# Package 'inteq'

May 8, 2026

**Type** Package

**Title** Numerical Solution of Integral Equations

**Version** 1.1

**Description** An R implementation of Matthew Thomas's 'Python' library 'inteq'. First, this solves Fredholm integral equations of the first kind ($f(s) = \int_a^b K(s, y) g(y) dy$) using methods described by Twomey (1963) <doi:10.1145/321150.321157>. Second, this solves Volterra integral equations of the first kind ($f(s) = \int_0^s K(s,y) g(t) dt$) using methods from Betto and Thomas (2021) <doi:10.48550/arXiv.2106.08496>. Third, this solves Voltera integral equations of the second kind ($g(s) = f(s) + \int_a^s K(s,y) g(y) dy$) using methods from Linz (1969) <doi:10.1137/0706034>.

**Suggests** knitr, rmarkdown, testthat

**VignetteBuilder** knitr

**License** MIT + file LICENSE

**RoxygenNote** 7.3.2

**Encoding** UTF-8

**NeedsCompilation** no

**Author** Mark Clements [aut, cre],
Aaron Jehle [aut],
Matthew Thomas [ctb]

**Maintainer** Mark Clements <mark.clements@ki.se>

**Repository** CRAN

**Date/Publication** 2026-04-25 14:00:02 UTC

## Contents

---

`diag_ext` *Internal function to generate diagonal matrices with possibly an offset with possibly mirrored diagonal*

---

## Description

Internal function to generate diagonal matrices with possibly an offset with possibly mirrored diagonal

## Usage

```
diag_ext(x, index, mirror = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | vector |
| `index` | integer offset index for the diagonal (can be negative) |
| `mirror` | logical for whether to mirror at the diagonal |

## Value

diagonal matrix

---

`fredholm_solve` *Solve a Fredholm equation of the first and second kind*

---

## Description

Solve a Fredholm equation of the first and second kind

## Usage

```
fredholm_solve(
  k,
  f = function(x) x,
  a = -0,
  b = 1,
  num = 41L,
  smin = 0,
  smax = 1,
  snum = 41L,
  gamma = 0.001
)
```

## Arguments

| | |
|---|---|
| k | kernel function of two time scales |
| f | left hand side function with f(a)=0 |
| a | lower bound of the grid for the integral approximation |
| b | upper bound of the grid for the integral approximation |
| num | number of points for the grid of the integral approximation |
| smin | lower bound of enforcement values for equation |
| smax | upper bound of enforcement values for equation |
| snum | number of points for the grid for the equation |
| gamma | regularization parameter |

## Value

data-frame with evaluation points 'ygrid' and calculations 'ggrid'

## Examples

```
# Define the kernel function
k <- function(s, t) {
    ifelse(abs(s - t) <= 3, 1 + cos(pi * (t - s) / 3), 0)
}

# Define the right-hand side function
f <- function(s) {
    sp <- abs(s)
    sp3 <- sp * pi / 3
    ((6 - sp) * (2 + cos(sp3)) + (9 / pi) * sin(sp3)) / 2
}

# Define the true solution for comparison
trueg <- function(s) {
    k(0, s)
}

# Solve the Fredholm equation
res <- fredholm_solve(
    k, f, -3, 3, 1001L,
    smin = -6, smax = 6, snum = 2001L,
    gamma = 0.01
)

# Plot the results on the same graph using base graphics
plot(
    res$ygrid, res$ggrid,
    type = "l",
    col = "blue",
    xlim = c(-3, 3),
    #ylim = c(-1, 1),
    xlab = "s",
```

```
    ylab = "g(s)",
    main = "Fredholm Equation Solution"
)
# add the true solution
lines(res$ygrid, trueg(res$ygrid), col = "red", lty = 2)
legend(
    "topright",
    legend = c("Estimated Value", "True Value"),
    col = c("blue", "red"),
    lty = c(1, 2)
)
```

---

`indexing` *Internal function to convert (row,col) to vector index*

---

## Description

Internal function to convert (row,col) to vector index

## Usage

```
indexing(row, col, nrows)
```

## Arguments

| | |
|---|---|
| `row` | integer vector the rows |
| `col` | integer vector for the columns |
| `nrows` | integer for the number of rows |

## Value

an index vector for the cells

---

`makeH` *Make H Matrix as in (Twomey 1963)*

---

## Description

Make H Matrix as in (Twomey 1963)

## Usage

```
makeH(dim)
```

### Arguments

| | |
|---|---|
| dim | integer for the number of dimensions of H (i.e. number of nodes for integral approximation) |

### Value

a matrix

---

## simpson *Internal function to calculate integration weights for Simpson's rule*

---

### Description

Internal function to calculate integration weights for Simpson's rule

### Usage

```
simpson(dim)
```

### Arguments

| | |
|---|---|
| dim | integer for the number of nodes |

### Value

a double vector for the integration weights

---

## smooth *Internal function to smooth a vector of values using two-point average*

---

### Description

Internal function to smooth a vector of values using two-point average

### Usage

```
smooth(v)
```

### Arguments

| | |
|---|---|
| v | double vector to be smoother |

### Value

smoothed double vector

`volterra_solve` *Solve a Volterra equation of the first kind*

## Description

Solve a Volterra equation of the first kind

## Usage

```
volterra_solve(
  k,
  f = function(x) x,
  a = 0,
  b = 1,
  num = 1000L,
  method = c("midpoint", "trapezoid")
)
```

## Arguments

| | |
|---|---|
| k | kernel function of two time scales |
| f | left hand side (free) function with f(a)=0 |
| a | lower bound of the integral |
| b | upper bound of the integral |
| num | integer for the number of evaluation points |
| method | string for the method |

## Value

data-frame with evaluation points 'sgrid' and calculations 'ggrid'

## Examples

```
k <- function(s,t) {
        cos(t-s)
}
trueg <- function(s) {
    (2+s**2)/2
}

res <- volterra_solve(k,a=0,b=1,num=1000)

plot(
    res$sgrid, res$ggrid,
    type = "l",
    col = "blue",
    xlim = c(0, 1),
```

```
    #ylim = c(-1, 1),
    xlab = "s",
    ylab = "g(s)",
    main = "Volterra Equation Solution first kind"
)
# add the true solution
lines(res$sgrid, trueg(res$sgrid), col = "red", lty = 2)
legend(
    "topright",
    legend = c("Estimated Value", "True Value"),
    col = c("blue", "red"),
    lty = c(1, 2)
)
```

---

`volterra_solve2` *Solve a Volterra equation of the second kind*

---

## Description

Solve a Volterra equation of the second kind

## Usage

```
volterra_solve2(
  k,
  f = function(x) x,
  a = 0,
  b = 1,
  num = 1001L,
  method = c("trapezoid", "midpoint")
)
```

## Arguments

| | |
|---|---|
| `k` | kernel function of two time scales |
| `f` | left hand side (free) function with f(a)=0 |
| `a` | lower bound of the integral |
| `b` | upper bound of the integral |
| `num` | integer for the number of evaluation points |
| `method` | string for the method |

## Value

data-frame with evaluation points 'sgrid' and calculated values 'ggrid'

## Examples

```
k <- function(s,t) {
        0.5 * (t-s)** 2 * exp(t-s)
}
free <- function(t) {
    0.5 * t**2 * exp(-t)
}
true <- function(t) {
    1/3 * (1 - exp(-3*t/2) * (cos(sqrt(3)/2*t) + sqrt(3) * sin(sqrt(3)/2*t)))
}

res <- volterra_solve2(k,free,a=0,b=6,num=100)

plot(
    res$sgrid, res$ggrid,
    type = "l",
    col = "blue",
    xlim = c(0, 6),
    #ylim = c(-1, 1),
    xlab = "s",
    ylab = "g(s)",
    main = "Volterra Equation Solution second kind"
)
# add the true solution
lines(res$sgrid, true(res$sgrid), col = "red", lty = 2)
legend(
    "topright",
    legend = c("Estimated Value", "True Value"),
    col = c("blue", "red"),
    lty = c(1, 2)
)
```

# Index