# Package 'HOIF'

June 24, 2026

**Type** Package

**Title** Higher-Order Influence Function Estimators for the Average Treatment Effect

**Version** 0.2.0

**Description** Implements Higher-Order Influence Function (HOIF) estimators of the Average Treatment Effect (ATE), following Robins et al. (2008) <doi:10.1214/193940307000000527>, Liu et al. (2017) <doi:10.48550/arXiv.1705.07577> and Liu and Li (2023) <doi:10.48550/arXiv.2302.08097>. Estimators of any order are supported, with optional covariate basis transformations (B-splines, Fourier) and optional K-fold sample splitting (cross-fitting) for improved finite-sample performance. The core higher-order U-statistics are computed exactly via the 'ustats' package, an R interface to the 'Python' package 'u-stats'; the underlying algorithm and its computational complexity are analyzed in Chen, Zhang and Liu (2025) <doi:10.48550/arXiv.2508.12627>. A pure R implementation (up to order 6) is also provided as a fallback that does not require 'Python'.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Depends** R (>= 4.0.0)

**Imports** splines, corpcor, SMUT, ustats (>= 0.1.5)

**Suggests** MASS, testthat (>= 3.0.0), reticulate, knitr, rmarkdown

**URL** https://cxy0714.github.io/HOIF/, https://github.com/cxy0714/HOIF

**BugReports** https://github.com/cxy0714/HOIF/issues

**SystemRequirements** For the default Python backend: Python (>= 3.11) with the 'u-stats', 'numpy' and 'torch' packages (provisioned automatically on first use via 'reticulate', or via ustats::setup_ustats()). Not needed when pure_R_code = TRUE.

**Config/testthat/edition** 3

**RoxygenNote** 7.3.3

**VignetteBuilder** knitr

**Language** en-US

**NeedsCompilation** no

**Author** Xingyu Chen [aut, cre],
Lin Liu [aut]

**Maintainer** Xingyu Chen <xingyuchen0714@sjtu.edu.cn>

**Repository** CRAN

**Date/Publication** 2026-06-24 08:40:10 UTC

# Contents



---

`build_Ej` *Build E_j tensor structure*

---

## Description

Constructs a nested list representing the tensor structure [1, [1,2], ..., [j-1,j], j] used in Higher-Order Influence Function (HOIF) calculations.

## Usage

```
build_Ej(j)
```

## Arguments

| | |
|---|---|
| `j` | Integer greater than or equal to 2 specifying the dimension |

## Value

A nested list with j+1 elements representing the tensor structure: - First element: scalar 1 - Middle elements: vectors [1,2], [2,3], ..., [j-1,j] - Last element: scalar j

## Examples

```
build_Ej(3)
```

`calculate_u_statistics_pure_r_six`
*Compute U-statistics HOIF-type from Order 2 to 6 in pure R code.*

## Description

This function serves as a core computational component in higher-order influence function (HOIF) estimators in pure R code.

## Usage

```
calculate_u_statistics_pure_r_six(Vector_1, Vector_2, A1, A2, A3, A4, A5)
```

## Arguments

| | |
|---|---|
| `Vector_1` | Numeric column vector of length $n$. |
| `Vector_2` | Numeric column vector of length $n$. |
| `A1`, `A2`, `A3`, `A4`, `A5` | Numeric $n \times n$ kernel matrices of the same dimension. |

## Details

Internally, the function constructs kernel matrices for orders 2 through 6 using recursive matrix operations and removes diagonal contributions to ensure degenerate U-statistics.

All diagonal elements of intermediate kernel matrices are removed to avoid self-interactions. Matrix multiplications are performed via 'eigenMapMatMult()' and element-wise products via 'hadamard()'. The exact formula of the output is:

$$\mathbb{U}_{n,m} = \frac{1}{\binom{n}{m} m!} \sum_{i_1 \neq \cdots \neq i_m} Vector_1[i_1] \cdot A1[i_1, i_2] \cdot A1[i_2, i_3] \cdots A1[i_{m-1}, i_m] \cdot Vector_2[i_m]$$

## Value

A named list containing numeric U-statistic estimates:

**U2** Second-order U-statistic

**U3** Third-order U-statistic

**U4** Fourth-order U-statistic

**U5** Fifth-order U-statistic

**U6** Sixth-order U-statistic

## compute_basis_matrix *Compute basis projection matrices*

### Description

Compute basis projection matrices

### Usage

```
compute_basis_matrix(Z, A, Omega1, Omega0)
```

### Arguments

| | |
|---|---|
| Z | Basis matrix (n x k) |
| A | Treatment vector (n x 1) |
| Omega1 | Inverse Gram matrix for treatment group |
| Omega0 | Inverse Gram matrix for control group |

### Value

List with B1 and B0 (projection matrices)

### Examples

```
n <- 100
Z <- cbind(1, matrix(rnorm(n * 2), n, 2))
A <- rbinom(n, 1, 0.5)
Omega <- compute_gram_inverse(Z, A)
B <- compute_basis_matrix(Z, A, Omega$Omega1, Omega$Omega0)
dim(B$B1)
```

## compute_gram_inverse *Compute inverse of weighted Gram matrix*

### Description

Compute inverse of weighted Gram matrix

### Usage

```
compute_gram_inverse(Z, A, method = "direct")
```

### Arguments

| | |
|---|---|
| Z | Basis matrix (n x k) |
| A | Treatment vector |
| method | Character: "direct" (Cholesky-based inversion, falling back to the Moore-Penrose inverse from **MASS** if the Gram matrix is not positive definite), "nlshrink" (nonlinear shrinkage), or "corpcor" (pseudoinverse via **corpcor**) |

### Value

List with Omega1 and Omega0 (inverse Gram matrices)

### Examples

```
n <- 100
Z <- cbind(1, matrix(rnorm(n * 2), n, 2))
A <- rbinom(n, 1, 0.5)
Omega <- compute_gram_inverse(Z, A)
str(Omega)
```

---

## compute_hoif_estimators

*Compute HOIF Estimators for ATE*

---

### Description

Compute HOIF Estimators for ATE

### Usage

```
compute_hoif_estimators(
  residuals,
  B_matrices,
  m = 7,
  backend = "torch",
  pure_R_code = FALSE,
  dtype = NULL
)
```

### Arguments

| | |
|---|---|
| residuals | A list containing the computed residuals: ‘R1‘, ‘r1‘, ‘R0‘, and ‘r0‘. |
| B_matrices | A list containing the projection-like basis matrices: ‘B1‘ and ‘B0‘. |
| m | Integer. The maximum order of the HOIF estimator. |

| | |
|---|---|
| backend | Character. The computation backend used by ustat; either "torch" (default) or "numpy". If PyTorch is not available, 'ustats' falls back to "numpy" with a warning. |
| pure_R_code | Logical. Whether to use a native R implementation. This serves as a fallback when the Python environment used by 'ustats' (via 'reticulate') is not available. Note: The pure R implementation only supports up to the 6th order (m = 6). |
| dtype | Optional character string passed to ustat controlling the numeric precision of the Python backend: "float32" or "float64". The default NULL selects the precision automatically (float32 on a CUDA GPU, float64 otherwise). Ignored when pure_R_code = TRUE. |

## Value

A list of HOIF estimators (ATE, HOIF, IIFF) for orders l = 2, ..., m.

## Examples

```
# Pure R example (no Python required), up to order 6
n <- 100
Z <- cbind(1, rnorm(n))
A <- rbinom(n, 1, 0.5)
Y <- A + Z[, 2] + rnorm(n)
residuals <- compute_residuals(A, Y, mu1 = 1 + Z[, 2], mu0 = Z[, 2],
                               pi = rep(0.5, n))
Omega <- compute_gram_inverse(Z, A)
B <- compute_basis_matrix(Z, A, Omega$Omega1, Omega$Omega0)
est <- compute_hoif_estimators(residuals, B, m = 6, pure_R_code = TRUE)
est$ATE

## Not run:
# Python backend (requires the 'ustats' Python dependencies), any order
est <- compute_hoif_estimators(residuals, B, m = 7, backend = "torch")

## End(Not run)
```

---

# compute_residuals *Compute residuals for both treatment groups*

---

## Description

Compute residuals for both treatment groups

## Usage

```
compute_residuals(A, Y, mu1, mu0, pi)
```

### Arguments

| | |
|---|---|
| A | Treatment vector (0 or 1) |
| Y | Outcome vector |
| mu1 | Predicted outcomes under treatment (mu(1, X)) |
| mu0 | Predicted outcomes under control (mu(0, X)) |
| pi | Propensity scores |

### Value

List with R1, r1, R0, r0

### Examples

```
n <- 100
A <- rbinom(n, 1, 0.5)
Y <- rnorm(n)
res <- compute_residuals(A, Y, mu1 = rep(0.5, n), mu0 = rep(-0.5, n),
                         pi = rep(0.5, n))
str(res)
```

---

## hoif_ate *Main function: HOIF estimators for ATE with optional sample splitting*

---

### Description

Computes the higher-order influence function terms of orders 2 to m, which estimate the estimable bias of the standard first-order doubly robust (AIPW) estimator of the ATE and are used to debias it.

### Usage

```
hoif_ate(
  X,
  A,
  Y,
  mu1,
  mu0,
  pi,
  transform_method = "none",
  basis_dim = NULL,
  inverse_method = "direct",
  m = 7,
  sample_split = FALSE,
  n_folds = 2,
```

```
  backend = "torch",
  seed = 42,
  pure_R_code = FALSE,
  dtype = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| X | Covariate matrix (n x p) |
| A | Treatment vector (n x 1) |
| Y | Outcome vector (n x 1) |
| mu1 | Predicted outcomes under treatment (predictions supplied by the user, ideally estimated on a separate, independent sample) |
| mu0 | Predicted outcomes under control (see ‘mu1‘) |
| pi | Predicted propensity scores (see ‘mu1‘) |
| transform_method | Character: method to transform covariates before constructing basis functions. - "splines": use basis splines expansion - "fourier": use Fourier basis expansion - "none": no transformation (use raw covariates) |
| basis_dim | Integer: number of basis functions to generate when using "splines" or "fourier" transformations. Higher values provide more flexible approximations but may increase variance. |
| inverse_method | Character: regularization method for Gram matrix inversion. - "direct": Cholesky-based inversion (falls back to the Moore-Penrose inverse from **MASS** when the Gram matrix is not positive definite) - "nlshrink": nonlinear shrinkage estimator (Ledoit-Wolf type) - "corpcor": shrinkage via the corpcor package (for high-dimensional settings) |
| m | Maximum order for HOIF (up to 6 when pure_R_code = TRUE) |
| sample_split | Logical: whether to cross-fit the inverse weighted Gram matrix against the U-statistics. If ‘TRUE‘ (the eHOIF case), the sample is split into ‘n_folds‘ folds; for each fold, the Gram matrix is estimated on the remaining folds, the U-statistics are computed on that fold, and the results are averaged across folds. If ‘FALSE‘ (the sHOIF case), both are computed on the same sample, without distinction. Note this is not a cross-fitting of the nuisance functions, whose predictions are supplied via ‘mu1‘, ‘mu0‘, ‘pi‘. |
| n_folds | Number of folds for sample splitting (if used) |
| backend | Character: computation backend used by ustat; "torch" (default) or "numpy". Ignored when pure_R_code = TRUE. |
| seed | Random seed for reproducibility (for sample splitting) |
| pure_R_code | Logical: if ‘TRUE‘, the higher-order U-statistics are computed with a pure R implementation (no Python required), which supports orders up to m = 6. If ‘FALSE‘ (default), they are computed by the ’ustats’ package, whose Python dependencies are provisioned automatically on first use (see the package README). |

| | |
|---|---|
| dtype | Optional character string ("float32" or "float64") controlling the numeric precision of the Python backend; 'NULL' (default) selects the precision automatically. Passed to ustat; ignored when pure_R_code = TRUE. |
| ... | Additional arguments passed to transform_covariates |

## Details

Conceptually, HOIF estimation involves three estimation tasks, and ideally each uses its own, independent part of the data: (1) estimating the nuisance functions mu(1, X), mu(0, X) and pi(X); (2) estimating the inverse weighted Gram matrix; (3) computing the higher-order U-statistics. This package does not implement task (1): 'hoif_ate()' only takes the nuisance *predictions* 'mu1', 'mu0' and 'pi' as inputs, so the overall three-way cross-fitting is left to the user. The 'sample_split' argument controls only the split between tasks (2) and (3); see its description below.

## Value

An object of class "hoif_ate": a list with components

| | |
|---|---|
| ATE | ATE estimates for orders 2 to m |
| HOIF1, HOIF0 | HOIF estimates for the treated/control arm |
| IIFF1, IIFF0 | Incremental influence function terms for the treated/control arm |
| orders | The orders 2 to m |
| convergence_data | Data frame with the ATE estimate per order |

## See Also

compute_hoif_estimators for the lower-level estimation routine.

## Examples

```
# A small, self-contained example using the pure R backend
set.seed(1)
n <- 100
X <- matrix(rnorm(n), ncol = 1)
A <- rbinom(n, 1, 0.5)
Y <- as.numeric(A + X %*% 0.5 + rnorm(n, 0, 0.1))
mu1 <- as.numeric(1 + X %*% 0.5)
mu0 <- as.numeric(X %*% 0.5)
pi <- rep(0.5, n)

fit <- hoif_ate(X, A, Y, mu1 = mu1, mu0 = mu0, pi = pi,
                transform_method = "none", m = 6,
                pure_R_code = TRUE)
print(fit)

## Not run:
# Python backend (provisioned automatically on first use), order m = 7,
# with 2-fold sample splitting
fit <- hoif_ate(X, A, Y, mu1 = mu1, mu0 = mu0, pi = pi,
```

```
                transform_method = "none", m = 7,
                sample_split = TRUE, n_folds = 2, seed = 123,
                backend = "torch")
print(fit)
plot(fit)

## End(Not run)
```

---

`plot.hoif_ate` *Plot convergence of ATE estimates*

---

## Description

Plot convergence of ATE estimates

## Usage

```
## S3 method for class 'hoif_ate'
plot(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | Object of class hoif_ate |
| `...` | Additional arguments passed to plot |

## Value

No return value, called for its side effect of drawing a convergence plot of the ATE estimates against the order.

---

`print.hoif_ate` *Print method for hoif_ate objects*

---

## Description

Print method for hoif_ate objects

## Usage

```
## S3 method for class 'hoif_ate'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | Object of class hoif_ate |
| `...` | Additional arguments (unused) |

### Value

The input object x, invisibly. Called for its side effect of printing a summary of the estimates to the console.

---

`transform_covariates` *HOIF Estimators for Average Treatment Effect*

---

### Description

This file implements the Higher Order Influence Function (HOIF) estimators for Average Treatment Effect (ATE) estimation with nuisance functions.

### Usage

```
transform_covariates(X, method = "splines", basis_dim, degree = 3, period = 1)
```

### Arguments

| | |
|---|---|
| X | Matrix of covariates (n x p) |
| method | Character: "splines", "fourier", or "none" |
| basis_dim | Integer: dimension of basis expansion per covariate (ignored if method = "none"; must be at least degree + 1 for "splines" and at least 2 for "fourier") |
| degree | Integer: degree for B-splines (default 3; ignored if method != "splines") |
| period | Numeric: period for Fourier basis (default 1; ignored if method != "fourier") |

### Value

Matrix of transformed covariates. For method = "none" the input X is returned unchanged (n x p); for "splines" and "fourier" the basis expansions of all covariates are column-bound together with an intercept column.

### Author(s)

Xingyu Chen Transform covariates to basis functions

### Examples

```
X <- matrix(rnorm(40), nrow = 20, ncol = 2)
Z_splines <- transform_covariates(X, method = "splines", basis_dim = 5)
Z_fourier <- transform_covariates(X, method = "fourier", basis_dim = 4)
Z_raw <- transform_covariates(X, method = "none")
dim(Z_splines)
dim(Z_fourier)
```

# Index