(X^*)) = \rho(\phi'(X^*))$.

(c) $Q(X) = \text{diag}(\phi'(X))$. ($Q(X)$ is the null matrix with the same diagonal elements as $\phi'(X)$). This is Wegstein's method. In sub-chapter 3.2.2 Wegstein's method is shown to obtain successively better approximations of the partial derivatives of each variable with respect to itself as the solution is approached. These derivatives are the diagonal elements of $\phi'(X)$ at $X = X^*$.

(d) Q(X) = block diag $\Phi'(X)$. (Q(X) is the null matrix and has the same block diagonal elements as $\Phi'(X)$, which are chosen in the same way as the B_{Ti} of equation 5.10).

In sub-chapter 5.4 a method of solving linear systems by a sequential simultaneous technique was developed based on the characteristics of linear ore dressing plants. In general it is expected that the unit models will not be highly non-linear, as the discretization of the streams (sub-chapter 4.1) is intended to minimize the interaction between torn variables. Therefore applying a similar technique to non-linear systems should allow fast convergence. (Note: for linear systems $\Phi'(X) = B$ of chapter 5).

This block diagonal choice of Q(X) will be called the 'Reduced Newton Method' and its convergence characteristics are considered in the next sub-chapter.

6.5 The Reduced Newton Method

6.5.1 Convergence of the Reduced Newton Method

From sub-chapter 6.3 it can be expected that when $\Phi'(X)$ is non-negative or has a few negative elements then the l_1 norm of $\Phi'(X^*)$ will be less than one ($||\Phi'(X^*)||_1 < 1$) and hence that $\rho(\Phi'(X^*)) < 1$.

Working from the basis that $\Phi'(X^*) \geq 0$ and $\rho(\Phi'(X^*)) < 1$ it is proved in appendix A.1 that $\rho(G'(X^*)) < 1$ for the reduced Newton method. Therefore in this case the reduced Newton method guarantees local convergence.

Appendix A.2 takes this a little further and proves that if $\rho(|\Phi'(X^*)|) < 1$ then $\rho(G'(X^*)) < 1$ for the reduced Newton method when only $Q(X^*) \geq 0$ instead of $\Phi'(X^*) \geq 0$.

Ortega and Rheinboldt (1970) p41 define the l_1 and l_∞ norms of matrices as:-

$$||A||_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$$

and $$||A||_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|$$

where $A \in R^{n \times m}$

So that if either of these norms of $\Phi'(X^*)$ are less than one then $\rho(|\Phi'(X^*)|) < 1$ (as $||A||_1 = |||A|||_1$ and $||A||_\infty = |||A|||_\infty$) and the proof in appendix A.2 will hold.

The requirement that $\Phi'(X^*) \geq 0$ or certainly $Q(X^*) \geq 0$ will usually be satisifed as the mass flows should dominate any small negative non-linearities, so the local convergence of the reduced Newton method in normal circumstances is virtually ensured. However a completely general proof, that this (or any other) iterative convergence scheme will always exhibit local convergence on any non-linear system, does not seem possible.

In appendices A.3 and A.4 a very lengthy proof eventually shows that the reduced Newton method always exhibits local convergence on any non-linear system for which $||\Phi'(X^*)||_\infty < 1$.

It has not been possible to demonstrate that $||\Phi'(X^*)||_\infty < 1$, but it has been found as a corrollary of the proof in appendix A.4 that:-

$$||G_{RN}||_\infty < ||\Phi'(X^*)||_\infty \qquad \text{---------- (6.24)}$$

where $G_{RN} = G'(X^*)$ for the reduced Newton method. Although equation 6.24 does not necessarily imply that $\rho(G_{RN}) < \rho(\Phi'(X^*))$ and hence by the 'linear convergence theorem' (sub-chapter 6.1.2) that the reduced Newton method converges at a faster rate than direct substitution, it is a strong indication in its favour. A second, but also not necessary, implication is that if direct substitution converges (as is highly likely by sub-chapter 6.3) so should the reduced Newton method.

6.5.2 Determination of Q for the Reduced Newton Method

In chapter 5 the product of mass fractions of 'similar' variables (see sub-chapter 4.1) along the paths between tears are the elements B_{Ti} required to solve equation 5.10. Similarly by the chain rule for partial derivatives the elements of the Jacobian (Φ') which are used in the reduced Newton method, are the products of partial derivatives of 'similar' variables for each unit along the paths between the tear streams.

The calculation of these partial derivatives by the path product method is quite involved in terms of computer logic. However it does provide the actual derivatives and if by having the actual derivatives one iteration can be saved, then in medium or large size plants the cost of using this method is more than compensated for. Another advantage of the reduced Newton method is that for a linear system the elements B_{Ti}

of equation 5.10 are found and this equation can then be solved exactly.

For this technique to work the partial derivatives of each output variable with respect to the same input variable must be found for each unit. For linear separation units these partial derivatives are the diagonal elements of T_j (see chapter 5).

The transformation matrix of a linear separation unit is diagonal and hence

$$t_{ij} = w_{ij}/v_i$$

where t_{ij} is the diagonal element in the i^{th} row and i^{th} column of T_j for the unit

w_{ij} is variable i of output stream j from the unit

v_i is variable i of the input stream to the unit

The partial derivatives required for linear separation units can therefore be obtained externally from the unit models by using the input and outputs. For non-linear separation units the fraction w_{ij}/v_i will not be exactly the required partial derivative but if the unit is nearly linear it will be a good approximation. This is similar to the Modified Newton Method (Ortega and Rheinboldt (1970) p187) where the Jacobian is not updated every iteration and the same Jacobian is used for several iterations. This method is based on the assumption that the Jacobian does not change much each iteration. This idea of using the mass fraction w_{ij}/v_i to approximate the partial derivatives of a unit is shown to work in TEST EXAMPLE II. Here the separation unit is the King flotation model which involves an internal iteration and the determination of the actual derivatives is very awkward.

These mass fractions will not be good approximations to the derivatives for transformation units as the transformation matrix even in the linear case is not diagonal. These derivatives should always be generated internally by the unit model.

6.5.3 Grouping Non-Linear Systems into Subplants

If as with linear systems (sub-chapter 5.4.1) it is possible to group units so that there is no feed-back of material from each group to any of the preceding groups, and the rows and columns in the system Jacobian are arranged so that the M_1 torn variables in the first group correspond to the first M_1 rows and columns, the M_2 torn variables in the second group correspond to the second M_2 rows and columns etc then the system Jacobian $\Phi'(X)$ is lower block triangular. This is because,

by the definition of a group, there is no material flow to preceding groups and hence changing the values of the torn variables in any group cannot have any effect on the torn variables of preceding groups.

In the iterative scheme defined by equations 6.1 and 6.21, $Q(X)$ approximates $\Phi'(X)$. As $\Phi'(X)$ is lower block triangular it is only necessary to find the lower block triangular elements of $Q(X)$. If the whole system is to be solved simultaneously (in the sense that all groups are solved at the same time), and $\Phi'(X)$ is arranged as suggested in paragraph one, then:-

$$Q^A = Q(X^*) = \begin{bmatrix} Q_{11} & 0 & \cdots & 0 \\ Q_{22} & Q_{22} & & 0 \\ \vdots & & \ddots & \vdots \\ Q_{N1} & \cdots & \cdots & Q_{NN} \end{bmatrix}$$

where Q_{ij} are $M_i \times M_j$ matrices, and there are N groups in the system

The rate of convergence of this scheme is given in sub-chapter 6.4 as:-

$$\rho(G'(X^*)) = (I - Q^A)^{-1}.(\Phi'(X^*) - Q^A)$$

Here the block diagonal matrices of $(I - Q^A)^{-1}$ are $(I - Q_{ii})^{-1}$ (Lapidus (1962) p252) and the lower off-diagonal matrices are represented by P_{ij}.

So

$$(I-Q^A)^{-1}.(\Phi'-Q^A) = \begin{bmatrix} (I-Q_{11})^{-1} & 0 & \cdots & 0 \\ P_{21} & (I-Q_{22})^{-1} & & \vdots \\ \vdots & & \ddots & \vdots \\ P_{N1} & \cdots & \cdots & (I-Q_{NN})^{-1} \end{bmatrix} \begin{bmatrix} (\Phi_{11}-Q_{11}) & \cdots & \cdots & 0 \\ (\Phi_{21}-Q_{21}) & & & \vdots \\ \vdots & & \ddots & \vdots \\ (\Phi_{N1}-Q_{N1}) & \cdots & & (\Phi_{NN}-Q_{NN}) \end{bmatrix}$$

where

$$\Phi' = \Phi'(X^*) = \begin{bmatrix} \Phi_{11} & \cdots & 0 \\ \Phi_{21} & \Phi_{22} & \vdots \\ \vdots & \ddots & \vdots \\ \Phi_{N1} & \cdots & \Phi_{NN} \end{bmatrix}$$

hence

$$(I-Q^A)^{-1}.(\Phi'-Q^A) = \begin{bmatrix} (I-Q_{11})^{-1}.(\Phi_{11}-Q_{11}) & 0 & \cdots & 0 \\ R_{21} & (I-Q_{22})^{-1}.(\Phi_{22}-Q_{22}) & & \vdots \\ \vdots & & \ddots & \vdots \\ R_{N1} & \cdots & & (I-Q_{NN})^{-1}.(\Phi_{NN}-Q_{NN}) \end{bmatrix}$$

where R_{ij} are the $M_i \times M_j$ matrices corresponding to the off diagonal lower block triegonal matrices of $(I-Q)^{-1}.(\Phi'-Q)$

Then from Appendix B

$$\rho((I-Q^A)^{-1}.(\Phi'-Q^A)) = \max_{1\leq i\leq N} \rho((I-Q_{ii})^{-1}.(\Phi' - Q_{ii})) \quad \text{---------- (B.25)}$$

If instead of solving the groups simultaneously each group is solved in sequence (1 to N), then the interaction of torn variables in group i on those in group j, for i<j are not considered and Q(X) is block diagonal.

For this case

$$Q^B = Q(X^*) = \begin{bmatrix} Q_{11} & 0 & \cdots & 0 \\ 0 & Q_{22} & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & & Q_{NN} \end{bmatrix}$$

So that

$$(I-Q^B)^{-1}.(\Phi'-Q^B) = \begin{bmatrix} (I-Q_{11})^{-1}.(\Phi_{11}-Q_{11}) & 0 & \cdots & 0 \\ 0 & (I-Q_{22})^{-1}.(\Phi_{22}-Q_{22}) & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & & (I-Q_{NN})^{-1}.(\Phi_{NN}-Q_{NN}) \end{bmatrix}$$

and

$$\rho((I-Q^B)^{-1}.(\Phi'-Q^B)) = \max_{1\leq i\leq N} \rho((I-Q_{ii})^{-1}.(\Phi_{ii}-Q_{ii})) \quad \text{---------- (B.26)}$$

Therefore by equation B.25 and B.26

$$\rho((I-Q^A)^{-1}.(\Phi'-Q^A)) = \rho((I-Q^B)^{-1}.(\Phi'-Q^B))$$

Consequently by the linear convergence theorum (sub-chapter 6.1.2) the rate of convergence of both these methods are identical. This means that if a system can be partitioned into groups, then each group can be solved separately, without altering the rate of convergence detrimentally, while reducing the problem to several smaller ones. This can be a great advantage as it can reduce the storage requirements and computation effort of a simulator considerably.

6.5.4 Effect of Problem Size on the Reduced Newton Method

It is anticipated from the nature of the reduced Newton method that the computation per iteration will increase linearly with the number of torn variables. However increasing the number of variables per stream will increase the computation in each unit model and increasing the number of units will also increase the computation effort per iteration.

The total computation time is the sum of computation times:-

(a) within the unit models
(b) within the convergence method
(c) within the convergence loop other than (a) and (b)
(d) outside the convergence loop

If the increase of computation time within the models is linear with increasing number of variables and increasing number of units then:-

$$Y = A.X_1.X_2.X_3 + B.X_2.X_3 + C.X_3 + D \qquad \text{---------- (6.27)}$$

(a) (b) (c) (d)

where Y = total computation time

X_1 = number of units in group

X_2 = number of variable per stream

X_3 = number of iterations

so that

A = number of seconds/unit/variable/iteration

B = number of seconds/tear/variable/iteration

C = number of seconds (other than (a) and (b))/iteration

D = number of seconds

In order to verify the assumptions of linearity used in creating equation 6.27 a flotation plant was simulated for sixteen sets of units and stream variables. The schematic flow diagram of this plant for simulation is shown in figure 1.

Consequently by the linear convergence theorum (sub-chapter 6.1.2) the rate of convergence of both these methods are identical. This means that if a system can be partitioned into groups, then each group can be solved separately, without altering the rate of convergence detrimentally, while reducing the problem to several smaller ones. This can be a great advantage as it can reduce the storage requirements and computation effort of a simulator considerably.

6.5.4 Effect of Problem Size on the Reduced Newton Method

It is anticipated from the nature of the reduced Newton method that the computation per iteration will increase linearly with the number of torn variables. However increasing the number of variables per stream will increase the computation in each unit model and increasing the number of units will also increase the computation effort per iteration.

The total computation time is the sum of computation times:-

(a) within the unit models
(b) within the convergence method
(c) within the convergence loop other than (a) and (b)
(d) outside the convergence loop

If the increase of computation time within the models is linear with increasing number of variables and increasing number of units then:-

$$Y = A.X_1.X_2.X_3 + B.X_2.X_3 + C.X_3 + D \qquad \text{---------- (6.27)}$$

(a) (b) (c) (d)

where Y = total computation time

X_1 = number of units in group

X_2 = number of variable per stream

X_3 = number of iterations

so that

A = number of seconds/unit/variable/iteration

B = number of seconds/tear/variable/iteration

C = number of seconds (other than (a) and (b))/iteration

D = number of seconds

In order to verify the assumptions of linearity used in creating equation 6.27 a flotation plant was simulated for sixteen sets of units and stream variables. The schematic flow diagram of this plant for simulation is shown in figure 1.

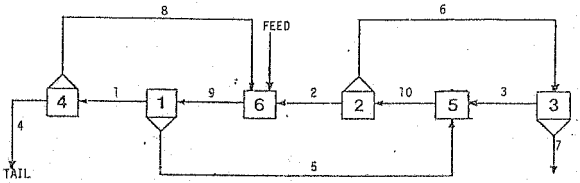


FIGURE 1 FLOW DIAGRAM OF FLOTATION PLANT

Units one to four are banks of flotation cells modelled by the adaption of Sutherland's model (see Part III). In this way the number of cells (units) in each bank can be varied without changing the plant configuration, also this model is non-linear so that the simulation is not too specific. Units five and six are mixing nodes.

In table 3 the sets representing problem size are indexed with a two digit number. The first digit relates to the number of variables per stream and the second digit to the number of units. For each set the number of iterations and total computation time has been found. This data was fitted by means of a least square regression to equation 6.27.

$A = .339 \times 10^{-3}$, $B = .315 \times 10^{-2}$, $C = .[illegible] \times 10^{-1}$, $D = .892$

are the coefficients found for equation 6.27 and the average percentage error for all the points is 1.98%. This extremely low error justifies the assumptions of linearity.

TABLE 3 RESULTS OF FLOTATION PLANT SIMULATIONS

SET	NUMBER UNITS	NUMBER VARIABLES	NUMBER ITERATIONS	COMPUTATION TIME SEC
11	8	7	13	1.60
12	16	7	4	1.11
13	24	7	3	1.12
14	32	7	3	1.26
21	8	13	11	2.08
22	16	13	4	1.42
23	24	13	3	1.43
24	32	13	3	1.61
31	8	19	11	2.54
32	16	19	4	1.71
33	24	19	3	1.76
34	32	19	2	1.71
41	8	25	10	2.91
42	16	25	4	2.05
43	24	25	3	2.08
44	32	25	2	1.97

Several points should be noted:-

(a) The stream variables are categorized into 3 G-classes, 2 S-classes while the number of D-classes are varied from one to four.

(b) The extra variable in each stream is the water flow.

(c) The computation time can vary when running the same system at different times of day, depending on the system loading. This variation can be up to 0.1 seconds, which in this case is about 5% of the total computation time. The error of 2% is well within this r

(d) The number of iteratio[illegible]ired to reach convergence decreases with an increasing number of cells per bank. There are two reasons for this. The first is that in this particular case the initial guess of torn variables (each set to unity)

happens to be closer to the solution when the number of cells is larger. The second is that the non-linearity of the Sutherland model is due to the limitation set on concentrate flow from each cell, so that in a bank with a large number of cells only the first few are effected by this limitation and the remainder behave linearly. This implies that the overall system is becoming more linear and eventually the number of iterations required is two (Set 44), the same number that would be required if the system was linear (one iteration for solids convergence and the second for the water balance).

From the results, equation 6.27 can be written as

$$Y = .639 \times 10^{-3}.X_1.X_2.X_3 + .315 \times 10^{-2}.X_2.X_3 + .132 \times 10^{-1}X_3 + .692$$

with the Average Error = 1.98%. This equation in three variables (X_1, X_2, X_3) cannot simply be diagrammed in two dimensions. However the effect of the third term is small, so if C is set to zero in equation 6.27 then:-

$$Y = AA.X_1X_2X_3 + BB.X_2X_3 + DD \qquad \text{---------- (6.28)}$$

Doing a least squares regression on this equation gives:-

$$AA = .574 \times 10^{-3} \quad BB = .400 \times 10^{-2} \quad DD = .769$$

with the Average Error = 2.34%

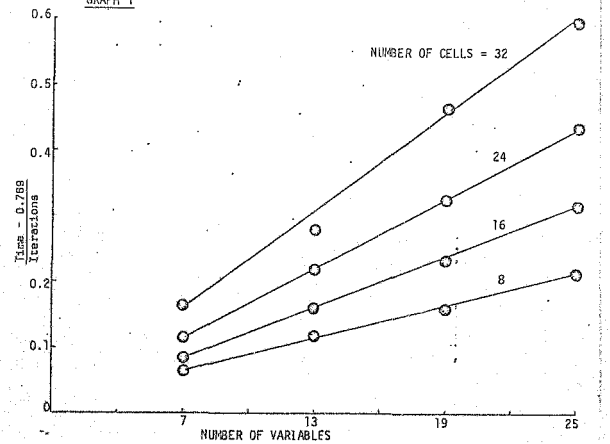
This is also a good fit and now equation 6.28 can be written:-

$$\frac{Y - DD}{X_3} = (AA.X_1 + BB).X_2 = (AA.X_2).X_1 + (BB.X_2) \qquad \text{---------- (6.29)}$$

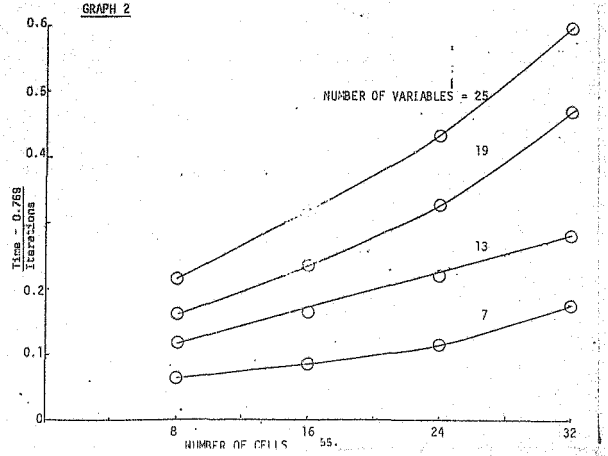
so that if $\frac{Y - .769}{X_3}$ is plotted against either X_2 with X_1 constant or against X_1 with X_2 constant straight lines should result. These plots have been done in Graphs 1 and 2. It can be seen that when the number of cells (X_1) is held constant the plot is very nearly linear however when the number of variables is held constant the plots are slightly curved particularly the top end. On examining equation 6.28 it can be seen that this implies that both AA and BB are independent of X_2 but are slightly dependent on X_1. This dependancy can be partly explained by the earlier discussion of the variation of non-linearity of the flotation cell model with variation in the number of cells.

GRAPH 1

EFFECT OF PROBLEM SIZE - CELLS & VARIABLES



GRAPH 2



It can be concluded both from the regression fits and graphs that the computation effort required by the reduced Newton method increases linearly with problem size. This is a strong point in the favour of this method as the problem size can become very large.

6.6 Comparison of the Reduced Newton Method with Other Convergence Methods

The two convergence methods often used by chemical plant simulators are direct substitution and bounded Wegstein. The analysis in sub-chapter 6.5.1 suggests that the reduced Newton should be faster converging than direct substitution. The bounded Wegstein method does not consider the interaction between variables in different tears, so that if there is more than one tear in the system the reduced Newton should be superior.

To demonstrate the reduced Newton's superiority the same system as used in sub-chapter 6.5.4 is simulated for SETS 11, 22, 33, 44 when using direct substitution and bounded Wegstein as convergence methods. These results are summarized in table 4, and Graphs 3 and 4.

TABLE 4 COMPARISON OF CONVERGENCE METHODS

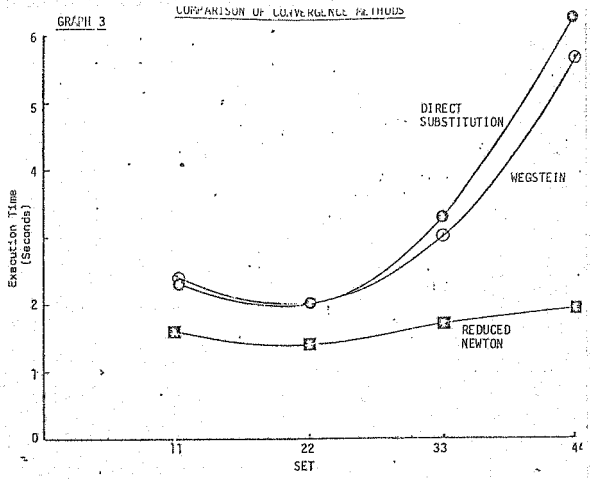
SET	PROBLEM SIZE		REDUCED NEWTON		BOUNDED WEGSTEIN		DIRECT SUBSTITUTION	
	NUMBER UNITS	NUMBER VARIABLES	NUMBER ITERATIONS	TIME SECONDS	NUMBER ITERATIONS	TIME SECONDS	NUMBER ITERATIONS	TIME SECONDS
11	8	7	13	1.60	29	2.39	29	2.33
22	16	13	4	1.42	9	1.87	9	1.97
33	24	18	3	1.76	8	3.04	11	3.40
44	32	25	2	1.97	12	5.75	14	6.43

When the problem size is small, even though the reduced Newton needs far fewer iterations to reach convergence the computational cost of using this method means that the overall saving in computer time is low. However as the problem size increases the great savings that can be made with this method become very apparent.

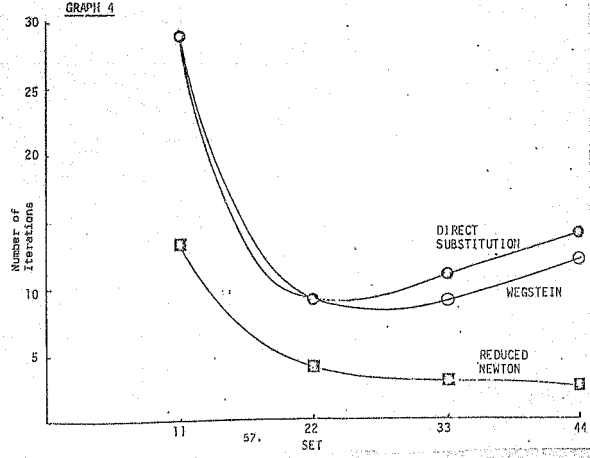
At small problem sizes there is no difference between direct substitution and bounded Wegstein but as the problem size increases the bounded Wegstein becomes more competitive.

GRAPH 3

COMPARISON OF CONVERGENCE METHODS



GRAPH 4



These results are exactly what would be expected from the theory. All three convergence methods will be available to the user of the simulator. In the case of a linear system the reduced Newton method should be used, as global and exact convergence will be achieved.

For non-linear systems only local convergence has been shown and of the three methods, direct substitution is expected to have the largest radius of convergence. However it is recommended that in all circumstances the reduced Newton method should be tried first, if it fails better estimates of the torn streams should be inserted. If convergence is still not achieved, or if it is not possible to estimate the values of the torn streams, either direct substitution, or bounded Wegstein can be tried in combination with the reduced Newton. This combination implies that a few iterations using direct substitution or bounded Wegstein are done and this should so improve the values of the torn streams that it will then be possible for the reduced Newton to rapidly converge to a solution. If this combined approach also fails, and there is only a single tear, the bounded Wegstein may succeed alone and, as a last resort, direct substitution alone can be tried.

All the non-linear library modules, except the adaption of King's flotation model, (see Part III) provide the partial derivatives required by the reduced Newton method. If the user wishes to create his own unit module he should also provide these derivatives as this is generally easily done. In the event that the partial derivatives are not provided and the system does not converge then adding these derivatives may allow convergence.

Another possible way of achieving convergence is to do a dummy simulation with linear models which approximate the real ones and then use these calculated values of the torn streams to start the real simulation.

7 PLANT TOPOLOGY

7.1 Representation of a System of Process Units

A 'process flow diagram' of a plant depicts the equipment and pipes connecting this equipment in graphical form. The pipes are shown as arrows pointing in the direction of material flow.

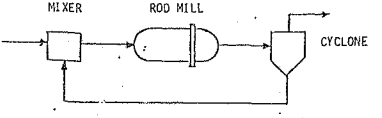


FIGURE 2 PROCESS FLOW DIAGRAM

An example of a simple process flow diagram is shown in figure 2. The process flow diagram must be encoded in numerical form if it is to be used in a computer simulation. This is done in two steps, the first is to construct an 'information flow diagram' and the second is to put this diagram into numerical form. The information flow diagram of the process flow diagram in figure 2 is shown in figure 3.

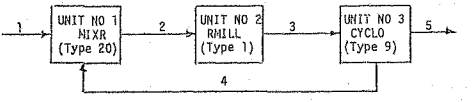


FIGURE 3 INFORMATION FLOW DIAGRAM

The information flow diagram represents a flow of information, via *streams*, between models of unit modules. It is constructed as follows:-

(a) Represent each item of equipment by a square box

(b) Each box is given the name of the corresponding unit module

(c) The flow of information between units are drawn as directed lines (streams), with the arrows indicating the direction of flow

(d) The streams and boxes are numbered separately, usually in ascending order in the direction of flow. The numbering is arbitrary but no two units or streams may have the same number

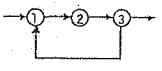
In this example the feed stream has been numbered 1 and the product stream 5. The mixer, rod mill and cyclone have been numbered 1, 2 and 3 respecitvely. In addition the names of the subroutines that represent these units are MIXR, RMILL and CYCLO. Because *FORTRAN* is used as the *programming language* and list processing is not easily done, the unit modules also have *TYPE numbers* to simplify *manipulation*, these are 20, 1 and 9 respectively.

Although the information flow diagram will generally resemble the process flow diagram there will be some stream and units which are not in both diagrams. For example surge tank pumps have no effect on steady state mass balances so can be ommitted. Also, to facilitate the standardization of the unit modules, separation and transformation units will only be allowed one input stream, so that mixers will be the only unit type with more than one input stream. Mixer units are only allowed one output stream. Because of this convention a separator or transformation unit with more than one input in the process flow diagram becomes two units, a mixer plus the separator or transformation unit, in the information flow diagram. This convention is needed to make the calculation of split fractions from outside the unit model possible.

7.2 Conversion to Numerical Form

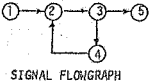
7.2.1 Relationships Between Streams or Units

To represent the information flow diagram in numerical form a Boolean matrix of ones and zeroes, called the adjacency matrix can be used. First the information flow diagram must be converted to a 'digraph' or directed graph which consists of vertices and edges, the edges join two vertices and are directed from one vertex to another. For the example in figure 3 there are two possible digraphs, one with the units as vertices and streams as edges and the other with the streams as vertices and units as edges. This second digraph is called a 'signal flow graph'. Both the digraphs and associated adjacency matrices are shown in figure 4.



DIGRAPH
(UNITS AS VERTICES)

TO VERTEX	FROM VERTEX 1	2	3
1	0	0	1
2	1	0	0
3	0	1	0



SIGNAL FLOWGRAPH
(STREAMS AS VERTICES)

TO VERTEX	FROM VERTEX 1	2	3	4	5
1	0	0	0	0	0
2	1	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	0
5	0	0	1	0	0

FIGURE 4 DIGRAPHS AND THEIR ASSOCIATED ADJACENCY MATRICES

The adjacency matrices in figure 4 are very sparse and storing all the zeroes is wasteful. If the vertices which are joined to each vertex are listed, then these lists can be stored in the 'arc operator matrix' as shown in figure 5. It is very unlikely that any vertex will be joined to all the others therefore the number of columns in the

arc operator matrix will be less than the number of vertices and hence the storage required is reduced.

FROM VERTEX			
1	2	0	0
2	3	0	0
3	4	5	0
4	2	0	0
5	0	0	0

FIGURE 5 ARC OPERATOR MATRIX (FOR SIGNAL FLOW GRAPH)

However this matrix is also sparse and a better way of storing the lists of vertices reached is by storing the lists in a string called the 'stream vector' and then having an index to distinguish the lists.

(2 3 4 5 2) Stream Vector

(1 2 4 5 5) Index

FIGURE 6 STREAM VECTOR AND ITS INDEX.

The stream vector and its index for the example in figure 3 is given in figure 6. Beginning with vertex 1 in the arc operator matrix, the non zero elements in row 1 are inserted into the first position of the stream vector. When all the non-zero elements have been inserted the final position filled is noted in the 'index' as its first element. This procedure is repeated for each row of the arc operator matrix and the elements are catenated to those already in the stream vector. For each vertex the position of the last element stored in the stream vector is noted in the index. Jain and Eskman (1972) claim storage savings of 80% to 95% on the adjacency matrix.

7.2.2 Relationships Between Streams and Units

For any simulator to work the input and output stream numbers for every unit must be known. The digraphs with their adjacency matrices do not provide this data. One way of describing the interconnection between streams and units is by means of the 'incidence matrix'.

		STREAM NUMBER				
		1	2	3	4	5
UNIT NUMBER	1	+1	-1	0	+1	0
	2	0	+1	-1	0	0
	3	0	0	+1	-1	-1

FIGURE 7 INCIDENCE MATRIX

The incidence matrix for the example in figure 3 is shown in figure 7. Here '+1' in position i,j shows that the stream number represented by column j is an input to the unit number represented by row i. Conversely '-1' indicates the stream is an output. In this example stream number 2 is an output from unit 1 and is an input to unit 2, therefore the column for stream 2 has '-1' in row 1 and '+1' in row 2.

The two main problems with using the incidence matrix are:-

(a) Storage is wasted, because it is sparse

(b) The stream number corresponding to the first, second or third output from the unit is not provided. (The unit model must know which stream number is concentrate, middling or tail).

Both these difficulties are overcome by the Stream Connection Matrix (SCM). This matrix is an array with three elements per row, the first entry is the stream number, the second and third are the unit numbers of the units from which the stream comes and to which it is going, respectively. The stream connection matrix for the information flow diagram of figure 3 is given in figure 8.

STREAM NUMBER	FROM UNIT NUMBER	TO UNIT NUMBER
1	0	1
2	1	2
3	2	3
4	3	1
5	3	0

FIGURE 8 STREAM CONNECTION MATRIX

The conventions that: feed and product streams come from and go to unit 0 respectively; and the order in which the stream numbers appear, starting with row 1, is the order of output streams for each unit; are used. So in this example stream 4 is output 1 and stream 5 is output 2 from unit 3.

7.2.3 Relationships Between Unit Number and Unit Type

Both the stream connection matrix and the incidence matrix do not relate the unit number to the unit type and unit name. A popular method of transferring all this information is the 'Process Matrix'.

UNIT NUMBER	UNIT TYPE	UNIT NAME	ASSOCIATED STREAMS
1	20	MIX	1 4 -2 0
2	1	RMILL	2 -3 0 0
3	9	CYCLO	3 -4 -5 0

FIGURE 9 PROCESS MATRIX

The process matrix for the example is given in figure 9. The ASSOCIATED STREAMS are given as a list related to each unit. The input stream numbers, with positive signs, are given first in the order of inputs and these are followed by the output stream numbers, with negative signs, in the order of outputs.

7.2.4 Selection of Numerical Form

An advantage of using the process matrix is that it is easily understood by the engineer. The disadvantages are that it contains blanks (zero's) and it is not very flexible. The inflexibility is due to the fact that each row of the matrix is not independent of the others so that when changes in stream number or stream destination are wanted several changes must be made and this can cause errors. Using the stream connection matrix each row is independent and changes can readily be made. The stream connection matrix also contains no blanks. However it does not relate unit numbers to unit types but this can easily be done by another array. In the simulator the subroutine name is related to the unit number and the unit type number is allocated by the computer to each name the first time it is presented during input.

The stream connection matrix and unit name array have been chosen in preference to the process matrix for data input. The stream vector plus index is used for internal storage of the signal flow graph. In fact most of the variable length lists are stored as vector strings with index.

8 PARTITIONING

8.1 Definitions of: Partitioning, Groups, Nodes, Maximal Nets

In both linear and non-linear systems grouping of units into sub-plants with no feed-back of material have been discussed in sub-chapters 5.4.1 and 6.4.3. If the system is divided into the maximum possible number of sub-plants then this is called 'Partitioning', as opposed to 'Grouping' where the system is simply split into sub-plants.

In the case of the partitioned system, each sub-plant must be either a single unit or several units which are linked by recycles in such a way that every unit has at least one of its inputs and one of its outputs connected to other units in the sub-plant. Here single unit sub-plants are called 'simple nodes' and multiple unit sub-plants are called either 'complex nodes' or 'maximal cyclic nets'.

It was shown in sub-chapter 6.4.3 that grouping or partitioning the system so that each node is solved sequentially in the order of mass flow does not degrade the convergence behaviour of the system even though the problem size is then reduced considerably.

8.2 History of Partitioning Algorithms

Methods for partitioning have been developed based on 'path searching' and Boolean operations performed on the adjacency matrix.

Sargent and Westerberg (1964) trace the information flow of a digraph backwards until a vertex (unit) that has been encountered before is encountered again. All vertices traced between these two encounters form a cyclic net, and are lumped into a pseudo vertex. The tracing is repeated until no more cyclic nets are found and all vertices have been traced. This method is difficult to program and Sargent and Westerberg (1964) use list processing to program their algorithm.

Christensen and Rudd (1969) simplified this algorithm by initially deleting vertices that have no input and vertices that have no output, and so cannot be included in a cyclic net. They also added forward tracing.

Forder and Hutchison (1969) programmed a modification of the method of Christensen and Rudd (1969). This program is very complex.

Steward (1965) proposes a path searching method similar to that of Sargent and Westerberg. However his search is conducted along an adjacency matrix of the digraph. This simplifies programming.

Billingsley according to Kahat and Shacham (1973b) used the adjacency matrix to search for the shortest path that included all the elements of the maximal cycle net.

A method using powers of the adjacency matrix has been suggested by Harary (1959) and improved by Norman (1968). Himmelblau (1966) applied the method of powers of an adjacency matrix to chemical process flow-sheets. He also showed that the union of the 'reachability matrix' (the sum of the powers of the adjacency matrix, from one until there is no further change in the elements) and its transpose presents an ordered picture of the cyclic nets. However if the system is not very sparse a full matrix may result.

Lewin and Himmelblau (1970) reduced the storage requirements by storing the results of the matrix operation in the original locations.

Kehat and Shacham (1973b) have proposed an algorithm which reduces storage by using an index array and combines the best features of Himmelblau (1966) and Norman (1968).

Jain and Eakman (1972) also use powers of the adjacency matrix but suggest path searching to seperate the cycles.

Batstone and Prince (1970) compared the efficiency of the path searching and basic adjacency matrix method for a particular problem and found the path searching algorithm superior.

8.3 Selection of the Partitioning Algorithm

For the simulator, tear streams must be found, and one of the tearing criteria needs to have all the cycles in each complex node. As the cycles are required for the tearing algorithm, using these cycles, to find the maximal cyclic nets, will mean that the cycle finding algorithm will serve a dual purpose, and so save computation time.

The algorithm for the enumeration of all the cycles in a system is described in chapter 9.

Combining the cycles into cyclic nets is easily done once it is recognised that if any two cycles have one or more common streams then both cycles belong to the same complex node, and if there are no common streams then the cycles belong to different complex nodes. (If units were allowed to simultaneously have more than one input and more than one output then the second part of the above statement would not be true, because in this case it would be possible for two cycles to have a common unit, implying their belonging to the same maximal net, while not having a common stream).

The algorithm operates as follows:-

A cycle is chosen and each stream in this cycle is tested to check whether it is also a member of another cycle. If a stream is found to be common to one or more of the other cycles then these cycles are added to the node. The new streams which are added to the node in this

way are also tested for membership of cycles not already in the node.

When all the stream members in the node have been tested then the list of cycles searched for a cycle which is not yet a member of a node. If one is found the procedure is repeated to obtain another maximal net.

For the logic flow diagram see sub-chapter 13.3.2, ALGORITHM 2.

9 DECOMPOSITION

When an engineer is solving a material balance he uses his intuition to work out a calculation scheme. An improper scheme often results in serious numerical difficulty during the course of solution. Whenever failure is encountered the engineer must reexamine his scheme. Generally this examination is time consuming and such use of a computer is inefficient and suggests a more logical way of ordering the calculation is needed.

In the iterative procedure to solve a complex node each unit must be evaluated once per iteration. This implies knowing all the inputs to each unit before its outputs can be determined. In a complex node this is impossible and it is necessary to guess some inputs to start the iteration. These guessed inputs are called 'tear streams'.

In any system there are many permutations of the order in which the unit modules can be evaluated, and depending on the order chosen different tear streams will be needed. Just as choosing the order fixes the tear streams, choosing tear streams which render the system acyclic will fix the order of unit evaluations. This selection of tear streams to render the system acyclic is called 'decomposing' the system.

The decomposition must be done so that the resulting calculation will converge to a solution with the least possible computer time and storage.

9.1 History of Decomposition Procedures

The problem of finding the optimal decomposition has been approached differently by many investigators. The most usual criteria used have been based on:-

(a) Tearing the minimum number of streams.

(b) Tearing the minimum number of variables (same as (a) when all the streams have the same number of variables).

(c) Tearing so as to obtain the minimum sum of weighting factors, where each stream is weighted by the user.

One of the earliest discussions of such minimizations is by Rubin (1962). He argued that the ordering which requires the lowest number of torn variables is the most desirable and developed an algorithm to achieve this. He started with an arbitary ordering, then interchanged the units in succession until any further interchanges increased the number of torn variables. However he discovered that as this is not an exact algorithm in some instances only a local minimum was found.

This problem was first successfully solved by Sargent and Westerberg (1964). They used dynamic programming to arrive at the desired order.

This algorithm unlike that of Rubin (1962) finds a global minimum. Sargent and Westerberg (1964) also introduced the concept of 'inelegible streams' and this was later developed by Christensen and Rudd (1969). The methods of Rubin (1962) and Sargent and Westerberg (1964) approach the problem from the point of view of unit ordering.

Other methods do not consider unit ordering during the search phase, rather unit orderings are obtained only after the set of tear streams has been found. These methods can be further subdivided into two classes:

(a) Those that require the cycle/streams matrix*

(b) Those that do not

*The cycle/stream matrix is the lists of streams making up each cycle.

Among the first type are the algorithms of Lee and Rudd (1966) and Upadye and Grens (1972). Lee and Rudd (1966) propose an exact algorithm based on the concept of 'containment' of streams and sets of streams in other streams and sets of streams. If some stream 'opens'* all the cycles opened by another stream and has a smaller weight, then the second stream is contained in the first and need no longer be considered, therefore it is eliminated.

* 'Opening' a cycle means that the stream being tested as a possible tear stream is a member of the cycle and using it will make that cycle acyclic .

This same concept can be applied to sets of streams.

For systems with many streams a large number of combinations must be tested for containment, resulting in a severe combinational problem. Upadye and Grens (1972) have also presented an algorithm of the first type which is based on dynamic programming. This application is different to the earlier one of Sargent and Westerberg (1964) as advantage is taken of the important structural properties of the cycle/stream matrix. Upadye and Grens (1972) test the combinations of cycles opened, while Sargent and Westerberg (1964) test the combinations of unit orderings which is a far larger problem. In the examples given by Upadye and Grens (1972) their algorithm is proved to be superior to that of Lee and Rudd (1966) and Sargent and Westerberg (1964). The major disadvantage of this method is the large amount of storage required. ($3 \times 2^{C-1}$ where C = number of cycles).

Amongst the methods that do not require the cycle/stream matrix are those by Christensen and Rudd (1969), Christensen (1970) and Barkley and Motard (1972). Christensen and Rudd (1969) and later Christensen (1970)

introduced and developed the concept of 'indexing' and further developed the idea of ineligible streams first introduced by Sargent and Westerberg (1964). The algorithms of Christensen and Rudd (1969) are heuristic in nature: Whenever an ordering is produced it is optimal, however, as pointed out by the authors in some cases it will not produce an ordering at all.

Christensen's algorithm is exact and can be used for ordering algerbraic equations as well. Both these algorithms are combinational and may become cumbersome for large processes.

Berkley and Motard (1972) used signal flow graphs, while Pho and Lapidus (1973) have also presented a branch and bound method which guarantees optimality if the signal flow graph reduction fails. Their method can be applied to both the cycle/streams matrix and the adjacency matrix.

8.2 Choice of Tearing Criterion

The criteria for choice of tear sets discussed so far are not sufficiently descriptive of the efficiency of solution. A better definition of an optimal decomposition objective function may be arrived at by realizing that the real measure of computational efficiency is the rate of convergence of the decomposed network.

In sub-chapter 8.1.2 it has been shown that the spectral radius of the Jacobian of the decomposed network interfaced with the convergence algorithm is a measure of the rate of convergence. Therefore given a choice of alternative tear sets it will be best to select the set which minimizes the spectral radius (i.e. minimizing $\rho(G'(X^*))$).

From sub-chapter 8.4

$$G'(X^*) = (I-Q)^{-1}(\Phi'-Q) \qquad \text{---------- (8.1)}$$

and the convergence method is determined by the choice of Q.

If Q = 0 the method is Direct Substitution

Q = diag (Φ') Bounded Wegstein

Q = block diag (Φ') Reduced Newton

and Q = Φ' Newton

(Note: $\Phi' \equiv \Phi'(X^*)$ and $Q \equiv Q(X^*)$)

For the full Newton method substituting $Q = \Phi'$ into equation 6.22 gives $G'(X^*) = 0$ and hence $\rho(G'(X^*)) = 0$, for every tear set and is therefore independant of the choice of tear set. All the other methods mentioned are effected by the choice of tear set, hence it is required to find the tear set which minimizes $\rho([I-Q)^{-1}(\Phi'-Q])$. This is not easily done because one must be at the solution before $G'(X^*)$ can be found and $\rho(G'(X^*))$ evaluated for each tear set.

This generalized criterion does not appear to have been considered before.

9.2.1 Effect of $\rho(\Phi')$ on $\rho(G'(X^*))$

What has been done is to try and find a tear set which minimizes $\rho(\Phi')$, based on the assumption that if the convergence characteristics are improved for direct substitution ($G'(X^*) = \Phi'$) then it is more than likely that they will improve for other convergence methods (Westerberg and Edie (1971)).

Genna and Motard (1975) suggest by analogy to linear systems behaviour under direct substitution that the sensitivity of the decomposed network to the torn variables should be minimized and claim that as the Newton method involves a local linearization of the system, Φ' is the same as the sensitivity matrix and so should have eigenvalues less than one and be as small as possible. In other words they are suggesting that if a tear set is chosen for which $\rho(\Phi')$ is minimized then the rate of convergence of the Newton method will be maximized.

Genna and Motard (1975) use an adaptive search for the tear set with the minimum spectral radius. The Jacobian is calculated for each tear set every few iterations, and then the following iteration is done using the tear set with the smallest spectral radius. This procedure may be more time consuming than the solution with any arbitrary tear set since it requires the evaluation of the Jacobian for several tear sets. In practice they use Broyden's quasi-Newton method and suggest that instead of calculating the eigenvalues of each Jacobian, computation effort can be saved if the tear set with the minimum trace $(J^2) = \sum_{i=1}^{n} \lambda_i^2$ is chosen. (J is the approximation to the Jacobian, see sub-chapter 3.2.3).

Westerberg and Edie (1971) have also suggested an adaptive search for the output set (similar to tear set, but comprises a set of variables used when solving sets of equations) which gives the best convergence properties. They use the norm of Φ' to give a return function which is presumed to minimize $\rho(\Phi')$.

The suggested norms are the l_1 and l_2 norms:

$l_1 \equiv$ Column Sum $\quad ||\phi'||_1 = \max_j \left(\sum_{i=1}^{n} |\phi_{ij}| \right)$

$l_2 \equiv$ Sum all elements $||\phi'||_2 = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} (\phi_{ij})^2 \right)^{\frac{1}{2}}$

The output set which minimizes the chosen norm is selected. However they do warn that minimizing the norm does not necessarily minimize $\rho(\phi')$, but claim that this is a good indication that such a system might have relatively reasonable convergence characteristics when compared to other variations with larger bounds.

In a later paper a return function of the form:

$\max \prod_{i=1}^{n} |\phi_{ii}|$ is recommended, which they claim selects the output set so that $\rho(\phi')$ is favourably effected and likely to be minimized.

Orbeck and Crowe (1971) recognized that the dominant eigenvalue of the Jacobian of the decomposed network controls the rate of convergence. Their method concentrates on identifying the dominant value for extrapolation to the solution.

All these authors have presumed to minimize $\rho(G'(X^*))$ by minimizing $\rho(\phi')$. This is not necessarily true, but consider the situation when ϕ' is non-negative.

(a) Decreasing any element not common to both ϕ' and Q decreases the corresponding element of $(\phi'-Q)$, while $(I-Q)^{-1}$ is unaltered. When $\phi' \geq 0$ it has been shown in Appendix A that both $(\phi'-Q)$ and $(I-Q)^{-1}$ are non-negative. This implies that whenever some non-common element of ϕ' is decreased then some elements of $(I-Q)^{-1}.(\phi'-Q)$ are similarly decreased. Therefore decreasing an element of ϕ' and consequently $\rho(\phi')$ will not increase the spectral radius of $G'(X^*)$.

(b) Decreasing any element common to both ϕ' and Q does not effect $(\phi'-Q)$ and decreases some elements of $(I-Q)^{-1}$ as:

$$(I-Q)^{-1} = \lim_{k \to \infty} \sum_{i=0}^{k} Q^i$$ (see Appendix A)

Therefore some elements of $(I-Q)^{-1}.(\phi'-Q)$ will decrease and the spectral radius of $(I-Q)^{-1}.(\phi'-Q)$ will not increase.

Hence it can be concluded that when ϕ' is non-negative then $\rho(\phi')$ and $\rho(G'(X^*))$ are directly linked and choosing a tear set which minimizes $\rho(\phi')$ will also minimize $\rho(G'(X^*))$.

If ϕ' is non-negative and some elements are reduced so that they become small negative numbers then as the eigenvalues of a matrix vary continuously with the elements (Brauer (1963) p27) it is expected that the relationship between $\rho(\phi')$ and $\rho(G'(X^*))$ will still hold.

Therefore in the context of process plants where it is expected that the Jacobian will contain only a few small negative elements, if any, one can be reasonably confident that choosing the tear set which minimizes $\rho(\phi')$ will also minimize $\rho(G'(X^*))$.

9.2.2 'No Double Tear' Criterion

Using $\rho(\phi')$ directly as a measure of convergence has the difficulty that the solution must be known before $\rho(\phi')$ can be calculated and compared for each tear set. This may be suitable if several simulations of the same system are to be done, but in the 'one-off' case a criterion which can be determined in advance is needed.

Upadye and Grens (1975), working on direct substitution have demonstrated via the "Replacement Rule', that decompositions can be grouped into families, all of whose members have the same convergence rate. These families are sub-divided into categories which contain 'double tear' decompositions and those which don't. To understand the meaning of 'double tear' decompositions consider:-

A valid decomposition is one which renders the system acyclic. To achieve this every cycle must be opened at least once.

A decomposition can be described using a cycle vector where each element of this vector represents the number of times that cycle is opened. Therefore a cycle vector for which each element is unity is a valid decomposition and is called a 'no double tear' decomposition. All other valid decompositions will have at least one double tear.

Upadye and Grens (1975) claim that all decompositions in a single family have the same cycle vector and conversely (based on extensive examination of cases) that the vector uniquely corresponds to a single family. Therefore all 'no double tear' decompositions belong to the same family and as such have the same convergence rate.

Upadye and Grens (1975) have gone further and claim that $\rho(\phi')$ increases if a cycle which is already torn at least once is torn again. This is

proved for ϕ' non-negative and claimed to be true in general. The arguments used for the general proof are based on the fact that the Jacobian derived from chemical processes is expected, from the necessarily positive sensitivities arising from material and energy conservation, to be positive or to have only a few small negative elements. Further for all simulations tested they found that a double tearing of one, or more, of the streams led to degraded convergence behaviour. The processes tested ranged from very simple to moderately complex and involved most types of interactions between variables and interconnections between units that are encountered in chemical plants.

From the above it can be concluded that choosing a tear set which involves no double tears means that $p(\phi')$ for the decomposition is smaller than that for any decomposition containing a double tear. This reduces the problem of choosing a tear set from all possible sets to choosing it from those with no double tears.

If direct substitution is used as the convergence method then all members of the 'no double tear' family have the same convergence rate, however this may not be true for other convergence methods. When the convergence rate is the same for two decompositions the one which requires the least computational effort is the one with the least torn variables. For lack of a better criterion for choosing amongst the 'no double tear' decompositions it has been decided to resort to using either :-

(a) Minimum Torn Streams

(b) Minimum Torn Variables

8.2.3 Implementation of Tearing Criteria

A possible way to choose a tear set is for the user to weight each stream and then select the decomposition on the basis of the set with the minimum weighted sum. This is a very flexible method and it is possible to include all previously mentioned criteria as follows:-

Let the weighting factor for stream k be W_k then for the criterion:- (see Appendix C).

(a)	Minimum torn streams	$W_k = 1$	for all k
(b)	Minimum torn variables	$W_k = NV_k$	for all k
(c)	Minimum double tears	$W_k = m_k$	for all k
(d)	Minimum double tears plus minimum torn streams within family	$W_k = A.m_k+1$	for all k and A>NC-2
(e)	Minimum double tears plus minimum torn variables within family	$W_k = B.m_k+NV_k$	for all k and B>NC.NV_{max}-2

where

NV_k = number of variables in stream k

NV_{max} = maximum number of variables in any stream

NC = number of cycles

m_k = number of cycles which contain stream k

Using this 'minimum weighted sum' criterion has the further advantage that it is likely to be possible to fit any new criterion developed with it.

9.3 Cycle Finding Algorithms

Having decided that some form of the 'no double tear' criterion is the best available for choosing a tear set at present it is necessary to find m_k (the number of cycles which contain stream k). To do this all the cycles in the system must be found first.

9.3.1 Methods Using the Adjacency Matrix

There has been a constant development of methods for enumerating cycles. Harary (1959, 1960) showed that if the adjacency matrix is raised to the power p each element will represent two vertices that are separated by p steps, and elements on the diagonal will represent vertices that are part of a recycle not of p vertices. The problem arises when two or more cycles of size p belong to the same system and so appear simultaneously on the diagonal. For such cases Norman (1965) suggests removing one of the vertices which have appeared on the diagonal from the original adjacency matrix and then putting this matrix to the power p. The members of the cycle from which the vertex was removed will now not appear on the diagonal and so in this way the cycle is identified. If there were only two cycles of size p then the elements remaining on the diagonal will be the vertices of the other cycle, for N_p cycles of size p this procedure must be repeated N_p-1 times.

Ledet and Himmelblau (1970) reduced the storage requirements of this method by storing the results of the matrix operation in the original storage locations.

Kehat and Shacham (1973b) overcame most of the storage problems by storing the adjacency matrix and its powers in the form of an 'index matrix'. A program of this type was written but the problem of separating similarly sized cycles still existed, and the method suggested by Norman (1965) is very involved.

5.3.2 Methods Using Path Searching

'Path searching' is a procedure which takes some stream as a root stream and then traces a path along the direction of material flow adding each stream in the path to a list sequentially. 'Backtracking' is the process of retracing ones steps back along the path towards the root stream.

The first example of this technique was presented by Tiernan (1970) and he claimed his algorithm was theoretically as efficient as possible. This is not true but Tiernan's (1970) approach did lay a base from which Weinblatt (1972) and Tarjan (1973) could develop improved path searching with backtracking algorithms.

Further improvements which restrict the backtracking procedure to fruitful paths only, have been presented by Read and Tarjan (1975) and Johnson (1975).

Mateti and Deo (1976) have reviewed algorithms for enumerating all cycles of a graph. In table 5, time and space bounds have been extracted for each method.

TABLE 5 TIME AND SPACE BOUNDS OF CYCLE FINDING ALGORITHMS

AUTHOR	TIME BOUND	SPACE BOUND	METHOD
Danielson (1968)	$n\ (const)^n$	$n\ (const)^n$	power adj mat
Tiernan (1970)	$n\ (const)^n$	n+e	backtrack
Weinblatt (1972)	$n\ (const)^n$	n.c	backtrack
Tarjan (1973)	n.e.c	n+e	backtrack
Johnson (1975)	(n+e).c	n+e	backtrack
Read & Tarjan (1975)	(n+e).c	n+e	backtrack

n = number of vertices e = number of edges (streams) c = number of cycles

The number of time units consumed by these algorithms depends on the intricacy of the structure of the graph. The worst case time bounds given in table 5 are in general pessimistic. Thus if T_A and T_B are upper bounds on the time required by two algorithms A and B, respectively, and if $T_A < T_B$ it cannot be said with certainty that algorithm A is faster than algorithm B. In general the worst case graphs for two algorithms are not the same.

Mateti and Deo (1976) conclude that, of all the algorithms analysed, the Johnson algorithm, on time, is the asymtotically fastest algorithm. The success of this backtrack algorithm is due to a very effective pruning technique which avoids much of the fruitless searching present in earlier algorithms.

9.3.3 Selection of Cycle Finding Algorithm

Because of the above discussion Johnson's (1975) algorithm will be used for enumerating the circuits of a graph. The essence of this algorithm's operation is:-

> Elementary circuits are constructed from a root stream S. At each step a stream is added so extending the elementary path. To avoid duplicating circuits no stream with a stream number greater than S can be used. Further a stream v is blocked when it is added to some elementary path beginning in S. It stays blocked as long as every path from v to S intersects the current elementary path at a stream other than S.

A more detailed logic flow diagram of this algorithm is given in Part II (sub-chapter 13.3.1 - ALGORITHM 1).

9.4 Tearing Algorithms

Once the cycle/stream matrix has been found it is possible to partition the system and weight the streams according to some 'no double tear' criterion. Having achieved this the next step is to find a valid tear set with the minimum weighted sum.

The history of these tearing algorithms have already been discussed (sub-chapter 9.1) and there appears to have been two main lines of attack:-

(a) Dynamic Programming technique of Sargent and Westerberg (1964) and Upadye and Grens (1972).

(b) Graph Simplification and Two Way Edge Reduction. The most advanced algorithm of this type being by Pho and Lapidus (1973).

Upadye and Grens (1972) compare their algorithm with that of Sargent and Westerberg (1964) and show that theirs is the more efficient. While Pho and Lapidus (1973) have incorporated the best of all previous algorithms of that type into theirs.

9.4.1 Selection of the Tearing Algorithm

In these circumstances there are only two competing algorithms, that of Pho and Lapidus (1973) and that of Upadye and Grens (1972).

Pho and Lapidus's method can be split into three parts

(a) Basic Tearing Algorithm (BTA)

(b) Two Way Edge Reduction

(c) Branch and Bound

The BTA simplifies and reduces the graph by using the "Ineligibility Theorum' and then if the graph is not yet acyclic, two way edge reduction is used. If after applying both BTA and two way edge reduction the graph is not acyclic then a branch and bound method is used to complete the decomposition. Pho and Lapidus (1973) claim this technique is very efficient and can be used on large systems.

However the 'Ineligibility Theorem' can be stated as follows:-

'If stream vertex S_m has a weighting factor W_{S_m} which is larger than or equal to the weighted sum of its immediate successors $\Gamma(S_m)$ or its immediate precessors $\Gamma^{-1}(S_m)$ then it is ineligible'.

The proof of this theorem revolves around the fact that the set of cycles that pass through S_m must also be contained in the set of cycles that pass through $\Gamma(S_m)$ or $\Gamma^{-1}(S_m)$.

If the no double tear criteria, which minimises the number of torn streams or variables (sub-chapter 9.2.3 criterions d, and e), are used then:-

$$W_{S_k} = A.m_{S_k} + X \qquad \text{where } X>0,\ A>0 \qquad \text{(see Appendix C)}$$

Setting Γ as the set of streams representing either $\Gamma(S_k)$ or $\Gamma^{-1}(S_k)$ and n_Γ as the number of streams in $\Gamma(S_k)$ or $\Gamma^{-1}(S_k)$ then:-

$$\sum_{i\in\Gamma} W_{S_i} = \sum_{i\in\Gamma} (A.m_{S_i} + X)$$

$$= \sum_{i\in\Gamma} A.m_{S_i} + \sum_{i\in\Gamma} X$$

$$= A.\sum_{i\in\Gamma} m_{S_i} + n_\Gamma.X$$

From the proof used by Pho and Lapidus (1973) $m_{S_k} \leq \sum_{i \epsilon \Gamma} m_{S_i}$

$$\therefore \quad Ws_k - \sum_{i \epsilon \Gamma} W_{S_i} = A.m_{S_k} + X - A. \sum_{i \epsilon \Gamma} m_{S_i} - n_r.X$$

$$= A(m_{S_k} - \sum_{i \epsilon \Gamma} m_{S_i}) + X(1-n_r)$$

$$\leq X(1-n_r)$$

$$< 0 \qquad \text{when } n_r > 1$$

$$\therefore \quad Ws_k < \sum_{i \epsilon \Gamma} W_{S_i} \qquad \text{when } n_r > 1$$

Then when using this type of tearing criterion stream S_k can never be 'inelegible' (except when $n_r = 1$) and consequently the BTA can only simplify the graph but never reduce the number of cycles.

Relying on two way edge reduction and the branch and bound method to reduce the graph is a trial and error procedure. Therefore the exact dynamic programming approach should be more reliable, and generally more efficient.

The method of Upadye and Grens (1972) has been chosen.

9.4.2 Description of the Tearing Algorithm

The algorithm of Upadye and Grens (1972) has been tested and found to be very fast, its only drawback is the large storage requirements, 3.2^{NC-1} where NC = number of cycle in the complex node. If NC is greater than about 15 the method becomes impractical (≥50 000 storage locations). However on examination of the Denver Flowsheets (1982) for mineral processing plants no system was found to have more than five cycles per complex node and so storage is unlikely to be a problem. If the situation should arise where either the number of cycles per complex node is very large or the computer very small then the user must supply the tear set or alternatively the calculation order.

Upadye and Grens's (1972) algorithm can be described as follows:-

The problem is viewed as having two definite terminal states, initial and final. The initial state corresponds to the original system with no cycles opened. The final state corresponds to the acyclic system, with all cycles opened. In between these two terminal states exist many intermediate states, corresponding to the various combinations of cycles opened. The solution to the problem may be considered to progress through these states.

These states are represented as vertices on a graph and the streams selected to be torn are arcs joining the vertices. Associated with each arc is a cost (or weight).

In general there exists many paths from the initial to final states, the optimum one is the path with the least sum of costs for its arcs, that is the least total cost. For any intermediate state, the path through this from initial to final states can be divided into two sub-paths. One sub-path from the initial state to the intermediate state and the second sub-path from the intermediate state to the final state, and each sub-path of the optimum path must itself be an optimum path between the states it connects. (If this were not so one of the sub-paths could be replaced by an alternate sub-path with a smaller cost, and this would give rise to a path between the terminal states with a smaller total cost than the original path, in violation of the assumption that the original path was optimum).

These conditions are precisely those which apply to dynamic programming.

E(q) is defined as the cost to reach state q and M(q) the optimum set of streams to reach state q. Now suppose stream l is combined with M(q) to yield a new state p. This state p may or may not have been reached before from some state other than q. If it has not previously been reached then the current optimum set of streams and cost for state p are:-

$$M(p) = M(q) \cup l$$

$$E(p) = E(q) + W_l \quad \text{where } W_l \equiv \text{cost of stream } l$$

If state p has been reached previously then a value for E(p) already exists. This value is checked against the new value $E(q) + W_l$. If $E(p) \leq E(q) + W_l$ then the set $M(q) \cup l$ is no better than the previous set M(p). However if $E(p) > E(q) + W_l$ then E(p) is replaced by $E(q) + W_l$ and M(p) by $M(q) \cup l$. This same procedure is repeated for state q and the next stream (l+1). When all available streams have been considered the procedure is repeated for the next state (q+1).

The amount of computation is reduced by nearly half by a simple modification. In reaching state q every permutation of the streams in a possible path will be tested by the above procedure. Each of these permutations have the same cost and set of form streams, therefore there is a duplication of effort. This duplication is avoided by indexing the current optimum path to reach a particular state with the stream number of the last stream in the path, and then extending the path from this state only with streams which have a stream number larger than the index for the state. This ensures that only one of the possible permutations are tested (i.e. the one for which the stream numbers on the path increase from smallest to largest).

So,starting from the initial state,new states are reached by tearing each stream in turn. Then from each of these states each stream with a larger stream number than the index for the state is torn. These new states are checked to establish whether the new path has a lower cost than the old one. If it has the new path, cost and index replace the old. This is repeated until the final state is reached. The path thus found will be the optimum having the minimum total cost.

See sub-chapter 13.3.6 for the logic flow diagram of this algorithm (ALGORITHM 6).

10 SUMMARY OF PART I

Basic to every process is the understanding and analysis of the system. This has been made possible by computer simulation with the advent of quantitative models and digital computers. However, so far, only flotation and comminution circuits have been simulated, and these with a limited choice of plant configuration.

Chemical plant simulators have been evolving since 1955 and the only successful ones have been those which are easy to use, have a modular structure and have a rigorous physical property package. The simulators have been used at every stage of a project with different levels of complexity. That is from research, through pilot plant, design of the full scale plant and simulation of the full scale plant during production. Large savings in cost have been made by using simulators, however, the trap of being over-rigorous should be avoided.

A simulator can be structured to operate in either design or simulation modes. For the former mode, feed and unit operating parameters are calculated to provide the desired products, while the latter mode uses the feeds and unit parameters to calculate products. The simulation mode is the more stable, numerically, (i.e. a solution can usually be found) and consequently design is done by iterated simulation.

The computer can be programmed to use either an equation solving approach or a modular approach, where an executive controls the flow of information to each module (which corresponds to a unit model) and, which in turn solves its own equations to find related outputs. This latter structure is the one most often used as it is flexible, efficient in the use of manpower, easy to maintain and suitable for programming.

Having chosen this modular approach the executive program must have a method of solving the overall mass balance based on the inputs to and outputs from each module. This is readily done for a sequential process (i.e. no recycles), but, for a plant with recycles a simultaneous or sequential method can be used. For the simultaneous method the units are linearized about an assumed steady state and the resulting equations solved simultaneously. A new steady state is assumed and the process repeated until no further changes are possible. For the sequential method the values of variables in the recycle streams are guessed and then the values of variables in all other streams are found by serially calculating the outputs from inputs for each module. If the guessed and calculated recycle stream values are not sufficiently similar the calculations are repeated with a new set of guessed variables.

The iterative method, whereby the calculated set of variables becomes the new guessed set, is called 'direct substitution'. The rate of convergence of this method is often slow and to accelerate convergence methods, like 'bounded Wegstein', or 'dominant eigenvalue', have been used. Quasi-Newton methods are used where the partial derivatives of the Jacobian are approximated by secants and updated each iteration. Broyden's method updates the inverse Jacobian directly.

Several authors have found that direct substitution is faster to converge than the simultaneous method and, further, expect it to be globally convergent. Other authors have found that Wegstein's method is faster converging than direct substitution in all of their applications. Newton-like methods are the fastest converging of all but have disadvantages in that the initial guess must be near the solution, and the solution must be non singular. Furthermore the overhead computing costs are high.

For computer simulation the process flow diagram must be converted to a form understandable to the computer. The information flow diagram assigns numbers to the streams and units and associates a mathematical model in the form of a computer subroutine with each unit module. The information flow diagram is then drawn as a digraph which can be represented in numerical terms either by an 'adjacency matrix', an 'arc operator matrix', or a 'stream vector' with index. However, the 'stream connection matrix' and 'incidence matrix' give a more complete representation of the information flow diagram as they include a connection between streams and units, as well as a connection between type of output stream (concentrate, middling, tail) and stream number. The 'process matrix' further associates unit number with unit computer subroutine name, however for flexibility of input the stream connection matrix in conjunction with a unit type matrix was found to be most suitable and the stream vector in conjunction with other relevant arrays was found to be most suitable for internal computer use.

Finding the system tears is done in two stages. First the system units are grouped into nodes, so that there is no feedback of material except within nodes and secondly the set of streams which render each node acyclic is found.

Methods of partitioning have been based on 'path searching' and Boolean operations on the adjacency matrix. However for this simulator a cycle/stream matrix is required for a tearing criterion and so can be also used to partition the system. The algorithm uses the fact that, if any two cycles have a common stream, then they both belong to the

same complex node.

The choice of tear streams is based on the minimum sum of stream weightings of streams which render the node acyclic. The stream weightings can be modified so that any of; minimum torn streams minimum torn variables, minimum double tears or minimum user weightings are used as the tear set. The criterion used for selecting tear streams can effect the rate of convergence of the system. It is shown that, by using the minimum double tear criterion, that tear set which allows direct substitution to converge fastest is likely to be selected. It is further shown that, if the selected tear set allows direct substitution to converge fastest, then it can be expected to allow both the Wegstein and reduced Newton methods to converge fastest.

In order to use the minimum double tear criterion the cycle/stream matrix must be found, this can be done very efficiently using Johnson's 'path searching' and 'backtracking' algorithm. This cycle/stream matrix can also be used with dynamic programming or graph simplification and two way edge reduction to find the tear set. However it is shown that the latter algorithm is not suitable when the 'minimum double tear' criterion is to be used.

The major difference between chemical plants and ore dressing plants is the fact that the streams in ore dressing plants carry solids. It is impossible to consider the behaviour of every particle so that particles must be grouped according to their physical properties. This allows the particles to be represented by a continuous distribution and described by a mathematical function with comparatively few parameters. However, because of the inherent nature of ore dressing operations, the distributions change irregularly and a mathematical function with only a few parameters does not have sufficient flexibility to fit the changing distributions. The alternative of a discretized distribution is very satisfactory. The alternative of a discretized distribution is very satisfactory as it can retain an exact mass balance and uses values which can be measured. The disadvantage of a discretized distribution is the large number of variables per stream that may be required.

The discretized distribution is split into a three dimensional grid based on particle size, grade and a third, arbitrarily chosen property, usually related to the relative surface area of each component mineral.

Another feature of ore dressing plants is that there are only two possible transformation units, comminution units and conditioners. The conditioner need not always be a transformation unit and it is preferable, from a convergence point of view, to use it as a 'non-transformation unit' whenever possible.

The physical properties used as separation criteria in ore dressing plants are hardly ever affected by temperature or pressure so that a physical property package should provide property values for pure minerals and show how to relate this information to mixed particles. This type of data is either readily available, e.g. specific gravity, or unknown, thus requiring laboratory tests; making a physical property packages of little value to an ore dressing simulator at present.

In many situations process units can adequately be represented by linear models, as the choice of variables is designed to minimize interaction and, hence, linearize. This is particularly true when the range of operation of the equipment is small.

If all the system describing equations are set up as a set of simultaneous linear equations, then solving these equations simultaneously, as they stand, is generally impractical (as very few computers have the capacity to invert a matrix of say 100 x 100). However due to the nature of ore dressing plants it is possible to reduce the problem to one of inverting several very manageable matrices. First the system is reduced to nodes which can be solved sequentially in the order of mass flow. Secondly as particles are only reduced in size it is possible to solve for variables in each size class in turn, from largest to smallest. Thirdly (except in the unusual case of a conditioner acting as a transformation unit) within each size class similar variables from each tear are independent of the other variables so that each set of similar variables can be solved for independently. This implies that instead of solving the complete system simultaneously, the problem is reduced to solving each node sequentially and, within each node, instead of inverting a $(n_G.m_G \times n_G.m_G)$ matrix, (where n_G is the number of variables per stream and m_G the number of tears in the node), n_G matrices of size $m_G \times m_G$ are inverted. Generally $n_G >> 1$, and the computation time for inversion of a matrix increases with the cube of its size so that this is a considerable reduction in computation effort.

Instead of using the simultaneous method an iterative scheme such as direct substitution can be used. The necessary and sufficient condition for convergence of this scheme is for the spectral radius of the system transformation matrix to be less than one. It is proved for linear ore dressing systems that this condition holds except possibly when certain variables equal zero at the solution. By using the null vector as an initial guess this possible exception is eliminated and convergence can always be expected.

For non-linear systems an iterative method must be used to solve the *mass balance*. If direct substitution is used as the convergence scheme then having the spectral radius of the system Jacobian less than one is sufficient for local convergence. For other iterative schemes the spectral radius of the system Jacobian, in conjunction with the convergence scheme must be less than one. Although it has not been conclusively proved that the spectral radius of non-linear ore dressing plants are always less than one it has been shown that this condition can generally be expected to hold.

Based on the remarkable saving in computational effort, when the block diagonal form of the linear system transformation matrix is used, a convergence method called the 'reduced Newton' is developed for non-linear systems. The reduced Newton is likely to exhibit local convergence whenever direct substitution converges and it is expected to converge at a faster rate than direct substitution.

Linear systems have been reduced to groups to make the problem more manageble. It is shown that this can also be done for non-linear systems, using an iterative solution scheme, without affecting the convergence rate at all.

The required elements of the system Jacobian are calculated from the sums of products of the partial derivatives of output variables with respect to input variables for each unit model, along the paths between the tear streams. For linear systems these partial derivatives are constant and equal to the mass fraction of each input variable appearing as a similar variable in the output stream. For non-linear systems these derivatives should be calculated within the unit model, however on a non-linear flotation plant, where mass fractions are used instead of the derivatives, the reduced Newton still converges.

Examination of the reduced Newton method indicates that the computation time to reach solution should increase only *linearly* with problem size. By using a non-linear flotation plant as an example, because of ease in changing the total number of cells without affecting the configuration, this linearity is found to hold within the limits of possible error. This is a strong point in favour of the reduced Newton when compared to other Newton-like methods.

The same flotation system has also been used to compare the reduced Newton with both direct substitution and bounded Wegstein methods. The reduced Newton is found to be far superior to both other methods in terms of total computation time and number of iterations required to reach solution.

PART II

PROGRAMMING

11 INTRODUCTION TO PROGRAMMING

In Part I the background to computer simulation is described and algorithms to partition, tear and solve the system are selected. In this second part these decisions are implemented and careful consideration is given to minimising both storage requirements and computation time, while still keeping the input simple, and flexible, and the output brief, yet comprehensive.

In what follows the principles and techniques used will be described in some detail. A chapter on program usage explains how to set up the input data file followed by instructions in running the program. However, it should be noted that the file processing methods are specific to the 'Wits' IBM system and will probably have to be modified for other machines.

12 OVERALL CONDITIONS

12.1 Programming Language

One of the features of an ore dressing simulator is the great variability in the problem size. The system may use only two variables per stream (e.g. larger and smaller than some size) in a plant with no recycles and few streams, or it may be a full production plant simulation with hundreds of variables per stream, tens of streams and complex recycle loops.

In a normal FORTRAN program the dimensions must be fixed, and if they are set to cope with large problems then users with small ones will be paying for excess storage. PL/1 is a computer language which can handle variable dimensions, however, as there is no standard form (as with FORTRAN) it is machine specific. An advantage of FORTRAN over PL/1, however, is that it is far more familiar to engineers, who are the anticipated users of the simulator. It is possible to interface these two languages (PL/1, the subroutines written by the user, as unit models, can still be in FORTRAN. The transportability of the program is important, however, and a method of solving the variable dimensions problem in FORTRAN has been found which offers other advantages, which will be discussed (12.2).

FORTRAN was therefore, selected as the programming language.

12.2 Structure of the Program

The overall program is divided into four phases and detailed descriptions are given in chapter 13. The underlying object is that each phase will set up the dimensions of the next phase so that computer usage is overlayed.

Running phase one creates a set of dimensions and calling statements for phase two. Phase two (input phase) reads the user supplied data, modifies it into a form suitable as input data to phases three and four and echoes the data to permit the user a check on its validity. The set of dimensions and calling statements for phase three are created.

Phase three (ordering phase) accepts data from phase two, orders the calculations and outputs the calculated tear streams and unit calculation order for each node, both for inspection, and for use as data in phase four. The set of dimensions and calling statements for phase four are created.

Phase four accepts data from phases two and three and then does the mass balance calculation and outputs the results.

The overall storage requirement in phase two is not big even for large plants. This has made it possible to choose a standard set of dimension sizes created by phase one, which are able to handle most plant configurations.

If the system can be broken into several sub-plants then the storage requirements will only be that of the largest sub-plant. In this way it may be possible to simulate even extremely large plants using the standard set of dimensions. This breakage into sub-plants will be advantageous in all circumstances as it reduces the storage requirements in phases three and four and reduces the computational effort in phase three (ordering) by doing some of its work.

By using the standard dimensions for phase two and the created dimensions for phase three and four, which potentially have very large storage requirements, the dimensions are effectively variable so that no storage is wasted.

The division into phases allows all input and preprocessing to be treated seperately from the mass balance calculation, so that all non-essential coding and operations are removed from phase four. This means that this, possibly expensive, phase executes efficiently.

12.3 Precompilation of Programs

The four main programs corresponding to the input phase, ordering phase, calculation phase and library unit model subroutines are all long, each having 600 to 1200 lines of coding. The compilation of the programs is expensive (approximately R20/program) and, to avoid incurring this cost each time they are run, these programs have been compiled and then stored in this form.

The need for variable dimensions complicates this task but, by writing each program as a subroutine and including calling statements with the created dimension statements (so becoming the main program for each phase) the subroutines can be compiled and stored.

12.4 Information Storage

The major reduction in storage requirements (as described in sub-chapter 12.2) is achieved by overlaying, variable dimensions, and grouping into sub-plants.

Another saving is obtained in the calculation phase by using the dynamic storage of stream variables. Instead of storing the variables in all the streams of the plant simultaneously, only those which are still needed as inputs(to units not yet computed in the iteration) are retained. This dynamic storage of stream variables can reduce the storage requirements dramatically. For example if each stream has 100 variables and the plant is a simple sequential process with a single recycle and 10 streams in the loop, then the dynamic storage method will only require 100 storage locations compared to the complete storage method with 1000 storage locations.

This reduction in storage is,however,paid for by reduced flexibility of output, as stream variables for each stream can only be printed in the order that they are calculated rather than in any arbitrary order. This is not a big drawback when compared to the savings in storage achieved.

In all the phases of the program there are many sets of variables, of different lengths,that could be stored in a single two dimensional array. For example,the streams in each cycle of the plant must be stored However,the number of streams in a cycle can vary from two to a hundred. If there are say ten cycles in the plant then the two dimensional array will have to have 10 x 100 = 1000 storage locations. This large array will have many unused locations and so be wasteful. To overcome this the sets are stored in a one-dimensional string with an associated index to indicate the position of each set.

Say there are four cycles containing the following streams

CYCLE 1 STREAMS 1 2 3 4
CYCLE 2 STREAMS 2 5
CYCLE 3 STREAMS 3 4 9 7
CYCLE 4 STREAMS 7 8 6 10 12

then the 2-D array would be:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 0 \\ 2 & 5 & 0 & 0 & 0 \\ 3 & 4 & 9 & 7 & 0 \\ 7 & 8 & 6 & 10 & 12 \end{bmatrix}$$

while the string would be:

(1 2 3 4 2 5 3 4 9 7 7 8 6 10 12)

and its index:

(4 6 10 15)

So if the streams in cycle 3 are required, they are found in positions 7 to 10 (as 3 4 9 7) of the string.

This means that there is no waste of storage. The use of indices makes programming more complicated because this would normally be done internally by the computer. However, it is this fact which ensures that explicit indexing will only be slightly more expensive in terms of computer effort, while giving great savings in storage.

12.5 Information Transmission

In the program, as structured, information is read in phase two, processed into convenient arrays and then outputted. This output is then read by the calculation phase and stored.

For a unit model in the calculation phase to calculate its outputs it must have the inputs, physical properties, size and grade of the material and the parameters of the model. A simple method of transmitting this data from the executive program to the unit model is in a COMMON storage block, which is available to all the subroutines. However any array with variable dimensions cannot be put into a COMMON block. Therefore the only solution is to pass this data through the argument list.

A further facility for data transmission is provided to solve such problems as the following:- A user has the data in a form which must be processed before use in the unit model and wishes to avoid

repeating this processing every time this model type is called. Alternately a user has a large table which is awkward and inefficient to read, store and write at the input phase followed by rereading and restoration at the calculation phase before passing on to the relevant unit subroutine'.

What has been done is to allow user written data to be read at the calculation phase. During the input phase a flag is set that indicates that data is to be read at the calculation phase. Then at the calculation phase subroutine READER is called. This subroutine, written by the user, has no argument list, and simply reads the data (from the user written data file DATR) in a form chosen by the user. It then performs any preprocessing that is required before storing the required data in a COMMON. This data can then only be used in a user written subroutine which contains a similar COMMON statement. In this way data transferrance is accomplished.

In the READER subroutine, data for several unit models can be entered and preprocessed if so desired. It should be noted that READER is called, at most, only once.

12.6 Calling Unit Model Subroutines

In setting up a plant configuration the streams and units are numbered and, in the calculation phase, the computer must be told which unit model corresponds to the unit number and then fetch this model to do the calculation.

The obvious method is to relate a unit 'type number' to each 'unit number', and a 'subroutine name' to each 'type number'. In the input phase the 'unit numbers' are connected to 'type numbers' and in the calculation phase when unit 'type i' is required the program goes to statement i (using a computed GO TO) which is the CALL to the required unit model. This unfortunately means that the names of all possible unit model subroutines must be known in advance, which, in turn, requires that there be a fixed number and choice of names which the user can call his subroutine. It also means that each of these imaginary subroutines must be created before the program will run which, in turn, implies that, when a user writes a subroutine, he must also eliminate the existing dummy subroutine of that name.

To overcome these disadvantages use can be made of the split structure of the program. In the input phase the user specifies a four letter unit model name which is related to the unit number. The program allocates type numbers in the order in which the names are specified.

each type name only being allocated one number. A subroutine is then written which contains the CALL's to each unit model subroutine based on the specified names and allocated type numbers. In the calculation phase, when a CALL is made to a unit model, this input phase-written subroutine is CALLed, which in turn, CALLs the correct unit model.

The technique has an advantage in that only unit model subroutines, which are actually used by the simulator, need be compiled. Also the user can specify model names which have some physical significance to him so making the input data more understandable and thus easier to check.

12.7 Changing Stream Variables - 'Converter'

It was originally thought desirable to allow every stream to have any combination of D, G and S-classes. On closer examination, however, this means that every stream would have to be specified individually. 'Converter units' would have to be placed in the system to change from one stream type to another and the user would have to ensure that streams of different types were not inputs to the same unit.

In order to use dynamic storage the streams must have the same number of variables. Furthermore the reduced Newton method is based on the products of partial derivatives along a path between the 'same' variables in to tears. If each stream has a different set of variables then this method becomes difficult to implement as well as inefficient.

To avoid these problems it was decided that plant streams must all be of the same type. This is not a very important restriction because a system can be broken into several plants with each plant as small as one simple and complex nodes in the system. In a complex node there are recycles and the most common use of this recycle would be to regrind the material. If, for example, the regrind unit operated only on D-classes and the remainder of the units in the node on D and G-classes then, in the first method, 'converter units' would have to be positioned before and after the mill, and the information for expanding D-classes to D and G-classes, after the mill, would have to be stored somewhere. To simulate a similar plant without 'converter units' the mill model need only be written to handle the D and G-classes, a simple task following which the G-class mass balance will be found to be automatically correct.

All the library unit modules have been written to handle D, G and S-classes regardless of what the actual stream variables are in the plant. The library modules cover the complete spectrum of ore dressing units but it is conceivable that a model could be so specialized that

it cannot be written to handle other class types. In the event of two of these specialized units, requiring different stream types, occuring in the same complex node, this simulator will not be able to cope. This is a rather remote possibility.

If, as could frequently happen , a system consists of a milling plant which uses only D-classes followed by a seperation plant which uses D, G and S-classes (e.g. flotation plant), then, at the connection between the plants, a conversion in stream variables is necessary. It is impossible to do this expansion from D to D, G and S-classes without additional information, although conversion in the reverse direction, (D, G and S-classes to D-classes) is easy. This required information must be supplied by the user as a distribution which is stored directly in those locations reserved for that feed stream to the separation plant. In this way no storage is wasted in creating extra arrays for this purpose as storage for plant feeds is essential in any event.

13 PROGRAM DESCRIPTION

13.1 Phase One - Overall Dimensions

In this phase the maximum problem size is defined and the dimension and calling statements are produced for phase two. The values for dimension sizes are set within the program. The standard values are:-

Maximum number of:	Physical properties	50
	Cycles in each complex node	8 ($2^8 = 256$)
	Plants in the system	30
	Connections between plants	30
	Mineral species	7
	Unit types	100
	Units in the system	100
	Streams in the system	100
	Size classes (D)	25
	Grade classes (G)	25
	Surface classes (S)	25
	Variables per stream	200
	*Physical property elements	1000
	Parameters	1000

*Each physical property, e.g. specific gravity, will be associated with all stream variables. If there are 10 stream variables per stream each with a different specific gravity, then 10 values must be stored as physical property elements for this physical property. The standard dimensions allow a maximum of 1000 physical property elements to represent all the physical properties.

It is expected that these sizes will satisfy almost any possible situation so that, as long as the output from this phase exists, this phase can be ignored by the user.

13.2 Phase Two - Input

The aim of this phase is to make data input easy and flexible for the user. This is achieved by:-

(a) Using descriptive keywords as part of the data. This makes it readable and easy to check.

(b) Making all non-essential data optional. This eliminates the need to provide dummy data.

(c) Echoing the data in readable form. This allows the user to check that the data received by the computer is the intended data. This can be particularly useful in picking up faulty data allignment.

(d) Doing internal checking for consistency of data, where possible, and warning the user if a fault is found.

(e) Converting data from a form easily understood by the user to one which is more suitable for storage and use in the computer. This should help reduce user errors.

The data is divided into three types.

13.2.1 System Data

This data applies to the whole system and cannot be changed within a single job run. Table 6 gives a summary of keywords related to system data, their meaning and default values if not specified. The significance of the default values and full usage of the keywords is given in chapter 14.

TABLE 6 SYSTEM KEYWORDS

KEYWORD	MEANING	DEFAULT
READ	Instruction to call READER	No call
NOGS	Set of stream variables for each plant	Essential
MINS	Number of mineral species	1-Ore
CNCT	Connections between plants	No connection
SIZE	Actual particle size in each G-class	Nothing
SPGR	Specific gravity of each mineral	Nothing
GRDM	Mass destribution of minerals in each G-class	Nothing
GRDV	Volume distribution of minerals in each G-class	Nothing
PHYP	Physical property	None

13.2.2 Plant Data

This data must be provided for each and every plant in the system, and cannot be changed within a single job run. Table 7 gives a list of keywords related to plant data,their meaning and default values if not specified.

TABLE 7 PLANT KEYWORDS

KEYWORD	MEANING	DEFAULT
PLNT	Plant number	Essential
SCMA	Stream connection matrix	Essential
WTCR	Weighting criterion for tear algorithm	2
STWT	Manual weighting of streams	all 1
WTRF	Water feed	None
STRM	Feed, tear and convertor streams	Feed: Essential Tear: all variables = 1 Converter: Essential

13.2.3 Run Data

This data refers to each plant in turn, and can be changed several times within a single job. Each time a parameter or several parameters are changed a new set of run data is associated with the plant. All runs for a particular plant must be completed before the next plant can run. This facility of being able to do several runs can be very useful:-

For example in RUN 1 the user can request five iterations using direct substitution to get near to the solution and in RUN 2 continue with another convergence method. Or perhaps do ten iterations and then request an intermediate printout. Or use a simple output code until convergence and then, after the system has converged, do another run to get a more detailed output listing. Or, after converging with a linear plant, change the units to non-linear models for a more realistic answer. Or do runs to see the effect of changing unit model parameters.

Table 8 gives a listing of keywords for run data and their meaning.

TABLE 8 RUN KEYWORDS

KEYWORD	MEANING	DEFAULT
CONV	Convergence method	3
ITER	Maximum allowable iterations	8
TOLM	Tolerance limit for convergence	10^{-4}
TYPE	Relates unit type to a name and parameters	Essential for first run
COPY	Copies unit type and parameters	-
OUTC	Output code	0001001655

For further description of the output code see sub-chapter 14.4.3.

13.2.4 Programming

Tables 6, 7 and 8 show that most of the data is optional, this means that the program has to read a keyword and then move to that part of the program which treats the related parameters. The reading of the parameters into arrays and their processing is simple and the program is, roughly, sequentially ordered with respect to the keywords.

More insight into the operation of this phase will be given in chapter 14.

13.3 Phase Three - Ordering

In this phase as much preprocessing as possible is done to keep the load on the calculation phase as small as possible. The program can be split into fourteen sequential operations:
(for definition of maximal nets, nodes and groups see chapter 8.1)
(for definition of stream connection matrix see sub-chapter 7.2.2)

(i)	Read data	
(ii)	Extraction of working arrays from the stream connection matrix	
(iii)	Find all cycles in the plant	ALGORITHM 1
(iv)	Weight plant streams	
(v)	Group cycles into maximal nets	ALGORITHM 2
(vi)	Find inputs and outputs to each node	ALGORITHM 3
(vii)	Order the units for calculation	ALGORITHM 4
	(a) Renumber streams and cycles in complex nodes	ALGORITHM 5
	(b) Find tears in complex nods	ALGORITHM 6
(viii)	Find paths between tears in each complex node	ALGORITHM 7
(ix)	Set up vector of paths affected by each stream	ALGORITHM 8
(x)	Set up index for dynamic storage of streams	
(xi)	Set up index for storage of feed and tear streams	
(xii)	Write input for the calculation phase	
(xiii)	Set up dimensions for calculation phase	
(xiv)	Set up calling statements for calculation phase	

Many of these operations are straightforward. The ones that need further explanation have been numbered as algorithms. In the following sub-chapters each algorithm is explained by means of a logic flowchart and a verbal description.

To help describe the operation of some of the algorithms the information flow diagram given below will be used.

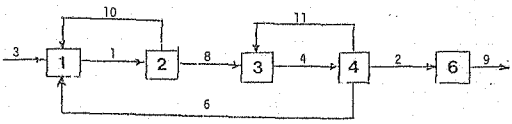


FIGURE 10 EXAMPLE INFORMATION FLOW DIAGRAM

The stream connection matrix, (SCM), for this example is given in figure 11 that there are gaps in both the stream and unit numbers.

STREAM		TO UNIT	STREAM	FROM UNIT	TO UNIT
3	0	1	4	3	4
10	2	1	2	4	6
1	1	2	9	6	0
11	4	3	6	4	1
8	2	3			

FIGURE 11 STREAM CONNECTION MATRIX CORRESPONDING TO EXAMPLE IN IN FIGURE 10

Although the order for streams in the SCM is arbitrary the relative position of the outputs from units 2 and 4 are important for stating which stream corresponds to which output.

In this example stream 10 is output 1 and stream 8 output 2 from unit 2 and streams 11, 2 and 8 correspond to outputs 1, 2 and 3 respectively, of unit 4.

In the logic flow diagrams that follow several verbs are used to describe steps in the program. In order to be consistant and clear, table 8 defines the meaning of the verbs as used in this context.

TABLE 8. DEFINITION OF VERBS USED IN LOGIC FLOW DIAGRAMS

VERB	MEANING
SET UP	search through array to find all elements of a new array
FIND	search through array until find some element/s
TAKE	increment/decrease counter to give next stream, cycle etc.
MOVE	increment/decrease position counter in any array
SET	extract element/s from array directly
PUT	insert element/s into array directly
CHECK	test whether condition holds
TEST	start procedure to do series of checks
RECORD	add several elements to string of elements
LIST	add elements to list

13.3.1 ALGORITHM 1 - Enumerate all Cycles in each Plant

The basis of this algorithm is pathsearching and backtracking, as described in sub-chapter 9.3.2, and uses Johnson's (1975) method as described in sub-chapter 9.3.3. The programming technique limits the number of cycles to a maximum of 32 per plant, however this should present no difficulty as it should always be possible to make the plants smaller than this (see sub-chapter 9.4.2).

As the path of streams from a root stream is traced each stream is tested to ensure that:-

(i) It is not a plant output stream

(ii) It is not already in the path

(iii) It does not have stream number larger than the root stream number

(iv) It is not blocked

before it is added as an extension to the path. If condition (ii) applies, so that the test extension stream must already be in the path, then a check is made to see whether the test extension is actually the root stream. If it is, a cycle has been found.

Once any of the conditions hold the path cannot be extended and the algorithm tests the next output from the last stream in the path, if there are no more outputs the algorithm backtracks along the path to the preceding stream.

Consider the example shown in figure 10, when the root stream is stream 8.

The path builds up to streams (8, 4) then tests stream 11, and finds that 11 is greater than 8 so that condition (iii) holds. Stream 2 is tested next and the path becomes (8, 4, 2). Stream 8 is a plant output so that condition (i) holds. Stream 6 and then stream 1 are added to the path, stream 10 is tested and condition (iii) holds. The next output stream from stream 1 is stream 8 and condition (ii) holds, however stream 8 is a root stream, therefore a cycle has been found. The algorithm then backtracks to stream 1 and, finding no more outputs from this stream, continues to backtrack to stream 6, then stream 4 and finally to the root stream 8. At this stage a new root stream must be found and the procedure repeated.

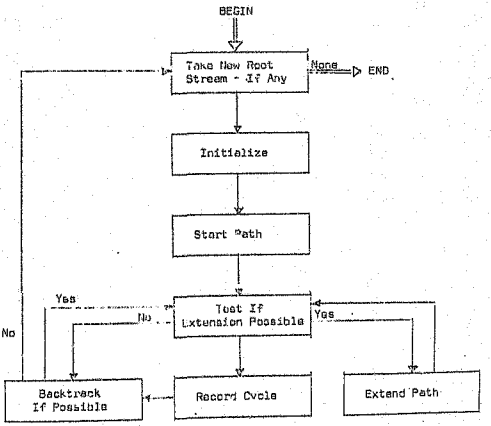


FIGURE 12 . LOGIC FLOW DIAGRAM - ALGORITHM 1

13.3.2 ALGORITHM 2 - Group Cycles into Maximal Nets

A verbal description of the algorithm was given in sub-chapter 8.3. The logic flow diagram follows:-

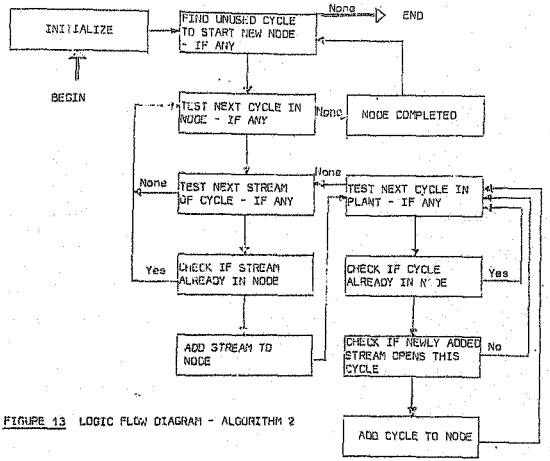


FIGURE 13 LOGIC FLOW DIAGRAM - ALGORITHM 2

In the example shown in figure 10 the streams in cycles one to three are (11, 4), (10, 1) and (8, 4, 6, 1) respectively.

Starting with cycle one (11, 4), stream 11 is selected. As there are no streams in any node at this stage stream 11 becomes the first member. The other cycles are checked for the membership of stream 11 and as stream 11 is not a member of cycles two or three no cycle can be added to the node. Next stream 4 is added to the node and it is found to be a member of cycle three, so cycle three is added to the current node. There are no more streams in cycle one so the next cycle in the node (cycle three) is tested. Stream 8 is added to the node but

is not a member of any new cycles.

Next, stream 4, is already in the node so the algorithm moves onto stream 6. Stream 6 is added to the node but belongs to no new cycle. Stream 1 is added to the node and is found to be a member of cycle two, therefore this cycle is added to the n, s (which now contains cycles one, two and three). Cycle three has no more elements and the algorithm moves to the next cycle, (cycle two). Stream 10 is added to the node without introducing any new cycles. Stream 1 is already a member of the node so the algorithm, finding no more cycles in the node, registers that the node is now a maximal net. No unused cycles can be found and the algorithm ends.

The system in figure 10 has been found to have only one maximal net consisting of cycles one, two and three and streams (11, 4, 8, 6, 1, 10).

13.3.3 ALGORITHM 3 - Find Inputs and Outputs for all Nodes

The maximal cyclic nets of ALGORITHM 2 are in fact 'complex' nodes while the remaining single units are 'simple' nodes (see sub-chapter 8.1). This algorithm finds the inputs and outputs for all nodes so that the interconnections between the nodes are available and the order of calculation of the nodes can be found.

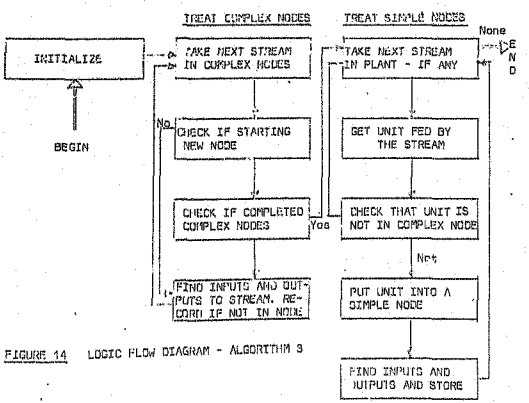


FIGURE 14 LOGIC FLOW DIAGRAM - ALGORITHM 3

The inputs and outputs for a simple node are readily available but the inputs and outputs of complex nodes are more difficult to obtain.

Each stream, in the list of streams in maximal nets, is checked in turn, to see if its input and output streams are members of complex nodes. If any are not members then they must be either an input or an output to the current node. Node output streams are recorded in a string along with an index. The input streams are stored in an array which indicates that node to which the stream is an input. Simple node inputs and outputs are added to the array and list respectively.

In the example of figure 10 the input to complex node (11, 4, 10, 1, 8, 6) is stream 3 and the output stream from this node is stream 2. Stream 2 is also the input to simple node, unit 6, and the output stream from this simple node is stream 9.

13.3.4 ALGORITHM 4 - Find the Calculation Order

The aim of all previously described algorithms has been to build up a series of working arrays so that the ordering of the units for calculation can be efficiently achieved.

This ordering algorithm is split into two distinct procedures. The first orders the nodes so that there is only a feed forward of information and the second orders the units within the complex nodes.

The number of inputs to each node has already been found, these numbers are initially reduced for the units which are fed by plant feeds. When some node has all its inputs available it is added to a list. Each node in the list is treated in turn. If it is a simple node the corresponding unit is added to the calculation order. If it is a complex node the units in the node are ordered first, after finding its tear streams using algorithms 5 and 6, and then added in order, to the unit calculation order. Once a node has been added (in terms of units) to the unit calculation order its outputs become available. These outputs are the feeds to other nodes so the number of unavailable inputs to these other nodes are reduced. New nodes with all inputs available may have been created. These are added to the list. This procedure is repeated until all the nodes and hence units are ordered.

The ordering of units within a complex node is achieved in exactly the same manner as the ordering of nodes. The only difference being that the first nodes with all inputs available, are obtained by removing plant feeds, while within the complex node, the first units with all inputs available are obtained by removing the node's tears.

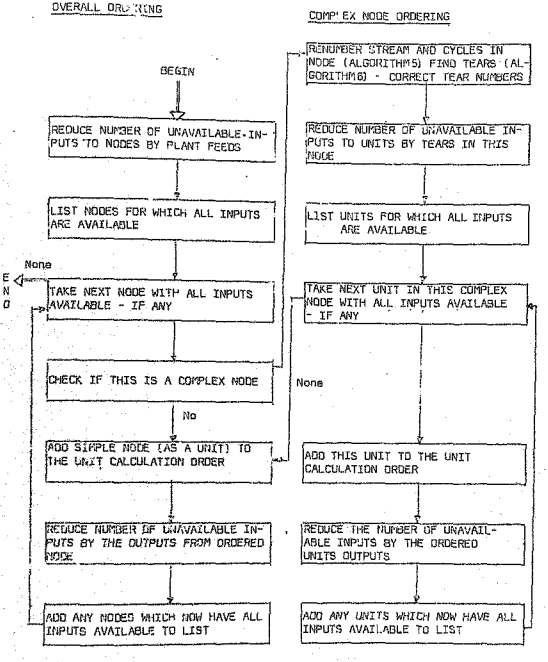


FIGURE 15 LOGIC FLOW DIAGRAM - ALGORITHM 4

Refering to figure 10 if the plant feed, stream 3 is removed, then all inputs to the complex node (11, 4, 10, 1, 8, 8) are available. Algorithms 5 and 8 find the tear streams as streams 11 and 1. Then removing these tear streams all inputs to unit 2 are available and it can be put into the unit calculation order list. The outputs from this unit, streams 10 and 8 are now available and unit 3 can be added to the unit calculations order. Now all the inputs to unit 4 are available and stream 6 can be found, and finally streams 3, 10 and 8 are available allowing the calculation of stream 1. Thus the unit calculation order is (2, 3, 4, 1) for the complex node.

The simple node unit 8 can now be added to the calculation order giving the plant unit calculation order as (2, 3, 4, 1, 8).

13.3.5 ALGORITHM 5 - Renumber Streams and Cycles in Complex Node

The biggest problem with the tearing algorithm (ALGORITHM 8) is the large storage requirements, $3 \times 2^{C-1}$ where C is the maximum cycle number (see sub-chapter 9.4.2). If there are numbers between one and C which do not refer to cycles in this node then storage is being wasted and renumbering the cycle will be worthwhile. Renumbering the streams from one to the total representing the number of streams saves storage.

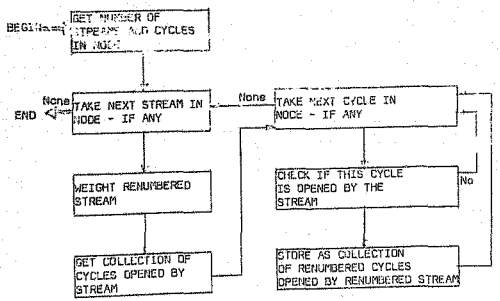


FIGURE 16 LOGIC FLOW DIAGRAM - ALGORITHM 5

The streams are *renumbered* by their position in the vector string in which they are stored. In the example of figure 10 the streams in the com node are in the order (11, 4, 10, 1, 8, 8) so that renumbered stream 1 corresponds to stream 11, renumbered stream 2 to stream 4, etc.

Similarly the cycles are renumbered by the order in which they are stored. In this example there is no need for renumbering the cycles, (11, 4), (10, 1) and (8, 4, 6, 1) as they are already numbered 1, 2 and 3 respectively. However, whenever there is more than one complex node in the plant, renumbering of cycles will be done.

It is possible to store a set of several small integers as a single number. For example 2, 4, 5 can be converted to:-

$$2^{2-1} + 2^{4-1} + 2^{5-1} = 2 + 8 + 16 = 26$$

then if one wishes to know whether 4 is a member of the set the quotient $26/2^{4-1} = 3$ is found, if the quotient is odd then the number is a member of the set and if the quotient is even then it is not. As 3 is odd 4 is a member of the set, and $26/2^{3-1} = 6$ which is even so 3 is not a member of the set.

This number (26, in the example) has been called a 'collection' and it is very useful in saving storage and eliminates the need for list searching to discover whether a number is a member of the set or not.

The cycles of the example in figure 10 are (11, 4), (10, 1) and (8, 4, 6, 1). The cycles becomes (1, 2), (3, 4) and (5, 2, 6, 4) when using renumbered streams.

In this algorithm the streams are also weighted according to the chosen criterion (see sub-chapter 9.2.3). If for the example criterion two is chosen then the weighting will be as follows:

(with $A = 6$ in $W_K = A.m_K + 1$)

Original Stream	Renumbered Stream	Weight
11	1	7
4	2	13
10	3	7
1	4	13
8	5	7
6	6	7

13.3.6 ALGORITHM 6 - Find Tear Streams of Complex Nodes

The algorithm used has been adapted from that of Upadye and Grens (1972) as described in sub-chapter 9.4.2, using the renumbered streams and cycles of ALGORITHM 5.

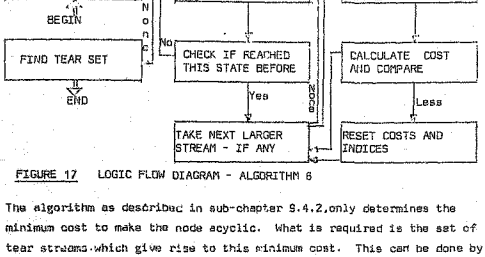


FIGURE 17 LOGIC FLOW DIAGRAM - ALGORITHM 6

The algorithm as described in sub-chapter 9.4.2,only determines the minimum cost to make the node acyclic. What is required is the set of tear streams which give rise to this minimum cost. This can be done by associating an optimum tear set to each state. However this involves the storage of a list of streams for each state. A neater method which also uses far less storage,involves associating each state with another parameter. This parameter comprises the previous state along the optimum path. There are now three parameters associated with each state:

(i) The minimum cost to reach this state

(ii) The largest (and last) stream number in the path to the state

(iii) The previous state along the optimum path

Once the final state has been reached,with the minimum cost, it is possible to backtrack along the optimal path using parameters (ii) and (iii) and so obtain the optimum tear set.

The state reached by tearing streams is determined by the cycles opened by the streams. For example if the torn streams open cycles 2 and 4 then the state reached will be $2^{2-1} + 2^{4-1} = 10$. It should be noted that even if the streams open some cycles more than once the same state is reached.

For the example in figure 10 the following list shows which state is reached when each combination of cycles are opened.

TABLE 10 POSSIBLE STATES FOR COMBINATION OF OPENED CYCLES

Cycles open	0	1	2	1,2	3	1,3	2,3	1,2,3
State	0	1	2	3	4	5	6	7

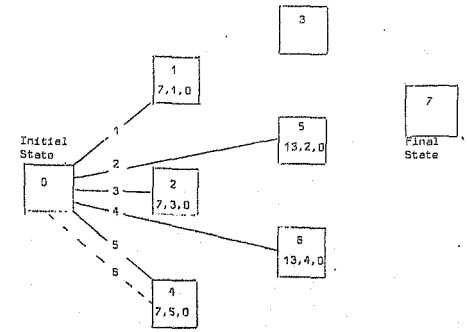
where Cycle 1 is streams (1, 2)
Cycle 2 is streams (3, 4)
Cycle 3 is streams (5, 2, 6, 4)

The weighting for this example were found in sub-chapter 13.3.5 to be:-

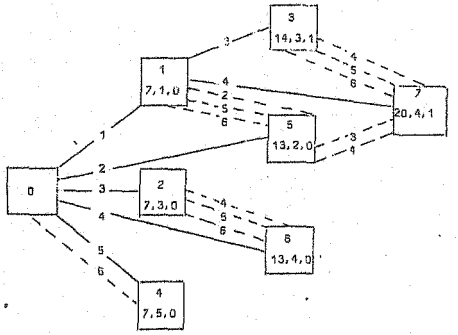
TABLE 11 STREAM WEIGHTINGS

Original Stream Number	11	4	10	1	8	6
Renumbered Stream Number	1	2	3	4	5	6
Weighting	7	13	7	13	7	7

In this example there are three cycles in the complex node and therefore there are $2^3 = 8$ possible states as shown in table 10. The algorithm procedure can be visualized through figures 18(a) and 18(b) which represent the states by vertices and the streams torn by arcs connecting the vertices. In the diagram states are numbered according to table 11. In the blocks representing each vertex is the state number and below it the three parameters associated with that state. These values are the ones corresponding the optimum path from the initial state to that state. The arcs are labelled by the numbers of the corresponding torn streams. Figure 18(a) shows all the states reached in one step from the initial state. Figure 18(b) illustrates the continued application of the algorithm. In both figures the current optimum path to all vertices (states) are shown by solid lines and non-optimum path are shown by dotted lines (these non-optimum paths are not recorded in the computer). In case there exists more than one optimum path to a vertex all paths found after the first such path are considered non-optimum.



(a) First Step from Initial State



(b) Complete Decomposition

FIGURE 18 STATE DIAGRAM ILLUSTRATING APPLICATION OF ALGORITHM 6

The optimum set of renumbered torn streams is then found to be (1, 4) with a cost of 20.

Using table 11 this set of torn streams, using the numbering of figure 10, is (11, 1). It is shown in sub-chapter 13.3.4 that this tear set is valid as they do render the complex node acyclic. The cost of 20 is clearly optimum as this configuration needs at least two tear streams, and there is no combination of two streams in table 10 which gives a lower cost. Tear set (4, 10) also has a cost of 20 but as set (11, 1) was found first this other set is considered non-optimal.

Tear set (10, 11, 6) has a cost of 21 which is greater than 20 (although it does not double tear any cycle it uses more tear streams) showing that the algorithms choice of tear set has been correct.

13.3.7 ALGORITHM 7 - Find Paths between Tears in each Complex Node

The products of partial derivatives of variables along paths from every tear to itself and every other tear is the basis of the reduced Newton method. These paths are numbered from one to NP, where NP is the number of paths in the plant, rather than from one to NPN_i, where NPN_i is the number of paths in complex node i, so that it is possible to differentiate between the paths of each node.

For this logic flow diagram BASEST is the base stream from which possible extensions are tested. The stream being tested is TESTST. BASEST will always be the last stream in the current path.

The algorithm takes each node in sequence and checks whether it is complex. If it is complex, starting with each tear of that node, a path is traced until the possible extension is either a tear stream, or not a member of the current complex node. When the TESTST is a tear a path has been found and algorithm 8 is called to organize the information in a form suitable for use in phase 4. In either event the next output of the BASEST becomes the TESTST. When no more outputs are available from a BASEST then the algorithm backtracks to the previous member in the path. This is continued until the root stream is reached and no further backtracking is possible. At this stage the algorithm takes the next tear stream of the node, if there are any that have not been used as the root stream, and repeats the procedure.

Once all the tears of a particular node have been used as root streams then the algorithm moves onto the next node.

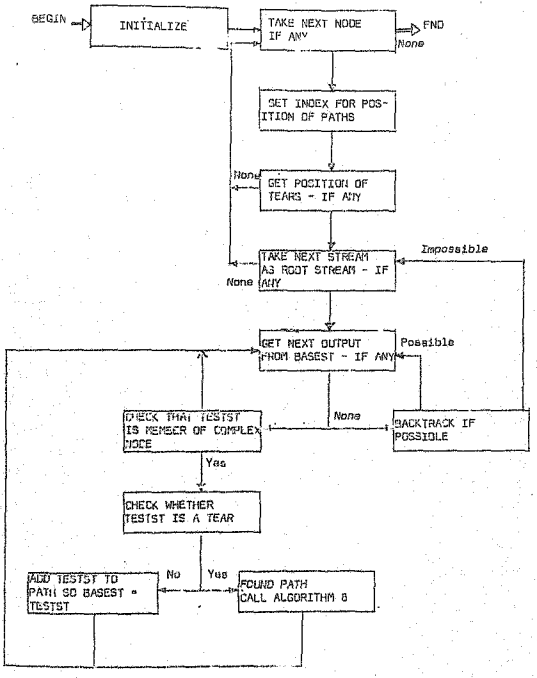


FIGURE 13 LOGIC FLOW DIAGRAM - ALGORITHM 7

For the example of figure 10 the first node consists of streams 11, 4, 10, 1, 8, 6 and is complex. The second node, unit 8, is simple. The tear streams were found in algorithms 6 as (11, 1). So using stream 11 as the root stream the path 11, 4 is traced following which 11 becomes the TESTST. Thus the path (11, 4, 11) has been found. The next output from BASEST. 4 is stream 2 and, as this stream is not a member of the node, the algorithm moves to the next output, stream 6, which is then added to the path. The output from stream 6 is stream 1 and as this is a tear stream another path (11, 4, 6, 1) is found. There are no other outputs from stream 6 so the algorithm backtracks to stream 4, which has no untested output, and then stream 11 which also has no untested output. Putting stream 1, the next tear, as root stream, the path (1, 10, 1) is found, then path (1, 8, 4, 11) followed by path (1, 8, 4, 6, 1). Having used all the tears as root stream the algorithm moves to the second node, which is simple and so contains no paths.

To summarize the paths found for this example plant:

Path 1 (11, 4, 11)
Path 2 (11, 4, 6, 1)
Path 3 (1, 10, 1)
Path 4 (1, 8, 4, 11)
Path 5 (1, 8, 4, 6, 1)

13.3.8 ALGORITHM 8 - Set Up Vector String of Paths affected by each Stream

When using the reduced Newton method to assist the convergence of a complex node the partial derivatives of output variables, with respect to input variables, are determined for each output stream of every unit. The convention that the only unit which can have more than one input stream is a mixer unit implies that the mixer unit outputs are the only output streams which can have several sets of partial derivatives associated with them, (one for each input stream). For a mixer unit these partial derivatives will always be both unity and the same for each input, therefore it will not affect the product of partial derivatives along any path, and so need not be considered. All other output streams must be from units with only one input stream and so each output stream will have only one set of associated partial derivatives. Therefore each stream can be associated with only one set of partial derivatives and, if each stream is also associated with the set of paths, (from tear to tear), of which it is part, then, directly a unit computation has been done, the pro-

duct of partial derivatives for each path affected can be updated. Storing information about paths as a vector string containing the numbers of the paths affected by each stream will therefore prove to be efficient when used in phase 4.

PTHVEC is the vector string of paths affected by each stream, and is the array that will be transferred to phase 4 along with its index.

The algorithm is in two parts. The first orders the stream numbers in the path so that they are in ascending order. This is necessary to make the second part, which actually finds PTHVEC, efficient.

The ordering is done by comparing each member of the path with the next member. If the first is smaller than the algorithm moves onto the next member. If the first is larger the two are swopped. This is repeated for every member of the path until the last member of the path is reached. The algorithm then returns to the first member of the partially reordered path and repeats the process. If no swops are made, or, if only the first two members are swopped, then the path is ordered as required.

Consider path 5, found for the example in figure 10, in algorithm 7, (1, 8, 4, 6, 1). The first member can be ignored for the purpose of finding PTHVEC as it is not an output stream. Therefore the streams that must be ordered are (8, 4, 6, 1). Stream 8 when compared with stream 4, proves to be larger so 8 and 4 are swopped giving (4, 8, 6, 1). Next stream 8 is compared with stream 6. It is larger and therefore swopped. This is repeated with stream 1 and the resulting partially ordered path is (4, 6, 1, 8). Returning to the start of this path, stream 4 is compared with stream 6. It is smaller so no change. Next stream 6 is compared with stream 1. It is larger so it is swopped. The path is now (4, 1, 6, 8). Stream 6 is now compared to stream 8. It is smaller so no swop is done. Returning to the first member of the path, stream 4 is compared to stream 1 and swopped. Stream 4 is then compared to stream 6, and stream 6 to stream 8 and, in both cases, no swop occurs. The path is now completely ordered as (1, 4, 6, 8).

The second part starts with the largest stream number and checks if the number is the same as the J^{th} member of the ordered path (where J is the number of elements in the path). If it is the same then the path number is inserted into the $(POS+J)^{th}$ position of PTHVEC, (where POS is the number of elements in PTHVEC before any elements of the new path have been inserted), all the remaining elements of PTHVEC effected by this stream are shifted (J-1) positions. If it is not the same then each element of PTHVEC effected by this stream is shifted J positions. This

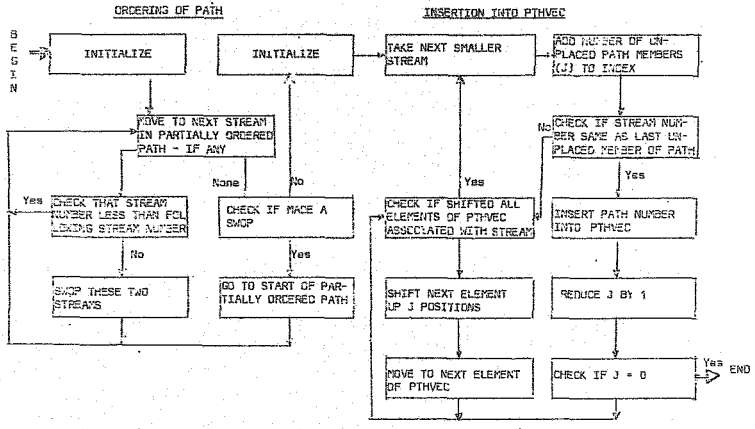


FIGURE 20 LOGIC FLOW DIAGRAM - ALGORITHM 8

procedure is repeated for each stream number in turn, from largest to smallest, until J equals zero which implies that all members of the path have been inserted.

The paths, with the first element ignored, found by algorithm 7 for the plant in figure 10 are ordered to give:

Path 1: (4, 11); Path 2:(1, 4, 6); Path 3:(1, 10); Path 4:(4, 8, 11) and Path 5:(1, 4, 8, 8).

After path 1 has been inserted into PTHVEC, PTHVEC is $1/_{4}1/_{11}$, where the long stroke with the subscript separates the paths effected by each stream and the subscript indicated the number of the stream which effects the path numbers to the left of the stroke and before the next stroke. So in this example stream 11 effects path 1 as does stream 4.

After inserting path 2, PTHVEC becomes $2/_{1}1,2/_{4}2/_{6}1/_{11}$ so that now stream 4 is seen to effect both paths 1 and 2, while stream 1 and 6 effect path 2 and stream 11 path 1 only. To demonstrate the operation of the algorithm the insertion of path 3 will be used. Path 3 has 2 elements therefore J = 2. Starting with stream 11 it is compared to element J (=2) of the path which is 1. They are not the same so the paths effected by stream 11 are shifted two positions to the right in PTHVEC. Next stream 10 is the same as the second element of the path so a 3 is inserted in the seventh position (original 6 + 2(=J) - 1 (only one path effected by stream 11)) of PTHVEC and J becomes equal to 1. No other paths already in PTHVEC are effected by streams 10 to 7, but when the stream number is 6, it effects path 2. This element must therefore be shifted J (=1) spaces to the right. Stream 5 effects no paths but the paths effected by stream 4 (1, 2) are both shifted one position to the right. No further changes take place for streams 3 and 2. Stream 1, however, is the same as the first element of the path so a 3 is inserted in the first position and J becomes J-1 = 0. The algorithm is thus completed. PTHVEC at this stage corresponds to:

$2,3/_{1}1,2/_{4}2/_{6}3/_{10}1/_{11}$

after inserting path 4 it becomes

$2,3/_{1}1,2,4/_{4}2/_{6}4/_{8}3/_{10}1,4/_{11}$

similarly after path 5:

$2,3,5/_{1}1,2,4,5/_{4}2,5/_{6}4,5/_{8}3/_{10}1,4/_{11}$

The validity of this result can be seen if each path is searched through to find in which path a particular stream can be found. For example stream 4 is found in paths 1, 2, 4 and 5 which agrees with PTHVEC.

This type of direct search to find the elements of PTHVEC was not used for two reasons. First the search method would require storage for all the paths simultaneously instead of only needing storage for one path at a time as with the chosen method. Secondly searching through all the elements of all the paths for each stream in turn is inefficient.

13.4 Phase Four - Calculation

The preceding phases have organized the data in such a way that the operation of this phase can be very simple and fast. This can be seen in the following logic flow diagram where each statement is a simple operation or set of operations (figure 21).

The FORTRAN names PTVEC, STRVAL and CNCVAL are the names of the arrays which store: the plant feeds and tears, the plant streams dynamically and the plant outputs, respectively. It is necessary to have a separate array for the feeds and tears so that the computation can be started for another run. The array CNCVAL stores the outputs so that these streams can be used as feeds to the following plants, STRVAL is not used for this purpose as the situation can arise where one plant's outputs are feeds to several different plants and storing values for different plants in STRVAL would cause confusion.

The array ALFA is used for storing the product of partial derivatives when the reduced Newton method is required and for storing the values of the torn variables from the previous iterations when the bounded Wegstein method is used. Besides these two methods, direct substitution is also available. All three convergence methods are described in sub-chapters 3.2.1, 3.2.2 and 6.5.

The RMS error is the square root of the sum of the squares of the differences between torn variables in two successive iterations divided by the latest values of torn variables, all divided by the number of torn variables.

$$\text{i.e. RMS} = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\left[\frac{y_i - x_i}{y_i}\right]^2} \qquad y_i \neq 0$$

where x_i is torn variable i at the start of the iteration and y_i is the calculated torn variable i at the end of this iteration. This RMS error is compared with the user selected tolerance limit TOLM to decide whether a solution has been reached or not.

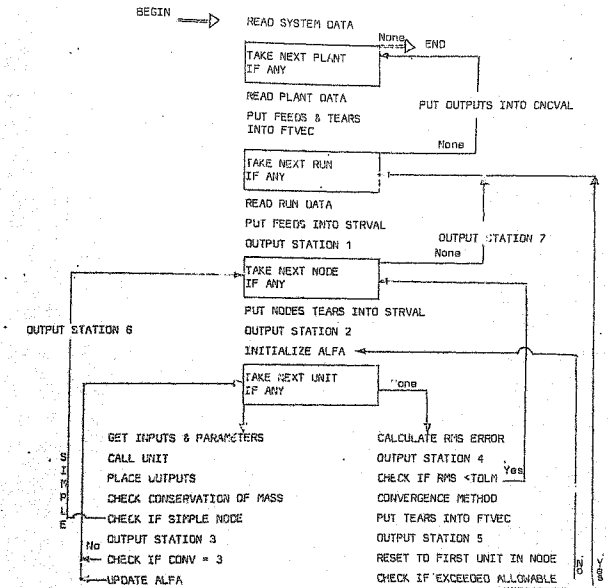


FIGURE 21 LOGIC FLOW DIAGRAM - CALCULATION PHASE

The keyword CONV gives the convergence method which has been chosen for the particular run. When CONV = 3 then the reduced Newton method is used.

The statements 'OUTPUT STATION i' correspond to those in sub-chapter 14.4.3 where their [illegible] are explained. However this logic flow diagram shows how they appear in every stage of the program so allowing very comprehensive output [illegible]ection.

This phase revolves around those four statements which have a block drawn around them. Each of these starts a loop within a loop. For a complex node the program computes the outputs for each of its units, in turn, for several iterations and, when this node has converged to a solution, the program moves to the next node. When all the nodes in a plant have been completed the program starts the next run for this plant and when the runs are all done the next plant is treated in the same fashion. The only exception to this procedure occurs if a complex node does not converge within the prescribed number of iterations. In this case the program checks to see if another run is requested for the plant containing this node and, if it is requested, the new run is started. If not the program moves onto the next plant. This exception allows the user to use one convergence method (e.g. direct substitution) to get near to the solution and then use an accelerated method to reach the solution. However, a disadvantage is that, if the user has not requested another run for this purpose, and the node has not converged to a solution, subsequent nodes in the calculation order will be operating on incorrect inputs. The output from the program will permit a user to ascertain if this has happened.

14 USAGE OF SIMULATOR

In the previous chapter the aims of the input phase were given. Here a detailed step by step method of using the simulator will be described.

14.1 Conversion of Process Flowsheet to Schematic Diagram

The conversion of the process flowsheet to schematic diagram has been discussed before (sub-chapter 7.1). The process flowsheet units are represented as blocks and the interconnections between the units as directed streams. This is very simple but it must be noted for the simulator that the only unit that can have more than one input is a mixer. So if two or more streams feed a separator or other non-mixer unit these streams must be first joined in a mixer unit before being fed to the non-mixer. This applies to water feeds as well. The user is not allowed to write a unit model with more than one feed as the simulator only provides for one feed to the unit.

14.2 Separation of System into Plants

The system is the complete schematic block diagram, whereas a plant can be a single node (maximal cyclic net). So, although the system can be a single plant, even if the system is made up of several nodes, there will sometimes be advantages in splitting the system:-

(a) Each plant can have a different set of stream variables

(b) The storage of the calculation arrays is overlayed, so allowing the treatment of a larger system

14.3 Numbering of Units and Streams

The numbering of streams and units are independent for each plant in the system. In other words the same stream or unit number can appear in two or more or all of the plants. The numbers can be arbitrarily assigned to units and streams but it must be born in mind that used numbers between one and the largest stream (or unit) number not corresponding to streams (or units), will waste storage.

A further point to note in numbering streams is that if this is done from small to large in the direction of flow of the streams, then this will make the cycle finding algorithm more efficient.

14.4 Creating the Input Data File

The input data is built around keywords presented in a fixed order. To make the data more understandable it is split into three types with each type prefaced by the one of the keywords 'SYSTEM DATA', 'PLANT DATA' and 'RUN DATA'. 'PLANT DATA' is repeated for each plant and 'RUN DATA'

is repeated for every run of each plant.

The format for all keyword data cards is: (A4,4I4),
except for TYPE and COPY for which the format is: (A4,I4,1X,A4,I4).
All other data cards are: (8G10.4),
except the output code which is: (20I4).

14.4.1 System Data

This refers to the system as a whole, appears only once and cannot be changed during the simulation.

(i) READ - optional - default: no reading of DATR

This keyword, when used, instructs the program to call subroutine READER which, in turn, will read the data in file DATR. Subroutine READER is user written and can be used to transform the raw data into a more suitable form.

(ii) NDGS I1 I2 I3 I4 - compulsory

The set of stream variables for each plant.

Plant I1, has I2 D-classes, I3 G-classes and I4 S-classes. This information must be given for every plant in the system. The order is irrelevant. Note one is the minimum number of D, G or S-classes.

(iii) MINS I1 - optional - default: one mineral called ORE

I1 gives the number of mineral species. The following I1 cards contain the names of each of the minerals.

(iv) CNCT I1 I2 I3 I4 - optional - default: no interplant connections.

Plant I1 stream I2 is connected to plant I3 stream I4. Repeated for each connection.

(v) SIZE - optional - default: nothing

The size of particles in each D-class need only be given if the values are required in the unit models. The library subroutines expect size 1 to be the largest. The sizes are given on the following card (cards).

(vi) SPGR - optional - default: *nothing*

Specific gravity is a physical property but the option of presenting it at this stage is given so that if the distribution of minerals in each G-class is given in terms of mass, then the distribution, in terms of volume can be determined immediately and vice versa. Usually the specific gravity of each mineral is known but if SG is to be used as a separating property then the SG of each G-class is what is required. This is done automatically if SPGR is given when the mineral distribution, in terms of mass or volume is also given.

(vii) GRDM - optional - default: *nothing*

This keyword indicates that the next N cards, where N is the number of G-classes, will contain a mineral distribution in terms of the mass fraction of each mineral in that G-class. The order of mass fractions must be the same as that for the minerals following MINS.

If SPGR has been given this mineral distribution in terms of volume fractions will be calculated, also the specific gravity of the material in each G-class will be stored as physical property one.

This is necessary if output code 5 or 6 is to be used.

(viii) GRDV - optional - default: nothing

This is the same as GRDM except related to volume instead of mass fractions. If SPGR is given but GRDM has not been given it will be calculated, the specific gravity of each G-class is put into physical property one.

If both GRDM and GRDV are given plus SPGR and they have conflicting values, GRDV will dominate.

(ix) PHYP I1 I2 - optional - default: nothing

Physical properties can be related to , G or S-classes or to some combinations of classes, but the fundamental property is that of each mineral. If the user wishes to read in a set of values to be used in a unit model then

I2 gives the number of elements to be read from the next card (cards). I2 must be less than,or equal to,one thousand. I1 is the property number.

A further facility is provided. Frequently the property of a pure mineral is known and the value of this property, for a mixed grain,is given by one of:

1001: $\Sigma M_i P_i$ - Sum of pure property in proportion to mass fraction

1002: $\Sigma V_i P_i$ - Sum of pure properties in proportion to volume fractions

1003: $1/\Sigma(M_i/P_i)$ - Sum of pure properties in proportion to inverse mass fractions

1004: $1/\Sigma(V_i/P_i)$ - Sum of pure properties in proportion to inverse volume fractions

When I2 is one of 1001-1004 then NMIN (the number of mineral species) elements are read on the following card and converted to the property for each G-class in accordance with the code. For example if the specific gravity of each mineral was given with the code 1002 or 1003 then property I1 would contain the specific gravity of each G-class.

To use this facility GRDM or GRDV or both must have been entered.

The property numbers, I1, need not be presented in order and unused property numbers are allowable. If SPGR has been given along with either GRDM or GRDV then property one, will refer to the specific gravity in each G-class, and should not be reused.

(x) A blank card to indicate the end of the system data.

14.4.2 Plant Data

This is data related to each plant individually and cannot be changed during the simulation.

(i) PLNT I1 - compulsory

I1 is the number of the plant for which data is to be given. I1 can be any number but if gaps are left storage is wasted. The plant data for each plant must be given in the order of calculation, so that all the feeds to each plant are available when its turn for calculation comes.

(ii) SCNA I1 I2 I3 - compulsory

This is the stream connection matrix and gives the plant configuration. Stream I1 links unit I2 to unit I3. This is repeated for every stream in the plant. Feeds to and outputs from the plant come from and go to unit zero. Outputs 1, 2 and 3 from units correspond to the order of presentation of streams in the stream connection matrix.

(iii) WTCR I1 - optional - default: 2

Tells the program which weighting criterion is to be used to weight the streams for the tearing algorithms.

I1 = 0 Minimum number of tears $W_k = 1$ all k, no change in ordering phase

I1 = 1 Minimum number of double tears $W_k = m_k$ in ordering phase

I1 = 2 Minimum number of double tears plus set with fewest tears $W_k = A.m_k + 1$ in ordering phase

(iv) SIWT I1 I2 - optional - default: $W_k = 1$ all k

In the input phase $W_k = 1$ for all k, is the weighting to which all the streams are initially set. Using this keyword the initial weightings can be changed. Stream I1 becomes weighted I2. So the user can weight all the streams, or simply weight preferred tear streams, to be zero. If the weighting criterion is zero then the weightings are unchanged. If the criterion is one, or two, then only streams with zero weight are left as zero. The remainder are weighted according to the criterion in the ordering phase.

(v) WTRF I1 - optional - default: none

This keyword indicates that there is a water feed to unit I1. The following card contains two items of information.

The first gives the mass of water feed to the unit and the second says whether this mass feed is fixed or if it can vary to give a fixed percent solids in the output stream. If the second item of information is set to zero then the first bit of information will be the fixed water feed. If the second is non zero then this will represent the percent solids required in the output stream.

As only mixer units may have more than one input stream these water feeds can only be added to MIXR units.

This facility can be used to model filters, thickeners and dryers, if the specification of percent solids in the underflow is adequate. In these cases the water feed will have a negative value as water is being removed from the system.

(vi) STRM I1 I2 - compulsory for system feeds

This is used to give an initial value to stream I1. If the stream is not a feed, or tear, it will be ignored. I2 tells the simulator how the data will be presented.

I2 = 1 Read the mass in each category. Column i corresponds to D-class i. The first NDC (number of G-classes) rows correspond to S-class 1 and the k^{th} NGC rows correspond to S-class k.

I2 = 2 Read mass fraction distribution.

The streams are discretized in terms of size, grade and surface categories. Each is first distributed into the D-classes. Then each D-class is distributed into G-classes and each G-class is distributed into S-classes. If some stream has NDC D-classes, NGC G-classes and NSC S-classes, then the first row of data will have NDC elements giving the distribution of particles into D-classes, the next row the distribution of D-class one into the NGC, G-classes (NGC elements) and the following NDC rows the distribution of D-class one and G-class one to NGC into S-classes (NSC elements per row). This is repeated for D-class two, three, etc. to NDC.

The sum of the elements in each row must be one as they are mass fractions. This implies that if there is only one D-class, then the first row giving the D-class distribution will have only one element equal to one. This also applies to the G and S-classes. The program has been written so that these single ones need not be put in the data. If for example there is only one D and one G-class and four S-classes then the data will consist of a single row with four elements giving the distribution of particles among S-classes.

Immediately after the keyword STRM the total mass of solids and percent solids in the stream are given. When I2 = 1 then the sum of masses in each category is checked against the total mass. Note zero per cent solids is impossible and not allowed.

When the stream is an output from one plant and a feed to another and the stream variables need conversion then a further consideration is necessary to describe the stream. In the 'converter' the previous plant output is condensed to a common base with the required feed stream, and then expanded to the required stream variables using the distribution given by STRM. If, for example, the common base are D and G-classes and it is required to expand this to D, G and S-classes then only the S distributions for each D, G-class must be given. First the total mass of solids is set to one. Then, where normally D and G distributions are given, every element is set to one. This has the desired effect.

In general all distributions which are in the common base have their elements set to one. Further details of this are given in a later sub-chapter, 14.8.

(vii) A blank card to indicate end of plant data.

14.4.3 Run Data

This data is related to the immediately preceeding plant data and is connected only to this one plant. However each plant may have several sets of run data, allowing the user the change convergence method, output codes and unit model types and parameters.

(i) CONV I1 - optional - default: 3

The convergence method is chosen by I1.

If I1 = 1 Direct Substitution
If I1 = 2 Bounded Wegstein
If I1 = 3 Reduced Newton

(ii) ITER I1 - optional - default: 8

The maximum number of iterations allowed for this run is I1.

(iii) TOLM I1 - optional - default:-4

The tolerance limit for convergence is 10^{I1}. There is no purpose in setting the tolerance limit less than 10^{-4} because machine roundoff will not provide better accuracy.

The order of presentation of CONV, ITER and TOLM is irrelevant.

(iv) TYPE I1 NAME I2 - compulsory for first plant run

Unit I1 requires I2 parameters and the model is stored in subroutine NAME. The I2 parameters are read from the following card (cards). This is repeated for each unit. In subsequent runs TYPE statements with parameters need only be given for those units whose type or parameters are changed.

(v) COPY I1 FROM I2 - optional

This statement instructs the program to copy the type and parameters for unit I1 from unit I2. The TYPE statement of unit I2 must be previously given. In general the COPY statements can be dispersed amongst the TYPE statement. The purpose of the COPY statement is to avoid repetition of long parameter lists.

(vi) OUTC - optional - default: 0 0 0 1 0 0 1 8 8 8

There are ten elements in the output code each can have several values and correspond to a particular output.

In the following list the 'output station' refers to the point in program CALC where results can be outputted, and correspond to those in the logic diagram in sub-chapter 13.4. Elements

seven to ten fix the output types of feeds, products and tears for the plant at convergence.

OUTPUT STATION		TYPE	AVAILABILITY
1	Element 1	Plant feeds	Initially
2	Element 2	Node tears	Initially for each node
3	Element 3	Unit outputs	Each unit, every iteration, complex node
4	Element 4	Iteration number RMS error Water feeds	
5	Element 5	Node tears	Every iteration, complex nodes
6	Element 6	Unit outputs	Simple nodes
7	Element 7	Water feeds	At convergence only
7	Element 8	Plant feeds	At convergence only
7	Element 9	Plant outputs	At convergence only
7	Element 10	Plant tears	At convergence only

The elements can be divided into three types:-

(a) Elements 1, 2, 3, 5, 6, 8, 9, 10 refer to streams and can have types of outputs depending on the code.

Code = 0 Nothing printed

Code = 1 Total mass solids, water and mass in each category, every iteration

Code = 2 Total mass solids and water only, every iteration

Code = 3 Same as code 1 but only at convergence

Code = 4 Same as code 2 but only at convergence

Code = 5 Total mass solids, water plus mass and grade of each mineral species at convergence only

Code = 6 Same as code 5 with the mass in each category added

Thus the case of element 9 if codes 5 or 6 are requested the recovery of each mineral species to each output stream from the feed streams will be given. The proviso in getting these recoveries is that element 1 or element 8 has code 5 or 6, so that the amount of each mineral in the feeds can be calculated.

(b) Element 4 has four possible output codes:

Code = 0 Nothing printed

Code = 1 RMS error, iteration number every iteration

Code = 2 Same as code 1 but only at convergence

Code = 3 RMS error, iteration number and water feeds every iteration

(c) Element 7 has only two possible codes:

Code = 0 Nothing printed

Code = 1 Water feeds at convergence

At the end of each run the output code returns to the default values unless told otherwise.

(vii) A blank card to end run data.

If several runs are required the new run data for the plant follows immediately prefaced by a data line reading 'RUN DATA' as before.

14.4.4 End the Data File

STOP - compulsory

This keyword is inserted at the end of the last run of the last plant.

14.5 Writing of Subroutines

When none of the library subroutines are suited to a user's needs it is possible for the user to write his own model. The first statements of the subroutine must be as follows:-

```
SUBROUTINE NAME (TMSF, TMS1, TMS2, TMS3, FEED, OUT1, OUT2, OUT3, DER1,
DER2, DER3, NDI, NGC, NSC, WTR, WTR1, WTR2, WTR3, SIZE, PARAM, PPROP,
INOPP, IFL, NPP, GROM, GROV, NMIN, NGCM) .

INTEGER IFL, INOPP(NPP,2)

REAL FEED(1), OUT1(1), OUT2(1), OUT3(1), DER1(1), DER2(1), DER3(1)
REAL SIZE(1), PARAM(1), PPROP(1), GROM (NGCM, NMIN), GROV (NGCM, NMIN)
COMMON NPLNT, NUNIT, ITER, IW
```

The *feed, output* and derivative streams are strings but it is possible for the user to make them into 2 or 3 dimensional arrays. To do this the dimension statement becomes

REAL FEED (NDC, NGC, NSC), OUT1 (NDC, NGC, NSC) .....

or

REAL FEED (NDC, NGC), OUT1 (NDC, NGC).....

or

REAL FEED (NDC, NSC), OUT1 (NDC, NSC] .....

or

REAL FEED (NGC, NSC), OUT1 (NGC, NSC) .....

depending on the stream variables for the particular plant.

SIZE(I) I = 1,NDC gives the sizes of each D-class in the original order.

PARAM(I) is the unit model parameters in the order given in the data.

PPROP(I) is a string of all physical properties for the system.

Individual properties can be found using the index INDPP(I,J). Property I has its first element in string PPROP in position INDPP(I,1) and has INDPP(I,2) elements.

The COMMON is simply a convenience to inform the subroutine of the plant unit and iteration numbers, and the output code for write statements (WRITE (IW, ...)). If none of this information is required the COMMON may be left out, however if further unlabelled COMMONS are to be used to create dummy arrays for computation or to transfer data between READER and the subroutine this COMMON should remain.

IFL is a flag that is set to zero by the simulator, if the user wishes to use derivatives in the convergence procedure, he must calculate these derivatives and then set IFL to 1.

Another use of IFL is to inform the simulator that a fatal fault has been diagnosed in the subroutine and the simulation must be stopped. To do this IFL is set to 3.

14.6 Features and Limitations of the Converter

Although each plant can have its own set of stream variables the size categories for the system are fixed, therefore it is unreasonable to change the size distribution in going from one plant to another. The only possible sensible D-class conversion is to remove D-classes which no longer contain significant amounts of material. A common occurrence

of this situation is in moving from the milling plant to the separating plant. If the milling plant is operating effectively there should be very little material in the top few D-classes of the stream being feed to the separation plant. If the user of the simulator knows this to be the case he can reduce storage and computation effort by eliminating the top few D-classes in the separation plant. To facilitate this elimination the 'converter' assumes that if the number of D-classes is reduced in going from one plant to the next, that the first (NDM-NDN) D-classes (largest) are to be eliminated, where NDM and NDN are the number of D-classes in the feeding and receiving plants respectively. If the receiving plant has only one D-class (NDN=1) then normal condensation into (summing of elements in every D-class for each G and S-class) a single D-class is done. (The implicit assumption being that size is irrelevant in the receiving plant).

Similarly to size categories there is no simple way of converting from one grade distribution to another. Even knowing the mass fraction of every mineral in each G-class for the feed plant, and being given this for the receiving plant, the material distribution in the G-classes of the receiving plant will not be defined. If on the other hand the distribution of material amongst the G-classes of the receiving plant is fixed the mass balance for each mineral component will not hold.

This means that the only sensible conversions for G-classes are increasing it from one to several or reducing it from several to one.

If the user wishes to make a conversion other than the two mentioned it can be done at his own peril. However in the case when both the feeding and receiving plants have the same number of G-classes the 'converter' will assume that they are the same and no conversion of G-classes will take place. If in this case the user intended conversion and fed data under keyword STRM to effect the redistribution of material between G-classes erroneous results will occur without warning.

It is not unreasonable for two plants to have the same number of S-classes even though the S-classes may refer to different properties. As with G-classes the 'converter' will not realize that conversion is intended and faulty values will result. If these problems should arise a way of getting around it is to insert a plant consisting of a single unit between the original two plants. The single unit is a mixer and the stream variables of this plant are set to the common classes of the original plants. For example if the feeding plant has 5 D, 3 G and 4 S-classes and the receiving plant 5 D, 1 G and 4 S-classes, the

inserted plant should have 5 D, 1 G and 1 S-class. Then the output from the inserted plant can be converted satisfactorily to the required 5 D, 1 G and 4 S-classes. This technique can be applied to G-classes and D-classes as well if unusual conversions must be made.

14.7 Running the Simulator

The method of setting up a job varies from machine to machine, therefore the method outlined in this sub-chapter applies only to the 'Wits' IBM installation. However the general layout will be a useful guide in fitting the simulator elsewhere.

The 'Wits' system is divided into two major parts,

(i) OS - Operating System - batch jobs

(ii) WITS - Interactive System .

It is usually more efficient computer time-wise to run on 'OS' particularly for large problems, however it is only in special circumstances that files can be stored under OS. This has meant storing the programs on WITS and then transferring the programs and data files to OS for execution.

14.7.1 Running on OS

It has been described in chapter 13 how the simulator is split into four phases, and in sub-chapter 12.2 how each phase relies on a previous phase or phases to provide the *dimensions* and *calling statements* as well as data. This information manipulation is done on the computer using 'files' each of which has a name. Table 12 summarizes these files. The blocked file names DIMNP, INPT, ORDER, CALC and UNIT are the names of the files containing the source listing of the simulator and the double underlined files DATT and DATR are data files that must be created by the user. Sub-chapter 14.4 details the form of DATT, and the form of DATR is explained in sub-chapter 12.5. DATR is optional but even if there is no data to be inputted via DATR, the empty file DATR must still exist. The *singly underlined file name* UNITS is the WITS file into which unit model subroutines must be inserted. It should exist but be empty when there are no user written subroutines. The remaining files are working files which are used to transfer information from phase to phase. Before the job can run all these files PHI1, PHI2, PHI3, PHI4, PHI5, PHO1, PHO2, DMINP and PHC must exist, even if they are empty.

The source programs INPT, ORDER, CALC and UNIT have been written as subroutines to allow their storage as compiled versions (see sub-chapter 12.3). These compiled version are stored on OS so that when phases 2,

3 and 4 are run these large files are not transferred from WITS to OS and are not compiled each time. This helps reduce costs but means that DMINP, PHI2, and PHO2 besides containing the dimensions for the respective phases also contain CALL statements.

TABLE 12 SUMMARY OF SIMULATORS FILES AND THEIR FUNCTIONS FOR RUNNING ON OS

Phase	Run Instruction	Data Files	Dimensions and Calls	Program	Output Use	Output Files
1	/OS CMPDR			DIMINP	Dimensions for PHASE 2	DMINP
2	/OS CMPIR	DATI	DMINP	INPT	Echo of rearranged data	PHI1
					Dimensions for PHASE 3	PHI2
					Data for PHASE 3	PHI3
					Data for PHASE 4	PHI4
					Unit subroutine calling program	PHI5
3	/OS CMPOR	PHI3	PHI2	ORDER	Results	PHO1
					Dimensions for PHASE 4	PHO2
					Data for PHASE 4	PHO3
4	/OS CMPCR	DATR	PHO2	CALC	Results	PHC
		PHI4		UNIT		
		PHO3		PHI5		
				UNITS		

To run a job on OS a file containing the necessary JCL (Job Control Language) must be written. These are CMPDR, CMPIR, CMPOR and CMPCR for phases one to four respectively.

The steps in running the simulator on OS can be summarized as follows:-

(i) Ensure that files PHI1, PHI2, PHI3, PHI4, PHI5, PHO1, PHO2, PHO3, PHC, DMINP, UNITS, DATT and DATR exist.

(ii) Insert data into DATT as described in sub-chapter 14.4.

(iii) Insert data into DATR if required.

(iv) Insert user written unit subroutines into UNITS.

(v) /OS CMPDR if standard dimensions given in sub-chapter 13.1 are sufficient, otherwise modify program DIMINP.

(vi) /OS CMPIR then check result (file PHI1) for consistency with the problem - possibly there is an error in the data.

(vii) /OS CMPUR then check result (file PHO1) for consistency with the problem - possibly there is an error in the data.

(viii) /OS CMPCR this gives final mass balance (file PHC).

Note Files INPT, ORDER, CALC and UNIT must be available in the compiled form for this JCL.

14.7.2 Running on WITS

Although running on OS is what the simulator is designed for it is possible also to run on WITS. This has the advantage of letting the user obtain results immediately but has the disadvantage of higher costs.

Compilation on WITS is very fast when compared to OS but this is paid for by slower execution, so making running on WITS only suitable for small to moderately sized plants. The conveniences of getting results immediately is great so modifications were made to the simulator to allow running on WITS.

One of the major problems is the fact that on WITS it is only possible to output results into a single file, and in phase two and three the simulator requires five and three output files respectively.

A second major problem is the fact that on WITS data can only be entered from one data file and in phase four, two are required. Both these problems were solved by eliminating phase one and combining phases two and three while using fixed dimensions for this combined phase. This allows the output data from the combined phase to be in the correct order for inputting as a single file to the calculation phase. The use of standard dimensions is necessary as the ordering phase dimensions can no longer be calculated in advance of compilation, but this should

present no serious restriction for running on WITS, as these dimensions can handle up to a moderately large system.

The output from the combined phase must still be split to provide the dimension and call statements and the unit subroutines calling statements as well as the data for the calculation phase. This is achieved by tagging the first element of each line of output with markers to distinguish between the types of output. This tagged output file is then UPDATED to extract the relevant sections, and remove the markers.

Just as in table 12 the file names and their functions are given for running on OS so does Table 13 give similar information when running on WITS.

TABLE 13 SUMMARY OF SIMULATORS FILES AND THEIR FUNCTIONS FOR RUNNING ON WITS

Phase	Run Instruction	Data Files	Dimensions and Calls	Program	Output Use	Output Files
2+3	/XEQ XIO	DATI		INPORD	Echo and results Dimensions for PHASE 4 Data for PHASE 4 Unit subroutine calling program	P1 P2 P43 P5
4	/XEQ XC	DATR P43	P2	CALC UNIT P5 UNITS	Results	

Data files DATI and DATR are identical to those when running on OS. The slightly rearranged combined input and ordering phase is now file INPORD, CALC and UNIT remained as before. The EXEC files activated by the instruction /XEQ contain the verbs for file manipulation and are specific to the 'WITS' systems.

The output file from the calculations phase (PHASE 4) is not stored automatically, but can be viewed by issuing the command /DOUT. If the user wishes to store the result this can be done with the command /KEEP FILENAME. The working files P1, P2, P43 and P5 are purged each time /XEQ XIO is requested and then automatically recreated.

The steps in running the simulator on WITS can be summarized as follows:-

(i) Ensure that files UNITS, DATT and DATR exist.

(ii) Insert data into DATT as described in sub-chapter 14.4.

(iii) Insert data into DATR if required, if not ensure that it is empty

(iv) Insert user written unit model subroutines into file UNITS, if there are none ensure that UNITS is empty.

(v) /XEQ XIO then check result (file P1) for corrections.

(vi) /XEQ XC to get the final mass balance calculation done.

(vii) /DOUT to see the results of PHASE 4.

(viii) /KEEP 'FILENAME' to store the results.

PART III

UNIT MODELS

15 INTRODUCTION TO UNIT MODELLING

In chapters 2 and 4.1 after discussion two fundamental decisions were made regarding the structure of the simulator. The first, that the unit model must calculate its outputs given the inputs, has been based on the facts that this improves the stability (likelyhood of convergence) of the calculations and allows a system to be built from modules. The second, that the streams must be discretized, ensures an accurate mass balance even when the materiel properties are distributed.

These two decisions impose a certain structure on the unit models which must be adhered to. However within this framework many types of models are possible ranging from completely empirical to completely mechanistic. In the next two chapters models are presented for both general and specific types of ore dressing equipment.

Two approaches to the modelling of equipment are possible. For the first approach the equipment's physical dimensions plus operating parameters are specified and from the mass balance the separation characteristics are determined. For the second approach the situation is reversed, separation characteristics are specified for the model, the mass balance done and consequently the equipment sized.

Both approaches have their difficulties. The former because models of this complexity are not generally available. Even if they are they will probably only hold over a small range of feed and operating conditions so that if the initial starting guess is far from the solution nonsensical results could be obtained if convergence is achieved at all. The latter because the separation characteristics specified may not be physically possible. However if the designer limits himself to separation characteristics which are known to be feasible for a certain unit operation this problem will not arise. Alternatively testwork could be done to determine the limits of the separation characteristics on the type of equipment being considered.

A common situation would be for the engineer to select feasible partition curves for various types of equipment and then obtain mass balances for several competing flowsheets. Having decided on the optimum flowsheet configuration the equipment can be sized. Then by using models for which the operating variables are parameters and equipment dimensions fixed it should be possible to find operating conditions which give the required products. Very few models are sufficiently well developed to permit this kind of simulation. But as the flowsheet is fixed and mass flows nearly

fixed the unit models need only be accurate over a fairly small range of conditions. This means it may be possible to do a regression fit to the set of experimental data for the range of conditions and get good simulation results. Alternatively the partition curve which would normally vary with the feed to the unit may be only slightly changed over the conditions required and so can be considered fixed for a particular set of operating conditions.

If the partition curve does very significantly with feed conditions then a feasible partition curve can be chosen for the expected conditions and the simulation done. Depending on whether the conditions have changed significantly new partition curves corresponding to the new conditions can be used for another simulation. This procedure can be repeated in an iterative fashion until the equipment partition curves correspond to the feed conditions.

The techniques mentioned above have been discussed to emphasize the possibility of doing a simulation even when sufficiently descriptive models are not available in the literature or even when the physics of the equipment are not understood. These general purpose models are discussed in the next chapter (16).

16 GENERAL PURPOSE MODELS OF UNITS

16.1 Partition (Tromp) Curve Models

16.1.1 Background

The use of partition curve modelling is fairly widespread and has been suggested by Imazumi and Inoue (1963) for flotation circuits, Loveday (1970) for gravity units, Plitt (1976), Harris (1972) and Luckie and Austin (1973) for classifiers and Rose (1977) and Brereton (1970) for screens. The use of partition curves on magnetic and electrostatic separators and filters does not appear to have been explicitly suggested but it will be seen in sub-chapter 17.4 that these are excellent candidates for this type of treatment.

The S-shaped partitions curve $R_a(x)$ is shown in figure 22 and the corrected partition curve, which excludes the bypass fractions A and B, $R_c(x)$ in figure 23. x is the relevant physical property (e.g. size) and $R(x)$ the fraction of material of property x which reports to output 1 of the separator. Output 1 can be either a concentrate or tailing stream depending on the users requirements. Because various properties can have different ranges of interest in different situations the abscissa is plotted as x/x_{50} for the corrected partition curve, where particles which split equally between concentrate and tail (overflow and underflow) on the corrected curve have the physical property value corresponding to $x = x_{50}$, i.e. $R_c(x_{50}) = 0.5$.

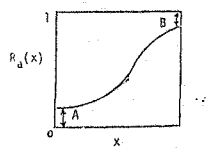


FIGURE 22 ACTUAL PARTITION CURVE

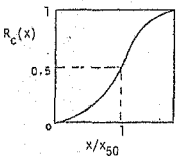


FIGURE 23 CORRECTED PARTITON CURVE

So $R_a(x) = R_c(x).(1-A-B) + A$

The corrected partition curve has been put into functional form by several authors. These are generally two parameter functions. The first is the x_{50} value which is used as a reference value of the property and the second is a measure of the efficiency of the separation.

Three corrected partition curves have been chosen for use in the simulator. They are those of Rao (1966) from Lynch and Rao (1975), Plitt (1976) based on work of Reid and Rosin-Rammler-Sperling, and Tarjan (1974) who has modified a Gates-Gaudin-Schuhmann function. These equations are summarized in table 15.

The parameters of these equations must be found experimentally. Statistical techniques of doing this are presented by Luckie and Austin (1973), Brereton (1970) and Petho and Tompos (1974). A more simple technique which can readily be done manually is to fit the partition curve by eye and then read off x_{75} ($R_c(x_{75}) = .75$), and x_{25} ($R_c(x_{25}) = .25$) and x_{50} ($R_c(x_{50}) = .50$). Then the 'imperfection' defined as:-

$$I = \frac{x_{75} - x_{25}}{2.x_{50}}$$

can be found. Also the other parameters for the equations in table 15 can be found as:-

$$\left.\begin{aligned} \beta &= 0.79/I \\ \gamma &= 0.70/I \end{aligned}\right\} \quad \text{by Tarjan (1974)}$$

and

$$\alpha = 1.54\beta - 0.47 \quad \text{by Plitt (1976)}$$

so that

$$\alpha = 1.22/I - 0.47$$

This procedure determines a partition curve for a particular set of conditions. If one has data for several sets of conditions it may be possible by regression to model the change in parameters due to change in conditions. This is essentially what has been done by Plitt (1976) for his cyclone model.

The user who is doing a prefeasibility study may be satisfied to use general values of the imperfection for various types of equipment so as to avoid doing costly testwork on a process which may be completely impractical. Some of these general values of imperfections have been extracted from the literature to act as a guide to a user of the simulator, these are summarized in table 14.

TABLE 14 IMPERFECTION RANGES FOR SEVERAL EQUIPMENT TYPES

SOURCE	EQUIPMENT TYPE	IMPERFECTION RANGE
Petho & Tompos (1974)	Heavy media bath	0.01 - 0.02
Petho & Tompos (1974)	Heavy media cyclone	0.02 - 0.04
Petho & Tompos (1974)	Jig coarse feed	0.05 - 0.08
Petho & Tompos (1974)	Jig fine feed	0.05 - 0.09
Zimmerman (1975)	Batac jig	0.40 - 0.80
Petho & Tompos (1974)	Hydrocyclone	0.20 - 0.40
Petho & Tompos (1974)	Dorr classifier	0.34 - 0.96
Petho & Tompos (1974)	Resonant screen	0.05 - 0.17

In this simulator the stream variables are three-dimensional, (see sub-chapter 4.1) and are related to the properties of size, grade and any third independent property. The partition curve as described is one dimensional. So to allow separations which are dependent simultaneously on all three dimensions the overall actual partition curve is given by:-

$$R_a^O(x_D, x_G, x_S) = R_a^D(x_D).R_a^G(x_G).R_a^S(x_S)$$

where the super and subscripts D, G and S apply to size, grade and surface classes respectively and O to overall. This equation implies that the separation with respect to each of the properties are independent of each other. Although this is far from perfect it does let the user apply a secondary and tertiary separation criterion to modify the products in a sensible fashion. For example if a separation is being done on the basis of specific gravity and there is a wide range of particle sizes it is possible using this model to reduce the efficiency of fines separation by including a partition curve relating to size which is nearly equal to unity for most of the sizes but drops off for fine sizes.

Another feature included in the model is to allow the partition curve related to size have a maximum. This has been done because several authors like Burt and Ottley (1974) for gravity separators and King (1975), Treher and Warren (1976), Imaizumi and Inoue (1963) and Tomlinson and Fleming (1963) for flotation have found that size is an important secondary separation criterion and the curve showing the effect of size contains a maximum. The maximum in this model is obtained by putting

two partition curves 'back to back' as shown in figure 24.

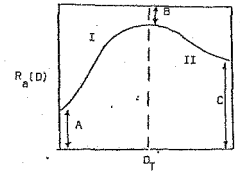


FIGURE 24 PARTITION CURVE FOR SECONDARY SIZE SEPARATION

Defining these two curves (I,II) requires eight parameters. This number of parameters would usually make the model quite hopeless for simulation but because of the ease with which the parameters can be obtained from the real partition curve this need not be a drawback. In any case the program has been written so that only those parameters which refer to required partition curves must be given, so that having the extra facility is not a disadvantage because it need never be used.

It can be seen in figure 23 that the corrected partition curve increases with the property value. So although it is possible to reverse the order of property values when they are provided at the input phase so that D, G or S class one can corresponds to either the largest or smallest physical property value with the remainder in either decending or ascending order respectively, a conflict can arise when two different units require the order in opposite directions. This difficulty is overcome by allowing four further codes in table 15. These extra codes are 11, 12, 13 and 14 and correspond to equations related to codes 1, 2, 3 and 4 respectively, but instead of using the equations directly to give the fractional recovery to output 1, the fraction obtained is first subtracted from one. This permits the material that would have gone to output 2 to now go to output 1, and so the relative directions of increasing recovery can be changed for the class types.

A last point to note is that the corrected partition curves are

written in terms of y in table 15, where

$$y = (x - \lambda)/(x_{50} - \lambda) \quad \text{where } \lambda = 1 \text{ for gravity separators and } \lambda = 0 \text{ for all others}$$

Usually y is x/x_{50} but for gravity separators the separation most often happens in a water medium which has a specific gravity of one, making the ratio $(SG-1)/(SG_{50}-1)$ a more meaningful term. However having this built in feature means that whenever this model is used for gravity separation the density values must be in terms of specific gravity rather than density. This unfortunately restricts the use of G-classes with this model to gravity separations. If the model must be used for some other separation not related to size or gravity the S-classes can be used, or alternatively each property value supplied during the input phase can be increased by one so that y will become $(x+1-1)/(x_{50}+1-1) = x/x_{50}$ during the gravity separation.

16.1.2 Subroutines TRMP - Partition Curve Library Module

(i) Parameters

Many of the parameters which are possible are not always required. For this reason the parameters are split into six sections only the first two of which must always be provided. The parameter order must be the same as that presented below:-

(a) The fraction of feed water reporting to output 1.

(b) Three codes must be entered as parameters and correspond to the D, G and S-classes respectively. The parameter value corresponds to the code and its associated equation in table 15. If the parameter is zero then the corresponding class is not used for separation and $R_a(x)$ for that class is fixed at unity. If more than one class is used it is not necessary to use the same form of the corrected partition curve equation for each.

(c) If G and/or S-classes are required for separation then the physical property number of the associated property must be given here. The order in which the property numbers must be given is G and/or S property, or neither. Note that no property number is given for the D-class separation as size is treated as a separate property, however when D-classes are used for separation the actual size of particles in each D-class must have been provided in the input phase.

(d) When D-classes are used for separation depending on whether both sections I and II or only section I of the curve in figure 24 is used the number of parameters required here will be either eight or five. The eight parameters are in order A, B, ∇_1, $D_{50,1}$, D_T C, ∇_2, $D_{50,2}$ where A,B,C, and D_T correspond to figure 23 and $D_{50,1}$ and $D_{50,2}$ are the D_{50} values for curve sections I and II respectively. ∇_1 and ∇_2 also correspond to sections I and II of the curve, and are the α, β, γ or I parameters for the chosen equation from table 15, depending on the selected code for D-classes.

If only section I of the curve is required then D_T is set to a value greater than the largest particle size given by the size property. For example if the top size particle is 1000 µm setting D_T to 2000 will ensure that the program will not expect C, ∇_2 and $D_{50,2}$.

(e) When G-classes are used for separation four parameters are required. A, B, ∇, X_{50} are the parameters as per figure 22. ∇ is one of α, ρ, γ or I depending on the code number chosen, for equations in table 15.

(f) When S-classes are used for separation four parameters are required here, A, B, ∇, X_{50} are the parameters as per figure 22. ∇ is one of α, β, γ or I depending on the code number chosen corresponding to equations in table 15.

(ii) Physical Properties

Size is required if D-classes are used for separation. When G and S-classes are used for separation the physical property values corresponding to the requested property number must be provided. The number of property values is the same as the number of G or S-class intervals respectively.

(iii) Equations

TABLE 15 CORRECTED PARTITION CURVE EQUATIONS WITH CODES

CODE	EQUATIONS $y = \frac{x^{-\lambda}}{x_{50}^{-\lambda}}$ $\lambda = 1$ for gravity separators $\lambda = 0$ otherwise	PARAMETER
0	Not used for separation	-
1	$R_c(x) = (\exp(\alpha y)-1)/(\exp(\alpha y) + \exp(\alpha)-2)$	α
2	$R_c(x) = 1.0 - \exp(-0.6931.y^{I/0.69})$	I
3	$R_c(x) = 1.0 - \exp(-0.6931.y^{\beta})$	β
4	$R_c(x) = 0.5y^{\gamma}$ $y \leq 1$ $= 1.0 - 0.5y^{-\gamma}$ $y > 1$	γ

(iv) Units

The units of the x_{50} values must correspond to those of the corresponding physical property supplied as data.

16.2 Empirical Models

Empirical model building involves a systematic approach in which independent variables are deliberately altered to determine their influence on a measured response. The fitting techniques are chosen in accord with the form of a model being entertained. The fitted model is then statistically assessed for validity.

In general a model can be represented by the relationship:-

$$y_i = f(x_{1i}, x_{2i}, \text{----}, x_{ni}; a_1, a_2, \text{-----}, a_k) + e_i$$

where y_i is the measured value of the response at the i^{th} settings of the independent variables $x_{1i}, x_{2i}, \text{----}, x_{ni}$. $a_1, a_2, \text{----}, a_k$ are constants and e_i is an error term known as a residual. A literature search can help determine an appropriate form of model. Frequently several alternatives will be proposed. In some cases 'old' process data plotted onto graph paper can serve to assist selection. However caution should be exercised as most 'old' data will have been collected under 'normal' conditions of operation and sometimes important variables will not appear to influence the response because they were not altered over a sufficiently wide range.

Some simple models may be:-

$$y_i = a_0 + a_1 x_{1i} + a_2 x_{2i} + e_i$$

or $$y_i = a_0 + a_1 x_{1i} + a_2 x_{2i}^2 + e_i$$

or $$y_i = -a_0 + a_1 \exp(a_2 x_i) + e_i$$

The experiments should now be conducted so as to vary the variables over as wide a range as required. Once these results have been obtained several methods are available for fitting the parameters. Systematic methods of indicating the experiments and fitting the models, e.g. least squares, 2^n factorial design have been suggested and described by Mular and Bull (1969) and Muler (1972). A multiple correlation method which does not require rigidly planned experiments has been described by Yarroll (1967).

An example of data being fitted to plant data to provide a model of the system is that of MacKay and Lloyd (1975). It appears to have given satisfactory results on the variation of gold losses with respect to grind, cyanidation temperature, pulp density and filter duty. In this example the regression fit has been done for the plant as a whole, however for simulation it will be more usual to model individual units.

Modelling of units in this way will be very easy if the equipment is available for testwork with the ore. If the range of conditions is small then it is very possible that a linear model will be adequate and in this case the model will be ideal for use in this simulator. Even when the range of conditions is not small, linear models are often used, however when the model is non-linear convergence will normally still be reasonably fast.

If this regression fit is to be done it may often be worthwhile considering relating the independent variables to the parameters of the partition curve, (as done by Plitt (1976) for the cyclone model). This may give a better fit as the shape of the curve will be closer to the actual separation curve.

16.3 Subroutine MIXR - Mixer Library Module

This module is the only one which is allowed more than one input stream. No parameters are required as it simply sums the same variables in each of the inputs to get the output stream.

The MIXR can be used as a point for adding or removing water to or from the system. This is done during the input phase using the keyword WTRF, (see sub-chapter 14.4.2(v)) and it is possible to specify total water

in the water feed stream or the percentage solids required in the MIXR output stream.

16.4 Subroutine SPLT - Splitter Library Module

This module allows a process stream to be split into two or three output streams so that a specified fraction of each of the input stream variables report to a particular output stream.

The parameters required are the mass fractions which should report to output streams 1, 2 and 3 respectively. The sum of these three parameters should be unity. If the user wishes to split a stream into two then the third parameter must be set to zero, so that the sum of the first two parameters equals unity.

17 SPECIFIC MODELS OF UNITS

17.1 Modelling of Comminution

For the past two decades power and the related mill size required for grinding operations have been determined from the specific energy necessary to reduce a feed material to a desired product size. Several attempts have been made to develop fundamental laws.

Kick assumed that energy for breakage is proportional to the volume broken, while Rittinger assumed that the energy is proportional to the surface area produced (Jowett (1971)). Both these assumptions can apply in very restricted circumstances and Bond has attempted a compromise using both assumptions. However Bond's work is essentially based on linear regression and is as such empirical.

The defects of these three 'laws' are discussed by Austin (1977), but in the absence of better information the energy-size relationships have served the industry well.

For simulation these energy size relationships are of no use as they fail to account for important grinding circuit subprocesses - breakage kinetics, material transport through the mill and size classification - in an explicit manner (Herbst et al (1973)).

17.1.1 Time Discontinuous Models

According the Herbst and Mika (1970), Kolmogorov and later Epstein used time discontinuous statistical models to describe breakage in terms of a series of discrete breakage events, which were characterized by two functions. These functions termed the selection and breakage functions, in the equivalent deterministic formulation of Broadbent and Callcott (1956), represent the probability of breakage for a given size particle in a given event and the resultant fragment size distribution from that breakage event, respectively.

Defining b_{ij} as the fraction of material in size interval j which falls into size interval i after breakage, and S_j as the probability of selection for breakage of particles of size j. Then in words; the mass size balance is:- 'The mass of material in interval i after one stage of grinding equals the sum of material broken into interval i plus the material in interval i to start, minus the material broken out of the interval'.

(Note: size 1 is largest size interval)

Or $$p_i = \sum_{j=1}^{i-1} b_{ij} S_j f_j + f_i - S_i f_i \quad \text{---------- (17.1)}$$

where f_i = the mass in interval i before the breakage event

and p_i the mass of particles in size interval i after one breakage event

In matrix form

$$P = (bS+I-S).F \quad \text{---------- (17.2)}$$

P and F are vectors, S is a diagonal matrix and b is a lower triangular matrix.

It might be expected that this type of model where the product can be obtained in terms of the feed after a single breakage event will be suitable for a situation where breakage occurs in a series of discrete events and the product of each event is the feed for the next, and particles act more or less independently of each other. This type of situation occurs in crushing where there is a fixed probability that particles in a particular size will be broken in passing through the crusher. For example particles larger than the gap size will have a probability of near one while those much smaller than the gap will have a probability of being selected for breakage near zero.

Now in the crusher there will be several breakage events during the course of a particle passing through so equation 17.2 can be written as:-

$$P = (bS+I-S)^m.F \quad \text{---------- (17.3)}$$

for m breakage events. Besides this within the crusher there is a classification effect as smaller particles undergo fewer breakage events than larger ones due to the smaller ones falling through more quickly. Consequently Whiten (1973) has proposed a cone crusher model which combines in close circuit a breakage model of the form in equation 17.2 and a classifier model. Whiten (1973) claims this approach was successful. Muler and Bull (1989) have successfully used an equation similar to equation 17.3.

An equation of the form of equation 17.2 is very useful for simulation because then b and S values can be selected by the user to give the expected product. Even when no actual data is available this model can be used along with educated guesses concerning the parameters to get a feel for the solution.

In the library module subroutine neither Whiten's (1973) nor equation 17.3 is used because it is very simple for the user to choose a configuration that has a classifier in close circuit with the crusher or has several crushers in series and this allows far more flexibility.

17.1.2 Time Continuous Models

The time discontinuous models discussed so far may be suitable for crushers where the flow approximates plug flow, but in other types of mills there is mixing and so some residence time distribution other than plug flow. Therefore a formulation continuously dependent on time is required.

According to Austin (1971), Sedlatschek and Bass by conventional statistical treatment showed that if a random probability of selection for breakage is applied to a large number of particles a 'FIRST ORDER LAW' will result. This 'law' has been shown to be true experimentally by Kelsall (1964) and Austin (1972) for a narrow size range of homogeneous material. This 'law' can be written as:-

'The rate of breakage of particles in a narrow size interval is proportional to the mass concentration of particles in this size range'.

Or $$r_j \propto w_j \qquad j = 1,n \qquad \text{---------- (17.4)}$$

where r_j is the rate of breakage of particles in size interval j

w_j the mass of particles of size j in the mill

and W the total mass of solids in the mill $W = \sum_{i=1}^{n} w_i$

r_j, w_j and W would normally be functions of time but for steady state simulation in a continuous mill this need not be explicitly shown.

The constant of proportionality in equation 17.4 is chosen as S_j. This S_j is called the specific rate of breakage of size j, and should not be confused with the probability of selection for breakage function used in sub-chapter 17.1.1.

Applying a mass balance over the mill:-

$$\text{OUTPUT} = \text{INPUT} + \text{FORMED} - \text{DISAPPEARED}$$

$$p_i = f_i + \sum_{j=1}^{i-1} b_{ij} S_j w_j - S_i w_i \qquad i=1,n \qquad \text{--------- (17.5)}$$

where the definition of b_{ij} is unchanged from sub-chapter 17.1.1 and is the fraction of material in size interval j which reports to size interval i after breakage.

Before equation 17.5 can be solved assumptions must be made concerning the relationships between p_i, f_i and w_i. Various assumptions and solutions are discussed by Luckie and Austin (1972) and Herbst and Mika (1970). However it has been found by Kelsall and Stewart (1974) and Austin (1977) that the mills residence time distribution is similar to that of several perfectly mixed tanks in series.

For a single perfectly mixed tank the mass concentration of particles of size i in the mill will be the same as that in the product stream, therefore:-

$$\frac{w_i}{W} = \frac{p_i}{\sum_{i=1}^{n} p_i}$$

Substituting for w_i in equation 17.5 gives:-

$$p_i = f_i + \sum_{j=1}^{i-1} b_{ij} S_j p_j \tau - S_i p_i \tau$$

hence $$p_i = \left(f_i + \sum_{j=1}^{i-1} b_{ij} S_j p_j \tau\right)/(1+S_i\tau) \quad i=1,n \quad \text{---------- (17.6)}$$

where $\tau = W/\sum_{i=1}^{n} p_i$ is the mean mill residence time. Equation 17.6 allows the calculation of p_i, i=1,n in sequence when n is the number of size intervals.

e.g. $p_1 = f_1/(1 + S_1\tau)$

$$p_2 = [f_2 + b_{21} p_1 S_1 \tau)/(1 + S_2\tau)$$

$$p_3 = [f_3 + \sum_{j=1}^{2} b_{3j} S_j \tau)/(1 + S_3\tau)$$

$$\vdots$$

$$p_n = (f_n + \sum_{j=1}^{n-1} b_{nj} p_j S_j \tau)/(1 + S_n\tau)$$

and in this way knowing f_i, b_{ij}, S_j and τ it is possible to calculate p_i. If several tanks in series give the best fit to the mill data then in the simulator this model can be used several times in series to simulate the real mill.

Equation 17.6 has been used as the basis for the simulation of cement milling circuits by Austin et al (1975) for autogenous milling by Stanley (1974) and to test the effect of classification in a milling circuit by Kelsall and Stewart (1971).

The determination of S and b values involves many experiments and has been described by Herbst and Mika (1970) and Austin (1977). A back calculation technique has been suggested by Klimpel and Austin (1977) which reduces the number of experiments required. The values are obtained from a series of size distribution measurements on a batch mill, however

functional forms of S and b are used. These are:-

$$S_i = ax_i^{\alpha} \qquad \text{---------- (17.7)}$$

and $$B_{ij} = \phi_j (x_{i-1}/x_j)^{\gamma} + (1 - \phi_j).(x_{i-1}/x_j)^{\beta} \qquad \text{---------- (17.8)}$$

where $\phi_j = \phi_1 (x_j/x_1)^{-\delta}$, $\beta \geq \gamma \geq 0$, x is the size and B_{ij} is the cumulative form of b_{ij}. Other functional forms for B have been used to avoid needing too many parameters. Broadbent and Calcott (1956) tried $B_{ij} = (1-e^{-x_i/x_j})/(1-e^{-1})$. Austin (1971) suggested and Fournier and Smith (1972) used $B_{ij} = (x_i/x_j)^{\alpha}$, while Bennet (1938) and Gaudin and Meloy (1962) have suggested $B_{ij} = 1-\exp(-x_i/\bar{x})$, where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

and $B_{ij} = 1-(1 - x_i/x_j)^{\Gamma}$ respectively. Austin (1971) has shown these last two forms to be equivalent.

The model discussed so far applies only to the particular mill and ore for which the S and B values were found. This is not normally adequate and what is required is a method of scaling up testwork done in a laboratory or pilot plant to the full scale plant.

B has been found by Meloy and Bergstrom (1964), Herbst and Fuerstenau (1973), Kelsall and Stewart (1971) and Austin (1977) to be virtually independent of mill conditions. So if the B values are found for the ore in a test mill then these same values apply to the full scale plant treating the same material. The S values however are strongly dependant on mill conditions. Austin (1977) has used equation 17.7 as the basis for scale up and claims that a can be assumed to be constant for the ore so making a the only variable. a is dependent on ball diameter (d), mill diameter (D), fractional powder filling (f_c), fractional ball filling (J), mill speed (ϕ_c), and ball density (ρ_B). They find

$$a_F = a_T.A.B.C.D$$

Subscript F for full scale and T for test scale

where $A = (d_T/d_F)^{1.5}$

$B = (D_F/D_T)^{0.6}$

$C = (2(0.159/f_{cF})^{0.55}-1)/(2(0.159/f_{cT})^{0.55}-1)$.

$D = m_{BF}/m_{BT}$

and $m_p = 10.7.M.D^{0.3}.(1.0 - 0.937J).(1.0 - 0.1/2^{(9-10\phi_c)})$

for M = mass of grinding media in short tons,

D in feet and m_p in kW/t

so $M = 0.47.J.L.D^2.\rho_B$ (L is the length of the mill).

This allows great flexibility in modelling mills. While factors A and B seem well proven, factors C and D are of rather doubtful value. Overall grinding models are well established and the problems of scale up are being closely scrutinized, this bodes well for models of this type.

17.1.3 Subroutine CRSH - Crusher Library Module

This subroutine is based on equation 17.2 with the modification that S becomes S' x τ where S' is the probability of selection for breakage per unit time and τ the mill residence time. As the probability cannot be more than one $S'x\tau \leq 1$ for each S' value. The reason for introducing this feature is so that the user can modify the probability of all the values in proportion by changing only one parameter, τ. Or if there is more than one crusher it may be desirable to have the same relative probability of selection between size classes but a variable overall values in each crusher.

Equation 17.2 becomes

$$P = (b - I).S'.\tau.F + F \qquad \text{---------- (17.8)}$$

(i) Parameters

All the following parameters must be supplied in the order of presentation:-

(a) NPN - The physical property number which contains the b values. If NPN=0 then the program will use the fixed form $B_{ij} = (1-e^{-x_i/x_j})/(1-e^{-1})$ to calculate the b values and none need be supplied.

(b) τ - The mill residence time.

(c) S' - The n probability for selection values. (One per size interval). Note: $S'_i x\tau \leq 1$ for i = 1,n - so the choice of S'_1 and τ are not completely arbitrary.

(ii) Physical Properties

The lower triangular matrix b is supplied in this way as it is assumed that this information will not be changed for the ore and it may be possible to use the same breakage values for several different units. b is written so that the fraction of material in size j breaking into size i will be in the j^{th} column and i^{th} row of the matrix. As there is no breakage possible from smaller to larger sizes the diagonal and upper triangular elements are zero. The elements of b are supplied to the computer in a string starting with the element in column one and row two. The remainder of the elements in column one follow sequentially and then the element in the third row of column two follows this, until all the lower triangular elements have been inserted into the string.

For example

$$b = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 4 & 0 & 0 \\ 3 & 5 & 6 & 0 \end{bmatrix}$$

then the elements in the string will be 1, 2, 3, 4, 5, 6. It should be noted that the sum of the elements in each column must equal one and if they do not the data will be rejected by the subroutine. This implies that the last element in the string (element 6 for the example) will always equal one. This last element need not be entered as data as the subroutine automatically sets it to one. If n is the number of size intervals then there will be (n(n-1)/2-1) elements in the string of b values.

(iii) Units

The mill residence time must have units corresponding to that of the mass flows. For example if the mill feed is in kg/s then τ must be in seconds.

17.1.4 Subroutine MILL - Milling Library Module

This subroutine is based on equation 17.8 and Klimpel and Austin's (1977) functional representation of the specific rate of breakage S_j and breakage matrix b_{ij}:-

$$S_i = S_1 (x_i/x_1)^{\alpha}$$

$$B_{ij} = \phi_j (x_{i-1}/x_j)^{\gamma} + (1 - \phi_j).(x_{i-1}/x_j)^{\beta}$$

$$\phi_j = \phi_1 (x_j/x_1)^{-\delta}$$

(i) Parameters

The parameters must be presented in the following order and the bracketed values give the range of parameters as found by Austin (1977) in his testwork on quartz, cement, coal and limestone.

(a) α $(0.6 \leq \alpha \leq 1.2)$

(b) β $(2.3 \leq \beta \leq 5.8)$

(c) γ $(0.5 \leq \gamma \leq 1.17)$

(d) δ $(0 < \delta \leq 0.4)$

(e) S_1 $(0.52 \leq S_1 \leq 1.0)$

(f) ϕ_1 $(0.24 \leq \phi_1 \leq 0.67)$

(g) τ

(ii) Physical Properties

The actual size of particles in each size interval with size 1 as the largest size interval.

(iii) Units

The size units can be in any convenient system, but must be consistent.

The residence time must correspond in units to that used for mass flows. For example if the mill feed is in kg/s then τ must be in seconds.

17.2 Modelling of Size Separators

Separation of particles by virtue of their size is an important unit operation, however there is no machine that can do this without having its products influenced by the density and shape of the particles. In this sub-chapter methods of separating particles based primarily on their size will be examined.

17.2.1 Screen Modelling

Gaudin (1939) using the assumptions that the particles are spherical, that the thickness of the screen wire is negligible and that no blinding occurs found that the probability of a particle of size d ($d<a$) passing through a square aperture of side a with wire diameter b is:-

$$p = \frac{(a-d)^2}{(a+b)^2} \qquad \text{---------- (17.10)}$$

This idea has been extended to cubic particles by Calango and Geiger (1973). Their result is a complicated expression which must be evaluated either numerically or graphically.

Whiten (1972) starting with Gaudin's expression (equation 17.10) has developed a complete screen model based on the screen geometry. He improves on Gaudin's method in three ways. First he provides an empirical method for determining the number of times a particle is presented to the screen. Second he claims that within a size interval there is a range of probabilities for particles to pass through the screen, and he solves this by calculating a weighted average value based on the error function. Third submesh material which reports to the oversize is allowed for by setting a fixed percentage of misplaced submesh material.

More recently Brereton and Dymott (1973) and Ferrara and Preti (1975) have split the screening process into two sub-processes, crowded screening where the particles interact and separate screening where there is no interaction.

Brereton and Dymott (1973) have tried to determine the probability of a particle passing through the screen for both crowded and separate screening but have had problems defining the change over or transition point.

Ferrara and Preti (1975) have taken a kinetic approach and relate the kinetic constants to the parameters of a Tromp type partition curve. The kinetic parameters in their turn have for the crowded region been related to d/a and two other readily obtained parameters. For the separate region no relationship has been proposed. This approach looks very promising for screen modelling as it is claimed that physical parameters like the vibration direction can be incorporated into the kinetic constants.

Rose (1977) in a very recent publication has shown that the progress of a screening operation is controlled by material present that is capable of 'blinding' the screen. The differential equations for both the 'blinding' process and the passage of the undersize materials, on the basis of this hypothesis, are deduced and their solutions verified by experiment. However the equations require numerical solution and so are not particularly suited to practical application. Because of this Rose (1977) himself suggests the use of a partition curve which is nearly identical to that of equation code 3 in table 15 and describes exactly how to obtain the equation parameters, from a single experiment, in terms of physically measurable quantities like screen width, length and feed rate.

This model of Rose (1977) introduces a unified theoretical treatment of screening and for the first time presents a basis upon which the performance of screening equipment can be analysed upon a sound theoretical basis.

The use of the partition curve for simulation is ideal and if 'rate of feed per unit width of screen', which is one of the variable parameters, is fixed for a particular situation. The model becomes linear and the simulation very fast. Setting the feed rate per width of screen to a fixed value is a reasonable step when in design mode, as it can then be fixed at a value which gives good screening efficiency. Once the feed rate is known it is possible to select the screen width corresponding to the flow.

A special screening model subroutine has not been included in the library of subroutines as all models discussed resort to the partition curve which is already available as a general purpose subroutine. Therefore, for screen modelling, it is recommended that Rose's (1977) paper is used as a basis to calculate the partition curve parameters, for any case that is of interest, and these derived parameters used in conjunction with subroutine TOMP. If however the user is going to need a lot of screens in his work it may well be worth his effort to write a screen model subroutine suited to his own requirements.

For a quick set of partition curve parameters it may be worthwhile to use a method suggested by Brereton (1970).

17.2.2 Classifier Modelling

Classifiers, other than cyclones, have been rather neglected from the point of view of mechanistic modelling. This could be due to the fact many authors have achieved excellent fits to data using the essentially empirical partition curve, (e.g. Yoshioka and Hotta (1955), Lynch and Rao (1975), Luckie and Austin (1973)) so making it unnecessary for people needing models to go any further.

Plitt (1971) trying to give some basis to the partition curve assumes the classifier can be represented by a perfect classifier, which splits the feed into size classes, followed by a perfect mixer which gives each size class a different residence time. The overflow is represented by the contents of the mixer and the underflow by the combined products. This has the effect of producing a partition curve with the functional form:-

$$R_c(x) = 1 - \exp\left(-0.0931(x/x_{50})^{6}\right) \qquad \text{---------- (17.11)}$$

Reid (1971) from a similar physical model obtains the same corrected efficiency curve.

Harris (1972) discusses both the Plitt (1971) and Reid (1971) models and suggests a more flexible three parameter curve:-

$$R_c(x) = 1 - (1 - (x/x_{50})^{\beta})^{r} \qquad \text{---------- (17.12)}$$

Reid, according to Mular and Runnels (1972), used the curve:-

$$R_c(x) = (\exp(\alpha.x/x_{50}) - 1)/(\exp(\alpha.x/x_{50}) + \exp(\alpha) - 2) \text{ ------ (17.13)}$$

and they show that this and equation 17.11 are special cases of a more general form.

Stewart and Restarick (1967) studied the dynamic flow characteristics of a spiral classifier by means of an impulse of tracer. The classifier was split into four zones each of which act as a perfect mixer with different residence times. Their model gives some insight into the physical routes by which particles find their way to each product and gave good results for a quartz/calcite mixture over a wide range of particle sizes.

The most detailed study of wet classification has been by Schubert and Neesse (1973) who use Reynold's number, eddy diffusion and residence time to show that pronounced turbulence can prevail and that this may be process determining. Differential equations are developed for both the pulp-partition and settling models, and both agree well with experiments. However parameter determination is difficult and the nett result is a fixed partition curve. This approach looks very promising and could form the basis of a mechanistic model.

The previous paragraph demonstrates that although attempts have been made to remove the empiricism from classifier modelling the analyses have always resulted in fixed partition curves (except Stewart and Restarick (1967)). Luckie and Austin (1973) present several other possible partition curves. It seems that given the present state of classifier modelling the partition curves of sub-chapter 18.1 will be as good a model to use as any other.

17.2.3 Cyclone Modelling

There have been several attempts to do a complete mathematical analysis of fluid flow through a cyclone, usually based on the Navier-Stokes

Author Ford Merrill Anthony
Name of thesis Simulation of ore dressing plants. 1979

PUBLISHER:
University of the Witwatersrand, Johannesburg
©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.