covariance matrix elements show the estimator is working well. The parameters are stable and only change if the operating point changes markedly or if the system changes (size disturbance). The transients in the first 80 minutes are due to unrealistic initial states for the grinding circuit (appendix H) and sump settling time.

The mill is driven, in about an hour, to the desired operating point giving a maximum power draft of 0.85 MW and a mill filling of 50 percent. After time 0 it takes about 9 input moves for the optimizer to find the optimum feed rate of 3.96 tonnes/hour and the optimum mill water addition.

Figure 13 on page 80 shows how, after the increased coarseness at 400 minutes, the optimizer increases the feed rate to increase the fractional filling to 63 percent at which maximum power of 0.9 MW is attained. The coarseness is increased by increasing the percentage grinding media in the feed by 20 percent. The mill water is decreased to increase the fractional filling of the grinding media interstices and achieve the best mill pulp loading. Note the transient increase in power.

Figure 14 on page 81 shows how, after the decreased coarseness at 400 minutes, the optimizer decreases the feed rate to decrease the fractional filling to 44 percent at which a maximum power of 0.76 MW is attained. The fraction of grinding media is decreased by 10 percent while maintaining the same mass flow rate in order to simulate a decrease in coarseness. The mill water addition is increased very slightly.

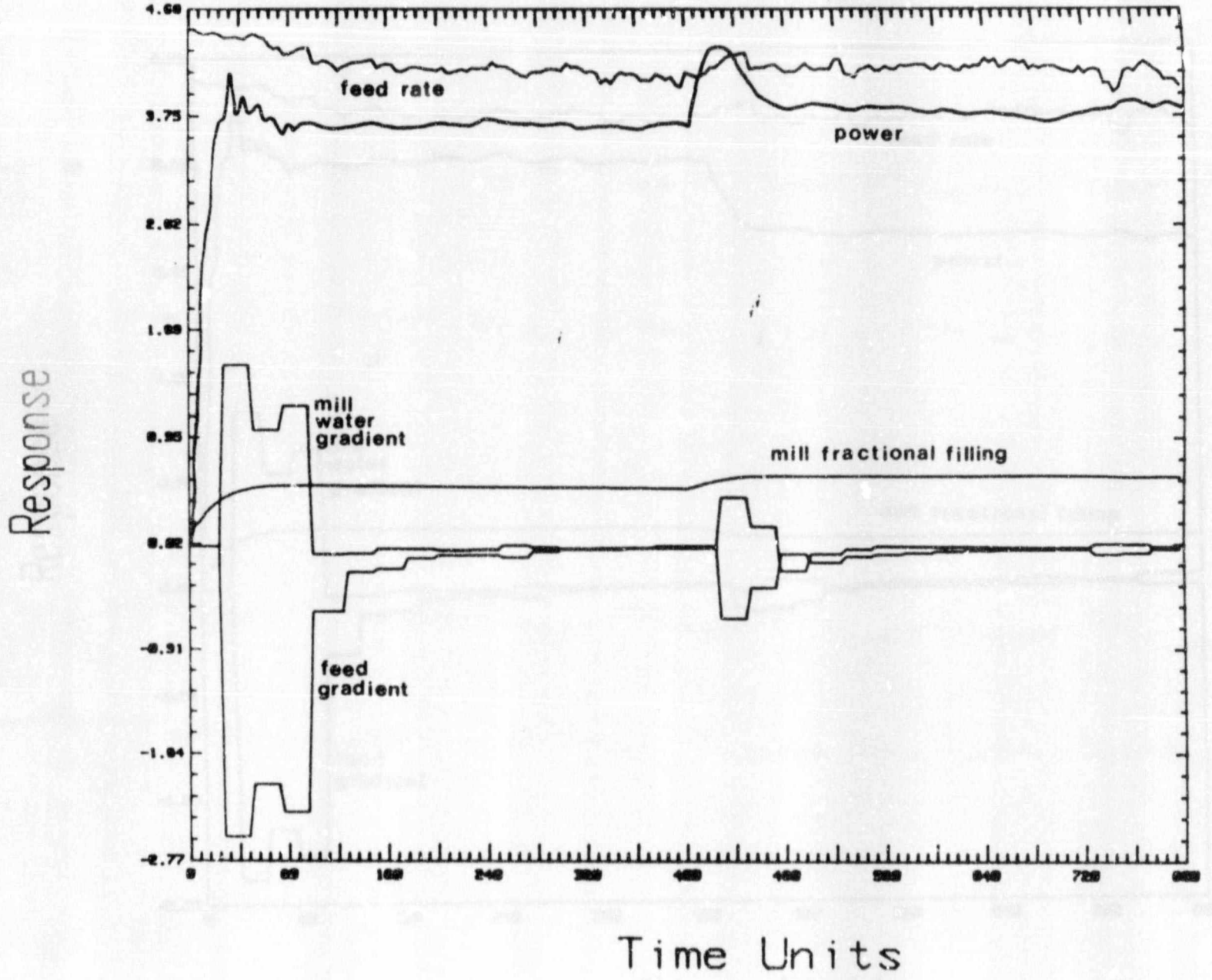


Figure 13. Simulation result using setup file: setup_6

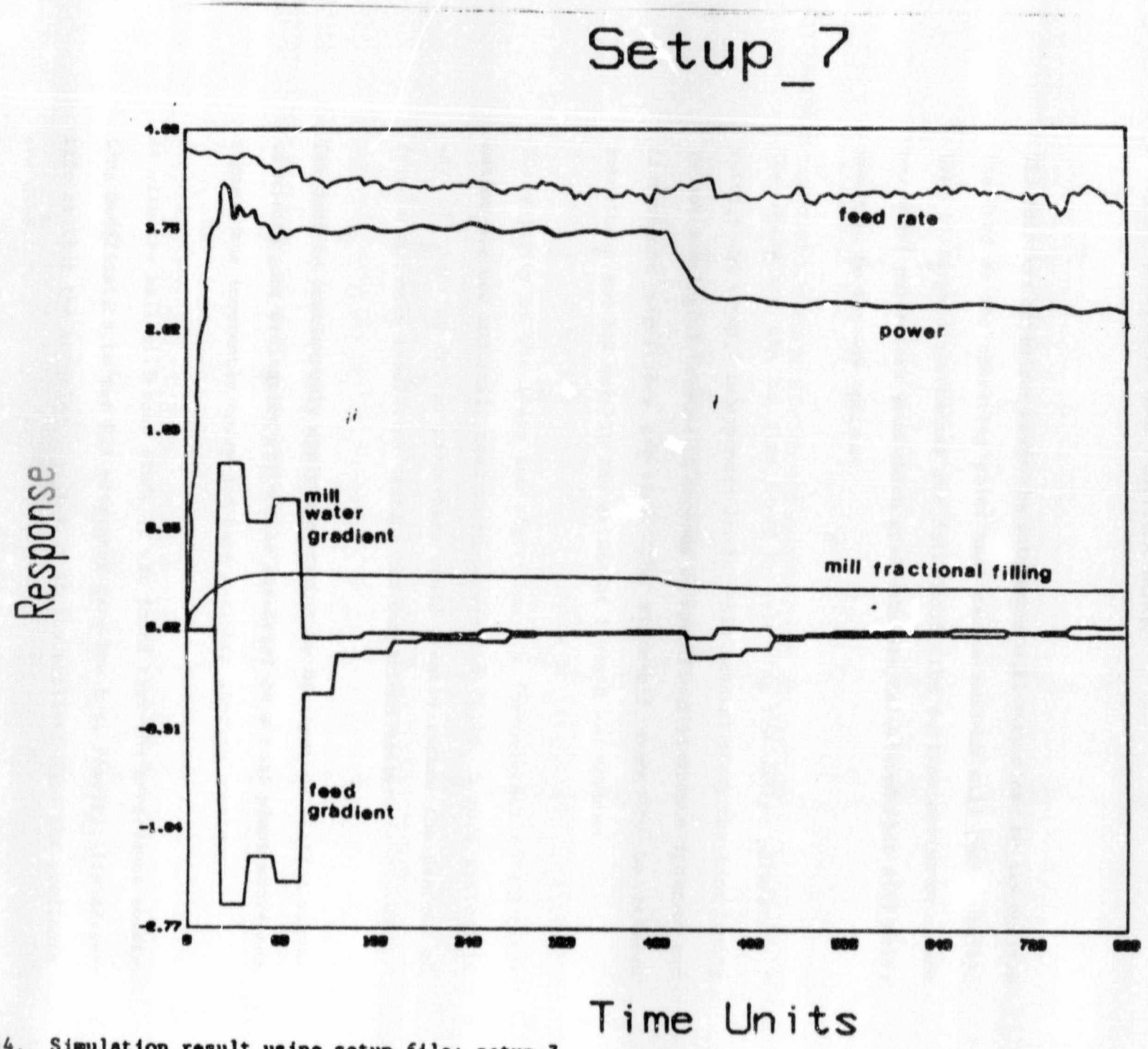


Figure 14. Simulation result using setup file: setup_7

5.6 CONCLUSIONS AND FURTHER WORK

The adaptive optimizer keeps the autogenous mill very close to its optimum (defined as the operating point that causes maximum mill power draft). Even if large disturbances are introduced, the on-line estimator finds new model parameters from which gradients are calculated that will drive the mill to the new optimum.

The optimizer can be fine tuned by selecting the design parameters (setup_6 and setup_7 in appendix J). For this general study the fine tuning is not meaningful because it depends on specific disturbance dynamics and disturbance magnitudes expected. The trade-off seems to be between robustness and how quickly the estimator finds a new optimum.

The majority of the ideas and algorithms for improvements to RLS estimation are new and still need to be understood fully. A more systematic way of choosing design parameters would greatly reduce the degrees of freedom and make simulation testing and comparison easier.

The need to continuously apply a perturbation or input excitation signal to ensure good estimation may not be essential on a real plant where the signals are inherently noisy and have a richer spectral content.

As with the ball mill case study it was found that the covariance resetting modification to the RLS estimator gave the best results. Its advantage is that the covariance matrix is at its smallest when the gradients are used.

6.0 CONCLUSIONS

The concepts and theory underlying an adaptive optimizing regulator have been presented. This adaptive optimizing regulator is intended to improve the performance of a plant where slowly varying disturbances change the desired plant operating point and it is possible to calculate a performance criterion from available measurements. Practical issues concerning the implementation for a real plant are also addressed.

A simulation case study of the adaptive optimizing regulator applied to a simplified ball mill was performed to test the fundamental operation of the optimizer. The simulation results show that the optimizer performs reliably and keeps the plant at its optimum in the face of economically significant disturbances. This study helped identify the conditions and requirements for correct optimizer operation.

Furthermore, it appears as though this kind of controller is directly applicable to the control of autogenous run-of-mine milling circuits and is suitable for implementation in a typical process control computer. It should be seen as complementary to conventional single-variable or multi-variable regulatory controllers in this application and provides a way of integrating such controllers.

The simulations performed to test the optimizer in the autogenous ROM application give very encouraging results, showing that the gradients derived from a simple on-line linear dynamic model describing the mill performance, can be used to drive the mill towards and track a shifting desired operating point. This part of the research has provided a framework for a more detailed, comprehensive and specific simulation

study that could then lead to implementation on an experimental autogenous grinding circuit.

In summary, the significance of this research is to show that a new proposal based on a combination of ideas from the field of modern digital control does have a lot to offer and can address real needs in connection with the control of grinding mills. The research has shown that further practical development of the optimizer is certainly worthwhile.

Limitations of this study include the lack of an overall stability analysis on the theoretical side, and on the practical side, the need to test the adaptive optimizer using real noisy signals.

7.0 REFERENCES

1. Bamberger, W. and Isermann, R. (1978) *Adaptive on-line steady-state optimization of slow dynamic processes* Automatica, vol. 14, 1978. pp.223-230.

2. Blackman, P.F. (1962) *Extremum-seeking Regulators* An Exposition of Adaptive Control, Westcott, A. New York, Pergamon, 1962.

3. Brdys', M. (1983) *Hierarchical Optimising Control of Steady-State Large Scale Systems under Model-Reality Differences of Mixed Type- A Mutually Interacting Approach* Proceedings of the 3rd IFAC Symposium, Large Scale Systems: Theory and Application, Warsaw, Poland, 1983.

4. Brdys', M. and Roberts, P. D. (1984) *Convergence and Optimality of a Modified Two-step Algorithm for Integrated System Optimisation and Parameter Estimation* The City University, Research Memorandum, CEC/MB-PDR/12

5. Bryson, A. E. and Ho, Y. C. (1975) *Applied Optimal Control* New York, Hemisphere, 1975.

6. Duckworth, G. A. and Lynch, A. J. (1982) *The effect of some operating variables on autogenous grinding circuits and their implications for control* Preprints, 14th International Mineral Processing Congress, C.I.M., 1982, III-1 - III-1.21.

7. Edler, J., Nikiforuk, P. N. and Tinker, E. B. (1970) *A comparison of the performance of techniques for direct, on-line optimization* Can. J. of Chem. Eng, vol. 48, 1970. p.432.

8. Ellis, J. E. and Roberts, P.D. (1985) *On the Practical Viability of Integrated System Optimisation and Parameter Estimation* IEE Control 85 Conference, Cambridge, June, 1985.

9. Flook, W.R. (1975) *Automatic control of feed of autogenous grinding media* J. S. Afr. Inst. Min. Metall. vol 76, 1975.

10. Flook, W. R. (1977) *The control of grinding loads in the autogenous and semi-autogencus milling* University of the Witwatersrand Vacation School of Grinding, Johannesburg, Aug. 1977. pp. 126-145.

11. Flook, W.R. and Plasket, R.P. (1982) *Design and control practice for run-of-mine mills in Unica Corporation* Proceedings, 12th CMMI Congress H.W. Glen(editor) Johannesburg, S. Afr. Inst. Min. Metall. (or Geol. Soc. S. Afr.), 1982.

12. Garcia, C.E and Morari, M. (1981) *Optimal Operation of Integrated Processing Systems - part 1.* AIChE J. vol. 27, 1981. pp. 960-968.

13. Garcia, C.E. and Morari, M. (1984) *Optimal Operation of Integrated Processing Systems - part 2.* AIChE J. vol. 30, 1984. pp. 226-234.

14. Goodwin,G.C. and Sin, Kwai Sang (1984) *Adaptive Filtering Prediction and Control* New Jersey: Prentice-Hall, 1984

15. Goodwin, G. C., Lozano Leal, R., Mayne, D.Q. and Middleton, R.H. (1986) *Rapprochement between Continuous and Discrete Model Reference Adaptive Control* Automatica vol.22, no.2, pp 199-207, 1986.

16. Haber, R. and Keviczky, L. (1978) *Identification of Nonlinear Dynamic Systems* Identification and System Parameter Estimation, Rajbman(ed.) North-Holland Publishing Company, 1978.

17. Haimes, Y. Y. and Wismer, D. A. (1972) *A computational approach to the combined problem of optimization and parameter estimation* Automatica, vol. 8, 1972. pp. 337-347.

18. Herbst, J. A. and Rajamani, K. (1982) *The application of modern control theory to mineral-processing operations* Proc. 12th CMMI Congress, Johannesburg, 1982, pp.779-792.

19. Hinde, A.L. (1977) *Control of milling circuits using simple feedback loops*. Vacation school "Grinding Theory and Practice" - South African institute of mining and metallurgy, 1977

20. Hulbert, D.G. and Barker, I.J. (1985) *The control and optimization of milling circuits* MINTEK 50 International Conference on Mineral Science and Technology Publication vol.1, 1985, pp.197-203

21. Jacobs, O. L. R. and Langdon, S. M. (1969) *An optimal extremal control system* Automatica, vol. 6, 1969. p. 297.

22. Keviczky, L., Kolostori, J. and Hilger, M. (1976) *On Simultaneous Optimal Control of a Raw Material Blending and Ball Grinding Mill* Proceed. IFAC Symposium, Johannesburg, pp.143-158, 1976.

23. Kramersh, I.L. (1987) *Modular Modelling of an Autogenous Grinding Circuit* M.Sc. Project Report, Department of Electrical Engineering, University of the Witwatersrand, South Africa, 1987

24. Lynch, A.J. (1977) *Mineral crushing and grinding circuits*. Elsevier Scientific Publishing Company, New York, 1977

25. Maarleveld, A. and Rijnsdorp, J. E. (1970) *Constraint control on distillation columns* Automatica, vol. 6, 1970, p. 51.

26. MacLeod, I. M. (1987) *Series of Private Discussions and Notes on Adaptive Control* - unpublished.

27. Magasarian, O. L. (1969) *Nonlinear Programming* R. Krieger Publ., Huntinton, NY, 1969.

28. Mokken, A. H. and Vlok, F. W. (1986) *Improved automation and control of a run-of-mine ore milling circuit for gold bearing rock at Kinross mines* Private communication, 1986.

29. Nachane, D. M. (1985) *The Integrated Problem of Identification and Optimization* IEE Control 85 Conference, Cambridge, June, 1985.

30. Narendra, K.S., Monopoli, R.V. (1980) *Applications of Adaptive Control* Academic Press, pp. 345-509, 1980.

31. Pauw, O. G., King, R.P., Garner, K. G. and van Aswegen, P. C. (1985) *The control of pebble mills at the Buffelsfontein Gold Mine by use of a multivariable peak-seeking controller* J. S. Afr. Inst. Min. Metall. vol. 85. Mar. 1985. pp. 89-96.

32. Plitt, L.R. (1976) *A Mathematical Model of the Hydrocyclone Classifier* CIM Bulletin, Dec 1976

33. Polak, E. (1971) *Computational Methods in Optimization: A Unified Approach* Academic Press, New York, 1971.

34. Roberts, P. D. and Williams, T. W. C. (1981) *On an algorithm for combined optimization and parameter estimation* Automatica, vol. 17, 1981. pp. 199-209.

35. Saridis, G. N. (1974) *Stochastic methods for identification and control - a survey* IEEE Trans. Automatic Control, vol. AC-19, 1974. p.798.

36. Savas, E.S. (1965) *Computer Control of Industrial Processes* New York, McGraw-Hill, 1965.

37. Sawaragi, Y., Takamatsu, K., Fukanaga, E., Nakanishi, E. and Tamura, H. (1971) *Dynamic version of steady-state optimizing control of a distillation column by trial method* Automatica, vol. 7, 1971. p. 509.

38. Seborg,D.E., Edgar,T.F. and Shah, S.L. (1986) *Adaptive Control Strategies For Process Control: A Survey* AIChe Journal. Vol. 32, No.6, June 1986 , pp.881-913.

39. Smith, C. L. (1972) *Digital Computer Process Control* Scranton, Intext Educational Publishers, 1972

40. Stanley, G.G. (1974a) *The autogenous mill, a mathematical model derived from pilot and industrial scale experiment* Ph.D. Thesis, University of Queensland, 1974.

41. Stanley, G.G. (1974b) *Mechanisms in the autogenous mill and their mathematical representation*. Journal of the South African institute of mining and metallurgy, November 1974.

42. Stanley, G.G. (1977) *The operation of autogenous and semi-autogenous milling circuits*. Vacation school "Grinding Theory and Practice" - South African institute of mining and metallurgy, 1977.

43. Sternby, J. (1980) *Extremum control systems - an area for adaptive control* Proc. 1980 Joint Automatic Control Conference, San Francisco, 1980.

44. van der Merwe, S. (1987) *The Implementation of a General Adaptive Controller Module* M.Sc.(Eng) Project Report, Department of Electrical Engineering, University of the Witwatersrand, 1987.

45. Webb, P. V., Lutter, B. E. and Hair, R. L. (1978) *Dynamic optimization of fluid cat crackers* Chemical Engineering Progress, vol. 74, 1978.

46. Williamson, J.E. (1975) *The automatic control of grinding medium in pebble mills*. J. S. Afr. Inst Min. Metall. Vol. 76, 1975.

47. Wong, K. Y. and Polak, E. (1967) *Identification of Linear Discrete Time Systems Using the Instrumental Variables Method* IEEE Trans. Autom. Contr., AC-12, 707, 1967.

APPENDIX A. AN EXAMPLE OF ROBUST MODEL IDENTIFICATION

An example of the implementation of the theory in section "Coping with Deterministic Disturbances and Plant Noise" on page 24 follows:

Consider the SISO case with one measurable disturbance included in the model. We want to model the objective measurement Ψ as a function of the plant input u (remember that this could be a lower level regulator setpoint or a direct plant input, in fact any input variable that can be used to control Ψ), and as a function of the measurable disturbance z. Choose the model as second order with deadtime d as a multiple of the model sampling time. The backward shift polynomial operators are:

$$A(q^{-1}) = a_0 + a_1 q^{-1} + a_2 q^{-2}$$
$$B(q^{-1}) = q^{-d}(b_0 + b_1 q^{-1})$$
$$C(q^{-1}) = q^{-d}(c_0 + c_1 q^{-1})$$

and the model is:

$$A(q^{-1})\Psi(t) = B(q^{-1})u(t) + C(q^{-1})z(t)$$

Assume the only known deterministic disturbance is a constant offset. This is often the case because the DARMA model is an incremental model. A model of the deterministic disturbance is:

$$r(t) = r(t-1)$$

or in q^{-1} operator notation:

$$(1-q^{-1})r(t) = 0$$

so in this case $O(q^{-1}) = (1-q^{-1})$

From $O(q^{-1})$ choose

$$Q(q^{-1}) = (1 - \varepsilon q^{-1})$$

a polynomial "close" to $O(q^{-1})$ where ε is just less than 1.

$Q(q^{-1})$ in the model denominator should filter out the dc term from the estimator inputs and the value the model predicts.

For the low pass filter ($1/E(q^{-1})$) we choose a highest frequency of interest $\omega_B = 2\pi$ rad/sec. or $f_B = 1$Hz. The minimum degree of E, ∂E must be equal to the degree of A, ∂A, which in this case is 2. Using the rectangular rule to approximate a digital equivalent of a continuous 2nd order low pass filter the Laplace variable can be related to the backward shift operator by:

$$s = \frac{1 - q^{-1}}{\Delta q^{-1}}$$

where $\Delta \equiv$ filter sampling period

and the equivalent of the filter:

$$\frac{1}{(s\tau + 1)^2}$$

where $\tau = \frac{1}{\omega_{3dB}}$

is:

$$\frac{1}{E(q^{-1})} = \frac{q^{-2}}{(e_0 + e_1 q^{-1} + e_2 q^{-2})}$$

where $h = \Delta\omega_B$ and

$$e_0 = 1/h^2$$

$$e_1 = 2(h-1)/h^2$$

$$e_2 = (h^2 - 2h + 1)/h^2$$

Now we have the pre-filter $\frac{Q}{EQ}$ and Ψ_f, u_f and z_f can be determined for the robust model:

$$A(q^{-1})y_f = B(q^{-1})u_f + C(q^{-1})z_f$$

APPENDIX B. EXAMPLE OF A SIMPLE OPTIMIZATION ALGORITHM

This appendix gives a simple example of how the steepest descent gradient search algorithm is applied to a SISO static plant model.

From the dynamic model derived in appendix A, and extracting the steady state model by setting $q^{-1} = 1$ we get:

$$(a_0 + a_1 + a_2)\Psi_f = (b_0 + b_1)u_f + (c_0 + c_1)z_f$$

If the dynamic model is constructed to predict $\Psi'(t)$ rather than $\Psi_f(t)$ (section entitled "Coping with Deterministic Disturbances and Plant Noise" on page 24) then the steady state model will include the filter coefficients e_i. The only difference between $\Psi_f(t)$ and $\Psi'(t)$ is that Ψ_f has been filtered by $1/E(q^{-1})$. Since all the variables are assumed to be in steady state it is valid that Ψ' equals Ψ_f.

The gradient of the objective with respect to the plant input u_f of this SISO, single disturbance, linear, static model is calculated analytically using simple calculus, to give:

$$\frac{\partial \Psi_f}{\partial u_f} = \frac{(b_0 + b_1)}{(a_0 + a_1 + a_2)}$$

If a non-linear model (section entitled "Choice of Model" on page 16) is used then the gradient would not be constant, as it is in the above case, and the derivative would have to be evaluated at the operating point $u_f(t)$.

Note also that if a plant output is included in the objective function calculation, and therefore also in the model, then the chain rule for differentiation needs to be applied.

Applying the steepest descent gradient search algorithm the plant moves are calculated from:

$$u_f(t) = u_f(t-1) + \mu \frac{\partial \Psi_f}{\partial u_f}$$

where μ is the step size to ensure convergence and the + sign finds a maximum and the - sign a minimum. In terms of the model parameters the above equation becomes:

$$u_f(t) = u_f(t-1) + \mu \frac{(b_0 + b_1)}{(a_0 + a_1 + a_2)}$$

Parameters a_i and b_i are calculated using a modified RLS estimator (section entitled "Estimation of Model Parameters" on page 20) with a relative deadzone (section entitled "Coping with Modelling Error within the Bandwidth of Interest" on page 29).

APPENDIX C. STEADY STATE RELATIONSHIPS FOR THE BALL MILL SIMULATION

Steady state relationships for ball mill simulation

variables: mt ...solids mass flow rate out of mill
product ...solids mass flow rate of product
gt ...separator underflow mass flowrate
ft ...fractional filling of the mill
lt ...total mass flow rate into the mill

constants: dd := 0.05 constant to simulate a disturbance caused by a change in ore characteristics or a change in size controller set point or ball or liner wear

$$df := 0.5 - \frac{\left[1 + dd - (dd \cdot (2 + dd))^{0.5}\right]^{0.5}}{2} \qquad df = 0.073$$

df is a variable to simulate a change in the discharge rate caused by one of the abovementioned disturbances. It is coupled so that an increase in discharge rate causes a corresponding increase in product

c := 4 constant for mill discharge calculation

a := 1 constant for product calculation

$$b := c \cdot \left[\frac{1 + dd}{2}\right]^2$$ constant for product calculation

$$n := \left[\left[\frac{1}{2}\right]^4 \cdot \left[\frac{2}{c}\right]\right]^{-1}$$ normalize the maximum product

n = 2

Optimum operating point with no disturbance:

$$product_{opt} := 1$$

$$ft_{opt} := 0.5$$

$$mt_{opt} := c \cdot \left[ft_{opt}\right]^2 + 0.4 \qquad mt_{opt} = 1.4$$

Optimum operating point if disturbance dd is not zero:

$$ft_{opt} := \frac{1 + dd}{2} \qquad ft_{opt} = 0.525$$

$$product_{opt} := n \cdot ft_{opt}^{2} \cdot \left[\frac{-a}{2} \cdot c^{2} \cdot ft_{opt}^{2} + b \cdot c\right] \qquad product_{opt} = 1.216$$

$$mt_{opt} := c \cdot \left[ft_{opt} + df\right]^{2} + 0.4 \qquad mt_{opt} = 1.83$$

$i := 1 \,..200$ iteration variable

$$ft_i := \frac{i}{200}$$ scaling of ft

Calculation of mill discharge:

$$mt_i := c \cdot \left[ft_i + df\right]^{2} + 0.4$$ and in the steady state:

$$lt_i := mt_i$$

Calculation of mill product:

$$product_i := n \cdot \left[\frac{-a}{2} \cdot c^{2} \cdot ft_i^{2} + b \cdot c\right] \cdot ft_i^{2}$$

Derivative of product w.r.t ft:

$$dproduct_i := n \cdot \left[-a \cdot c^{2} \cdot 2 \cdot ft_i^{3} + 2 \cdot b \cdot c \cdot ft_i\right]$$

Derivative of ft w.r.t lt:

$$dft_i := \frac{1}{2 \cdot c \cdot ft_i}$$

Gradient of product w.r.t regulator setpoint lt:

$$gradient_i := dproduct_i \cdot dft_i$$

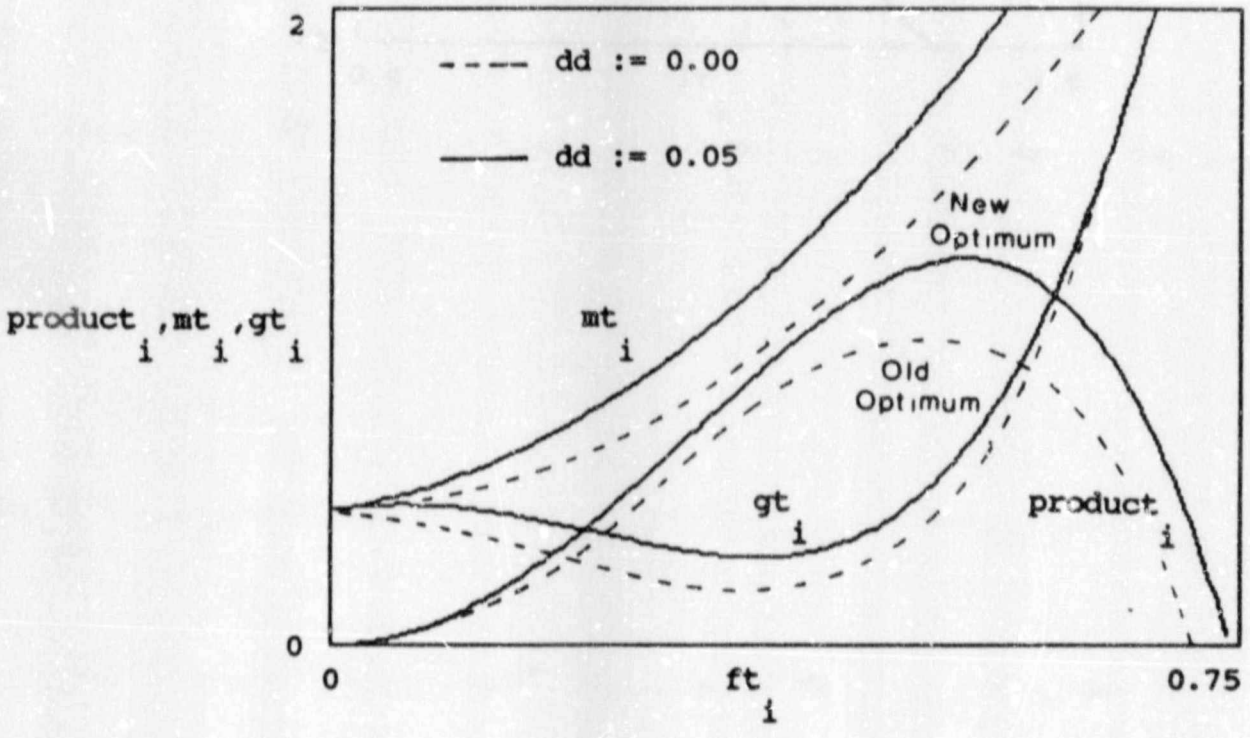
line indicating zero

$$z_i := 0$$

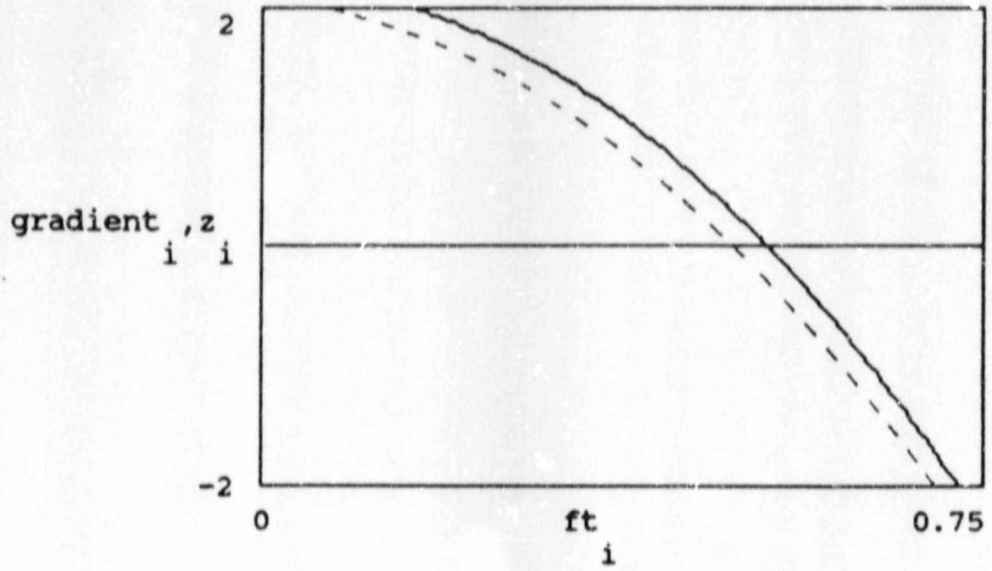
Separator Underflow is calculated as folows:

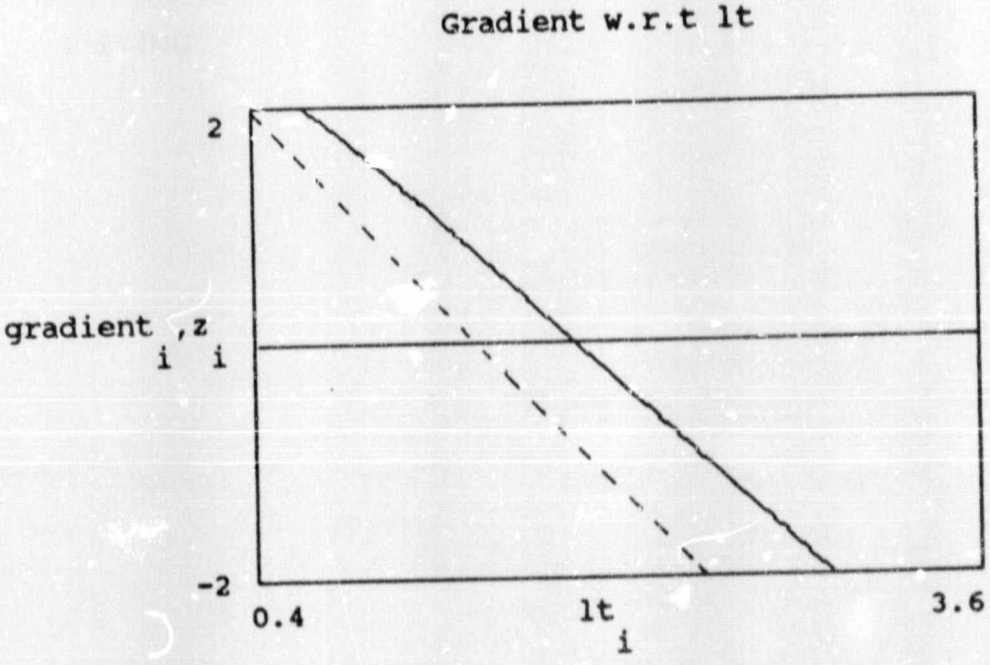
$$gt_i := mt_i - product_i$$

Mill steady state relationships



Gradient w.r.t ft




Gradient w.r.t lt
2
gradient ,z
i i
-2
0.4
lt
i
3.6

APPENDIX D. BALL MILL ADAPTIVE OPTIMIZER PASCAL PROGRAM LISTING

```
{**********************************************************************}
program simulation(input,output);

const no_par=5;
      pi=3.1416;

var
    { simulator variables }
      T:real;              {Time}
      TSTART:real;
      TSTOP:real;STOP:boolean;
      DELT:real; { integration step size for simulation }
      SAMPLE:integer; { controller sampling rate }
                      { as a multiple of DELT    }
      OPT:integer;    { optimizer update time }
                      { as a multiple of SAMPLE*DELT }
      LOG:integer;    { data logging time as a multiple }
                      { of SAMPLE*DELT                  }
      prev_time:real; { variable to store time }
                      { from previous iteration }

     { plant variables }
      x:array[1..2] of real; { x[1] plant state }
                     { x[2] regulator state }
      dx:array[1..2] of real;{ derivative of states }

      { inputs }
          feed_rate:real;
          rt:real; { regulator setpoint }
                   { in steady state rt=lt }
          size:real;

      { intermediate variables }
          lt:real; { flow into mill }
          mt:real; { flow out of mill }
          ft:real; { measure of mill load }
          gt:real; { separator underflow }
          store:array[1..3,0..1000] of real; { history to }
                                             { implement pure delay }
      { outputs }
          product:real; { quantity of product }
          power:real;   { mill power draft }

      { plant constants }
          A:real;  { integrator gain }
          dtm:integer; { mill transport delay }
                       { as a multiple of DELT }
          dts:integer; { separator transport delay }
                       { as a multiple of DELT }
          Ki:real; { regulator integrator gain }

       { variables to implement changes  }
       { in feed characteristics         }
         dist_m_prod:real; { disturbance }
         dist_t_prod:real; { start time for disturbance }
         dist_st_prod:real;{ stop time for disturbance }
         disturbance_store:real;
         dd,ddf:real;
         disturb_rate:real; { constant rate of change of model }

       { variables for noise generation }
          n,bn:longreal;
          bnn:real;
          mag_noise:real;
          tor:real;
          bandwidth:real;
```

```
        { controller variables }
          rt_mag:real;
          rt_max:real;
          init_rt:real;
          start:integer;
          size_sp:real;
          step_size:real;
          opt_start:real;  { optimization start time }
          gradient,max_grad,min_grad:real;

        { estimator variables }
          est_stop_time:real;
          est_start_time:real;
          dead_time:integer;
          sy:array[0..3] of real;
          su:array[0..50] of real;
          sv:array[0..50] of real;
          Pest:real;
          error:real;
          P_mat:array[1..20,1..20] of real;
          init_P_mat:real;
          covariance_resetting:boolean;
          constant_trace:boolean;
          par:array[1..20] of real;
          reg:array[1..20] of real;
          forgetting_factor:real;
          deadband_time_constant:real;
          beta:real;  { deadband scaling constant }
          dz:array[0..1] of real;
          k0,k1:real; { constants for constant trace algorithm }
          ep:real; { high pass filter constant }
          sfu,sfy:array[0..2] of real; { storage for filter }
          est_only:boolean; { to check estimator without the controller }
          debug:boolean;

$include 'plot_var.p'$

{*************************************************************************}
procedure read_setup;
   var path:strpath;
   file1:text;
begin
   writeln('Optimizing regulator applied to Ball Mill');
   writeln('Simulation Study');
   writeln('File to set up simulation: setup');
   path:='/users/bleloch/sim_dir/ball_mill_dir/set_plant';
   reset(file1,path);
      readln(file1,TSTART);writeln('Simulation start time ',TSTART:3:2);
      readln(file1,TSTOP);
      writeln('Stop time ',TSTOP:3:2);
      readln(file1,DELT);
      writeln('Plant simulation sampling interval ',DELT:2:2);
      readln(file1,dtm);
      writeln('Mill pure delay ',dtm);
      readln(file1,dts);writeln('Separator pure delay ',dts);
      readln(file1,A);writeln('Integrator gain ',A:2:2);
      readln(file1,Ki);writeln('Regulator integrator gain ',Ki:2:2);
      readln(file1,dist_m_prod);
      writeln('% disturbance ',(dist_m_prod*100):2:2);
      readln(file1,dist_t_prod);
      writeln('Disturbance start time ',dist_t_prod:3:2);
      readln(file1,dist_st_prod);
      writeln('Disturbance stop time ',dist_st_prod:3:2);
      readln(file1,disturb_rate);
      writeln('Time while model is changing ',disturb_rate:2:1);
```

```
      readln(file1,mag_noise);writeln('mag_noise ',mag_noise:2:2);
      readln(file1,bandwidth);writeln('bandwidth ',bandwidth:2:3);
    close(file1);
end;
{-------------------------------------------------------------------------}
procedure read_controller_setup;
   var path:strpath;
       file1:text;
       i,j:integer;
begin
   writeln('File containing controller settings: set_cont');
   path:='/users/bleloch/sim_dir/ball_mill_dir/set_cont';
   reset(file1,path);
      readln(file1,SAMPLE);
      writeln('Controller sampling period ',SAMPLE*DELT:3:2);
      readln(file1,OPT);
      writeln('Optimization update time ',OPT*SAMPLE*DELT:3:2);
      readln(file1,LOG);
      writeln('Logging interval ',LOG*SAMPLE*DELT:3:2);
      readln(file1,dead_time);
      readln(file1,forgetting_factor);
      readln(file1,ep);
      readln(file1,deadband_time_constant);
      readln(file1,beta);
      readln(file1,init_P_mat);
      readln(file1,covariance_resetting);
      readln(file1,constant_trace);
      readln(file1,step_size);
      readln(file1,max_grad);
      readln(file1,min_grad);
      readln(file1,rt_max);
      readln(file1,init_rt);
      readln(file1,rt_mag);
      readln(file1,size_sp);
      readln(file1,k0);
      readln(file1,k1);
      readln(file1,est_only);
      readln(file1,est_start_time);
      readln(file1,est_stop_time);
      readln(file1,opt_start);
      readln(file1,debug);
   close(file1);
   { initialize storage variables }
   for i:=0 to 50 do begin
      su[i]:=0;
      sv[i]:=0;
   end;
   sy[0]:=0;sy[1]:=0;sy[2]:=0;
end;
{-------------------------------------------------------------------------}
procedure read_initial_parameters;
var path:strpath;
    file1:text;
    i:integer;
begin
   path:='/users/bleloch/si  /ball_mill_dir/initial_parameters';
   reset(file1,path);
      for i:=1 to nu_par do readln(file1,par[i]);
   close(file1);
end;
{-------------------------------------------------------------------------}
function sample_time:boolean;
begin
   if round(T*10) mod round(SAMPLE*DELT*10)=0 then sample_time:=true
   else sample_time:=false;
end;
```

```
{-------------------------------------------------------------------------}
function opt_time:boolean;
begin
   if round(T*10) mod round(OPT*SAMPLE*DELT*10)=0 then opt_time:=true
   else opt_time:=false;
end;
{-------------------------------------------------------------------------}
function log_time:boolean;
begin
   if round(T*10) mod round(LOG*SAMPLE*DELT*10)=0 then log_time:=true
   else log_time:=false;
end;
{-------------------------------------------------------------------------}
function delay(store_no:integer;variable:real;time:integer):real;
var i:integer;
begin
        delay:=store[store_no,time];
        for i:=time downto 1 do store[store_no,i]:=store[store_no,i-1];
        store[store_no,0]:=variable;
end;
{-------------------------------------------------------------------------}
function limit(variable,max,min:real):real;
begin
   if (variable>max) or (variable<min) then begin
      if variable>max then limit:=max;
      if variable<min then limit:=min;
   end
   else limit:=variable;
end;
{-------------------------------------------------------------------------}
function srand48:longreal;external; { Bandlimited Gaussian Noise }
function drand48:longreal;external;
function noise(mag_noise,cutoff:real):real;
const sigma=0.1;
var i:integer;
    accum:longreal;
begin
   accum:=0;
   tor:=1/(2*pi*cutoff);
   { convert from linear distribution to Gaussian distribution }
   for i:=1 to 12 do accum:=accum + 2*(drand48-0.5);
   n:=accum/12;
   { limit bandwidth }
   bn:=bn*exp(-SAMPLE*DELT/tor)+sigma*n*sqrt(1-exp(-2*SAMPLE*DELT/tor));
   noise:=mag_noise*bn;
end;
{-------------------------------------------------------------------------}
function disturbance(disturb_time,disturb_stop:real;
                     disturb_mag:real;disturb_rate:real):real;
begin
   if T<disturb_time then disturbance_store:=0
   else if T<disturb_stop then
      disturbance_store:=disturbance_store
                         +disturb_mag/(round(2*disturb_rate/DELT));
   if T>(disturb_time+disturb_rate)
      then if T<disturb_stop then disturbance_store:=disturb_mag;
   if T>=disturb_stop then
      disturbance_store:=disturbance_store
                         -disturb_mag/(round(2*disturb_rate/DELT));
   if T>(disturb_stop+disturb_rate) then disturbance_store:=0;
   if disturbance_store<0 then disturbance_store:=0;
   disturbance:=disturbance_store;
end;
{-------------------------------------------------------------------------}
procedure reset_P(value:real);
var i,j:integer;
```

```
begin
  for i:= 1 to no_par do
      for j:= 1 to no_par do
         if i=j then P_mat[i,i]:=value
         else P_mat[i,j]:=0;
end;
{------------------- ----------------------------------------------------}
procedure model;
begin
   if T>prev_time then
      dd:=disturbance(dist_t_prod,dist_st_prod,dist_m_prod,disturb_rate);
   if T>prev_time then ft:=delay(1,x[1],dtm) { mill pure delay }
   else ft:=x[1];
   { discharge coupled to product disturbance }
   ddf:=0.5-sqrt(1+dd-sqrt(dd*(2+dd)))/2;
   mt:=4*sqr(ft+ddf)+0.4;  { discharge function }
   { product function }
   product:=2*((-1/2)*16*ft*ft+16*sqr((1+dd)/2))*ft*ft;
   { classifier underflow }
   gt:=mt-product;
   if gt<0 then gt:=0;
   if T>prev_time then gt:=delay(2,gt,dts); { separator pure delay }
   feed_rate:=x[2];
   { nett flow into mill }
   lt:=feed_rate+gt;
   { mill dynamics }
   dx[1]:=A*(lt-mt);
   { solids feedrate regulator dynamics }
   dx[2]:=Ki*(rt-lt);
   prev_time:=T;
end;
{-------------------------------------------------------------------------}
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
PROCEDURE Runge_Kutta;
CONST NSTATE=2;
VAR
    i: INTEGER;
     Xstart, K1, K2, K3:array[1..NSTATE] of real;
Begin
  FOR i:=1 to NSTATE do begin
    Xstart[i]:= X[i];
    K1[i]:= dX[i]*DELT;
    X[i]:= Xstart[i] + K1[i]/2;
  end;
  T:= T + DELT/2;
  model;
  FOR i:=1 to NSTATE do begin
    K2[i]:= dX[i]*DELT;
    X[i]:= Xstart[i] + K2[i]/2;
  end;
  model;
  FOR i:= 1 to NSTATE do begin
    K3[i]:= dX[i]*DELT;
    X[i]:= Xstart[i] + K3[i];
  end;
  T:= T + DELT/2;
  model;
  FOR i:=1 to NSTATE do
    X[i]:= Xstart[i] + (K1[i] + K2[i]*2 + K3[i]*2 + dX[i]*DELT)/6.0;
  T:=round(T*100)/100;
End;  { procedure Runge_Kutta }

{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
PROCEDURE integrator;
var i:integer;
Begin
```

```
   Runge_Kutta;
   IF (T >= TSTOP + DELT) then STOP:= TRUE;
 End;    { procedure integrator }
{+++++++++++++++++++++++++++++{++++++++++++++++++++++++++++++++++++++++++}
procedure initialize;
var i,j:integer;
begin
   STOP:=FALSE;
   step_no:=1;
   for i:=0 to 500 do
      for j:=1 to 2 do store[j,i]:=0;
   { initial states }
   x[1]:=0;
   x[2]:=0;
   rt:=init_rt;
   product:=0;
   bnn:=srand48; { seed random number generator }
   reset_P(init_P_mat); { reset covariance matrix }
   read_initial_parameters; { read initial model parameters }
end;
{------------------------------------------------------------------------}
function step(start:integer):integer;
begin
   if T>=start then step:=1 else step:=0;
end;
{------------------------------------------------------------------------}
procedure regression_vector(y,u,v:real);
var i:integer;
begin
  reg[1]:=sfy[1];
  reg[2]:=sfy[2];
  reg[3]:=sfu[1];
  reg[4]:=sfu[2];
  reg[5]:=1;

  { shift sample storage }
  sy[3]:=sy[2];
  sy[2]:=sy[1];
  sy[1]:=sy[0];
  sy[0]:=y;
  for i:= 0 to 49 do  begin
     su[50-i]:=su[50-i-1];
     sv[50-i]:=sv[50-i-1];
  end;
  su[0]:=u;
  sv[0]:=v;

  { filter high pass}
   sfy[2]:=sfy[1];
   sfy[1]:=sfy[0];
   sfy[0]:=(1-SAMPLE*DELT*ep)*sfy[0]+sy[0]-sy[1];

   sfu[2]:=sfu[1];
   sfu[1]:=sfu[0];
   sfu[0]:=(1-SAMPLE*DELT*ep)*sfu[0]+su[dead_time]-su[dead_time+1];
end;
{------------------------------------------------------------------------}
procedure trace;  { constant trace algorithm }
const     eps=0.005;
var trace:real;
   i,j:integer;
begin
   trace:=0;
   for i:=1 to no_par  do trace := trace+P_mat[i,i];
   for i:=1 to no_par do
      for j:= 1 to no_par do begin
```

```
            if i=j then P_mat[i,i]:=(k0/trace)*P_mat[i,i]+k1
            else begin
                 P_mat[i,j]:=(k0/trace)*P_mat[i,j];
                 if P_mat[i,j]<eps then P_mat[i,j]:=0;
                end;
        end;
end;
{-------------------------------------------------------------------------}
function multiply_vec:real;
var i,j:integer;
    dummy:real;
begin
  dummy:=0;
  for i:= 1 to no_par do dummy:=dummy+reg[i]*par[i];
  multiply_vec:=dummy;
end;
{-------------------------------------------------------------------------}
procedure estimator;
var denom:real;
    K,L:array[1..no_par] of real;
    i,j:integer;
    {=================================}
    function norm:real;
    var K:array[1..no_par] of real;
        i,j:integer;
        dummy:real;
    begin
       for i:= 1 to no_par do K[i]:=0;
       for j:= 1 to no_par do
          for i:=1 to no_par do
             K[j]:=K[j]+reg[i]*P_mat[i,j];
       dummy:=0;
       for i:= 1 to no_par do dummy:=dummy
                                  +K[i]*reg[i];
       norm:=dummy;
    end;
    {=======================================}
begin
   Pest:=multiply_vec;
   error:=sfy[0]-Pest;
   denom:=forgetting_factor+norm;
   for i:= 1 to no_par do begin
      K[i]:=P_mat[i,1]*reg[1]+P_mat[i,2]*reg[2]
           +P_mat[i,3]*reg[3]+P_mat[i,4]*reg[4]
           +P_mat[i,5]*reg[5];{+P_mat[i,6]*reg[6]
           +P_mat[i,7]*reg[7]+P_mat[i,8]*reg[8]
           +P_mat[i,9]*reg[9]+P_mat[i,10]*reg[10];}
   end;
   for i:=1 to no_par do
      L[i]:=reg[1]*P_mat[1,i]+reg[2]*P_mat[2,i]
           +reg[3]*P_mat[3,i]+reg[4]*P_mat[4,i]
           +reg[5]*P_mat[5,i];{+reg[6]*P_mat[6,i]
           +reg[7]*P_mat[7,i]+reg[8]*P_mat[8,i]
           +reg[9]*P_mat[9,i]+reg[10]*P_mat[10,i];}
   for i:= 1 to no_par do
      par[i]:=par[i]+K[i]*error/denom;
   for i:= 1 to no_par do
      for j:= 1 to no_par do
        P_mat[i,j]:=(1/forgetting_factor)* P_mat[i,j]-K[i]*L[j]/denom);
end;
{-------------------------------------------------------------------------}
procedure optimizer;
begin
   rt:=rt+step_size*gradient;
   rt:=limit(rt,rt_max,0);
end;
```

```
{-------------------------------------------------------------------------}
function dead_band:boolean;
const cdz0=0.00001;
      cdz1=0.00001;
      cdz2=0.00001;
      cdz3=0.00001;
      {cdz4=0.2;
      alpha=0.56;}
begin
   if T>SAMPLE*DELT then dz[1]:=dz[0] else dz[1]:=0;
   {if T<=SAMPLE then beta:=sqrt(cdz4+1/(1-alpha));}

   dz[0]:=deadband_time_constant*dz[1]
          +cdz0+cdz1*abs(product)
          +cdz2*abs(feed_rate)
          +cdz3*abs(size);
   error:=product-Pest;
   if abs(beta*dz[0])>abs(error) then dead_band:=true
   else dead_band:=false;
end;
{-------------------------------------------------------------------------}
procedure dump;
var i:integer;
begin
        writeln('Time ',T:3:2);
        writeln('                Regression vector: ');
         write('                ');
        for i:=1 to no_par do   write(reg[i]:2:3,'  ');writeln;
        writeln('                Parameters: ');
        for i:=1 to round(no_par/2) do
            writeln('                  ',par[2*i-1]:3:3,par[2*i]:3:3);
        writeln('               P_mat: ');write('                ');
        for i:=1 to no_par do write(P_mat[i,i]:6:2,'  ');writeln;
        writeln('               Product ',product:2:3);
        writeln('               Mill Power ',power:3:3);
        writeln('               Estimation error ',error:2:4);
        writeln('               Gradient ',gradient:2:3);
end;
{-------------------------------------------------------------------------}
procedure controller;
begin
   if est_only then if opt_time then rt:=rt + rt_mag;
   rt:=rt+noise(mag_noise,banduidth);  { add noise }
   regression_vector(product,rt,size);
   { call estimation procedure }
   if T< est_stop_time then
      if T>est_start_time then
         if not dead_band then estimator;
   { calculate gradient }
   if T>est_stop_time then Pest:=multiply_vec;
              gradient:=(par[3]+par[4])/(1-(par[1]+par[2]));
              {if abs(par[3]+par[4])<0.01 then
                 if abs(1-par[1]-par[2])>abs(par[3]+par[4]) then
                    gradient:=0;}
              gradient:=limit(gradient,max_grad,min_grad);
   if opt_time then if debug then dump; { dump variables }
   { call optimization procedure }
   if not est_only then
      if T>=opt_start  then
         if opt_time then
            if ft>0.3 then optimizer;
   if covariance_resetting then if opt_time then reset_P(init_P_mat);
   if constant_trace then trace;
end;
{-------------------------------------------------------------------------}
procedure log_points(curve_no,step_no:integer;value,T:real);
```

```
    begin
       datamatrix_x[curve_no,step_no]:=T;
       datamatrix_y[curve_no,step_no]:=value;
    end;
{-------------------------------------------------------------------------}
procedure log_data;
   var i:integer;
   begin
     log_points(1,step_no,size,T);
     log_points(2,step_no,product,T);
     log_points(3,step_no,ft,T);
     log_points(4,step_no,feed_rate,T);
     log_points(5,step_no,gradient,T);
     log_points(6,step_no,Pest,T);
     log_points(7,step_no,rt,T);
     log_points(10,step_no,bn,T);
     log_points(8,step_no,gt,T);
     log_points(9,step_no,mt,T);
     log_points(11,step_no,lt,T);
     step_no:=step_no+1;
  end;
{-------------------------------------------------------------------------}
procedure write_data;
   var
       path,path1:strpath;
       i,dummy:integer;
procedure write_file(name:strname;curve_no,no_of_points:integer);
          var directory:strpath;
              i:integer;
          begin
             directory:=path;
             strappend(directory,name);
             rewrite(file_var,directory);
             writeln(file_var,name);
             writeln(file_var,no_of_points);
             for i:=1 to no_of_points do
             writeln(file_var,datamatrix_x[curve_no,i],
                     datamatrix_y[curve_no,i]);
             close(file_var,'SAVE');
          end;
   begin
     path:='/users/bleloch/sim_dir/ball_mill_dir/';
         write_file('feed_rate',4,step_no-1);
         write_file('size',1,step_no-1);
         write_file('product',2,step_no-1);
         write_file('ft',3,step_no-1);
         write_file('Pest',6,step_no-1);
         write_file('mt',9,step_no-1);
         write_file('gt',8,step_no-1);
         write_file('gradient',5,step_no-1);
         write_file('noise',10,step_no-1);
         write_file('power',7,step_no-1);
         write_file('lt',11,step_no-1);
   end;
{*************************************************************************}
begin { main program }
   read_setup;
   read_controller_setup;
   initialize;
   writeln('Running simulation');
   log_data;
   REPEAT
      integrator;
      if sample_time then begin
         controller;
         if log_time then log_da.
```