

Package ‘SemiEstimate’

July 21, 2025

Title Solve Semi-Parametric Estimation by Implicit Profiling

Version 1.1.3

Description Semi-parametric estimation problem can be solved by two-step Newton-Raphson iteration. The implicit profiling method<[doi:10.48550/arXiv.2108.07928](https://doi.org/10.48550/arXiv.2108.07928)> is an improved method of two-step NR iteration especially for the implicit-bundled type of the parametric part and non-parametric part. This package provides a function semislv() supporting the above two methods and numeric derivative approximation for unprovided Jacobian matrix.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, numDeriv, purrr, rlang, testthat (>= 3.0.0), BB, nleqslv, splines2

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Jinhua Su [aut, cre] (ORCID: <<https://orcid.org/0000-0002-6344-0607>>)

Maintainer Jinhua Su <944866518@qq.com>

Repository CRAN

Date/Publication 2021-09-06 07:10:02 UTC

Contents

semislv	2
Index	4

semislv

*Solve Semi-parametric estimation by implicit profiling***Description**

Solve Semi-parametric estimation by implicit profiling

Usage

```
semislv(
  theta,
  lambda,
  Phi_fn,
  Psi_fn,
  jac = list(),
  intermediates = list(),
  method = "implicit",
  diy = FALSE,
  control = list(max_iter = 100, tol = 0.001),
  save = list(time = TRUE, path = FALSE),
  ...
)
```

Arguments

theta	the initial value of parametric part
lambda	the initial value of non-parametric part
Phi_fn	the equation function highly relevant to the parametric part
Psi_fn	the equation function highly relevant to the non-parametric part
jac	a list containing some of derivate info of Phi_der_theta_fn, Psi_der_theta_fn, Phi_der_lambda_fn, Psi_der_lambda_fn,
intermediates	a list containing the important variables for diy mode
method	"implicit" or "iterative"
diy	a bool value to decide to parse user designed function
control	a list like list(max_iter = 100, tol = 1e-3) to control the early stop
save	a list like list(time = FALSE, path = FALSE) to control saving setting
...	static parameter for Phi_fn, Psi_fn. Diy execution function.

Value

A save space containing final iteration result and iteration path

Examples

```

Phi_fn <- function(theta, lambda, alpha) 2 * theta + alpha * lambda
Psi_fn <- function(theta, lambda, alpha) 2 * lambda + alpha * theta
# build quasi jacobian by package NumDeriv
res <- semislv(1, 1, Phi_fn, Psi_fn, alpha = 1)
res <- semislv(1, 1, Phi_fn, Psi_fn, method = "iterative", alpha = 1)
# parsing all mathematical Jacobian function by user
res <- semislv(1, 1, Phi_fn, Psi_fn, jac = list(
  Phi_der_theta_fn = function(theta, lambda, alpha) 2,
  Phi_der_lambda_fn = function(theta, lambda, alpha) alpha,
  Psi_der_theta_fn = function(theta, lambda, alpha) alpha,
  Psi_der_lambda_fn = function(theta, lambda, alpha) 2
), method = "implicit", alpha = 1)
res <- semislv(1, 1, Phi_fn, Psi_fn, jac = list(
  Phi_der_theta_fn = function(theta, lambda, alpha) 2,
  Psi_der_lambda_fn = function(theta, lambda, alpha) 2
), method = "iterative", alpha = 1)
# parsing partial mathematical user-provided Jacobian, the rest will be generated by the NumDeriv
res <- semislv(1, 1, Phi_fn, Psi_fn,
  jac = list(Phi_der_theta_fn = function(theta, lambda, alpha) 2),
  method = "implicit", alpha = 1
)
res <- semislv(1, 1, Phi_fn, Psi_fn,
  jac = list(Phi_der_theta_fn = function(theta, lambda, alpha) 2),
  method = "iterative", alpha = 1
)
# use some package or solve the updating totally by the user
# Cases: (1) use thirty party package (2) save the intermediates
# use diy = True, then the package will be just a wrapper for your personalise code
# diy is an advanced mode for researchers, see more examples in our vigettee documents

```

Index

semisl_v, [2](#)