```
# ---------------------------------------
# STOCHASTIC PROGRAMMING PROBLEM

##############################################################################
#     UNITS
#
#     state             m3
#     flow              cumec
#     demand in block   MWh
#     generation        MW
#     cost              $/MWh
#     minzone           MWh
#
##############################################################################

#-----------------------------------------------------------------------
#                       Time
#-----------------------------------------------------------------------
# Week
param weeks_in_year, default 52;
set WEEKS := 1..weeks_in_year circular;
param start_week , >=1, <=weeks_in_year;  # start week of planning year (1-52)
param end_week := prev(start_week,WEEKS);       # end week of planning year

# Year to sample from
param start_year  ; param end_year  ;
set YEARS := start_year..end_year ordered ;
set WEEKSstart := ( if start_week==1 then WEEKS else {w in WEEKS:w>=start_week} ) ;
set WEEKSend := ( if start_week==1 then {} else {w in WEEKS:w<=end_week} ) ;
set YEARSstart := ( if start_week==1 then YEARS else {y in YEARS:y<end_year} ) ;
set YEARSend := (if start_week==1 then {} else {y in YEARS:y>start_year} ) ;
set YEARS_WEEKS_SAMPLE := (YEARSstart cross WEEKSstart) union (YEARSend cross WEEKSend ) ;

# Planning year and week
param plan_year ; # Year in which planning is to start
set YEARS_PLAN := if start_week==1 then {plan_year} else {plan_year,plan_year+1} ;
set YEARS_WEEKS_PLAN :=
if start_week==1
then ({plan_year} cross WEEKSstart) union ({plan_year} cross WEEKSend)
else ({plan_year} cross WEEKSstart) union ({plan_year+1} cross WEEKSend ) ;

set YEARS_WEEKS_ALL  := YEARS_WEEKS_SAMPLE union YEARS_WEEKS_PLAN ;

# Stages
param tau;
param T := weeks_in_year ;                      # number of stages

# Current year and week at stage
param current_year := if start_week+tau-1 <= weeks_in_year
then plan_year else plan_year+1 ;
param current_week := if start_week+tau-1 <= weeks_in_year
then start_week+tau-1 else start_week+tau-1-weeks_in_year ;

# Others
param seconds_in_hour >=0, default 3600 ;

#-----------------------------------------------------------------------
#                       Island
#-----------------------------------------------------------------------
set ISLAND ;                                # ISLAND

#-----------------------------------------------------------------------
#                       Demand
#-----------------------------------------------------------------------
# Load
set NODE;               # set of demand nodes
set BLOCK;                                  # demand usage blocks in each week
set NODE_ISLAND within {NODE,ISLAND} ;    # position of demand node in island

param demand {NODE,YEARS_WEEKS_PLAN,BLOCK}; # demand (MWh)
param hrs_per_block {YEARS_WEEKS_PLAN,BLOCK} ; # number of hours in each block

# Lost load
set LL_SECTOR ;                             # sectors
set LL_SEGMENT ;                            # segments
```

```
param LL_cost{LL_SECTOR, LL_SEGMENT};           # cost in $/MWh
param LL_sec_dist{LL_SECTOR,ISLAND};            # proportion of sector in each island
param LL_seg_bound{LL_SEGMENT};                 # proportion of segment

var LostLoad {1..T,LL_SECTOR, seg in LL_SEGMENT,NODE,BLOCK} >=0, default 0;   # LL in MWh

#-----------------------------------------------------------------
#                       Generation
#-----------------------------------------------------------------
### Hydro ###
set HYDRO;                    # hydro station
set AVE_HYDRO within HYDRO;     # hydro station with historical weekly average generation
capacity
set HYDRO_DERATING within {HYDRO} ; # hydro station with outage due to maintenance
set HYDRO_NODE within {HYDRO,NODE} ;        # position of hydro station

param hydro_cap{HYDRO} >=0;         # capacity (MW)
param ave_hydro_cap {AVE_HYDRO} >=0; # historical weekly average generation capacity
param conv_fac{HYDRO};   # conversion factor for generator MW/cumec
param scaling_fac{HYDRO,YEARS_WEEKS_PLAN};   # scaling factor for conversion factor
param hydro_derating{HYDRO,YEARS_WEEKS_PLAN}, default 0 ; # derating due to outage

var hydro_disp {s in HYDRO,BLOCK} >=0; # dispatch of hydro generator in MW

### Thermal ###
set THERMAL;                        # thermal generator
set FUEL ;
set THERMAL_FUEL within {THERMAL,FUEL};
set THERMAL_NODE within {THERMAL,NODE} ;  # position of thermal generator

param therm_cap {THERMAL} >= 0;     # capacity of each thermal plant, assuming plants always
runs at capicity(MW)
param therm_heat_rate {THERMAL} >= 0;     # GJ/MWh
param fuel_cost {FUEL,YEARS_WEEKS_PLAN} >= 0;   # cost to run thermal plant in $/GJ
param therm_derating{THERMAL,YEARS_WEEKS_PLAN}, default 0 ; # derating due to outage

param therm_start_year{THERMAL} , default 0; # The first year was available. 0=always
available
param therm_start_week{THERMAL} , default 0; # The first week was available. 0=always
available
param therm_end_year{THERMAL} , default 10000; # The last year was available. 10000=always
available
param therm_end_week{THERMAL} , default 10000; # The last week was available. 10000=always
available
set THERMAL_AVAILABLE :=
      {
      m in THERMAL :
      ( (current_year = therm_start_year[m] and current_week >= therm_start_week[m])
      or (current_year > therm_start_year[m]) )
      and
      ( (current_year = therm_end_year[m] and current_week <= therm_end_week[m])
      or (current_year < therm_end_year[m] ) )
      }
;

var therm_hrs {THERMAL,b in BLOCK} >= 0, <= hrs_per_block[current_year,current_week,b],
default 0;  # number of hours each thermal station fully dispatchs for (hr)

#-----------------------------------------------------------------
#                       Transmission
#-----------------------------------------------------------------
set LINK within {NODE,NODE};  # transmission line

param link_cap{LINK} >= 0;                # line capacity in transmission network (MW)
param link_derating {LINK,YEARS_WEEKS_PLAN}, default 0 ; # derating due to outage

var link_power{(i,j) in LINK,b in BLOCK} >=0, <= link_cap[i,j];   # power in transimission
network

#-----------------------------------------------------------------
#                       Reservoir and junction
#-----------------------------------------------------------------
set RES ;               # reservoirs (LAKES)
set JUNCTION;                           # junctions
```

```
set TRIB_JUNCTION within JUNCTION;      # junctions with inflows

set RES_TRIB_JUNCTION_Waikato within {RES union TRIB_JUNCTION} ;  # reservoir and
trib_junctions in waikato river
set RES_TRIB_JUNCTION_sf within {RES union TRIB_JUNCTION};  # to scale inflows

set RES_ISLAND within {RES,ISLAND} ;                  # position of reservoirs (LAKES)
set TRIB_JUNCTION_ISLAND within {TRIB_JUNCTION,ISLAND} ;               # position of trib
junctions

param ini_state {r in RES} >= 0;          # initial state of each reservoir (m3) of the year
param cand_state {RES,0..T};  # end state at each stage
param res_cap {RES} > 0;                  # the capacity of reservoirs (m3)
param specific_energy {RES} >0 ;    # convert factor for volume in m3 to energy in MWh
param energy_sf >0, default 3600 ;        # scaling factor for volume in m3 to energy in MWh
param end_water_value {RES} >= 0;   # end water value in $/MWh

param state_lb {r in RES} >= 0, default 0 ;     # state lower bound
param pc_state_lb >= 0 , default 0 ; # penalty cost for state lower bound violation in $/MWh

param minzone{1..weeks_in_year} >= 0;     # national minzone, minimum national storage in GWh
param minzone_nat {w in 1..weeks_in_year} := minzone[w]*1000; # national minzone in MWh
param minzone_SI {w in 1..weeks_in_year} := max(minzone[w]-250,0)*1000; # SI minzone in MWh
param pc_minzone ;                  # penalty cost for minzone violation in $/MWh

param minzone_taupo{1..weeks_in_year} >= 0;     # minzone for Taupo to meet Karapiro min flow
param pc_minzone_taupo ;            # penalty cost for Taupo minzone violation in $/MWh

var state {r in RES} >= 0, <= res_cap[r];

var pv_state_lb{r in RES} >= 0;  # penalty variable for state lower bound violation

var pv_minzone_nat >= 0; # penalty variable for national minzone violation in $/MWh
var pv_minzone_SI >= 0; # penalty variable for SI minzone violation in $/MWh
var pv_minzone_taupo >= 0; # penalty variable for Taupo minzone violation in $/MWh

#-----------------------------------------------------------------------
#                       Inflow
#-----------------------------------------------------------------------
### Inflows ###
# Historical inflows
param inflows_his_posneg {RES,YEARS_WEEKS_ALL}, default 0; # inflow into reservoirs (m3/s)
param inflows_trib_his_posneg {TRIB_JUNCTION,YEARS_WEEKS_ALL}, default 0; # tributary inflows

param inflows_his {r in RES,(y,w) in YEARS_WEEKS_ALL} := max(0,inflows_his_posneg[r,y,w]);
param inflows_trib_his {tj in TRIB_JUNCTION,(y,w) in YEARS_WEEKS_ALL} := max
(0,inflows_trib_his_posneg[tj,y,w]);

# Inflow at current stage, with current outcome
param inflow {1..T, RES};
param inflow_trib {1..T, TRIB_JUNCTION};

# Scale historical inflows
param inflows_pow, default 0 ;

param inflows_sf { y in ( setof{(yy,ww) in YEARS_WEEKS_ALL:ww==start_week} yy ) } :=
(sum {r in RES_TRIB_JUNCTION_sf inter RES} inflows_his[r,plan_year,start_week] +
sum {r in RES_TRIB_JUNCTION_sf inter TRIB_JUNCTION} inflows_trib_his[r,plan_year,start_week])
/
(sum {r in RES_TRIB_JUNCTION_sf inter RES} inflows_his[r,y,start_week] +
sum {r in RES_TRIB_JUNCTION_sf inter TRIB_JUNCTION} inflows_trib_his[r,y,start_week])
;

param inflows {r in RES,(y,w) in YEARS_WEEKS_ALL} :=
if w>=start_week then
inflows_sf [y] ^ (inflows_pow^(w - start_week)) * inflows_his[r,y,w]
else
inflows_sf [y-1] ^ (inflows_pow^(w + weeks_in_year - start_week)) * inflows_his[r,y,w]
;

param inflows_trib {r in TRIB_JUNCTION, (y,w) in YEARS_WEEKS_ALL} :=
if w>=start_week then
inflows_sf [y] ^ (inflows_pow^(w - start_week)) * inflows_trib_his[r,y,w]
else
inflows_sf [y-1] ^ (inflows_pow^(w + weeks_in_year - start_week)) * inflows_trib_his[r,y,w]
```

```
;

### Markov state, wetdry ###
param outcomes, default 1;          # number of Markov state
param transmat {1..T,1..outcomes,1..outcomes}, default 1 ;   # transition matrix
  check {t in 1..T,q1 in 1..outcomes:t<T} : 0.9998 <= sum {q2 in 1..outcomes} transmat
[t,q1,q2] <= 1.0002;
param wetdry{1..T};           # a particular Markov wetdry
param wetdry_his{YEARS_WEEKS_ALL}, default 1;
param startingwetdry := wetdry_his[plan_year,start_week];   # starting Markov wetdry in
plan_year

#---------------------------------------------------------------------
#                       Flow
#---------------------------------------------------------------------
set NODES := RES union {'SEA'} union HYDRO union JUNCTION;        # nodes in water network
set ARC within {NODES,NODES};            # arcs in water network

set ARC_LB within ARC ;
set ARC_UB within ARC ;
param flow_lb {ARC_LB} default 0, >=0;
param flow_ub {(i,j) in ARC_UB} default 1000000, >=0;
param pc_flow_lb >=0, default 500 ;
param pc_flow_ub >=0, default 50 ;

param pc_water_balance >= 0;        # cost of violating a water balance constraint in $/MWh

set ARC_SPILL within ARC ;
param spill_fac {ARC_SPILL} >= 0 ;
param pc_spill >=0, default 50 ;

var flow_on_arc {ARC,BLOCK} >= 0;   # flows on arcs, in cumecs

var pv_water_shortage{NODES,BLOCK} >= 0, default 0;
var pv_water_excess{NODES,BLOCK} >= 0, default 0;

var pv_flow_lb {ARC_LB,BLOCK} >= 0, default 0;
var pv_flow_ub {ARC_UB,BLOCK} >= 0, default 0;

#---------------------------------------------------------------------
#                       Cut
#---------------------------------------------------------------------
set RES_cut ;  # reservior with cuts for end value in each stage

param nCut {1..T,1..outcomes} >= 0 ;           # number of cuts in stage t
param alpha {t in 1..T,j in 1..outcomes,1..nCut[t,j]};            # cut coefficients
param beta  {RES_cut,t in 1..T,j in 1..outcomes,1..nCut[t,j]};

var theta {1..T}    >= 0;           # future cost

#---------------------------------------------------------------------
#                       Cost
#---------------------------------------------------------------------
# Record cost
var fuel_cost_total{1..T};
var LL_cost_total{1..T}  ;
var pc_water_balance_total{1..T} ;
var pc_minzone_total{1..T};
var pc_minzone_taupo_total{1..T};
var pc_state_lb_total{1..T};
var pc_flow_lb_total{1..T};
var pc_flow_ub_total{1..T};
var pc_spill_total{1..T};
var future_cost_total{1..T} ;

#---------------------------------------------------------------------
#                       Cut generation and simulation
#---------------------------------------------------------------------
param back ; # backward pass indicator

param sim_num;
param plan_year_only ;  # plan year inflows only in simulation if 1
param historical_only ; # historical inflows only in simulation if 1
param sampled_only ;    # sampled inflows only in simulation if 1
```

```
param water_value_cuts := 0; # water value cuts indicator


#---------------------------------------------------------------------
#                       Constraints for energy
#---------------------------------------------------------------------
subject to hydro_disp_defn {s in HYDRO diff AVE_HYDRO, b in BLOCK}:
      hydro_disp[s,b] = conv_fac[s] * scaling_fac[s,current_year,current_week] * sum{(i,s) in
ARC} flow_on_arc[i,s,b]
;

subject to hydro_disp_UB {s in HYDRO, b in BLOCK}:
      hydro_disp[s,b] <= hydro_cap[s] - hydro_derating[s,current_year,current_week]
;

subject to ave_hydro_disp_defn {s in AVE_HYDRO}:
      sum {b in BLOCK} hydro_disp[s,b]*hrs_per_block[current_year,current_week,b]
      <=
      ave_hydro_cap[s] * (sum {b in BLOCK} hrs_per_block[current_year,current_week,b])
;

subject to line_derating {(i,j) in LINK, b in BLOCK}:
      link_power[i,j,b] <= link_cap[i,j] - link_derating[i,j,current_year,current_week]
;

# Meet demand in each node and block
subject to MeetDemand {(i,isl) in NODE_ISLAND,b in BLOCK}:
      hrs_per_block[current_year,current_week,b]*
      (
      sum {(s,i) in HYDRO_NODE} hydro_disp[s,b]
      + sum{(j,i) in LINK}link_power[j,i,b] - sum{(i,j) in LINK}link_power[i,j,b]
      )
      +
      sum {(m,i) in THERMAL_NODE:m in THERMAL_AVAILABLE}
      therm_hrs[m,b]*(therm_cap[m]-therm_derating[m,current_year,current_week])
      >=
      demand[i,current_year,current_week,b]
      -
      sum{sec in LL_SECTOR,seg in LL_SEGMENT}LostLoad[tau,sec,seg,i,b]
;

# Lost load, i.e., energy saving
subject to LostLoadBound {(i,isl) in NODE_ISLAND,sec in LL_SECTOR,seg in LL_SEGMENT,b in
BLOCK}:
      LostLoad[tau,sec,seg,i,b] <= demand[i,current_year,current_week,b] * LL_sec_dist
[sec,isl]*LL_seg_bound[seg]
;

#---------------------------------------------------------------------
#                       Constraints for water
#---------------------------------------------------------------------
# Reservoir state = initial state + inflows - outflows
# New ABP code: units are cubic netres
# flow_on_arc is cumecs
# Correction is cumecs
subject to StateEqn {r in RES}:
      state[r] =
      cand_state[r,tau-1]
      +  sum{b in BLOCK}
      ( hrs_per_block[current_year,current_week,b]*seconds_in_hour*
            (
       - sum {(r,v) in ARC} flow_on_arc[r,v,b]
       + sum {(v,r) in ARC} flow_on_arc[v,r,b]
       + inflow[tau, r]
       +  pv_water_shortage[r,b] - pv_water_excess[r,b]
            )
      )
;

subject to MeetStateLB {r in RES}:
      state[r] + pv_state_lb[r] >= state_lb[r]
;

# For junctions/stations, inflows = outflows
# New ABP code  in cumecs
```

```
# Correction is in cumecs
subject to FlowBalance {i in HYDRO union JUNCTION, b in BLOCK}:
      sum{(i,k) in ARC} flow_on_arc[i,k,b]
      =
      sum{(h,i) in ARC} flow_on_arc[h,i,b] + (if (i in TRIB_JUNCTION) then inflow_trib[tau, i]
else 0)
      + pv_water_shortage[i,b] - pv_water_excess[i,b] ;
;

subject to MeetFlowLB {(i,j) in ARC_LB,b in BLOCK} :
flow_on_arc[i,j,b] + pv_flow_lb[i,j,b] >= flow_lb[i,j];

subject to MeetFlowUB {(i,j) in ARC_UB,b in BLOCK} :
flow_on_arc[i,j,b] - pv_flow_ub[i,j,b] <= flow_ub[i,j];

subject to MeetNationalMinzone :
      sum {r in RES} state[r] * specific_energy [r] / energy_sf
      +  pv_minzone_nat  >= minzone_nat[current_week] ;

subject to MeetSIMinzone :
      sum {(r,isl) in RES_ISLAND:isl=='SI'} state[r] * specific_energy [r] / energy_sf
      +  pv_minzone_SI  >= minzone_SI[current_week] ;

subject to MeetTaupoMinzone {r in RES:r=='TAUPO'} :
      state[r] + pv_minzone_taupo >= minzone_taupo[current_week] ;

#---------------------------------------------------------------------
#                       Constraints for cost
#---------------------------------------------------------------------

minimize StageCost:
      fuel_cost_total[tau] + LL_cost_total[tau] + pc_water_balance_total[tau]
      + pc_minzone_total[tau] + pc_minzone_taupo_total[tau] + pc_state_lb_total[tau]
      + pc_flow_lb_total[tau] + pc_flow_ub_total[tau] + pc_spill_total[tau]
      + future_cost_total[tau]
;

subject to CostCalculation1 :
      fuel_cost_total[tau] =
      sum {b in BLOCK} sum {(m,f) in THERMAL_FUEL} fuel_cost[f,current_year,current_week]
*therm_heat_rate[m]*therm_hrs[m,b]*(therm_cap[m]-therm_derating[m,current_year,current_week])
;

subject to CostCalculation2 :
      LL_cost_total[tau] = sum {b in BLOCK} sum{(i,isl) in NODE_ISLAND}
            sum{sec in LL_SECTOR} LL_sec_dist[sec,isl] *
            sum{seg in LL_SEGMENT} LostLoad[tau,sec,seg,i,b]*LL_cost[sec,seg]
;

subject to CostCalculation3 :
      pc_water_balance_total[tau] =
        pc_water_balance * (max{r in RES}(specific_energy [r] / energy_sf))
            * sum {n in NODES,b in BLOCK} (hrs_per_block[current_year,current_week,b] *

            seconds_in_hour*(pv_water_shortage[n,b]+pv_water_excess[n,b]))
;

subject to CostCalculation4 :
      pc_minzone_total[tau] = pc_minzone*(pv_minzone_nat+pv_minzone_SI)
;

subject to CostCalculation5 :
      pc_minzone_taupo_total[tau] = pc_minzone_taupo*(max{r in RES:r=='TAUPO'}specific_energy
[r]/energy_sf)*pv_minzone_taupo
;

subject to CostCalculation10 :
      pc_state_lb_total[tau] = sum{r in RES}specific_energy[r]/energy_sf*pv_state_lb[r]
*pc_state_lb
;

subject to CostCalculation6 :
      pc_flow_lb_total[tau] = pc_flow_lb*(max{r in RES}(specific_energy [r] / energy_sf))*

            sum {(i,j) in ARC_LB,b in BLOCK} (hrs_per_block[current_year,current_week,b]
```

```
*seconds_in_hour*pv_flow_lb[i,j,b])
;

subject to CostCalculation7 :
      pc_flow_ub_total[tau] = pc_flow_ub*(max{r in RES}(specific_energy [r] / energy_sf))*

            sum {(i,j) in ARC_UB,b in BLOCK} (hrs_per_block[current_year,current_week,b]
*seconds_in_hour*pv_flow_ub[i,j,b])
;

subject to CostCalculation9 :
      pc_spill_total[tau] = sum{(i,j) in ARC_SPILL,b in BLOCK} flow_on_arc[i,j,b] * spill_fac
[i,j] * pc_spill
;

subject to CostCalculation8 :
      future_cost_total[tau] = ( if tau<T then theta[tau] else
            ( if water_value_cuts == 1 then theta[tau] else
                  250000000 + sum {r in RES} (ini_state[r]-state[r]) * specific_energy[r]
                  / energy_sf * end_water_value[r]
            )       )
;


# Cut definition to calculate future cost theta
subject to CutDefn {k in 1..nCut[tau,wetdry[tau]]}:
      theta[tau] >= alpha[tau,wetdry[tau],k] + sum {r in RES_cut}(beta[r,tau,wetdry[tau],k] *
state[r]);


###############################################################################
#
#                              End
#
###############################################################################
```