```
      end;
    UNTIL STOP;
    write_data;
end.   { main program }
{*****************************************************************************}
```

APPENDIX E. FILE CONTAINING PLANT PARAMETERS

PLANT PARAMETERS

0.0	simulation start time
1000.0	simulation stop time
0.2	plant simulation sampling interval
20	twice mill pure delay
2	twice separator pure delay
0.04	integrator gain
1.00	gain of regulator integrator
0.05	magnitude of disturbance
400.0	disturbance start time
700.0	disturbance stop time
10.0	time to complete change in model
0.000	magnitude of noise
0.30	frequency of noise in hertz

APPENDIX F. FILE CONTAINING CONTROLLER PARAMETERS

CONTROLLER PARAMETERS

```
5        controller sampling interval as a multiple of delt
30       optimization update time
3        logging interval as a multiple of the sampling time
2        dead_time as a multiple of SAMPLE
1.00     forgetting factor
0.00     high pass filter constant
0.6      deadband_time constant
1.0      dead_band width scaling factor
100000   initial value for covariance matrix diagonal terms
true     covariance resetting
false    constant trace
0.20     input move step size
1.0      maximum gradient
-1.0     minimum gradient
6.00     rt max
0.8      initial rt
0.16     rt step size to test estimator
0        step start time
0.001    size set point
1000     constant for trace algorithm
150      constant for trace algorithm
false    test estimator only
0        start estimation after this time
1200     stop estimation after this time
50       start optimization
true     debug
```

APPENDIX G. AUTOGENOUS MILL ADAPTIVE OPTIMIZER PASCAL SIMULATION

```
{=========================================================================

{ feed characteristics constants }
    ore_density=2.7; { density of rock }
    water_density=1; { density of water }
    g=9.8;  { gravitational constant}
    frac_grind=0.7;
    frac_fines=0.2;
    frac_req_grind=0.1;
    fines_voidage=0.4;
    media_voidage=0.4;

{ mill model constants }
    L=10; { length of mill in meters }
    R=2.6; { radius of mill in meters }
    Max_load=0.60;
    power_max=1.00;
    overfill=0.1;
    pi=3.14159;

{ sump model constants }
    Vsump=100;  { volume of sump in cubic meters }

{hydrocyclone model constants}
    cut=0.002075; {required cut size in meters}

{physical constants}
    S=2.7;  {density of solids in pulp (g/cm^3)}
    Dc=70; {inside diameter of hydrocyclone(cm)}
    Di=20; {inside diameter of inlet (cm)}
    Du=5; {inside diameter of apex (cm)}
    Dv=40; {inside diameter of vortex finder (cm)}
    grate=0.01; {mill discharge grate size in meters}
    h=140; {free vortex height (cm)}

{statistical constants}
    mean=0.0001; {mean size of particles in fines in cm}
    SD=0.002;   {standard deviation of particles}
                {comprising fines in cm}
    intervals=10;{number of intervals that the}
                 {fines are to be divided into}
{=========================================================================
```

```
{================================================================================}
const
   max_states=10;
{================================================================================}
type
   vector=array[0..max_states] of real;
{================================================================================}
var
{ state variables }
   x:vector;
   dx:vector;

{ note SI units unless specified }
{ inputs }
        feed_rate:real; { solids feed rate }
        mill_water:real; { feed rate of water into mill }
        sump_water:real; { feed rate of water into the sump }
        pump_speed:real; { speed of sump pump }

 { outputs }
        power:real; { mill power draft }
        weight:real; { mass of mill contents in kg }
        sump_level:real; { level in the sump }
        flow:real;       { flow rate to cyclone }
        pulp_density:real;    { density of pulp }
        underflow:real; { volumetric flowrate of underflow }
        overflow:real; { volumetric flowrate of overflow }
        lt:real; { total mass flow into mill }

{ intermediate outputs }
{ mill }
        y0:real; { water discharge }
        y1:real; { required grind discharge }
        y2:real; { fines discharge }
{ intermediate variables }
{ mill }
    Vo:real; { internal volume of mill }
    fc:real;            { fractional volume of fines }
    J : REAL;           { fractional volume of grinding media
    load_volume: REAL;    { volume of load within the mill }
    A : REAL;           { constant for atrition breakage rate }
    T1:real; { volumetric feed rate of required grind }
    T2:real; { volumetric feed rate of fines }
    T3:real; { volumetric feed rate of pebbles }
    T4:real; { volumetric feed rate of fresh grinding media }
    s2:real; { breakage rate for attrition }
    s3:real; { breakage rate for abrasion }
    s4:real; { breakage rate for chipping }
    D : array[3..4,1..3] of REAL;  { breakage functions }
    Fpo: REAL;          { constant for mill discharge }
    coarseness:real; { coarseness of mill load }
    discharge:real;  {volumetric discharge rate of pulp from mill }
    fractional_filling:real; {fractional filling of mill}
    m1:real; { mass of water entering the mill }
    m2:real; { mass of feed entering the mill }
    power_correction:real; { power correction factor }
    permeability:real; { permeability of grinding media }
    mx:real; { slope for power curves }
    my:real; { slope for power curves }
    wear_slope:real; { change in wear rate due to change }
                     { in ore characteristics }
    U:real; {fractional filling of media interstices}

{ limits of operation }
    overload:array[1..4] of real; { power calcultion if not under }
                                  { normal operating conditions }
```

```
      max_coarseness: REAL; { maximum allowable load coarseness }
      min_coarseness:real;  { minimum allowable coarseness }
      minimum_load:real;    { minimum load for normal wear rates }
      excess_wear: REAL;    { increase in media wear rate due to underload }
      reduced_wear:real;    { reduction in media breakage rate }
      water_critical:real;  { critical level for water dilution }
      load_max_power:real;  { fractional filling causing max power draft}

  { steady state scaling constants }
      Pm:real; { mechanical power losses }
      Po:real; { constant for power draft }
      { sump variables }
      { x5 volume of water in the sump }
      { x7 volume of fines in the sump }
      { x6 volume of required grind in the sump }
          { outputs }
              z0:real; {flow of water out of sump}
              z2:real; {flow of fines out of sump}
              z1:real; {flow of required grind out of sump}
  {hydrocyclone model variables }
     {intermediate variables}
              d50,   {cut size}
              Ec,    {probability that particle goes to the underflow}
              m,     {sharpness of separation}
              P,     {pressure drop across the cyclone}
              Ph,    {pressure drop expressed in head of pulp}
              pflow, {hydrocyclone product flowrate}
              probability, {lognormal distribution of particles}
              Rv,    {feed volume to underflow (%)}
              SC,    {amount of feed that bypasses classification (%)}
              solids, {% solids by weight in pulp}
              V,     {volumetric % solids in feed }
              y,     {random variable representing particle size}
              {outputs}
              o1,     {proportion by volume of water in overflow}
              o2,     {proportion by volume of fines in overflow}
              o3,     {proportion by volume of required grind in overflow}
              u1,     {proportion by volume of water in underflow}
              u2,     {proportion by volume of fines in overflow}
              u3,     {proportion by volume of required grind in overflow}
              F1,     {proportion by volume of water in feed pulp}
              F2,     {proportion by volume of fines in feed pulp}
              F3,     {proportion by volume of required grind in feed pulp}
              u1b,u2b,u3b,o1b,o2b,o3b    {b suffix indicates the actual}
                                         {volume and not a fraction}
              :real;
  {=============================================================================
```

```
{**************************************************************************

program main(input,output);

type vec=array[1..20] of rea
     mat=array[1..20,1..20] of real;

const no_par=7; { number of model parameters }
$include 'model_const.p'$  { file containing model constants }

var
    { simulator variables }
    T:real;              { Time }
    TSTART:real;         { Simulation start time }
    TSTOP:real;          { Simulation stop time }
    LOG:real;            { data logging interval, }
                         { or controller sampling time }
    NSTP:integer;        { number of intersamples, }
                         { or plant simulation samling time }
    METHOD:char;         { integration method for solving }
                         { differential equations }
    NSTATE:integer;      { number of state variables }
    DELT:real;           { integration step length }
    i:integer;           { general counter variable }
    STOP:boolean;        { simulation finished }
    magnitude:real;      { magnitude of input fo step generation }
    SAMPLE:real;         { estimator sampling time }
    debug_m,debug_e,debug_c:boolean; { turns debugging on or off }

    { model variables }
    $include 'model_var.p'$  { file containing model variables }

    { controller global variables }
          x_u:array[1..2] of real; { optimizer storage variables }
          step_size1,step_size2:real;  { steepest descent step size }
          gradient1,gradient2:real;    { cost function gradient }
          grad1_max,grad1_min:real;    { gradient limits }
          opt_start:real; { optimization start time }
          opt_update:integer; { optimization update time as }
                              { multiple of estimation sampling rate }
          change_time:real; { disturbance start time }
          change:real; { magnitude of disturbance }
          disturb_req:real;    { variables to implement a disturbance in }
          disturb_fines:real;  { size distribution }
          disturb_grind:real;

   { estimator variables }
          dead_time:integer; { process dead time as an }
                             { integer multiple of sampling time SAMPLE }
          sy:array[0..3] of real; { store past values of y }
          su:array[1..2,0..20] of real; { store past values of u }
          sv:array[0..20] of real; { store past values of v }
          Pest:real; { model estimated ouput }
          error:real; { error between Pest and measured value }
          P_mat:mat;  { covariance matrix }
          par:vec;    { vector of parameters }
          reg:vec;    { regression vector }
          forgetting_factor:real;
          dz:array[0..1] of real; { realative deadzone dynamic variable }
          deadband_time_constant:real;
          beta:real;       { deadband width scaling constant }
          feed_inc:real;  { feed increment when testing estimator only }
          est_stop:real;  { estimation stop time }
          excit_mag_feed, { magnitude of excitation input }
          excit_mag_water:real; { magnitude of plant excitation input }
          est_only:boolean;     { perform estimation without any }
```

```
                                { optimization plant moves }
        covar_reset:boolean;  { perform covariance resetting }
        diag,off_diag:real;   { diagonal and off diagonal elements }
    { noise variables }
        n:longreal; { white Gaussian noise }
        nb:array[1..10] of real; { bandlimited Gaussian noise }

    { data storage for plots }
    $include 'plot_var.p'$
{*****************************************************************************
procedure read_init_state(filename:strname);
  var path:strpath;
      initial_states:text;
      file_no:integer;
      i:integer;
  begin
    path:='/users/bleloch/sim_dir/init_state_dir/';
    strappend(path,filename);
    path:=strtrim(path);
    reset(initial_states,path);
        readln(initial_states,file_no);
        writeln('File containing initial conditions: ',filename);
        readln(initial_states,NSTATE);
        For i:=0 to NSTATE-1 do begin
            readln(initial_states,x[i]);
        end;
    close(initial_states);
  end;
{-----------------------------------------------------------------------------
procedure read_const(filename:strname);
  var path:strpath;
      file_no:integer;
      const_file:text;
  begin
    path:='/users/bleloch/sim_dir/constants_dir/';
    strappend(path,filename);
    path:=strtrim(path); {remove any blanks from end of string}
    writeln('File containing plant constants: ',filename);
    reset(const_file,path);
                readln(const_file,A);
                readln(const_file,s3);
                readln(const_file,s4);
                readln(const_file,D[3,1]);
                readln(const_file,D[3,2]);
                readln(const_file,D[4,1]);
                readln(const_file,D[4,2]);
                readln(const_file,D[4,3]);
                readln(const_file,Fpo);
                readln(const_file,max_coarseness);
                readln(const_file,min_coarseness);
                readln(const_file,minimum_load);
                readln(const_file,excess_wear);
                readln(const_file,reduced_wear);
                readln(const_file,water_critical);
                readln(const_file,Pm);
                readln(const_file,Po);
    close(const_file);
  end;
{-----------------------------------------------------------------------------
procedure read_setup;
var file_no:string[3];
    path:strpath;
    filename:strname;
    setup:text;
  begin
    writeln(' Autogenous Grinding Circuit Simulation');
```

```
        write(' Setup file number? ');read(file_no);writeln;
        path:='/users/bleloch/sim_dir/setup_dir/setup_';
        strappend(path,file_no);
        path:=strrtrim(path);
        reset(setup,path);
                readln(setup,file_no);
                readln(setup,TSTART);writeln('Start time ',TSTART:3:2);
                readln(setup,TSTOP);writeln('Stop time ',TSTOP:3:2);
                readln(setup,LOG);writeln('Logging interval ',LOG:2:2);
                readln(setup,NSTP);
                readln(setup,METHOD);
                readln(setup,filename);read_init_state(filename);
                readln(setup,filename);read_const(filename);
                readln(setup,pump_speed);
                readln(setup,mill_water);
                readln(setup,sump_water);
                readln(setup,magnitude);
                readln(setup,step_size1);
                readln(setup,step_size2);
                readln(setup,grad1_max);
                readln(setup,grad1_min);
                readln(setup,opt_start);
                writeln('Optimization start time=',opt_start:3:2);
                readln(setup,SAMPLE);
                writeln('Estimator sampling time',SAMPLE:2:2);
                readln(setup,opt_update);
                writeln('Optimization update time=',opt_update*SAMPLE:2:2);
                readln(setup,dead_time);
                readln(setup,change_time);
                writeln('Disturbance start time ',change_time:3:2);
                readln(setup,change);
                writeln('Magnitude of change ',change:2:2);
                readln(setup,forgetting_factor);
                readln(setup,deadband_time_constant);
                readln(setup beta);
                readln(setup,est_stop);
                writeln('Estimation stop time ',est_stop:3:2);
                readln(setup,excit_mag_feed);
                write('Magnitude of feed_rate excitation signal ');
                writeln(excit_mag_feed:2:2);
                readln(setup,excit_mag_water);
                write('Magnitude of mill_water excitation ');
                writeln(excit_mag_water:2:2);
                readln(setup,est_only);
                writeln('Estimation only  ',est_only);
                readln(setup,covar_reset);
                writeln('Covariance resetting on: ',covar_reset);
                readln(setup,diag);
                readln(setup,off_diag);
                readln(setup,debug_m);
                readln(setup,debug_e);
                readln(setup,debug_c);
        close(setup);
        run_number:=file_no;
    end;
{=============================================================================
function log_time:boolean;
begin
  if round(T*10) mod round(LOG*10)=0 then log_time:=true
  else log_time:=false;
end;
{=============================================================================
function sample_time:boolean;
begin
   if round(T*10) mod round(SAMPLE*10)=0 then sample_time:=true
   else sample_time:=false;
```

```
end;
{=============================================================================}
function opt_time:boolean;
begin
   if round(T*10) mod (opt_update*round(SAMPLE*10))=0 then opt_time:=true
   else opt_time:=false;
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;
{-----------------------------------------------------------------------------}
$include 'model.p'$  { file containing dynamic grinding circuit model }
{-----------------------------------------------------------------------------}
$include 'output_calc.p'$ { file containing output calculations }
{-----------------------------------------------------------------------------}
function srand48:longreal;external;
function drand48:longreal;external;
{-----------------------------------------------------------------------------}
function noise(id:integer;cutoff:real):real;
const sigma=0.05;
var i:integer;
    tor:real;
    accum:longreal;
begin
   { calculate a Gaussian distribution from a normal distribution }
   accum:=0;
   tor:=1/(2*pi*cutoff);
   for i:=1 to 12 do accum:=accum+(2*drand48-1);
   n:=accum/12;
   nb[id]:=nb[id]*exp(-DELT/tor)+sigma*n*sqrt(1-exp(-2*DELT/tor));
   noise:=nb[id];
end;
{-----------------------------------------------------------------------------}
procedure dump_model;
var i:integer;
begin
   writeln('Time ',T:3:1);
   writeln('Model Variables');
   writeln('Inputs ');
   writeln('solids feedrate ',feed_rate:2:2);
   writeln('mill water ',mill_water:3:2);
   writeln;
   writeln('Outputs ');
   writeln('Power draft ',power:3:2);
   writeln('Inside mill');
   writeln('fractional filling of mill ',fractional_filling:2:2);
   writeln('fraction of grinding media ',J:2:2);
   writeln('fractional filling of interstices ',U:2:2);
   writeln('States');
   for i:=0 to NSTATE-1 do
      writeln('x[',i,']=',x[i]:3:2);
   writeln;
end;
{-----------------------------------------------------------------------------}
procedure regression_vector(y,u1,u2,v1:real;dead_time:integer);
var i,j:integer;
begin
   sy[3]:=sy[2];
   sy[2]:=sy[1];
   sy[1]:=sy[0];
```

```
    sy[0]:=y;

    for i:=1 to 2 do
       for j:=0 to 19 do su[i,20-j]:=su[i,20-j-1];
    su[1,0]:=u1;
    su[2,0]:=u2;

    for j:= 0 to 19 do sv[20-j]:=sv[20-j-1];
    sv[0]:=v1;

    reg[1]:=sy[1];
    reg[2]:=sy[2];

    reg[3]:=su[1,dead_time];
    reg[4]:=su[1,dead_time+1];

    reg[5]:=su[2,dead_time];
    reg[6]:=su[2,dead_ti  +1];

    reg[7]:=1;

    for i:= 8 to 10 do reg[i]:=0;

end;
{-------------------------------------------------------------------------------
function dead_band:boolean;
const cdz0=0.0001;
      cdz1=0.0001;
      cdz2=0.0001;
      cdz3=0.0001;
      cdz4=0.0001;
begin
   if T>SAMPLE then dz[1]:=dz[0] else dz[1]:=0;
   dz[0]:=deadband_time_constant*dz[1]
          +cdz0+cdz1*abs(su[1,0])+c 'z2*abs(su[2,0])
          +cdz3*abs(sy[0]);
   error:=sy[0]-Pest;
   if abs(beta*dz[0])>abs(error) then dead_band:=true
   else dead_band:=false;
end;
{-------------------------------------------------------------------------------
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    function multiply_vec:real;
    var i:integer;
        dummy:real;
    begin
       dummy:=0;
       for i:= 1 to no_par do
          dummy:=dummy+reg[i]*par[i];
       multiply_vec:=dummy;
    end;
{-------------------------------------------------------------------------------
procedure dump_controller;
var i:integer;
begin
         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('              mill Power ',power:3:3);
         writeln('              feed_rate    ',feed_rate:3:2);
         writeln('              mill_water   ',mill_water:3:2);
```

```
          writeln('                product      ',y1:3:2);
          writeln('                Gradients ',gradient1:2:3,gradient2:2:3);
          writeln('                Estimation error ',error:2:4);
end;
{-------------------------------------------------------------------------------
procedure estimator;
var denom:real;
    K,L:vec;
    i,j:integer;
    {+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
    function norm:real;
    var M:vec;
        i,j:integer;
        dummy:real;
    begin
       for i:= 1 to no_par do M[i]:=0;
       for j:= 1 to no_par do
          for i:=1 to no_par do
             M[j]:=M[j]+reg[i]*P_mat[i,j];
       dummy:=0;
       for i:= 1 to no_par do dummy:=dummy+M[i]*reg[i];
       norm:=dummy;
    end;
    {+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
begin
   Pest:=multiply_vec;
   error:=sy[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 reset_P(diag_value,off_diag_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,j]:=diag_value
          else P_mat[i,j]:=off_diag_value
end;

{-------------------------------------------------------------------------------
procedure level_controller(var speed:real;set_point:real;sump_level:real);
   const P=20;  { gain of proportional controller }
   var error:real;
begin
   error:=set_point-sump_level;
   speed:=pump_speed-P*error;
   speed:=limit(speed,800,5);
end;
```

```
{-------------------------------------------------------------------------
procedure opt_controller(var feed_rate,mill_water:real);
var
    i,j:integer; { array indexes }
begin
   x_u[1]:=x_u[1]+step_size1*gradient1;
   x_u[2]:=x_u[2]+step_size2*gradient2;
   x_u[1]:=limit(x_u[1],100,0);
   x_u[2]:=limit(x_u[2],100,0);
   feed_rate:=x_u[1];
   mill_water:=x_u[2];
end;
{-------------------------------------------------------------------------
PROCEDURE dynamics;
Begin
   { level controller }
   level_controller(pump_speed,0.5,sump_level);

   { calculate the plant outputs from the states }
   output_calc;

   regression_vector(Power,feed_rate,mill_water,0,dead_time);

   { call the parameter estimation procedure }
   if T<est_stop then begin
                        if not dead_band then estimator
                        else Fest:=multiply_vec;
                      end;

   { write important variables to screen to monitor simulation }
   if opt_time then if debug_m then dump_model;
   if opt_time then if debug_c then dump_controller;


   { calculate the gradients }
   if opt_time then begin
                      gradient1:=(par[3]+par[4])/(1-par[1]-par[2]);
                      gradient1:=limit(gradient1,grad1_max,grad1_min);
                      gradient2:=(par[5]+par[6])/(1-par[1]-par[2]);
                      gradient2:=limit(gradient2,grad1_max,grad1_min);
                    end;

   { call the optimization procedure }
   if not est_only then
      if T>opt_start then
         if opt_time then
            begin
               opt_controller(feed_rate,mill_water);

               { reset covariance matrix }
               if covar_reset then reset_P(diag,off_diag);
            end;

   { estimation only }
   if est_only then if opt_time then feed_rate:=feed_rate+feed_inc;

   { add an excitation signal }
   feed_rate:=feed_rate+excit_mag_feed*noise(1,1/3*SAMPLE);
   mill_water:=mill_water+excit_mag_water*noise(2,1/3*SAMPLE);

   { simulate a disturbance }
   { change the size ditribution of the fresh rock solids }
   { going into the mill }
   if T>change_time then begin
       disturb_req:=1;
       disturb_grind:=(1+change);
```

```
        { the change in amount of fines must not change the total }
        { mass of fresh rock flowing into the mill, this is done  }
        { by calculating the change in fines using the following formula }
        disturb_fines:=(1/frac_fines)*(1-frac_req_grind-frac_grind*(1+change)
       end
    else begin
           disturb_req:=1.0;
           disturb_fines:=1.0;
           disturb_grind:=1.0;
         end;

End;     { procedure dynamics }
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
PROCEDURE Euler;
VAR
    i: INTEGER;
Begin
  model;
  FOR i:=0 to NSTATE do
    X[i]:= X[i] + DELT*dX[i];
  T:= T + DELT;
End;     { procedure Euler }
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
PROCEDURE Runge_Kutta;
VAR
    i: INTEGER;
     Xstart, K1, K2, K3: vector;
Begin
  FOR i:=0 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:=0 to NSTATE do begin
    K2[i]:= dX[i]*DELT;
    X[i]:= Xstart[i] + K2[i]/2;
  end;
  model;
  FOR i:= 0 to NSTATE do begin
    K3[i]:= dX[i]*DELT;
    X[i]:= Xstart[i] + K3[i];
  end;
  T:= T + DELT/2;
  model;
  FOR i:=0 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
  CASE METHOD of
    'E': Euler;
    'R': Runge_Kutta;
  End;  { case }
  IF (T >= TSTOP + DELT/2) then STOP:= TRUE;
  for i:=0 to NSTATE-1 do if x[i]<0 then x[i]:=0;
End;     { procedure integrator }
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
{-------------------------------------------------------------------------------
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,feed_rate,T);
      log_points(2,step_no,mill_water,T);
      log_points(3,step_no,power,T);
      if T>opt_start then log_points(4,step_no,Pest,T)
         else log_points(4,step_no,0,T);
      log_points(5,step_no,fractional_filling,T);
      log_points(6,step_no,J,T);
      log_points(7,step_no,fc,T);
      log_points(8,step_no,U,T);
      log_points(9,step_no,gradient1,T);
      log_points(10,step_no,gradient2,T);
      log_points(11,step_no,y1,T);
      for i:=0 to NSTATE-1 do log_points(i+12,step_no,x[i],T);
      step_no:=step_no+1;
  end;
{-------------------------------------------------------------------------------
procedure write_data;
   var
       path,path1:strpath;
       i,dummy:integer;

  {++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
   procedure write_file(sub_dir:strpath;name:strname;
                        curve_no,no_of_points:integer);
        var directory:strpath;
              i:integer;
        begin
          directory:=path;
          strappend(directory,sub_dir);
          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 begin
                write(file_var,datamatrix_x[curve_no,i]);
                writeln(file_var,datamatrix_y[curve_no,i]);
          end;
          close(file_var,'SAVE');
        end;
  {++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

  begin
    path:='/users/bleloch/sim_dir/plot_dir_';
    strappend(path,run_number);path:=strtrim(path);
    strappend(path,'/selection_dir');
    { file inputs }
        write_file('/inputs_dir/','feed_rate',1,step_no);
        write_file('/inputs_dir/','mill_water',2,step_no);
    { file states }
        for i:=0 to NSTATE-1 do begin
                path1:='x';
                strwrite(path1,2,dummy,i:0);
                write_file('/states_dir/',path1,12+i,step_no);
        end;
    { file outputs }
        write_file('/outputs_dir/mill_dir/','power',3,step_no);
        write_file('/outputs_dir/mill_dir/','fractional_filling',5,step_no)
        write_file('/outputs_dir/mill_dir/','J',6,step_no);
        write_file('/outputs_dir/mill_dir/','fc',7,step_no);
        write_file('/outputs_dir/mill_dir/','s_feed',9,step_no);
```

```
          write_file('/outputs_dir/mill_dir/','s_water',10,step_no);

          write_file('/outputs_dir/mill_dir/','Jp',4,step_no);
          write_file('/outputs_dir/mill_dir/','U',8,step_no);

          write_file('/outputs_dir/cyclone_dir/','product',11,step_no);
      end;
{-------------------------------------------------------------------------------
procedure init_param;
var i,j:integer;
begin
  for i:=1 to 10 do
     for j:=1 to 10 do
        if i=j then P_mat[i,j]:=10000
        else P_mat[i,j]:=500;
  { initialize parameters }
  for i:=1 to 2 do par[i]:=0;
  for i:=3 to 10 do par[i]:=0;
  { initialize storage variables }
  for i:=0 to 20 do
        begin
           for j:=1 to 2 do su[j,i]:=0;
           sv[i]:=0;
        end;
  for i:= 0 to 2 do sy[i]:=0;
end;
{-------------------------------------------------------------------------------
PROCEDURE initialise;
   var i,j:integer;
Begin
  T:= 0;
  step_no:= 1;
  STOP:=FALSE;
  read_setup; { read all the simulation setup variables and constants }
  DELT:=LOG/NSTP; { simulation time step size }
  Vo:=pi*R*R*L; { mill volume }
  feed_rate:=magnitude;
  x_u[1]:=feed_rate;
  x_u[2]:=mill_water;
  n:=srand48;
End;  { procedure  initialise }
{===============================================================================
Begin  { main program }
  initialise;
  init_param; { initial model parameters }
  log_data;   { write initial data to global variables }
  writeln;writeln('Running Simulation');writeln;
  REPEAT
    integrator; { calculate states }
    if sample_time then dynamics; { plant output calculation }
                                  { and controller calculation }
    if log_time then log_data;    { store data in global arrays }
  UNTIL STOP;
  write_data; { write data to disk }
End.   { main program }
{===============================================================================
```

APPENDIX H. FILE CONTAINING AUTOGENOUS MILL INITIAL STATES

FILE INIT_1 (contains initial states)

```
:                    #file number
8                    #number of states
0.0                    #x[0]
0.0                    #x[1]
0.0                    #x[2]
10.0                   #x[3]
0.0                    #x[4]
10.0                   #x[5]
0.0                    #x[6]
0.0                    #x[7]
```

APPENDIX I. FILE CONTAINING GRINDING CIRCUIT CONSTANTS

```
FILE CONST_6 ( contains plant constants )

0.6      ---- A       constant for attrition breakage rate
0.05     ---- S3      breakage rate for abrasion
0.15     ---- S4      breakage rate for chipping
0.60     ---- D3,1    proportion that breaks from x3 into x1
0.40     ---- D3,2    proportion that breaks from x3 into x2
0.10     ---- D4,1    proportion that breaks from x4 into x1
0.10     ---- D4,2    proportion that breaks from x4 into x2
0.80     ---- D4,3    proportion that breaks from x4 into x3
0.05     ---- Fpo     discharge function
6.00     ---- max_coarseness
0.00     ---- min_coarseness
0.20     ---- minimum_load
0.0      ---- excess_wear
0.0      ---- reduced_wear
100.0    ---- water_critical
0.00     ---- Pm      mechanical power losses
1.0      ---- Po      constant for power draft calculation
```

APPENDIX J. FILE CONTAINING AUTOGENOUS MILL SIMULATION SETUP AND CONTROLLER PARAMETERS

SETUP FILE NUMBER 6

```
6               file number
0.0             TSTART  simulation start time
800.0           TSTOP   simulation stop time
4.0             LOG     logging interval
8               NSTP    number of integration steps
R               METHOD  numerical integration method [R,E]
init_1          #file containing initial states
const_6         #file containing constants
50              initial pump speed
5.0             initial mill_water
5               sump water
4.5             initial feed rate
0.1             optimization step size
0.1             optimization step size
4.0             maximum gradient
-4.0            minimum gradient
50.00           optimization start time
0.5             estimator sampling time
50              optimization update time
1               process dead time
400.0           disturbance introduced at this time
0.20            magnitude of disturbance
1.00            forgetting factor
0.2             deadband time constant
0.01            deadband width beta
800.0           estimation stop time
1.0             magnitude of excitation signal for feed_rate
1.0             magnitude of excitation signal for mill_water
false           estimation only
true            covariance resetting on
1000            diagonal element of covariance matrix
950             off diagonal element of covariance matrix
true            #debug model
true            #debug estimator
true            #debug controller
```

SETUP FILE NUMBER 7

```
7               file number
0.0             TSTART  simulation start time
800.0           TSTOP   simulation stop time
4.0             LOG     logging interval
8               NSTP    number of integration steps
R               METHOD  numerical integration method [R,E]
init_1          #file containing initial states
const_6         #file containing constants
50              initial pump speed
5.0             initial mill_water
5               sump water
4.5             initial feed rate
0.1             optimization step size 1
0.1             optimization step size 2
4.0             maximum gradient
-4.0            minimum gradient
50.00           optimization start time
0.5             estimator sampling time
50              optimization update time
1               process dead time
400.0           disturbance introduced at this time
-0.1            magnitude of disturbance
1.00            forgetting factor
0.2             deadband time constant
0.01            deadband width beta
800.0           estimation stop time
1.0             magnitude of excitation signal for feed_rate
1.0             magnitude of excitation signal for mill_water
false           estimation only
true            covariance resetting on
1000            diagonal element of covariance matrix
950             off diagonal element of covariance matrix
true            #debug model
true            #debug estimator
true            #debug controller
```

APPENDIX K. PASCAL INCLUDE FILE MODEL.P

```
{=======================================================================}

{=======================================================================}
  function Rose_Sullivan (x: REAL): REAL;
    begin
      Rose_Sullivan:= x + 1.5*sqr(x) - 6.7*sqr(x)*x + 4.25*sqr(sqr(x));
    end;   { function Rose_Sullivan }
{=======================================================================}
function pump(pump_speed:real):real;
   const kpump=1;
   begin
      pump:=kpump*pump_speed;
   end;
{=======================================================================}
function cut_point: REAL;
  var
    numerator, denomenator: REAL;
  begin
    if (S-pulp_density)<=0 then pulp_density:=1.3;
    numerator:= 14.8*exp(0.46*ln(Dc))*exp(0.6*ln(Di))*exp(1.21*ln(Dv))
                    *exp(0.063*V);
    denomenator:= exp(0.71*ln(Du))*exp(0.38*ln(h))*exp(0.45*ln(flow))
                    *exp(0.5*ln(S-pulp_density));
    cut_point:= 0.000001*numerator/denomenator;
  end;  { function cut_point }
{=======================================================================}
function distribution: REAL;
  var
  numerator, denomenator: REAL;
  Begin
    Ph:= P/(pulp_density * g);
    if Ph=0 then begin
                    writeln('Ph=0');
                    Ph:=0.1;
                    end;
    numerator:= 1.9*exp(3.31*ln(Du/Dv))*exp(0.54*ln(h))
                   *exp(0.36*ln(sqr(Du)+sqr(Dv)))*exp(0.0054*V);
    denomenator:= exp(0.24*ln(Ph))*exp(1.11*ln(Di));
    distribution:= numerator/denomenator;
  End;  { function distribution }
{=======================================================================}
function pressure: REAL;
  var
    numerator, denomenator: REAL;
  begin
    numerator:= 1.88*exp(1.78*ln(flow))*exp(0.0055*V);
    denomenator:= exp(0.37*ln(Dc))*exp(0.94*ln(Di))*exp(0.28*ln(h))
                    *exp(0.87*ln(sqr(Du)+sqr(Dv)));
    pressure:= numerator/denomenator;
  end;   { function pressure }
{=======================================================================}
function flow_rate: REAL;
  var
    numerator, denomenator: REAL;
  begin
    numerator:= 0.53*exp(0.56*ln(P)) * exp(0.21*ln(Dc))
                *exp(0.53*ln(Di)) * exp(0.16*ln(h))
                *exp(0.49*ln(sqr(Du) + sqr(Dv)));
    denomenator:= exp(0.0031*V);
    flow_rate:= numerator/denomenator;
  end;
{=======================================================================}
function lognormal (y,mean,SD: REAL): REAL;
  const
    b1 = 0.31938153;
    b2 = -0.356563782;
```

```
    b3 = 1.781477937;
    b4 = -1.821255978;
    b5 = 1.330274429;
    pi = 3.141592654;
    r  = 0.2316419;
  VAR
    t, x, fx: REAL;
  Begin
    if y=0 then
       begin
          writeln('y=0');
          y:=0.0000001;
       end;
    x:= (ln(y*(1+sqr(SD/mean))/mean))/sqrt(ln(1+sqr(SD/mean)));
    fx:= (1/sqrt(2*pi))*exp(-sqr(x)/2);
    t:= 1/(1 + r*x);
    lognormal:= 1 - fx*(b1*t + b2*sqr(t) + b3*sqr(t)*t
                        + b4*sqr(sqr(t)) + b5*sqr(sqr(t))*t);
  End; { function lognormal }
{=============================================================================}
procedure model;
   var
    i:integer;
    S4e, S3e: REAL;
Begin
  flow:=pump(pump_speed);

{ feed size distribution }
  T1:=frac_req_grind*feed_rate*disturb_req;
  T2:=frac_fines*feed_rate*disturb_fines;
  T4:=frac_grind*feed_rate*disturb_grind;

{ feasible states }
   for i:=0 to NSTATE-1 do
      if x[i]<0 then x[i]:=0;

{ sump output calculation
    if( x[5]+x[6]+x[7]) =0 then begin
                                  z0:=0;z1:=0;z2:=0
                                end
    else begin
           z0:=flow*x[5]/(x[6]+x[7]+x[5]);
           z1:=flow*x[6]/(x[6]+x[7]+x[5]);
           z2:=flow*x[7]/(x[5]+x[6]+x[7]);
           pulp_density:=water_density*x[5]/(x[5]+x[6]+x[7])
                         +ore_density*(x[6]+x[7])/(x[5]+x[6]+x[7]);
       end;
    sump_level:=(x[5]+x[6]+x[7])/Vsump;

{ hydrocyclone }
    F1:=z0*delt;
    F3:=z1*delt;
    F2:=z2*delt;

{ calculate volumetric % solids in feed }
if pulp_density <> 0 then
  solids:= S*(pulp_density - 1)/(pulp_density*(S - 1))
  else solids:=0;
  V:= pulp_density*solids/S;

{ calculate cut point of cyclone }
  d50 := cut_point;

{ calculate pressure drop across cyclone }
  P:= pressure;
```

```
{ calculate fraction of feed bypassing classification }
  SC:= distribution;
  Rv:= SC/(SC + 1);

{ calculate sharpness of separation }
  m:= 1.94*exp(-1.58*Rv)*exp(0.15*ln(sqr(DC*h/flow));

{ calculate pobability of particle satisfying the }
{ required grind reporting to the underflow }
  Ec:= 1 - exp(-0.6931*exp(m*ln(cut/d50)));
  {if debug_m then writeln('Ec ',Ec,' d50 ',d50,' cut ',cut);}

{ calculate volume of required grind undergoing classification
  that reports to the overflow and underflow }
  o3:= (1 - Rv)*(1 - Ec)*F3;
  U3:= (1 - Rv)*Ec*F3;

{ calculate distribution of fines undergoing classification }
  O2:= 0;
  U2:= 0;
  FOR i:=1 to intervals do begin
    IF (i=1)
      then probability:= 1
      else
       begin
        y:= (grate*(i-1)/intervals - mean)/SD;
        probability:= lognormal(y,mean,SD);
       end;
    y:= (grate*i/intervals - mean)/SD;
    probability:= lognormal(y,mean,SD) - probability;
    Ec:= -0.6931*exp(m*ln(grate*i/intervals/d50));
    IF (Ec < -20)
      then Ec:= 1
      else Ec:= 1 - exp(Ec);
    U2:= U2 + Ec*probability*(1 - Rv)*F2;
    O2:= O2 + (1 - Ec)*probability*(1 - Rv)*F2;
  end;  { for }

{ calculate overflow water }
  o1:=(1-Rv)*F1;

{ calculate components of underflow }
  U1:= Rv*F1;
  U2:= U2 + Rv*F2;
  U3:= U3 + Rv*F3;

{ mass balance }
  o1b:=o1;
  u1b:=u1;
if (u2+o2)=0 then begin
                    o2b:=0;
                    u2b:=0
                  end
else begin
       o2b:=(o2/(u2+o2))*F2;
       u2b:=(u2/(u2+o2))*F2;
    end;
if (u3+o3)=0 then begin
                    o3b:=0;
                    u3b:=0;
                  end
else begin
       o3b:=(o3/(u3+o3))*F3;
       u3b:=(u3/(u3+o3))*F3;
    end;
```

```
{ calculate total volume flowing into the mill }
underflow:=(u2b+u3b)/delt;
overflow:=(o2b+o3b)/delt;
if debug_m and opt_time then begin
   writeln('overflow ',overflow:3:2);
   writeln('underflow ',underflow:3:2);
end;

{ evaluate present status of internal variables }
     fc:= (X[1] + X[2])/(Vo*(1 - fines_voidage));
     J := (X[3] + X[4])/(Vo*(1 - media_voidage));
     IF (fc <=0.159)
      then S2:= A*Rose_Sullivan(J)
      else S2:= A*(2*exp(ln(0.159/fc)*0.55) - 1)*Rose_Sullivan(J);
 IF (fc=0) then wear_slope:= 1;
  IF (J=0) then wear_slope:=0;
  IF (fc<>0) and (J<>0) then
              wear_slope:=1-fc/(J*0.4);
  IF wear_slope < 0 then wear_slope:=0;
  IF wear_slope > 1 then wear_slope:=1;
  IF (X[0] + X[1] + X[2] = 0) then discharge:= 0
  else discharge:= Fpo*sqr(x[0]+x[1]+x[2] - 0.02*Vo)
                  *sqrt(x[0]/(x[0]+x[1]+x[2]));

{ differential equations for mill model }
{ chipping process }
  S4e:= wear_slope*excess_wear*S4 + S4;
  S4e:= S4e*overload[4];
  dX[4]:= T4 - S4e*X[4];

{ abrasion process }
  S3e:= wear_slope*excess_wear*S3 + S3;
  S3e:= S3e*overload[3];
  T3:= S4e*X[4]*D[4,3];
  dX[3]:= T3 - S3e*X[3];


{ crushing process }
  IF (X[0] + X[1] + X[2] = 0) then Y2:= 0
  else Y2:= discharge*X[2]/(X[0]+X[1]+X[2]);
  S2:= S2*overload[2];
  dX[2]:= T2 +(u2b/delt)+ S3*D[3,2]*X[3]
         + S4e*D[4,2]*X[4] - S2*X[2]-Y2;

  IF (X[0] + X[1] + X[2] = 0)
   then Y1:= 0
   else Y1:= discharge*X[1]/(X[0]+X[1]+X[2]);
  dX[1]:= T1+(u3b/delt)+S2*X[2] + S3*D[3,1]*X[3]
           + S4e*D[4,1]*X[4] - Y1;

{ water balance }
  IF (X[0] + X[1] + X[2] = 0)
   then Y0:= 0
   else Y0:= discharge*X[0]/(X[0]+X[1]+X[2]);
  dX[0]:= mill_water+u'b/delt-y0;

  { check feasible states }
  for i:=0 to NSTATE-1 do if x[i]<0 then x[i]:=0;
{ sump model }
    { x5  volume of water in the sump  }

    {inputs  }
    {       sump_water    sump water addition    }
    {       pump_speed    speed of pump          }

    {intermediate veriables                      }
```

```
    {       y0 flow of water from mill into the sump }
    {       y1 flow of required grind into the sump  }
    {       y2 flow of fines into the sump }

    {outputs                                          }
    {       z0 flow of water out of the sump          }
    {       z1 flow of required grind out of the sump }
    {       z2 flow of fines out of the sump }

  if (x[5]+x[6]+x[7])<>0 then begin
                        dx[5]:=y0+sump_water-flow*(x[5]/(x[5]+x[6]+x[7]));
                        dx[6]:=y1-flow*(x[6]/(x[5]+x[6]+x[7]));
                        dx[7]:=y2-flow*(x[7]/(x[6]+x[7]+x[5]));
                      end
  else begin
       dx[5]:=y0+sump_water;
       dx[6]:=y1;
       dx[7]:=y2;
     end;
  { end sump model }
End;     { procedure model }
{=============================================================================}

{=============================================================================
```