A R Code

A.1 Simulation Code

```
#============================================================
# SIMULATION STUDY:
# Projection-based multivariate Bergsma-Dassios tau-star
# and its Rao-Blackwell approximation using Stiefel matrices
#============================================================

rm(list = ls())
set.seed(20261105)

#------------------------------------------------------------
# 1. BASIC SETTINGS
#------------------------------------------------------------

n <- 12                 # Sample size
p <- 2                  # Dimension of X
q <- 2                  # Dimension of Y
d <- p + q              # Dimension of joint observation Z

L <- 100                # Number of projection-direction pairs
M <- 50                 # Number of Stiefel simulations
R <- 200                # Number of repeated-sampling replications

rho <- 0.50             # Cross-correlation parameter

#------------------------------------------------------------
# 2. CONSTRUCT THE POPULATION COVARIANCE MATRIX
#------------------------------------------------------------

# X and Y have identity marginal covariance matrices.
# Only corresponding first coordinates are correlated.
#
# Sigma = [ I_p    R_xy ]
#         [ R_xy'  I_q  ]

R_xy <- matrix(0, nrow = p, ncol = q)
R_xy[1, 1] <- rho

Sigma <- rbind(
  cbind(diag(p), R_xy),
  cbind(t(R_xy), diag(q))
)

# Check positive definiteness
if (min(eigen(Sigma, symmetric = TRUE)$values) <= 0) {
  stop("The population covariance matrix is not positive definite.")
}
```

```
cat("Population covariance matrix:\n")
print(Sigma)

#------------------------------------------------------------
# 3. GENERATE MULTIVARIATE NORMAL OBSERVATIONS
#------------------------------------------------------------

generate_data <- function(n, Sigma, mu = NULL) {

  d <- ncol(Sigma)

  if (is.null(mu)) {
    mu <- rep(0, d)
  }

  # R satisfies t(R) %*% R = Sigma
  Rchol <- chol(Sigma)

  G <- matrix(rnorm(n * d), nrow = n, ncol = d)

  Z <- G %*% Rchol

  # Add the population mean
  Z <- sweep(Z, 2, mu, FUN = "+")

  colnames(Z) <- paste0("Z", seq_len(d))

  return(Z)
}

#------------------------------------------------------------
# 4. GENERATE UNIFORM RANDOM DIRECTIONS ON A UNIT SPHERE
#------------------------------------------------------------

# Each column is an independent uniform direction
# on S^(dimension-1).

sphere_directions <- function(dimension, L) {

  G <- matrix(
    rnorm(dimension * L),
    nrow = dimension,
    ncol = L
  )

  norms <- sqrt(colSums(G^2))

  if (any(norms == 0)) {
    stop("A zero-length Gaussian direction was generated.")
```

```
  }

  U <- sweep(G, 2, norms, FUN = "/")

  return(U)
}

# Generate independent projection directions
# for the X and Y components.

generate_projection_directions <- function(p, q, L) {

  U <- sphere_directions(p, L)
  V <- sphere_directions(q, L)

  return(list(U = U, V = V))
}

#------------------------------------------------------------
# 5. RAW BERGSMA-DASSIOS SIGN KERNEL
#------------------------------------------------------------

# For four real numbers z1,z2,z3,z4:
#
# a(z1,z2,z3,z4)
# = sign{
#   |z1-z2| + |z3-z4|
#   - |z1-z3| - |z2-z4|
# }

bd_raw <- function(z1, z2, z3, z4) {

  value <- abs(z1 - z2) +
    abs(z3 - z4) -
    abs(z1 - z3) -
    abs(z2 - z4)

  return(sign(value))
}

#------------------------------------------------------------
# 6. PROJECTION-BASED MULTIVARIATE KERNEL
#------------------------------------------------------------

# For a fixed quadruple of multivariate observations,
# approximate the double sphere integral by
#
# h(z1,z2,z3,z4)
# = E_{U,V}[ a(U'X1,...,U'X4)
#             a(V'Y1,...,V'Y4) ].
```

```
#
# U and V are independent uniform sphere directions.
#
# The same observation ordering is used in both
# raw sign kernels.

projection_kernel <- function(Z4, p, U, V) {

  if (nrow(Z4) != 4) {
    stop("Z4 must contain exactly four observations.")
  }

  X4 <- Z4[, seq_len(p), drop = FALSE]

  Y4 <- Z4[, (p + 1):ncol(Z4), drop = FALSE]

  # Project the four X observations
  # onto the L directions.

  XP <- X4 %*% U

  # Project the four Y observations
  # onto the same number of independent directions.

  YP <- Y4 %*% V

  # Raw X sign kernel for all projection directions
  aX <- bd_raw(
    XP[1, ],
    XP[2, ],
    XP[3, ],
    XP[4, ]
  )

  # Raw Y sign kernel for all projection directions
  aY <- bd_raw(
    YP[1, ],
    YP[2, ],
    YP[3, ],
    YP[4, ]
  )

  # Monte Carlo approximation to the double
  # projection-sphere integral.
  #
  # Each pair (U[,l],V[,l]) is independently generated.

  return(mean(aX * aY))
}
```

```
#-------------------------------------------------------------
# 7. FOURTH-ORDER U-STATISTIC
#-------------------------------------------------------------

# The estimator is
#
# B = 1/C(n,4) * sum_{i<j<k<l} h(Zi,Zj,Zk,Zl).

projection_tau_star <- function(Z, p, U, V) {

  n <- nrow(Z)

  if (n < 4) {
    stop("At least four observations are required.")
  }

  # All four-element subsets
  subsets <- combn(n, 4)

  number_subsets <- ncol(subsets)

  kernel_values <- numeric(number_subsets)

  for (j in seq_len(number_subsets)) {

    ind <- subsets[, j]

    Z4 <- Z[ind, , drop = FALSE]

    kernel_values[j] <- projection_kernel(
      Z4 = Z4,
      p = p,
      U = U,
      V = V
    )
  }

  B <- mean(kernel_values)

  return(list(
    estimate = B,
    kernel_values = kernel_values,
    number_subsets = number_subsets
  ))
}

#-------------------------------------------------------------
# 8. GENERATE A UNIFORM CONSTRAINED STIEFEL MATRIX
#-------------------------------------------------------------
```

```
# Required constraints:
#
# Q'Q = I_d
# 1_n'Q = 0
#
# Construction:
#
# H = I_n - (1/n)11'
# W = HG
# Q = W(W'W)^(-1/2)
#
# This is the polar orthonormalization of a
# centered Gaussian matrix.

generate_stiefel <- function(n, d) {

  if (n - 1 < d) {
    stop("The condition n - 1 >= d is required.")
  }

  # Centering projection matrix
  H <- diag(n) - matrix(1 / n, n, n)

  # Standard Gaussian matrix
  G <- matrix(rnorm(n * d), nrow = n, ncol = d)

  # Centered Gaussian matrix
  W <- H %*% G

  # Symmetric eigendecomposition of W'W
  A <- crossprod(W)

  eig <- eigen(A, symmetric = TRUE)

  if (min(eig$values) <= 1e-12) {
    stop("Numerically rank-deficient centered Gaussian matrix.")
  }

  # Construct (W'W)^(-1/2)
  A_inv_half <- eig$vectors %*%
    diag(1 / sqrt(eig$values)) %*%
    t(eig$vectors)

  # Polar factor
  Q <- W %*% A_inv_half

  return(Q)
}

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

```
# 9. SYMMETRIC SQUARE ROOT OF THE SAMPLE SCATTER MATRIX
#-------------------------------------------------------------

symmetric_sqrt <- function(A) {

  eig <- eigen(A, symmetric = TRUE)

  if (min(eig$values) <= 0) {
    stop("The sample scatter matrix is not positive definite.")
  }

  A_half <- eig$vectors %*%
    diag(sqrt(eig$values)) %*%
    t(eig$vectors)

  return(A_half)
}

#-------------------------------------------------------------
# 10. RECONSTRUCT A SAMPLE GIVEN ITS SUFFICIENT STATISTIC
#-------------------------------------------------------------

# The observed sufficient statistic is
#
# S = (Zbar, C),
#
# where C = sum_i (Zi-Zbar)(Zi-Zbar)'.
#
# The reconstructed sample is
#
# Zstar = 1 Zbar' + Q C^(1/2).
#
# This preserves both the sample mean and sample scatter.

reconstruct_sample <- function(Q, sample_mean, C_half) {

  n <- nrow(Q)

  Zstar <- matrix(
    1,
    nrow = n,
    ncol = 1
  ) %*% t(sample_mean) +
    Q %*% C_half

  return(Zstar)
}

#-------------------------------------------------------------
# 11. MONTE CARLO RAO-BLACKWELL ESTIMATOR
```

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

rb_monte_carlo <- function(
    Z,
    p,
    U,
    V,
    M = 100,
    retain_simulations = TRUE) {

  n <- nrow(Z)
  d <- ncol(Z)

  if (n - 1 < d) {
    stop("The condition n - 1 >= d is required.")
  }

  #-------------------------------------------------------------
  # Observed sufficient statistic
  #-------------------------------------------------------------

  sample_mean <- colMeans(Z)

  Z_centered <- sweep(
    Z,
    2,
    sample_mean,
    FUN = "-"
  )

  # Sample scatter matrix (not divided by n-1)
  C <- crossprod(Z_centered)

  # Symmetric square root
  C_half <- symmetric_sqrt(C)

  #-------------------------------------------------------------
  # Generate M conditional Stiefel configurations
  #-------------------------------------------------------------

  simulated_B <- numeric(M)

  for (m in seq_len(M)) {

    # Uniform constrained Stiefel matrix
    Q <- generate_stiefel(n, d)

    # Reconstruct a sample with the same
    # observed mean and scatter matrix
    Zstar <- reconstruct_sample(
```

```
      Q = Q,
      sample_mean = sample_mean,
      C_half = C_half
    )

    # Evaluate the projection-based U-statistic
    simulated_B[m] <- projection_tau_star(
      Z = Zstar,
      p = p,
      U = U,
      V = V
    )$estimate
  }

  #------------------------------------------------------------
  # Monte Carlo Rao-Blackwell approximation
  #------------------------------------------------------------

  RB_estimate <- mean(simulated_B)

  # Conditional Monte Carlo standard error
  MCSE <- sd(simulated_B) / sqrt(M)

  # Estimated conditional simulation variance
  MC_variance <- var(simulated_B) / M

  result <- list(
    RB_estimate = RB_estimate,
    MCSE = MCSE,
    MC_variance = MC_variance,
    simulated_B = if (retain_simulations) simulated_B else NULL,
    sample_mean = sample_mean,
    sample_scatter = C
  )

  return(result)
}

#------------------------------------------------------------
# 12. CHECK THE STIEFEL CONSTRAINTS
#------------------------------------------------------------

Q_test <- generate_stiefel(n, d)

cat("\nStiefel constraint: Q'Q\n")
print(round(crossprod(Q_test), 8))

cat("\nStiefel constraint: 1'Q\n")
print(round(colSums(Q_test), 8))
```

```
#-------------------------------------------------------------
# 13. GENERATE ONE SAMPLE AND CALCULATE BOTH ESTIMATORS
#-------------------------------------------------------------

Z <- generate_data(
  n = n,
  Sigma = Sigma
)

directions <- generate_projection_directions(
  p = p,
  q = q,
  L = L
)

U <- directions$U
V <- directions$V

# Original U-statistic
B_original <- projection_tau_star(
  Z = Z,
  p = p,
  U = U,
  V = V
)$estimate

# Monte Carlo Rao-Blackwell estimator
RB_result <- rb_monte_carlo(
  Z = Z,
  p = p,
  U = U,
  V = V,
  M = M
)

cat("\nOriginal U-statistic:\n")
print(B_original)

cat("\nMonte Carlo Rao-Blackwell estimator:\n")
print(RB_result$RB_estimate)

cat("\nConditional Monte Carlo standard error:\n")
print(RB_result$MCSE)

cat("\nEstimated conditional simulation variance:\n")
print(RB_result$MC_variance)

#-------------------------------------------------------------
# 14. MONTE CARLO APPROXIMATION TO THE POPULATION PARAMETER
#-------------------------------------------------------------
```

```
# theta = E[h(Z1,Z2,Z3,Z4)]
#
# Generate independent quadruples and average their
# projection-based kernel values.

theta_reference_mc <- function(
    Sigma,
    p,
    U,
    V,
    N = 20000) {

  d <- ncol(Sigma)

  kernel_values <- numeric(N)

  for (b in seq_len(N)) {

    # Generate four independent observations
    Z4 <- generate_data(
      n = 4,
      Sigma = Sigma
    )

    # Evaluate the projection-integrated kernel
    kernel_values[b] <- projection_kernel(
      Z4 = Z4,
      p = p,
      U = U,
      V = V
    )
  }

  theta_hat <- mean(kernel_values)

  theta_se <- sd(kernel_values) / sqrt(N)

  return(list(
    theta = theta_hat,
    SE = theta_se,
    kernel_values = kernel_values
  ))
}

# Independent reference calculation
theta_result <- theta_reference_mc(
  Sigma = Sigma,
  p = p,
  U = U,
```

```
  V = V,
  N = 20000
)

theta_reference <- theta_result$theta

cat("\nApproximate population dependence parameter:\n")
print(theta_reference)

cat("\nMonte Carlo standard error of reference parameter:\n")
print(theta_result$SE)

#------------------------------------------------------------
# 15. REPEATED-SAMPLING SIMULATION
#------------------------------------------------------------

simulation_study <- function(
    R,
    n,
    Sigma,
    p,
    q,
    L,
    M,
    theta_reference) {

  d <- p + q

  # Generate common projection directions
  # for all repeated samples.
  #
  # This reduces extraneous variation due to
  # changing projection directions across replications.

  directions <- generate_projection_directions(
    p = p,
    q = q,
    L = L
  )

  U <- directions$U
  V <- directions$V

  # Storage
  results <- data.frame(
    Replication = seq_len(R),
    Original = numeric(R),
    Rao_Blackwell = numeric(R),
    MCSE = numeric(R)
  )
```

```
for (r in seq_len(R)) {

  # Generate a new independent dataset
  Z <- generate_data(
    n = n,
    Sigma = Sigma
  )

  # Original U-statistic
  B <- projection_tau_star(
    Z = Z,
    p = p,
    U = U,
    V = V
  )$estimate

  # Monte Carlo Rao-Blackwell approximation
  RB <- rb_monte_carlo(
    Z = Z,
    p = p,
    U = U,
    V = V,
    M = M,
    retain_simulations = FALSE
  )

  results$Original[r] <- B
  results$Rao_Blackwell[r] <- RB$RB_estimate
  results$MCSE[r] <- RB$MCSE

  if (r %% 10 == 0) {
    cat("Completed replication:", r, "of", R, "\n")
  }
}

#------------------------------------------------------------
# Calculate empirical performance measures
#------------------------------------------------------------

performance <- data.frame(
  Estimator = c(
    "Original U-statistic",
    "Monte Carlo Rao-Blackwell"
  ),

  Mean = c(
    mean(results$Original),
    mean(results$Rao_Blackwell)
  ),
```

```
    Bias = c(
      mean(results$Original) - theta_reference,
      mean(results$Rao_Blackwell) - theta_reference
    ),

    Variance = c(
      var(results$Original),
      var(results$Rao_Blackwell)
    ),

    RMSE = c(
      sqrt(mean((results$Original - theta_reference)^2)),
      sqrt(mean((results$Rao_Blackwell - theta_reference)^2))
    ),

    MAE = c(
      mean(abs(results$Original - theta_reference)),
      mean(abs(results$Rao_Blackwell - theta_reference))
    )
  )

  #-------------------------------------------------------------
  # Relative efficiency
  #-------------------------------------------------------------

  relative_efficiency <-
    var(results$Original) /
    var(results$Rao_Blackwell)

  # Empirical variance reduction
  variance_reduction <-
    100 * (
      1 -
        var(results$Rao_Blackwell) /
        var(results$Original)
    )

  # Average estimated conditional MC standard error
  average_MCSE <- mean(results$MCSE)

  return(list(
    results = results,
    performance = performance,
    relative_efficiency = relative_efficiency,
    variance_reduction = variance_reduction,
    average_MCSE = average_MCSE,
    theta_reference = theta_reference
  ))
}
```

```
#------------------------------------------------------------
# 16. EXECUTE THE REPEATED-SAMPLING EXPERIMENT
#------------------------------------------------------------

simulation_result <- simulation_study(
  R = R,
  n = n,
  Sigma = Sigma,
  p = p,
  q = q,
  L = L,
  M = M,
  theta_reference = theta_reference
)

cat("\n============================================\n")
cat("SIMULATION PERFORMANCE\n")
cat("============================================\n")

print(simulation_result$performance)

cat("\nRelative efficiency:\n")
print(simulation_result$relative_efficiency)

cat("\nPercentage variance reduction:\n")
print(simulation_result$variance_reduction)

cat("\nAverage conditional Monte Carlo standard error:\n")
print(simulation_result$average_MCSE)

#------------------------------------------------------------
# 17. HISTOGRAMS
#------------------------------------------------------------

par(mfrow = c(1, 2))

hist(
  simulation_result$results$Original,
  breaks = 25,
  probability = TRUE,
  main = "Original U-statistic",
  xlab = expression(hat(tau)[p,q]^"*"),
  border = "white"
)

abline(
  v = theta_reference,
  lty = 2,
  lwd = 2
```

```
)

hist(
  simulation_result$results$Rao_Blackwell,
  breaks = 25,
  probability = TRUE,
  main = "Monte Carlo Rao-Blackwell",
  xlab = expression(hat(tau)[RB,M]),
  border = "white"
)

abline(
  v = theta_reference,
  lty = 2,
  lwd = 2
)

par(mfrow = c(1, 1))

#-------------------------------------------------------------
# 18. BOXPLOT COMPARISON
#-------------------------------------------------------------

boxplot(
  simulation_result$results$Original,
  simulation_result$results$Rao_Blackwell,
  names = c("Original", "Rao-Blackwell"),
  main = "Sampling distribution comparison",
  ylab = "Estimator value"
)

abline(
  h = theta_reference,
  lty = 2,
  lwd = 2
)

#-------------------------------------------------------------
# 19. EMPIRICAL ESTIMATION ERRORS
#-------------------------------------------------------------

error_original <-
  simulation_result$results$Original - theta_reference

error_RB <-
  simulation_result$results$Rao_Blackwell - theta_reference

boxplot(
  error_original,
  error_RB,
```

```
  names = c("Original", "Rao-Blackwell"),
  main = "Estimation error comparison",
  ylab = "Estimator minus reference parameter"
)

abline(h = 0, lty = 2, lwd = 2)

#------------------------------------------------------------
# 20. CONVERGENCE OF THE RAO-BLACKWELL APPROXIMATION
#------------------------------------------------------------

rb_convergence <- function(
    Z,
    p,
    U,
    V,
    M_max = 1000) {

  n <- nrow(Z)
  d <- ncol(Z)

  sample_mean <- colMeans(Z)

  Z_centered <- sweep(
    Z,
    2,
    sample_mean,
    FUN = "-"
  )

  C <- crossprod(Z_centered)

  C_half <- symmetric_sqrt(C)

  simulated_B <- numeric(M_max)

  for (m in seq_len(M_max)) {

    Q <- generate_stiefel(n, d)

    Zstar <- reconstruct_sample(
      Q,
      sample_mean,
      C_half
    )

    simulated_B[m] <- projection_tau_star(
      Zstar,
      p,
      U,
```

```
      V
    )$estimate
  }

  # Cumulative Monte Carlo estimates
  cumulative_mean <- cumsum(simulated_B) /
    seq_len(M_max)

  # Running Monte Carlo standard error
  cumulative_MCSE <- rep(NA_real_, M_max)

  for (m in 2:M_max) {

    cumulative_MCSE[m] <-
      sd(simulated_B[seq_len(m)]) / sqrt(m)
  }

  return(data.frame(
    M = seq_len(M_max),
    Estimate = cumulative_mean,
    MCSE = cumulative_MCSE,
    Simulated_Value = simulated_B
  ))
}

# Apply to one fixed observed dataset
convergence_result <- rb_convergence(
  Z = Z,
  p = p,
  U = U,
  V = V,
  M_max = 500
)

# Plot convergence
plot(
  convergence_result$M,
  convergence_result$Estimate,
  type = "l",
  lwd = 2,
  xlab = "Number of Stiefel simulations (M)",
  ylab = "Monte Carlo Rao-Blackwell estimate",
  main = "Convergence of the conditional estimator"
)

# Final cumulative estimate as a reference line
abline(
  h = tail(convergence_result$Estimate, 1),
  lty = 2,
  lwd = 2
```

```
)

# Plot the estimated Monte Carlo standard error
plot(
  convergence_result$M[-1],
  convergence_result$MCSE[-1],
  type = "l",
  lwd = 2,
  xlab = "Number of Stiefel simulations (M)",
  ylab = "Estimated Monte Carlo standard error",
  main = "Monte Carlo standard error"
)

#------------------------------------------------------------
# 21. SENSITIVITY ANALYSIS WITH RESPECT TO L
#------------------------------------------------------------

L_values <- c(25, 50, 100, 250, 500, 1000)

projection_sensitivity <- data.frame(
  L = L_values,
  Estimate = NA_real_
)

for (j in seq_along(L_values)) {

  Lj <- L_values[j]

  directions_j <- generate_projection_directions(
    p = p,
    q = q,
    L = Lj
  )

  result_j <- projection_tau_star(
    Z = Z,
    p = p,
    U = directions_j$U,
    V = directions_j$V
  )

  projection_sensitivity$Estimate[j] <-
    result_j$estimate
}

print(projection_sensitivity)

plot(
  projection_sensitivity$L,
  projection_sensitivity$Estimate,
```

```
  type = "b",
  pch = 19,
  xlab = "Number of projection-direction pairs (L)",
  ylab = "Estimated dependence parameter",
  main = "Sensitivity to projection integration"
)

#------------------------------------------------------------
# 22. NUMERICAL VERIFICATION OF THE RECONSTRUCTION
#------------------------------------------------------------

sample_mean <- colMeans(Z)

Z_centered <- sweep(
  Z,
  2,
  sample_mean,
  FUN = "-"
)

C <- crossprod(Z_centered)

C_half <- symmetric_sqrt(C)

Q <- generate_stiefel(n, d)

Zstar <- reconstruct_sample(
  Q = Q,
  sample_mean = sample_mean,
  C_half = C_half
)

# Mean of the reconstructed sample
mean_original <- colMeans(Z)
mean_reconstructed <- colMeans(Zstar)

# Scatter matrix of the reconstructed sample
Zstar_centered <- sweep(
  Zstar,
  2,
  mean_reconstructed,
  FUN = "-"
)

C_reconstructed <- crossprod(Zstar_centered)

cat("\nOriginal sample mean:\n")
print(mean_original)

cat("\nReconstructed sample mean:\n")
```

```
print(mean_reconstructed)

cat("\nMaximum absolute difference in means:\n")
print(max(abs(mean_original - mean_reconstructed)))

cat("\nMaximum absolute difference in scatter matrices:\n")
print(max(abs(C - C_reconstructed)))

cat("\nOrthogonality error:\n")
print(max(abs(crossprod(Q) - diag(d))))

cat("\nCentering error:\n")
print(max(abs(colSums(Q))))
```