

# Package ‘RcppDynProg’

December 15, 2020

**Type** Package

**Title** 'Rcpp' Dynamic Programming

**Version** 0.2.0

**Date** 2020-12-15

**URL** <https://github.com/WinVector/RcppDynProg/>,  
<https://winvector.github.io/RcppDynProg/>

**BugReports** <https://github.com/WinVector/RcppDynProg/issues>

**Maintainer** John Mount <jmount@win-vector.com>

## Description

Dynamic Programming implemented in 'Rcpp'. Includes example partition and out of sample fitting applications. Also supplies additional custom coders for the 'vtreat' package.

**License** GPL-2 | GPL-3

**Depends** R (>= 3.4.0)

**Imports** wrapr (>= 2.0.4), Rcpp (>= 1.0.0), utils, stats

**LinkingTo** Rcpp, RcppArmadillo

**RoxygenNote** 7.1.1

**Suggests** tinytest, knitr, rmarkdown

**VignetteBuilder** knitr

**NeedsCompilation** yes

**Author** John Mount [aut, cre],  
Nina Zumel [aut],  
Win-Vector LLC [cph]

**Repository** CRAN

**Date/Publication** 2020-12-15 19:20:02 UTC

## R topics documented:

const_costs	2
const_costs_logistic	3
lin_costs	3
lin_costs_logistic	4
piecewise_constant	5
piecewise_constant_coder	5
piecewise_linear	6
piecewise_linear_coder	7
RcppDynProg	7
score_solution	8
solve_for_partition	8
solve_for_partitionc	10
solve_interval_partition	11
solve_interval_partition_k	12
solve_interval_partition_no_k	12
<b>Index</b>	<b>14</b>

---

const_costs	<i>const_costs</i>
-------------	--------------------

---

### Description

Built matrix of total out of sample interval square error costs for held-out means. One indexed.

### Usage

```
const_costs(y, w, min_seg, indices)
```

### Arguments

y	NumericVector, values to group in order.
w	NumericVector, weights.
min_seg	positive integer, minimum segment size ( $\geq 1$ ).
indices	IntegerVector, order list of indices to pair.

### Value

xcosts NumericMatix, for  $j \geq i$  xcosts(i,j) is the cost of partition element  $[i, \dots, j]$  (inclusive).

### Examples

```
const_costs(c(1, 1, 2, 2), c(1, 1, 1, 1), 1, 1:4)
```

---

const\_costs\_logistic    *const\_costs\_logistic*

---

**Description**

Built matrix of interval logistic costs for held-out means. One indexed.

**Usage**

```
const_costs_logistic(y, w, min_seg, indices)
```

**Arguments**

y	NumericVector, 0/1 values to group in order (should be in interval [0,1]).
w	NumericVector, weights (should be positive).
min_seg	positive integer, minimum segment size ( $\geq 1$ ).
indices	IntegerVector, order list of indices to pair.

**Value**

xcosts NumericMatix, for  $j \geq i$  xcosts(i,j) is the cost of partition element [i,...j] (inclusive).

**Examples**

```
const_costs_logistic(c(0.1, 0.1, 0.2, 0.2), c(1, 1, 1, 1), 1, 1:4)
```

---

lin\_costs                    *lin\_costs*

---

**Description**

Built matrix of interval costs for held-out linear models. One indexed.

**Usage**

```
lin_costs(x, y, w, min_seg, indices)
```

**Arguments**

x	NumericVector, x-coords of values to group.
y	NumericVector, values to group in order.
w	NumericVector, weights.
min_seg	positive integer, minimum segment size ( $\geq 1$ ).
indices	IntegerVector, ordered list of indices to pair.

**Value**

xcosts NumericMatix, for  $j \geq i$  xcosts(i,j) is the cost of partition element [i,...j] (inclusive).

**Examples**

```
lin_costs(c(1, 2, 3, 4), c(1, 2, 2, 1), c(1, 1, 1, 1), 1, 1:4)
```

---

`lin_costs_logistic`     *lin\_costs\_logistic deviance costs.*

---

**Description**

Built matrix of interval deviance costs for held-out logistic models. Fits are evaluated in-sample. One indexed.

**Usage**

```
lin_costs_logistic(x, y, w, min_seg, indices)
```

**Arguments**

x	NumericVector, x-coords of values to group.
y	NumericVector, values to group in order (should be in interval [0,1]).
w	NumericVector, weights (should be positive).
min_seg	positive integer, minimum segment size ( $\geq 1$ ).
indices	IntegerVector, ordered list of indices to pair.

**Value**

xcosts NumericMatix, for  $j \geq i$  xcosts(i,j) is the cost of partition element [i,...j] (inclusive).

**Examples**

```
lin_costs_logistic(c(1, 2, 3, 4, 5, 6, 7), c(0, 0, 1, 0, 1, 1, 0), c(1, 1, 1, 1, 1, 1, 1), 3, 1:7)
```

---

piecewise\_constant     *Piecewise constant fit.*

---

### Description

vtreat custom coder based on RcppDynProg::solve\_for\_partition().

### Usage

```
piecewise_constant(varName, x, y, w = NULL)
```

### Arguments

varName	character, name of variable to work on.
x	numeric, input values.
y	numeric, values to estimate.
w	numeric, weights.

### Examples

```
piecewise_constant("x", 1:8, c(-1, -1, -1, -1, 1, 1, 1, 1))
```

---

piecewise\_constant\_coder  
                          *Piecewise constant fit coder factory.*

---

### Description

Build a piecewise constant fit coder with some parameters bound in.

### Usage

```
piecewise_constant_coder(  
  penalty = 1,  
  min_n_to_chunk = 1000,  
  min_seg = 10,  
  max_k = 1000  
)
```

**Arguments**

penalty	per-segment cost penalty.
min_n_to_chunk	minimum n to subdivied problem.
min_seg	positive integer, minimum segment size.
max_k	maximum segments to divide into.

**Value**

a vtreat coder

**Examples**

```
coder <- piecewise_constant_coder(min_seg = 1)
coder("x", 1:8, c(-1, -1, -1, -1, 1, 1, 1, 1))
```

---

piecewise\_linear      *Piecewise linear fit.*

---

**Description**

vtreat custom coder based on RcppDynProg::solve\_for\_partition().

**Usage**

```
piecewise_linear(varName, x, y, w = NULL)
```

**Arguments**

varName	character, name of variable to work on.
x	numeric, input values.
y	numeric, values to estimate.
w	numeric, weights.

**Examples**

```
piecewise_linear("x", 1:8, c(1, 2, 3, 4, 4, 3, 2, 1))
```

---

`piecewise_linear_coder`*Piecewise linear fit coder factory.*

---

**Description**

Build a piecewise linear fit coder with some parameters bound in.

**Usage**

```
piecewise_linear_coder(  
  penalty = 1,  
  min_n_to_chunk = 1000,  
  min_seg = 10,  
  max_k = 1000  
)
```

**Arguments**

<code>penalty</code>	per-segment cost penalty.
<code>min_n_to_chunk</code>	minimum n to subdivied problem.
<code>min_seg</code>	positive integer, minimum segment size.
<code>max_k</code>	maximum segments to divide into.

**Value**

a vtreat coder

**Examples**

```
coder <- piecewise_linear_coder(min_seg = 1)  
coder("x", 1:8, c(1, 2, 3, 4, 4, 3, 2, 1))
```

---

`RcppDynProg`*RcppDynProg*

---

**Description**

Rcpp dynamic programming solutions for partitioning and machine learning problems. Includes out of sample fitting applications. Also supplies additional custom coders for the vtreat package. Please see <https://github.com/WinVector/RcppDynProg> for details.

**Author(s)**

John Mount

---

score_solution	<i>compute the price of a partition solution (and check is valid).</i>
----------------	--

---

**Description**

compute the price of a partition solution (and check is valid).

**Usage**

```
score_solution(x, solution)
```

**Arguments**

x	NumericMatix, for $j \geq i$ $x(i,j)$ is the cost of partition element $[i, \dots, j]$ (inclusive).
solution	vector of indices

**Value**

price

**Examples**

```
x <- matrix(c(1,1,5,1,1,0,5,0,1), nrow=3)
s <- c(1, 2, 4)
score_solution(x, s)
```

---

solve_for_partition	<i>Solve for a piecewise linear partiton.</i>
---------------------	---

---

**Description**

Solve for a good set of right-exclusive x-cuts such that the overall graph of  $y \sim x$  is well-approximated by a piecewise linear function. Solution is a ready for use with `base::findInterval()` and `stats::approx()` (demonstrated in the examples).

**Usage**

```
solve_for_partition(
  x,
  y,
  ...,
  w = NULL,
  penalty = 0,
  min_n_to_chunk = 1000,
```



```

    min_seg = 1,
    max_k = length(x)
  )

```

### Arguments

x	numeric, input variable (no NAs).
y	numeric, result variable (no NAs, same length as x).
...	not used, force later arguments by name.
w	numeric, weights (no NAs, positive, same length as x).
penalty	per-segment cost penalty.
min_n_to_chunk	minimum n to subdivied problem.
min_seg	positive integer, minimum segment size.
max_k	maximum segments to divide into.

### Value

a data frame appropriate for stats::approx().

### Examples

```

# example data
d <- data.frame(
  x = 1:8,
  y = c(1, 2, 3, 4, 4, 3, 2, 1))

# solve for break points
soln <- solve_for_partition(d$x, d$y)
# show solution
print(soln)

# label each point
d$group <- base::findInterval(
  d$x,
  soln$x[soln$what=='left'])
# apply piecewise approximation
d$estimate <- stats::approx(
  soln$x,
  soln$pred,
  xout = d$x,
  method = 'linear',
  rule = 2)$y
# show result
print(d)

```

---

solve\_for\_partitionc *Solve for a piecewise constant partition.*

---

### Description

Solve for a good set of right-exclusive x-cuts such that the overall graph of  $y \sim x$  is well-approximated by a piecewise linear function. Solution is a ready for use with `base::findInterval()` and `stats::approx()` (demonstrated in the examples).

### Usage

```
solve_for_partitionc(
  x,
  y,
  ...,
  w = NULL,
  penalty = 0,
  min_n_to_chunk = 1000,
  min_seg = 1,
  max_k = length(x)
)
```

### Arguments

x	numeric, input variable (no NAs).
y	numeric, result variable (no NAs, same length as x).
...	not used, force later arguments by name.
w	numeric, weights (no NAs, positive, same length as x).
penalty	per-segment cost penalty.
min_n_to_chunk	minimum n to subdivided problem.
min_seg	positive integer, minimum segment size.
max_k	maximum segments to divide into.

### Value

a data frame appropriate for `stats::approx()`.

### Examples

```
# example data
d <- data.frame(
  x = 1:8,
  y = c(-1, -1, -1, -1, 1, 1, 1, 1))

# solve for break points
```

```

soln <- solve_for_partition(c(d$x, d$y)
# show solution
print(soln)

# label each point
d$group <- base::findInterval(
  d$x,
  soln$x[soln$what=='left'])
# apply piecewise approximation
d$estimate <- stats::approx(
  soln$x,
  soln$pred,
  xout = d$x,
  method = 'constant',
  rule = 2)$y
# show result
print(d)

```

---

```
solve_interval_partition
```

*solve\_interval\_partition interval partition problem.*

---

### Description

Solve a for a minimal cost partition of the integers  $[1, \dots, \text{nrow}(x)]$  problem where for  $j \geq i$   $x(i,j)$  is the cost of choosing the partition element  $[i, \dots, j]$ . Returned solution is an ordered vector  $v$  of length  $k \leq k_{\max}$  where:  $v[1] = 1$ ,  $v[k] = \text{nrow}(x) + 1$ , and the partition is of the form  $[v[i], v[i+1])$  (intervals open on the right).

### Usage

```
solve_interval_partition(x, kmax)
```

### Arguments

<code>x</code>	square NumericMatrix, for $j \geq i$ $x(i,j)$ is the cost of partition element $[i, \dots, j]$ (inclusive).
<code>kmax</code>	int, maximum number of segments in solution.

### Value

dynamic program solution.

### Examples

```

costs <- matrix(c(1.5, NA ,NA ,1 ,0 , NA, 5, -1, 1), nrow = 3)
solve_interval_partition(costs, nrow(costs))

```

---

solve\_interval\_partition\_k

*solve\_interval\_partition interval partition problem with a bound on number of steps.*

---

### Description

Solve a for a minimal cost partition of the integers  $[1, \dots, \text{nrow}(x)]$  problem where for  $j \geq i$   $x(i,j)$  is the cost of choosing the partition element  $[i, \dots, j]$ . Returned solution is an ordered vector  $v$  of length  $k \leq kmax$  where:  $v[1] = 1$ ,  $v[k] = \text{nrow}(x) + 1$ , and the partition is of the form  $[v[i], v[i+1])$  (intervals open on the right).

### Usage

```
solve_interval_partition_k(x, kmax)
```

### Arguments

<code>x</code>	square NumericMatix, for $j \geq i$ $x(i,j)$ is the cost of partition element $[i, \dots, j]$ (inclusive).
<code>kmax</code>	int, maximum number of segments in solution.

### Value

dynamic program solution.

### Examples

```
costs <- matrix(c(1.5, NA ,NA ,1 ,0 , NA, 5, -1, 1), nrow = 3)
solve_interval_partition(costs, nrow(costs))
```

---

solve\_interval\_partition\_no\_k

*solve\_interval\_partition interval partition problem, no bound on the number of steps.*

---

### Description

Not working yet.

### Usage

```
solve_interval_partition_no_k(x)
```

**Arguments**

*x* square NumericMatrix, for  $j \geq i$   $x(i,j)$  is the cost of partition element  $[i, \dots, j]$  (inclusive).

**Details**

Solve a for a minimal cost partition of the integers  $[1, \dots, \text{nrow}(x)]$  problem where for  $j \geq i$   $x(i,j)$  is the cost of choosing the partition element  $[i, \dots, j]$ . Returned solution is an ordered vector  $v$  of length  $k$  where:  $v[1] = 1$ ,  $v[k] = \text{nrow}(x) + 1$ , and the partition is of the form  $[v[i], v[i+1])$  (intervals open on the right).

**Value**

dynamic program solution.

**Examples**

```
costs <- matrix(c(1.5, NA, NA, 1, 0, NA, 5, -1, 1), nrow = 3)
solve_interval_partition(costs, nrow(costs))
```

# Index

`const_costs`, [2](#)  
`const_costs_logistic`, [3](#)

`lin_costs`, [3](#)  
`lin_costs_logistic`, [4](#)

`piecewise_constant`, [5](#)  
`piecewise_constant_coder`, [5](#)  
`piecewise_linear`, [6](#)  
`piecewise_linear_coder`, [7](#)

`RcppDynProg`, [7](#)

`score_solution`, [8](#)  
`solve_for_partition`, [8](#)  
`solve_for_partitionc`, [10](#)  
`solve_interval_partition`, [11](#)  
`solve_interval_partition_k`, [12](#)  
`solve_interval_partition_no_k`, [12](#)