Replication SLP Catherine Grant

2025-11-03

Replication

```
# ============================================================================
# LIGHTWEIGHT RESERVOIR SLP MODEL
# Reduced computational requirements for laptop/desktop


# SPECIFICATIONS:
# - Storage states: 5 (vs 21 original)
# - Inflow states: 3 (vs 7 original)
# - Time periods: 4 quarterly (vs 12 monthly)
# - Decision variables: 300 (vs 37,044 original)

# ============================================================================



set.seed(42)

# Load required package
library(lpSolve)

# ============================================================================
# PARAMETERS
# ============================================================================

# Discretization (reduced for computational feasibility)
N_S <- 5          # Storage states
N_Q <- 3          # Inflow states
T <- 4            # Time periods (quarterly)

# Time
Delta_t <- 2160   # Hours per quarter (90 days × 24 hours)

# Physical parameters
h_min <- 50       # Minimum hydraulic head (meters)
alpha <- 0.5      # Head coefficient
eta <- 0.00981    # Energy conversion factor
U_max <- 5000     # Maximum turbine capacity (m³/s)
scale_factor <- 10000

# Performance targets
Y <- 0.5          # Power generation target (GW)
beta <- 0.90      # Reliability threshold
nu <- 0.05        # Vulnerability threshold (GW)

print("Model parameters set")
```

```
## [1] "Model parameters set"
```

```
print(paste("Variables:", N_S * N_Q * N_S * T))
```

```
## [1] "Variables: 300"
```

```
# ============================================================================
# GENERATING SYNTHETIC INFLOW DATA
# ============================================================================

generate_synthetic_inflows <- function(n_years = 10, n_periods = 4) {
  inflows <- matrix(NA, nrow = n_years, ncol = n_periods)
  seasonal_means <- c(3000, 2500, 1500, 2000)  # Q1-Q4
  seasonal_sds <- c(600, 500, 400, 450)

  for (year in 1:n_years) {
    for (quarter in 1:n_periods) {
      inflows[year, quarter] <- max(500,
        rnorm(1, mean = seasonal_means[quarter], sd = seasonal_sds[quarter]))
    }
  }
  return(inflows)
}

inflow_data <- generate_synthetic_inflows(n_years = 10, n_periods = T)
print("Synthetic data generated")
```

```
## [1] "Synthetic data generated"
```

```
# ============================================================================
# DISCRETIZING INFLOWS
# ============================================================================

discretize_inflows <- function(inflow_data, N_Q) {
  n_periods <- ncol(inflow_data)
  inflow_states <- matrix(NA, nrow = n_periods, ncol = N_Q)

  for (t in 1:n_periods) {
    period_data <- inflow_data[, t]
    quantiles <- quantile(period_data, probs = seq(0, 1, length.out = N_Q + 1))
    for (k in 1:N_Q) {
      inflow_states[t, k] <- mean(c(quantiles[k], quantiles[k + 1]))
    }
  }
  return(inflow_states)
}

inflow_states <- discretize_inflows(inflow_data, N_Q)
print("Inflow states:")
```

```
## [1] "Inflow states:"
```

```
print(round(inflow_states, 0))
```

```
##      [,1] [,2] [,3]
## [1,] 2498 3225 3916
## [2,] 1676 2232 2377
## [3,]  978 1557 1893
## [4,] 1611 2167 2673
```

```
# ============================================================================
# ESTIMATING MARKOV TRANSITIONS
# ============================================================================

estimate_transitions <- function(inflow_data, inflow_states) {
  N_Q <- ncol(inflow_states)
  n_periods <- ncol(inflow_data)
  n_years <- nrow(inflow_data)
  transitions <- array(0, dim = c(N_Q, N_Q, n_periods))

  for (year in 1:(n_years - 1)) {
    for (t in 1:(n_periods)) {
      current_inflow <- inflow_data[year, t]
      if (t < n_periods) {
        next_inflow <- inflow_data[year, t + 1]
      } else {
        next_inflow <- inflow_data[year + 1, 1]
      }

      k <- which.min(abs(inflow_states[t, ] - current_inflow))
      next_t <- ifelse(t < n_periods, t + 1, 1)
      j <- which.min(abs(inflow_states[next_t, ] - next_inflow))

      transitions[k, j, t] <- transitions[k, j, t] + 1
    }
  }

  # Normalize
  for (t in 1:n_periods) {
    for (k in 1:N_Q) {
      row_sum <- sum(transitions[k, , t])
      if (row_sum > 0) {
        transitions[k, , t] <- transitions[k, , t] / row_sum
      } else {
        transitions[k, , t] <- 1 / N_Q
      }
    }
  }
  return(transitions)
}

transition_matrices <- estimate_transitions(inflow_data, inflow_states)
print("Transition matrix Q1->Q2:")
```

```
## [1] "Transition matrix Q1->Q2:"
```

```
print(round(transition_matrices[, , 1], 3))
```

```
##       [,1] [,2]  [,3]
## [1,] 0.667 0.00 0.333
## [2,] 0.000 0.50 0.500
## [3,] 0.000 0.75 0.250
```

```
# ============================================================================
# CREATING STORAGE STATES
# ============================================================================

storage_states <- seq(0, 1, length.out = N_S)

# ============================================================================
# COMPUTING POWER GENERATION G_iklt
# ============================================================================

G <- array(NA, dim = c(N_S, N_Q, N_S, T))

for (t in 1:T) {
  for (i in 1:N_S) {
    for (k in 1:N_Q) {
      for (l in 1:N_S) {
        S_begin <- storage_states[i]
        S_end <- storage_states[l]
        Q <- inflow_states[t, k]

        S_avg <- 0.5 * (S_begin + S_end) * scale_factor
        h <- h_min + alpha * sqrt(S_avg)

        R <- (S_begin - S_end) / Delta_t + Q
        R <- max(0, R)
        R <- min(R, U_max)

        G[i, k, l, t] <- eta * h * R
      }
    }
  }
}

print(paste("Power generation computed. Average:", round(mean(G), 3), "GW"))
```

```
## [1] "Power generation computed. Average: 1832.832 GW"
```

```r
# ============================================================================
# OPTIMIZATION SETUP
# ============================================================================

n_vars <- N_S * N_Q * N_S * T
obj_coef <- as.vector(G) * Delta_t

get_index <- function(i, k, l, t) {
  return(i + (k - 1) * N_S + (l - 1) * N_S * N_Q + (t - 1) * N_S * N_Q * N_S)
}

constraints <- list()
rhs <- c()
dirs <- c()

# Constraint 1: Normalization
for (t in 1:T) {
  constraint <- rep(0, n_vars)
  for (i in 1:N_S) {
    for (k in 1:N_Q) {
      for (l in 1:N_S) {
        idx <- get_index(i, k, l, t)
        constraint[idx] <- 1
      }
    }
  }
  constraints[[length(constraints) + 1]] <- constraint
  rhs <- c(rhs, 1)
  dirs <- c(dirs, "=")
}

# Constraint 2: Reliability
reliability_constraint <- rep(0, n_vars)
for (t in 1:T) {
  for (i in 1:N_S) {
    for (k in 1:N_Q) {
      for (l in 1:N_S) {
        if (G[i, k, l, t] >= Y) {
          idx <- get_index(i, k, l, t)
          reliability_constraint[idx] <- 1 / T
        }
      }
    }
  }
}
constraints[[length(constraints) + 1]] <- reliability_constraint
rhs <- c(rhs, beta)
dirs <- c(dirs, ">=")

# Constraint 3: Vulnerability
vulnerability_constraint <- rep(0, n_vars)
for (t in 1:T) {
```

```
  for (i in 1:N_S) {
    for (k in 1:N_Q) {
      for (l in 1:N_S) {
        if (G[i, k, l, t] < Y) {
          idx <- get_index(i, k, l, t)
          vulnerability_constraint[idx] <- (Y - G[i, k, l, t]) / T
        }
      }
    }
  }
}
constraints[[length(constraints) + 1]] <- vulnerability_constraint
rhs <- c(rhs, nu)
dirs <- c(dirs, "<=")

constraint_matrix <- do.call(rbind, constraints)
print(paste("Constraints built:", nrow(constraint_matrix)))
```

```
## [1] "Constraints built: 6"
```

```
# ============================================================================
# SOLVING
# ============================================================================

print("Solving optimization...")
```

```
## [1] "Solving optimization..."
```

```
result <- lp("max", obj_coef, constraint_matrix, dirs, rhs, all.bin = FALSE)

# ============================================================================
# RESULTS
# ============================================================================

if (result$status == 0) {
  print("OPTIMIZATION SUCCESSFUL!")

  P_solution <- array(result$solution, dim = c(N_S, N_Q, N_S, T))

  # Calculate metrics
  total_generation <- sum(P_solution * G * Delta_t)

  reliability_sum <- 0
  for (t in 1:T) {
    for (i in 1:N_S) {
      for (k in 1:N_Q) {
        for (l in 1:N_S) {
          if (G[i, k, l, t] >= Y) {
            reliability_sum <- reliability_sum + P_solution[i, k, l, t]
          }
        }
      }
    }
  }
  reliability_actual <- reliability_sum / T

  vulnerability_sum <- 0
  for (t in 1:T) {
    for (i in 1:N_S) {
      for (k in 1:N_Q) {
        for (l in 1:N_S) {
          if (G[i, k, l, t] < Y) {
            vulnerability_sum <- vulnerability_sum +
              P_solution[i, k, l, t] * (Y - G[i, k, l, t])
          }
        }
      }
    }
  }
  vulnerability_actual <- vulnerability_sum / T

  # Print results
  print("")
  print("=== RESULTS ===")
  print(paste("Expected generation:", round(total_generation, 2), "MWh/year"))
  print(paste("Reliability:", round(reliability_actual, 4), "(target:", beta, ")"))
  print(paste("Vulnerability:", round(vulnerability_actual, 4), "MW (target:", nu, ")"))

  # Sample policy (Q1)
  print("")
```

```
  print("=== SAMPLE POLICY (Q1) ===")
  state_names <- c("Low", "Med", "High")
  for (i in 1:N_S) {
    for (k in 1:N_Q) {
      probs <- P_solution[i, k, , 1]
      best_l <- which.max(probs)
      if (probs[best_l] > 0.01) {
        cat(sprintf("Storage=%d%%, Inflow=%s -> End=%d%% (prob=%.3f, gen=%.2f MW)\n",
                    storage_states[i] * 100, state_names[k],
                    storage_states[best_l] * 100, probs[best_l],
                    G[i, k, best_l, 1]))
      }
    }
  }

  # Quarterly generation
  print("")
  print("=== GENERATION BY QUARTER ===")
  for (t in 1:T) {
    period_gen <- sum(P_solution[, , , t] * G[, , , t]) * Delta_t
    cat(sprintf("Q%d: %.2f MWh\n", t, period_gen))
  }

} else {
  print(paste("OPTIMIZATION FAILED. Status:", result$status))
}
```

```
## [1] "OPTIMIZATION SUCCESSFUL!"
## [1] ""
## [1] "=== RESULTS ==="
## [1] "Expected generation: 23010296.04 MWh/year"
## [1] "Reliability: 1 (target: 0.9 )"
## [1] "Vulnerability: 0 MW (target: 0.05 )"
## [1] ""
## [1] "=== SAMPLE POLICY (Q1) ==="
## Storage=100%, Inflow=High -> End=100% (prob=1.000, gen=3841.65 MW)
## [1] ""
## [1] "=== GENERATION BY QUARTER ==="
## Q1: 8297970.62 MWh
## Q2: 5036141.12 MWh
## Q3: 4012027.05 MWh
## Q4: 5664157.26 MWh
```

```
print("")
```

```
## [1] ""
```

```r
# ============================================================================
# FUNCTION TO SOLVE MODEL WITH GIVEN PARAMETERS
# ============================================================================

solve_slp_model <- function(Y, beta, nu, scenario_name) {

  cat("\n", strrep("=", 70), "\n", sep="")
  cat("SCENARIO:", scenario_name, "\n")
  cat("Parameters: Y =", Y, "GW, β =", beta, ", v =", nu, "GW\n")
  cat(strrep("=", 70), "\n", sep="")

  n_vars <- N_S * N_Q * N_S * T
  obj_coef <- as.vector(G) * Delta_t

  get_index <- function(i, k, l, t) {
    return(i + (k - 1) * N_S + (l - 1) * N_S * N_Q + (t - 1) * N_S * N_Q * N_S)
  }

  constraints <- list()
  rhs <- c()
  dirs <- c()

  # Constraint 1: Normalization
  for (t in 1:T) {
    constraint <- rep(0, n_vars)
    for (i in 1:N_S) {
      for (k in 1:N_Q) {
        for (l in 1:N_S) {
          idx <- get_index(i, k, l, t)
          constraint[idx] <- 1
        }
      }
    }
    constraints[[length(constraints) + 1]] <- constraint
    rhs <- c(rhs, 1)
    dirs <- c(dirs, "=")
  }

  # Constraint 2: Reliability (only if beta > 0)
  if (beta > 0) {
    reliability_constraint <- rep(0, n_vars)
    for (t in 1:T) {
      for (i in 1:N_S) {
        for (k in 1:N_Q) {
          for (l in 1:N_S) {
            if (G[i, k, l, t] >= Y) {
              idx <- get_index(i, k, l, t)
              reliability_constraint[idx] <- 1 / T
            }
          }
        }
      }
```

```
  }
  constraints[[length(constraints) + 1]] <- reliability_constraint
  rhs <- c(rhs, beta)
  dirs <- c(dirs, ">=")
}

# Constraint 3: Vulnerability (only if nu < Inf)
if (nu < Inf) {
  vulnerability_constraint <- rep(0, n_vars)
  for (t in 1:T) {
    for (i in 1:N_S) {
      for (k in 1:N_Q) {
        for (l in 1:N_S) {
          if (G[i, k, l, t] < Y) {
            idx <- get_index(i, k, l, t)
            vulnerability_constraint[idx] <- (Y - G[i, k, l, t]) / T
          }
        }
      }
    }
  }
  constraints[[length(constraints) + 1]] <- vulnerability_constraint
  rhs <- c(rhs, nu)
  dirs <- c(dirs, "<=")
}

constraint_matrix <- do.call(rbind, constraints)

# Solve
result <- lp("max", obj_coef, constraint_matrix, dirs, rhs, all.bin = FALSE)

if (result$status == 0) {
  P_solution <- array(result$solution, dim = c(N_S, N_Q, N_S, T))

  # Calculate metrics
  total_generation <- sum(P_solution * G * Delta_t)

  reliability_sum <- 0
  for (t in 1:T) {
    for (i in 1:N_S) {
      for (k in 1:N_Q) {
        for (l in 1:N_S) {
          if (G[i, k, l, t] >= Y) {
            reliability_sum <- reliability_sum + P_solution[i, k, l, t]
          }
        }
      }
    }
  }
  reliability_actual <- reliability_sum / T

  vulnerability_sum <- 0
```

```
    for (t in 1:T) {
      for (i in 1:N_S) {
        for (k in 1:N_Q) {
          for (l in 1:N_S) {
            if (G[i, k, l, t] < Y) {
              vulnerability_sum <- vulnerability_sum +
                P_solution[i, k, l, t] * (Y - G[i, k, l, t])
            }
          }
        }
      }
    }
    vulnerability_actual <- vulnerability_sum / T

    cat("✓ OPTIMIZATION SUCCESSFUL\n")
    cat("Expected generation:", round(total_generation, 2), "GWh/year\n")
    cat("Reliability achieved:", round(reliability_actual, 4), "(target:", beta, ")\n")
    cat("Vulnerability:", round(vulnerability_actual, 4), "GW (target:", nu, ")\n")

    return(list(
      status = "success",
      generation = total_generation,
      reliability = reliability_actual,
      vulnerability = vulnerability_actual
    ))

  } else {
    cat("✗ OPTIMIZATION FAILED. Status:", result$status, "\n")
    return(list(status = "failed", generation = NA, reliability = NA, vulnerability = NA))
  }
}

# ============================================================================
# RUN ALL SCENARIOS
# ============================================================================

# Create results storage
results_table <- data.frame(
  No = integer(),
  Y = numeric(),
  beta = numeric(),
  nu = numeric(),
  Model_GWh = numeric(),
  Reliability = numeric(),
  Vulnerability = numeric(),
  stringsAsFactors = FALSE
)

# BASELINE: No RV constraints
cat("\n\n")
```

```
cat(strrep("=", 70), "\n", sep="")
```

```
## ======================================================================
```

```
cat("RUNNING BASELINE SCENARIO (No RV Constraints)\n")
```

```
## RUNNING BASELINE SCENARIO (No RV Constraints)
```

```
cat(strrep("=", 70), "\n", sep="")
```

```
## ======================================================================
```

```
baseline <- solve_slp_model(Y = 0, beta = 0, nu = Inf, "Baseline (No RV)")
```

```
##
## ======================================================================
## SCENARIO: Baseline (No RV)
## Parameters: Y = 0 GW, β = 0 , v = Inf GW
## ======================================================================
## ✓ OPTIMIZATION SUCCESSFUL
## Expected generation: 23010296 GWh/year
## Reliability achieved: 1 (target: 0 )
## Vulnerability: 0 GW (target: Inf )
```

```
baseline_generation <- baseline$generation

# SCENARIO 1: (1.5, 0.98, 0.01)
result1 <- solve_slp_model(Y = 1.5, beta = 0.98, nu = 0.01, "Scenario 1")
```

```
##
## ======================================================================
## SCENARIO: Scenario 1
## Parameters: Y = 1.5 GW, β = 0.98 , v = 0.01 GW
## ======================================================================
## ✓ OPTIMIZATION SUCCESSFUL
## Expected generation: 23010296 GWh/year
## Reliability achieved: 1 (target: 0.98 )
## Vulnerability: 0 GW (target: 0.01 )
```

```
results_table <- rbind(results_table, data.frame(
  No = 1, Y = 1.5, beta = 0.98, nu = 0.01,
  Model_GWh = round(result1$generation, 0),
  Reliability = round(result1$reliability, 4),
  Vulnerability = round(result1$vulnerability, 4)
))

# SCENARIO 2: (1.5, 0.98, 0)
result2 <- solve_slp_model(Y = 1.5, beta = 0.98, nu = 0, "Scenario 2")
```

```
##
## ======================================================================
## SCENARIO: Scenario 2
## Parameters: Y = 1.5 GW, β = 0.98 , v = 0 GW
## ======================================================================
## ✓ OPTIMIZATION SUCCESSFUL
## Expected generation: 23010296 GWh/year
## Reliability achieved: 1 (target: 0.98 )
## Vulnerability: 0 GW (target: 0 )
```

```
results_table <- rbind(results_table, data.frame(
  No = 2, Y = 1.5, beta = 0.98, nu = 0,
  Model_GWh = round(result2$generation, 0),
  Reliability = round(result2$reliability, 4),
  Vulnerability = round(result2$vulnerability, 4)
))

# SCENARIO 3: (1.5, 0, 0.01)
result3 <- solve_slp_model(Y = 1.5, beta = 0, nu = 0.01, "Scenario 3")
```

```
##
## ======================================================================
## SCENARIO: Scenario 3
## Parameters: Y = 1.5 GW, β = 0 , v = 0.01 GW
## ======================================================================
## ✓ OPTIMIZATION SUCCESSFUL
## Expected generation: 23010296 GWh/year
## Reliability achieved: 1 (target: 0 )
## Vulnerability: 0 GW (target: 0.01 )
```

```
results_table <- rbind(results_table, data.frame(
  No = 3, Y = 1.5, beta = 0, nu = 0.01,
  Model_GWh = round(result3$generation, 0),
  Reliability = round(result3$reliability, 4),
  Vulnerability = round(result3$vulnerability, 4)
))

# SCENARIO 4: (2.0, 0.7, 0.3)
result4 <- solve_slp_model(Y = 2.0, beta = 0.7, nu = 0.3, "Scenario 4")
```

```
##
## ========================================================================
## SCENARIO: Scenario 4
## Parameters: Y = 2 GW, β = 0.7 , v = 0.3 GW
## ========================================================================
## ✓ OPTIMIZATION SUCCESSFUL
## Expected generation: 23010296 GWh/year
## Reliability achieved: 1 (target: 0.7 )
## Vulnerability: 0 GW (target: 0.3 )
```

```
results_table <- rbind(results_table, data.frame(
  No = 4, Y = 2.0, beta = 0.7, nu = 0.3,
  Model_GWh = round(result4$generation, 0),
  Reliability = round(result4$reliability, 4),
  Vulnerability = round(result4$vulnerability, 4)
))

# SCENARIO 5: (2.0, 0.6, 0.3)
result5 <- solve_slp_model(Y = 2.0, beta = 0.6, nu = 0.3, "Scenario 5")
```

```
##
## ========================================================================
## SCENARIO: Scenario 5
## Parameters: Y = 2 GW, β = 0.6 , v = 0.3 GW
## ========================================================================
## ✓ OPTIMIZATION SUCCESSFUL
## Expected generation: 23010296 GWh/year
## Reliability achieved: 1 (target: 0.6 )
## Vulnerability: 0 GW (target: 0.3 )
```

```
results_table <- rbind(results_table, data.frame(
  No = 5, Y = 2.0, beta = 0.6, nu = 0.3,
  Model_GWh = round(result5$generation, 0),
  Reliability = round(result5$reliability, 4),
  Vulnerability = round(result5$vulnerability, 4)
))

# ============================================================================
# PRINT RESULTS
# ============================================================================

cat("\n\n")
```

```
cat(strrep("=", 70), "\n", sep="")
```

```
## ========================================================================
```

```
cat("TABLE 3 REPLICATION\n")
```

```
## TABLE 3 REPLICATION
```

```
cat(strrep("=", 70), "\n\n", sep="")
```

```
## ======================================================================
```

```
cat("BASELINE: ", round(baseline_generation, 0), "GWh/year\n\n")
```

```
## BASELINE:  23010296 GWh/year
```

```
print(results_table, row.names = FALSE)
```

```
##  No   Y beta   nu Model_GWh Reliability Vulnerability
##   1 1.5 0.98 0.01  23010296           1             0
##   2 1.5 0.98 0.00  23010296           1             0
##   3 1.5 0.00 0.01  23010296           1             0
##   4 2.0 0.70 0.30  23010296           1             0
##   5 2.0 0.60 0.30  23010296           1             0
```

```
# Calculate losses
results_table$Pct_Change <- round(
  100 * (results_table$Model_GWh - baseline_generation) / baseline_generation, 2
)

cat("\n\nPERCENTAGE CHANGES:\n")
```

```
##
##
## PERCENTAGE CHANGES:
```

```
print(results_table[, c("No", "Y", "beta", "nu", "Model_GWh", "Pct_Change")],
      row.names = FALSE)
```

```
##  No   Y beta   nu Model_GWh Pct_Change
##   1 1.5 0.98 0.01  23010296          0
##   2 1.5 0.98 0.00  23010296          0
##   3 1.5 0.00 0.01  23010296          0
##   4 2.0 0.70 0.30  23010296          0
##   5 2.0 0.60 0.30  23010296          0
```

```
# Comparison with Chen et al.
cat("\n\nCOMPARISON WITH CHEN ET AL.:\n")
```

```
##
##
## COMPARISON WITH CHEN ET AL.:
```

```
chen_losses <- c(-1.11, -0.91, -0.80, -1.29, -0.54)
cat(sprintf("%-8s %-12s %-12s\n", "Scenario", "Chen Loss %", "Our Loss %"))
```

```
## Scenario Chen Loss %  Our Loss %
```

```
cat(sprintf("%-8s %-12s %-12s\n", "--------", "-----------", "----------"))
```

```
## -------- -----------  ----------
```

```
for (i in 1:5) {
  cat(sprintf("%-8d %-12.2f %-12.2f\n", i, chen_losses[i], results_table$Pct_Change[i]))
}
```

```
## 1        -1.11        -0.00
## 2        -0.91        -0.00
## 3        -0.80        -0.00
## 4        -1.29        -0.00
## 5        -0.54        -0.00
```

```
cat("\n✓ Done!\n")
```

```
##
## ✓ Done!
```

```r
# ============================================================================
# EXTENSION: DROUGHT RELEASE CONSTRAINTS
# ============================================================================

# Checking baseline Q3 releases to set appropriate limits
baseline_releases <- numeric()
for (i in 1:N_S) {
  for (k in 1:N_Q) {
    for (l in 1:N_S) {
      R <- (storage_states[i] - storage_states[l]) / Delta_t + inflow_states[3, k]
      if (R > 0) baseline_releases <- c(baseline_releases, R)
    }
  }
}

max_release <- max(baseline_releases)
moderate_limit <- round(max_release * 0.9)
severe_limit <- round(max_release * 0.8)
extreme_limit <- round(max_release * 0.7)

# ============================================================================
# DROUGHT SOLVER
# ============================================================================

solve_drought <- function(Q3_limit) {

  n_vars <- N_S * N_Q * N_S * T
  obj_coef <- as.vector(G) * Delta_t

  get_index <- function(i, k, l, t) {
    i + (k-1)*N_S + (l-1)*N_S*N_Q + (t-1)*N_S*N_Q*N_S
  }

  constraints <- list()
  rhs <- c()
  dirs <- c()

  # Normalization
  for (t in 1:T) {
    constraint <- rep(0, n_vars)
    for (i in 1:N_S) for (k in 1:N_Q) for (l in 1:N_S) {
      constraint[get_index(i,k,l,t)] <- 1
    }
    constraints[[length(constraints) + 1]] <- constraint
    rhs <- c(rhs, 1); dirs <- c(dirs, "=")
  }

  # Reliability
  rel_constraint <- rep(0, n_vars)
  for (t in 1:T) for (i in 1:N_S) for (k in 1:N_Q) for (l in 1:N_S) {
    if (G[i,k,l,t] >= Y) {
      rel_constraint[get_index(i,k,l,t)] <- 1/T
```

```
    }
  }
  constraints[[length(constraints) + 1]] <- rel_constraint
  rhs <- c(rhs, beta); dirs <- c(dirs, ">=")

  # Vulnerability
  vul_constraint <- rep(0, n_vars)
  for (t in 1:T) for (i in 1:N_S) for (k in 1:N_Q) for (l in 1:N_S) {
    if (G[i,k,l,t] < Y) {
      vul_constraint[get_index(i,k,l,t)] <- (Y - G[i,k,l,t])/T
    }
  }
  constraints[[length(constraints) + 1]] <- vul_constraint
  rhs <- c(rhs, nu); dirs <- c(dirs, "<=")

  # DROUGHT CONSTRAINTS
  if (Q3_limit < Inf) {
    for (i in 1:N_S) for (k in 1:N_Q) for (l in 1:N_S) {
      R <- (storage_states[i] - storage_states[l]) / Delta_t + inflow_states[3, k]
      if (R > Q3_limit) {
        constraint <- rep(0, n_vars)
        constraint[get_index(i,k,l,3)] <- 1
        constraints[[length(constraints) + 1]] <- constraint
        rhs <- c(rhs, 0)
        dirs <- c(dirs, "<=")
      }
    }
  }

  # Solve
  result <- lp("max", obj_coef, do.call(rbind, constraints), dirs, rhs)

  if (result$status == 0) {
    P <- array(result$solution, dim = c(N_S, N_Q, N_S, T))
    total <- sum(P * G * Delta_t)
    q <- sapply(1:T, function(t) sum(P[,,,t] * G[,,,t]) * Delta_t)
    return(list(total = total, quarterly = q, status = "success"))
  } else {
    return(list(total = NA, quarterly = rep(NA, T), status = "failed"))
  }
}

# ============================================================================
# RUNning SCENARIOS
# ============================================================================

d0 <- solve_drought(Inf)                # Baseline
d1 <- solve_drought(moderate_limit)    # Moderate
d2 <- solve_drought(severe_limit)      # Severe
d3 <- solve_drought(extreme_limit)     # Extreme

# ============================================================================
```

```r
# STORING RESULTS
# ==============================================================================

drought_results <- data.frame(
  Scenario = c("Baseline", "Moderate", "Severe", "Extreme"),
  Q3_Limit_m3s = c(Inf, moderate_limit, severe_limit, extreme_limit),
  Annual_GWh = c(d0$total, d1$total, d2$total, d3$total) / 1000,
  Q1_GWh = c(d0$quarterly[1], d1$quarterly[1], d2$quarterly[1], d3$quarterly[1]) / 1000,
  Q2_GWh = c(d0$quarterly[2], d1$quarterly[2], d2$quarterly[2], d3$quarterly[2]) / 1000,
  Q3_GWh = c(d0$quarterly[3], d1$quarterly[3], d2$quarterly[3], d3$quarterly[3]) / 1000,
  Q4_GWh = c(d0$quarterly[4], d1$quarterly[4], d2$quarterly[4], d3$quarterly[4]) / 1000
)

# Calculating changes
drought_results$Loss_GWh <- drought_results$Annual_GWh - drought_results$Annual_GWh[1]
drought_results$Loss_Pct <- 100 * drought_results$Loss_GWh / drought_results$Annual_GWh[1]
drought_results$Q3_Change <- drought_results$Q3_GWh - drought_results$Q3_GWh[1]
drought_results$Q4_Change <- drought_results$Q4_GWh - drought_results$Q4_GWh[1]
drought_results$Cost_Million <- abs(drought_results$Loss_GWh) * 60

# Printing results
print(drought_results)
```

```
##   Scenario Q3_Limit_m3s Annual_GWh   Q1_GWh   Q2_GWh   Q3_GWh   Q4_GWh
## 1 Baseline          Inf   23010.30 8297.971 5036.141 4012.027 5664.157
## 2 Moderate         1704   22296.87 8297.971 5036.141 3298.599 5664.157
## 3   Severe         1515   21069.60 8297.971 5036.141 2071.332 5664.157
## 4  Extreme         1325   21069.60 8297.971 5036.141 2071.332 5664.157
##     Loss_GWh  Loss_Pct  Q3_Change Q4_Change Cost_Million
## 1     0.0000  0.000000     0.0000         0         0.00
## 2  -713.4281 -3.100473  -713.4281         0     42805.69
## 3 -1940.6946 -8.434027 -1940.6946         0    116441.68
## 4 -1940.6946 -8.434027 -1940.6946         0    116441.68
```

```r
# ============================================================================
# SHADOW PRICE ANALYSIS
# ============================================================================

solve_drought_with_shadow <- function(Q3_limit) {

  n_vars <- N_S * N_Q * N_S * T
  obj_coef <- as.vector(G) * Delta_t

  get_index <- function(i, k, l, t) {
    i + (k-1)*N_S + (l-1)*N_S*N_Q + (t-1)*N_S*N_Q*N_S
  }

  constraints <- list()
  rhs <- c()
  dirs <- c()

  for (t in 1:T) {
    constraint <- rep(0, n_vars)
    for (i in 1:N_S) for (k in 1:N_Q) for (l in 1:N_S) {
      constraint[get_index(i,k,l,t)] <- 1
    }
    constraints[[length(constraints) + 1]] <- constraint
    rhs <- c(rhs, 1); dirs <- c(dirs, "=")
  }

  rel_constraint <- rep(0, n_vars)
  for (t in 1:T) for (i in 1:N_S) for (k in 1:N_Q) for (l in 1:N_S) {
    if (G[i,k,l,t] >= Y) {
      rel_constraint[get_index(i,k,l,t)] <- 1/T
    }
  }
  constraints[[length(constraints) + 1]] <- rel_constraint
  rhs <- c(rhs, beta); dirs <- c(dirs, ">=")

  vul_constraint <- rep(0, n_vars)
  for (t in 1:T) for (i in 1:N_S) for (k in 1:N_Q) for (l in 1:N_S) {
    if (G[i,k,l,t] < Y) {
      vul_constraint[get_index(i,k,l,t)] <- (Y - G[i,k,l,t])/T
    }
  }
  constraints[[length(constraints) + 1]] <- vul_constraint
  rhs <- c(rhs, nu); dirs <- c(dirs, "<=")

  drought_constraint_start <- length(constraints) + 1

  if (Q3_limit < Inf) {
    for (i in 1:N_S) for (k in 1:N_Q) for (l in 1:N_S) {
      R <- (storage_states[i] - storage_states[l]) / Delta_t + inflow_states[3, k]
      if (R > Q3_limit) {
        constraint <- rep(0, n_vars)
        constraint[get_index(i,k,l,3)] <- 1
```

```
        constraints[[length(constraints) + 1]] <- constraint
        rhs <- c(rhs, 0)
        dirs <- c(dirs, "<=")
      }
    }
  }

  result <- lp("max", obj_coef, do.call(rbind, constraints), dirs, rhs, compute.sens = TRUE)

  if (result$status == 0) {
    drought_shadows <- result$duals[drought_constraint_start:length(constraints)]
    avg_shadow <- mean(abs(drought_shadows)) / Delta_t  # Convert to GWh

    return(list(
      shadow_price = avg_shadow
    ))
  } else {
    return(list(shadow_price = NA))
  }
}

s1 <- solve_drought_with_shadow(moderate_limit)
s2 <- solve_drought_with_shadow(severe_limit)
s3 <- solve_drought_with_shadow(extreme_limit)

shadow_table <- data.frame(
  Scenario = c("Moderate", "Severe", "Extreme"),
  Q3_Limit_m3s = c(moderate_limit, severe_limit, extreme_limit),
  Shadow_Price_GWh = round(c(s1$shadow_price, s2$shadow_price, s3$shadow_price), 1)
)

print(shadow_table, row.names = FALSE)
```

```
##  Scenario Q3_Limit_m3s Shadow_Price_GWh
##  Moderate         1704             92.9
##    Severe         1515            461.1
##   Extreme         1325            461.1
```