# Package ‘Rspc’

July 30, 2018

**Type** Package

**Title** Nelson Rules for Control Charts

**Version** 1.2.2

**Maintainer** Stanislav Matousek (MSD) <rspc@merck.com>

**Description** Implementation of Nelson rules for control charts in 'R'. The 'Rspc' implements some Statistical Process Control methods, namely Levey-Jennings type of I (individuals) chart, Shewhart C (count) chart and Nelson rules (as described in Montgomery, D. C. (2013) Introduction to statistical quality control. Hoboken, NJ: Wiley.). Typical workflow is taking the time series, specify the control limits, and list of Nelson rules you want to evaluate. There are several options how to modify the rules (one sided limits, numerical parameters of rules, etc.). Package is also capable of calculating the control limits from the data (so far only for i-chart and c-chart are implemented).

**BugReports** https://github.com/Merck/SPC_Package/issues

**Depends** R (>= 3.1.0)

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.0.1

**VignetteBuilder** knitr

**Suggests** knitr

**NeedsCompilation** no

**Author** Martin Vagenknecht (MSD) [aut],
Jindrich Soukup (MSD) [aut],
Stanislav Matousek (MSD) [aut, cre],
Janet Alvarado (MSD) [ctb, rev],
Merck Sharp & Dohme Corp. a subsidiary of Merck & Co., Inc.,
Kenilworth, NJ, USA [cph]

**Repository** CRAN

**Date/Publication** 2018-07-30 16:20:06 UTC

# R topics documented:



---

`CalculateLimits` *CalculateLimits*

---

### Description

Evaluates whether to use custom limits or calculate them from the data.

### Usage

```
CalculateLimits(x, lcl = NA, cl = NA, ucl = NA, type = "i",
  controlLimitDistance = 3)
```

### Arguments

| | |
|---|---|
| `x` | Numerical vector |
| `lcl` | Lower control limit, single value or NA |
| `cl` | Central line, single value or NA |
| `ucl` | Upper control limit, single value or NA |
| `type` | Type of control chart, either "i" for i-chart (default) or "c" for c-chart |
| `controlLimitDistance` | Multiple of st.dev to be used to calculate limits, possible values: 1, 2, 3 (default); this parameter affect the interpretation of lcl and ucl parameters |

### Details

If at least two limits were provided, the missing ones are calculated from the them. If one or zero limits were provided the rest is computed from data.

### Value

Named list with limits

### Examples

```
CalculateLimits(x = rnorm(10), lcl = NA, cl = 100, ucl = NA, type = 'i')
```

---

## CalculateZoneBorders *CalculateZoneBorders*

---

### Description

Some Nelson rules uses so-called zones. This function calculates the borders of the zones for given limits.

### Usage

```
CalculateZoneBorders(limits, controlLimitDistance = 3)
```

### Arguments

`limits` List of limits provided by `CalculateLimits`

`controlLimitDistance`
Multiple of st.dev to be used to calculate limits, possible values: 1, 2, 3 (default); this parameter affect the interpretation of lcl and ucl parameters

### Value

Vector of zones

### Examples

```
limits = CalculateLimits(x = rnorm(10), lcl = NA, cl = 100, ucl = NA, type = 'i')
CalculateZoneBorders(limits)
#limits is object created by CalculateLimits() function
```

EvaluateRules *EvaluateRules*

## Description

Evaluates the selected Nelson rules for a given numerical vector.

## Usage

```
EvaluateRules(x, type = "i", whichRules = 1:8, lcl = NA, cl = NA,
  ucl = NA, controlLimitDistance = 3, returnAllSelectedRules = F,
  parRules = NULL)
```

## Arguments

| | |
|---|---|
| `x` | Series to be evaluated, numerical vector |
| `type` | Type of control chart, either "i" for i-chart (default) or "c" for c-chart |
| `whichRules` | Selection of Nelson rules beeing evaluated, vector with numbers from 1 to 8 |
| `lcl` | Lower control limit, single numeric value (expected as mean - controlLimitDistance * sigma), if missing the function calculates it from data |
| `cl` | Central line, single numeric value (expected as mean), if missing the function calculates it from data |
| `ucl` | Upper control limit, single numeric value (expected as mean + controlLimitDistance * sigma), if missing the function calculates it from data |
| `controlLimitDistance` | Multiple of st.dev to be used to calculate limits, possible values: 1, 2, 3 (default); this parameter affect the interpretation of lcl and ucl parameters |
| `returnAllSelectedRules` | Resulting dataframe will contain all selected rules, either True or False, if missing only valid rules returned |
| `parRules` | Optional parameters for specific rules, for details see `SetParameters` |

## Details

# Only Rules 1-4 relevant for c-chart.
# Check for non negative data for c-chart.
# For controlLimitDistance less than or equal to 2 disable rule 5.
# For controlLimitDistance less than or equal to 1 disable rule 5,6,8.
# For returnAllSelectedRules=TRUE columns of invalid rules for given evaluation are filled with NAs.

## Value

Dataframe containing original vector and rules evaluation

## Examples

```
# Evaluate data, use all 8 Nelson rules, limits are specified by user
EvaluateRules(x = rnorm(10), whichRules = 1:8, lcl = 0, cl = 50, ucl = 100)
#Evaluate only rule 1, 3, 5, calculate limits from data using c-chart formula,
#use 2 sigma instead of 3, modify default behaviour of rule by pars variable
#created by function SetParameters()
pars = SetParameters()
EvaluateRules(x = rpois(10, lambda = 15), type = 'c', whichRules = c(1,3,5), lcl = NA, cl = NA,
ucl = NA, controlLimitDistance = 2, parRules = pars)
# pars is object of optional parameters created by SetParameters() function
```

---

`NelsonRules` *NelsonRules*

---

## Description

Auxiliary function to calling individual Rule functions.

## Usage

```
NelsonRules(ruleN, data, zoneB, limits, parRules = NULL, ...)
```

## Arguments

| | |
|---|---|
| `ruleN` | Name of individual Rule function "Rule1" to "Rule8" |
| `data` | Data to be checked, numerical vector |
| `zoneB` | Vector of zones created by `CalculateLimits` |
| `limits` | List of limit created by `CalculateLimits` |
| `parRules` | List of optional parameters for this particular rule |
| `...` | unspecified arguments of a function |

## Details

Handling the missing values:

Missing values are represented by the symbol NA - not available.

Rule 1: NAs do not violate this rule.

Rule 2-8: NAs are ignored, they do not break Rule evaluation. NA values are removed from the vector, the rule function is calculated and then the NAs are returned back to it's original position in the vector.

## Value

Result of individual Rule function with predefined parameters

`Rule1` *Rule 1*

## Description

One point beyond the control limits

## Usage

```
Rule1(x, lcl, ucl, sides, ...)
```

## Arguments

| | |
|---|---|
| `x` | Numerical vector |
| `lcl` | Lower control limit, single number |
| `ucl` | Upper control limit, single number |
| `sides` | Monitored side of the process: either "two-sided" (default), "upper" or "lower" |
| `...` | unspecified arguments of a function |

## Details

0 means: ok
1 means: violation

inequality used during evaluation

parametr sides is internally encoded as: 1 for "two-sided", 2 for "upper", 3 for "lower"

## Value

Vector of the same length as x

## Examples

```
Rule1(x = rnorm(10), lcl = 10, ucl = 100, sides = "two-sided")
```

`Rule2` *Rule 2*

## Description

Nine points in a row are on one side of the central line.

## Usage

```
Rule2(x, cl, nPoints = 9, ...)
```

## Arguments

| | |
|---|---|
| `x` | Numerical vector |
| `cl` | central line, single number |
| `nPoints` | Sequence of consequtive points to be evaluated |
| `...` | unspecified arguments of a function |

## Details

0 means: ok
1 means: violation

inequality used during evaluation

## Value

Vector of the same length as x

## Examples

```
Rule2(x = rnorm(20), cl=0, nPoints = 9)
```

`Rule3` *Rule 3*

## Description

Six points in a row steadily increasing or decreasing.

## Usage

```
Rule3(x, nPoints = 6, convention = 1, equalBreaksSeries = 1, ...)
```

### Arguments

| | |
|---|---|
| `x` | Numerical vector |
| `nPoints` | Sequence of consequtive points to be evaluated |
| `convention` | Calculation according to 'minitab' or 'jmp' (see details) |
| `equalBreaksSeries` | Equal values break consequtive series of points |
| `...` | unspecified arguments of a function |

### Details

0 means: ok
1 means: violation

parameter equalBreakSeries is internally encoded as: 1 for TRUE and 2 for FALSE

parameter convention is internally encoded as: 1 for 'minitab' and 2 for 'jmp'

Difference in convention parameter is as follows:
'minitab' - seven points in a row steadily increasing or decreasing
'jmp' - six points in a row steadily increasing or decreasing

### Value

Vector of the same length as x

### Examples

```
Rule3(x = rnorm(20), nPoints = 6, convention = 1, equalBreaksSeries = 1)
```

---

## Rule4 *Rule 4*

---

### Description

Fourteen or more points in a row alternate in direction, increasing then decreasing.

### Usage

```
Rule4(x, nPoints = 14, convention = 1, ...)
```

### Arguments

| | |
|---|---|
| `x` | Numerical vector |
| `nPoints` | Sequence of consequtive points to be evaluated |
| `convention` | Calculation according to 'minitab' or 'jmp' (see details) |
| `...` | unspecified arguments of a function |

### Details

0 means: ok
1 means: violation

parameter convention is internally encoded as: 1 for 'minitab' and 2 for 'jmp'

Difference in convention parameter is as follows:
'minitab' - 15 or more points (14 changes of direction) in a row alternate in direction, increasing then decreasing
'jmp' - 14 or more points (13 changes of direction) in a row alternate in direction, increasing then decreasing

### Value

Vector of the same length as x

### Examples

```
Rule4(x = rnorm(20), nPoints = 14,convention = 1)
```

---

## Rule5 *Rule 5*

---

### Description

Two out of three consecutive points beyond the 2*sigma limits on same side of center line.

### Usage

```
Rule5(x, zoneB, minNPoints = 2, nPoints = 3, ...)
```

### Arguments

| | |
|---|---|
| `x` | Numerical vector |
| `zoneB` | Vector of zone borders |
| `minNPoints` | Minimal number of points in a sequence violating a rule |
| `nPoints` | Sequence of consequtive points to be evaluated |
| `...` | unspecified arguments of a function |

### Details

0 means: ok
1 means: violation

inequality used during evaluation
Rule is violated also if the first two points are beyond the 2*sigma limits During calculation of EvaluateRules function wiht controlLimitDistance <= 2, the evaluation of this rule is suppressed

## Value

Vector of the same length as x

## Examples

```
limits = CalculateLimits(x = rnorm(10), lcl = NA, cl = 100, ucl = NA, type = 'i')
zones = CalculateZoneBorders(limits)
Rule5(x = rnorm(20), zoneB = zones, minNPoints = 2, nPoints = 3)
#zones is object created by function CalculateZoneBorders()
```

---

Rule6 *Rule 6*

---

## Description

Four or five out of five points in a row are more than 1 standard deviation from the mean in the same direction.

## Usage

```
Rule6(x, zoneB, minNPoints = 4, nPoints = 5, ...)
```

## Arguments

| | |
|---|---|
| x | Numerical vector |
| zoneB | Vector of zone borders |
| minNPoints | Minimal number of points in a sequence violating a rule |
| nPoints | Sequence of consequtive points to be evaluated |
| ... | unspecified arguments of a function |

## Details

0 means: ok
1 means: violation

inequality used during evaluation Rule is violated also if the first four points are beyond the 1 standard deviation from the mean During calculation of EvaluateRules function wiht controlLimitDistance <= 1, the evaluation of this rule is suppressed

## Value

Vector of the same length as x

## Examples

```
limits = CalculateLimits(x = rnorm(10), lcl = NA, cl = 100, ucl = NA, type = 'i')
zones = CalculateZoneBorders(limits)
Rule6(x = rnorm(20), zoneB = zones, minNPoints = 4, nPoints = 5)
#zones is object created by function CalculateZoneBorders()
```

---

`Rule7` *Rule 7*

---

## Description

Fifteen points in a row are all within 1 standard deviation of the mean on either side of the mean.

## Usage

```
Rule7(x, nPoints = 15, zoneB, ...)
```

## Arguments

| | |
|---|---|
| `x` | Numerical vector |
| `nPoints` | Sequence of consequtive points to be evaluated |
| `zoneB` | Vector of zone borders |
| `...` | unspecified arguments of a function |

## Details

0 means: ok
1 means: violation

equality used during evaluation

## Value

Vector of the same length as x

## Examples

```
limits = CalculateLimits(x = rnorm(10), lcl = NA, cl = 100, ucl = NA, type = 'i')
zones = CalculateZoneBorders(limits)
Rule7(x = rnorm(20), zoneB = zones, nPoints = 15)
#zones is object created by function CalculateZoneBorders()
```

Rule8 *Rule 8*

### Description

Eight points in a row outside 1 standard deviation of the mean in both directions.

### Usage

```
Rule8(x, nPoints = 8, zoneB, ...)
```

### Arguments

| | |
|---|---|
| `x` | Numerical vector |
| `nPoints` | Sequence of consequtive points to be evaluated |
| `zoneB` | Vector of zone borders |
| `...` | unspecified arguments of a function |

### Details

0 means: ok
1 means: violation

inequality used during evaluation During calculation of EvaluateRules function wiht controlLimitDistance <= 1, the evaluation of this rule is suppressed

### Value

Vector of the same length as x

### Examples

```
limits = CalculateLimits(x = rnorm(10), lcl = NA, cl = 100, ucl = NA, type = 'i')
zones = CalculateZoneBorders(limits)
Rule8(x = rnorm(20), zoneB = zones, nPoints = 8)
#zones is object created by function CalculateZoneBorders()
```

SetParameters *SetParameters*

## Description

Creates optional parameters with default settings.

## Usage

```
SetParameters()
```

## Details

The function is called without any parameter. If you want to modify any or the rules' setting, modify the result of this function and plug it to `EvaluateRules`'s parRules parameter.

## Value

List of optional parameters

## Examples

```
pars <- SetParameters()
pars$Rule1$sides <- "upper"
#function doos not need any input parameters
```

# Index