Package ‘pcts’

January 12, 2022

Type Package

Title Periodically Correlated and Periodically Integrated Time Series


Version 0.15.2

Date 2022-01-12

Author Georgi N. Boshnakov

Maintainer Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>

Depends R (>= 3.5.0)

Imports methods, sarima, Matrix, BB, PolynomF (>= 2.0-2), gbutils, zoo, xts, stats4, lagged (>= 0.2.2), mcompanion, Rdpack (>= 0.9), lubridate

Suggests testthat, fUnitRoots, partsm, knitr, rmarkdown

RdMacros Rdpack

LazyData yes

URL https://geobosh.github.io/pcts/ (website)

https://github.com/GeoBosh/pcts/ (devel)

BugReports https://github.com/GeoBosh/pcts/issues

License GPL (>= 2)

R topics documented:

RoxygenNote 7.1.1
VignetteBuilder knitr
NeedsCompilation no
Repository CRAN
Date/Publication 2022-01-12 13:22:42 UTC

R topics documented:

  pcts-package ......................................................... 4
  allSeasons ..........................................................  8
  as_date-methods ...................................................... 11
  as_datetime-methods .................................................. 11
  autocorrelations-methods .......................................... 12
  autocovariances-methods ........................................... 12
  availStart ........................................................... 13
  BareCycle-class ...................................................... 15
  BasicCycle-class ..................................................... 17
  BuiltinCycle-class ................................................... 18
  Cyclic ................................................................. 21
  Cyclic-class .......................................................... 23
  dataFranses1996 ...................................................... 23
  date<-methods ........................................................ 26
  ex1f ................................................................. 26
  filterCoef-methods ................................................... 27
  fitPM ................................................................. 28
  FittedPeriodicArmaModel-class ..................................... 31
  FittedPeriodicArModel-class ........................................ 32
  fit_trigPAR_optim .................................................... 33
  four_stocks_since2016_01_01 ........................................ 34
  Fraser2017 ........................................................... 35
  maxLag-methods ........................................................ 36
  mC.ss ................................................................. 36
  meanvarcheck .......................................................... 39
  modelCycle ............................................................ 40
  ModelCycleSpec-class ................................................ 41
  nCycles ............................................................... 41
  nSeasons-methods .................................................... 42
  num2pcpar ............................................................. 43
  parcovmatlist ........................................................ 45
  PartialCycle-class ................................................... 46
  PartialPeriodicAutocorrelations-class ........................... 47
  pc.filter ............................................................. 48
  pc.filter.xarma ....................................................... 50
  pc.hat.h ............................................................... 52
  pcacfMat .............................................................. 53
  pcacf_pwn_var ........................................................ 54
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pcalg1</td>
<td>55</td>
</tr>
<tr>
<td>pcalg1util</td>
<td>57</td>
</tr>
<tr>
<td>pcApply-methods</td>
<td>58</td>
</tr>
<tr>
<td>pcAr.ss</td>
<td>60</td>
</tr>
<tr>
<td>pcAR2acf</td>
<td>61</td>
</tr>
<tr>
<td>pcarma_acvf2model</td>
<td>62</td>
</tr>
<tr>
<td>pcarma_solve</td>
<td>63</td>
</tr>
<tr>
<td>pcarma_unvec</td>
<td>66</td>
</tr>
<tr>
<td>pcCycle-methods</td>
<td>68</td>
</tr>
<tr>
<td>pclsdf</td>
<td>71</td>
</tr>
<tr>
<td>pclspiar</td>
<td>73</td>
</tr>
<tr>
<td>pcMean-methods</td>
<td>74</td>
</tr>
<tr>
<td>pcPlot</td>
<td>76</td>
</tr>
<tr>
<td>pcTest-methods</td>
<td>77</td>
</tr>
<tr>
<td>Petime</td>
<td>78</td>
</tr>
<tr>
<td>pcts</td>
<td>81</td>
</tr>
<tr>
<td>pcts-deprecated</td>
<td>83</td>
</tr>
<tr>
<td>pcts_exdata</td>
<td>83</td>
</tr>
<tr>
<td>pc_sdfactor</td>
<td>84</td>
</tr>
<tr>
<td>pdSafeParOrder</td>
<td>85</td>
</tr>
<tr>
<td>PeriodicArmaFilter-class</td>
<td>86</td>
</tr>
<tr>
<td>PeriodicArmaModel-class</td>
<td>86</td>
</tr>
<tr>
<td>PeriodicArmaSpec-class</td>
<td>87</td>
</tr>
<tr>
<td>PeriodicArModel-class</td>
<td>88</td>
</tr>
<tr>
<td>PeriodicArModel-methods</td>
<td>88</td>
</tr>
<tr>
<td>PeriodicAutocorrelations-class</td>
<td>89</td>
</tr>
<tr>
<td>PeriodicAutocovarianceModel-class</td>
<td>90</td>
</tr>
<tr>
<td>PeriodicAutocovariances-class</td>
<td>91</td>
</tr>
<tr>
<td>PeriodicAutocovariances-class</td>
<td>92</td>
</tr>
<tr>
<td>PeriodicBIFilter-class</td>
<td>92</td>
</tr>
<tr>
<td>PeriodicFilterModel-class</td>
<td>94</td>
</tr>
<tr>
<td>PeriodicIntegratedArmaSpec-class</td>
<td>94</td>
</tr>
<tr>
<td>PeriodicInterceptSpec-class</td>
<td>95</td>
</tr>
<tr>
<td>PeriodicMaModel-class</td>
<td>96</td>
</tr>
<tr>
<td>PeriodicMTS-class</td>
<td>96</td>
</tr>
<tr>
<td>PeriodicMTS_ts-class</td>
<td>98</td>
</tr>
<tr>
<td>PeriodicMTS_zooreg-class</td>
<td>99</td>
</tr>
<tr>
<td>PeriodicSPFilter-class</td>
<td>99</td>
</tr>
<tr>
<td>PeriodicTimeSeries-class</td>
<td>100</td>
</tr>
<tr>
<td>PeriodicTS-class</td>
<td>101</td>
</tr>
<tr>
<td>PeriodicTS_ts-class</td>
<td>102</td>
</tr>
<tr>
<td>PeriodicTS_zooreg-class</td>
<td>102</td>
</tr>
<tr>
<td>PeriodicVector-class</td>
<td>103</td>
</tr>
<tr>
<td>periodic_acf1_test</td>
<td>105</td>
</tr>
<tr>
<td>permean2intercept</td>
<td>106</td>
</tr>
<tr>
<td>permodelmf</td>
<td>107</td>
</tr>
<tr>
<td>pilar2par</td>
<td>108</td>
</tr>
<tr>
<td>PiPeriodicArmaModel-class</td>
<td>109</td>
</tr>
</tbody>
</table>
### Description


---

**Index**

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PiPeriodicArModel-class</td>
<td>110</td>
</tr>
<tr>
<td>PiPeriodicMaModel-class</td>
<td>111</td>
</tr>
<tr>
<td>pwn_McLeodLjungBox_test</td>
<td>111</td>
</tr>
<tr>
<td>SamplePeriodicAutocorrelations-class</td>
<td>113</td>
</tr>
<tr>
<td>SamplePeriodicAutocovariances-class</td>
<td>113</td>
</tr>
<tr>
<td>seqSeasons-methods</td>
<td>114</td>
</tr>
<tr>
<td>sigmaSq-methods</td>
<td>115</td>
</tr>
<tr>
<td>SimpleCycle-class</td>
<td>115</td>
</tr>
<tr>
<td>sim_parAcf</td>
<td>116</td>
</tr>
<tr>
<td>sim_parCoef</td>
<td>117</td>
</tr>
<tr>
<td>sim_pc</td>
<td>119</td>
</tr>
<tr>
<td>sim_pwn</td>
<td>121</td>
</tr>
<tr>
<td>SiPeriodicArmaModel-class</td>
<td>123</td>
</tr>
<tr>
<td>SiPeriodicArModel-class</td>
<td>123</td>
</tr>
<tr>
<td>SiPeriodicMaModel-class</td>
<td>124</td>
</tr>
<tr>
<td>sl_utils</td>
<td>125</td>
</tr>
<tr>
<td>SubsetPM-class</td>
<td>127</td>
</tr>
<tr>
<td>test_piar</td>
<td>128</td>
</tr>
<tr>
<td>unitCycle-methods</td>
<td>130</td>
</tr>
<tr>
<td>unitCycle&lt;-methods</td>
<td>130</td>
</tr>
<tr>
<td>unitSeason-methods</td>
<td>131</td>
</tr>
<tr>
<td>unitSeason&lt;-methods</td>
<td>131</td>
</tr>
<tr>
<td>Vec</td>
<td>132</td>
</tr>
<tr>
<td>window</td>
<td>134</td>
</tr>
<tr>
<td>zoo-class</td>
<td>135</td>
</tr>
<tr>
<td>zooreg-class</td>
<td>136</td>
</tr>
<tr>
<td>[-]_methods</td>
<td>137</td>
</tr>
<tr>
<td>[&lt;-]_methods</td>
<td>138</td>
</tr>
<tr>
<td>[$]_methods</td>
<td>138</td>
</tr>
</tbody>
</table>

**pcts-package** *(Periodically Correlated and Periodically Integrated Time Series)*
Details

The underlying assumption is that the observations are made at regular intervals, such as quarter, month, week, day — or represent data for such intervals — and these intervals are nested into larger periods. In pcts we call the larger period a cycle and its parts seasons. Typical examples of season-cycle timing are months in a year, quarters in a year, days in a week (or business week). The number of seasons in a cycle is called frequency in class "ts" in base R.

Cycles in pcts keep not only the number of seasons (frequency) but other information, such as the names of the seasons and units of seasons. In pcts there are a number of builtin cycle classes for typical cases, as well as provision for creation of custom cycles on the fly. See pcCycle and BuiltinCycle for ways to create cycle objects, and allSeasons for further examples.

Periodic time series can be created with pcts, which accepts as input vectors, matrices and time series objects from base R and some other packages, including zoo and xts. When importing data, the time information is taken from the data and an attempt is made to guess the periodicity from the frequency (for time series objects that have it set) and an analysis of the datetime stamps, if present. pcts also has arguments for specifying the number of seasons or the cycle, as well as the start datetime.

The main periodic time series classes in pcts are PeriodicTS and PeriodicMTS, for univariate and multivariate time series, respectively. Standard time series functions can be used with them directly, see for example window and start.

The naming conventions are as follows. Names of classes generally consists of one or more words. The first letter of each word, is capitalised. Only the first letter of abbreviations for models, such as ARMA, is capitalised. Similarly for generic functions but for them the first word is not capitalised. In a few names PM stands for 'periodic model' and TS for 'time series'.

Significant portion of the code was written in 2005–2007. Many of the functions and classes have been renamed under the above conventions and most of those that are not are not exported but a few still are and they should be considered subject to change.

autocovariances and autocorrelations are one-stop generic functions for computation of these second order properties. What to compute is deduced from the type of the object. For models they compute theoretical quantities — periodic or non-periodic, scalar or multivariate. For time series they compute the corresponding sample counterparts.

Author(s)

Georgi N. Boshnakov

Maintainer: Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>

References


**See Also**

*mcompanion*

**Examples**

data(dataFranses1996)
class(dataFranses1996) # [1] "mts" "ts" "matrix"

pcfr <- pcts(dataFranses1996)
class(pcfr) # "PeriodicMTS"
nSeasons(pcfr) # 4
allSeasons(pcfr)
allSeasons(pcfr, abb = TRUE)

## subsetting
## one index, x[i], is analogous to lists
pcfr2to4 <- pcfr[2:4]; class(pcfr2to4) # "PeriodicMTS"
p CFR2toZ <- pcfr[2]; class(pcfr2toZ) # "PeriodicMTS"
pcfr2 <- pcfr[[2]]; class(pcfr2) # note '[[2]], "PeriodicTS"

## data for 1990 quarter 3
pcfr2to4[as_date("1990-07-01")]] # note: not "1990-03-01"!
pct1990_Q3 <- Pctime(c(1990, 3), pcCycle(pcfr2to4))
p CFR2to4[pct1990_Q3]

## with empty index, returns the underlying data
dim(pcfr[]) # [1] 148 19
dim(pcfr2toZ[]) # 148 1
length(pcfr2[]) # 148 (this is numeric)
allSeasons

Get names of seasons

Description

Functions and methods to get names of seasons and related quantities for objects from the cycle, periodic time series classes and other objects for which the concepts are defined.

Usage

unitSeason(x)
unitCycle(x)
seqSeasons(x)
allSeasons(x, abb = FALSE, prefix = "S", ...)

unitSeason ( x, ... ) <- value
allSeasons

unitCycle (x, ...) <- value
allSeasons (x, abb, ...) <- value

## S4 replacement method for signature 'SimpleCycle'
allSeasons(x, abb, prefix, ...) <- value

## S4 replacement method for signature 'Cyclic'
allSeasons(x, abb = FALSE, ...) <- value

Arguments

- **x**
  - a cycle, time series or other object for which the concept of seasons is defined.
- **abb**
  - if TRUE give the abbreviated names of the seasons.
- **prefix**
  - use this prefix for automatically generated names of seasons.
- **...**
  - further arguments for methods.
- **value**
  - a character string

Details

The cycle classes, i.e. classes inheriting from class `BasicCycle`, provide common functionality. In particular, they guarantee that the functions described in this topic are available. These functions work also for the periodic time series classes and may be defined for other classes where they make sense.

Methods

Methods for `allSeasons()`:

- signature(x = "BasicCycle", abb = "ANY")
- signature(x = "DayWeekCycle", abb = "logical")
- signature(x = "DayWeekCycle", abb = "missing")
- signature(x = "FiveDayWeekCycle", abb = "logical")
- signature(x = "FiveDayWeekCycle", abb = "missing")
- signature(x = "MonthYearCycle", abb = "logical")
- signature(x = "MonthYearCycle", abb = "missing")
- signature(x = "OpenCloseCycle", abb = "logical")
- signature(x = "OpenCloseCycle", abb = "missing")
- signature(x = "QuarterYearCycle", abb = "logical")
- signature(x = "QuarterYearCycle", abb = "missing")
- signature(x = "SimpleCycle", abb = "ANY")
- signature(x = "Cyclic", abb = "ANY")
- signature(x = "Every30MinutesCycle", abb = "logical")
- signature(x = "Every30MinutesCycle", abb = "missing")
- signature(x = "VirtualPeriodicModel", abb = "ANY")
Examples

```r
opcycle <- new("OpenCloseCycle")
## convert to SimpleCycle to change some names
siopcycle <- as(opcycle, "SimpleCycle")
## siopcycle inherits names from opcycle
unitSeason(siopcycle) # "Season"
unitCycle(siopcycle) # "Cycle"
allSeasons(siopcycle) # "Open" "Close"
allSeasons(siopcycle, abb = TRUE) # "O" "C"

allSeasons(siopcycle) <- c("Day", "Night")
allSeasons(siopcycle) # now: "Day" "Night"
## change also abbreviations
allSeasons(siopcycle, abb = TRUE) <- c("D", "N")
allSeasons(siopcycle, abb = TRUE) # now: "D" "N"

seasons <- new("SimpleCycle", 4)
unitSeason(seasons) # "Season"
unitCycle(seasons) # "Cycle"
allSeasons(seasons)
allSeasons(seasons, abb = TRUE)

unitCycle(seasons) <- "Year"
unitSeason(seasons)
allSeasons(seasons) <- c("Winter", "Spring", "Summer", "Autumn")
allSeasons(seasons)
allSeasons(seasons, abb = TRUE) <- c("Win", "Spr", "Sum", "Aut")
allSeasons(seasons, abb = TRUE)

## change autumn to Fall
allSeasons(seasons)[4] <- "Fall"
allSeasons(seasons, abb = TRUE)[4] <- "Fall"
allSeasons(seasons)
allSeasons(seasons, abb = TRUE)

## indexing of cycle objects is equivalent to allSeasons.
seasons[]
seasons[, abb = TRUE]

seasons[4] <- "Herbst"
seasons

unitCycle(seasons) <- "Jahre"
unitSeason(seasons)
unitSeason(seasons) <- "Jahreszeit"
seasons[] <- c("Winter", "Fruehling", "Sommer", "Herbst")
seasons[, abb = TRUE] <- c("W", "F", "S", "H")
seasons[]
```
as_date-methods

seasons

as_date-methods  Replace methods for as_date in package pcts

Description

Replace methods for as_date in package pcts.

Methods

signature(x = "PeriodicTimeSeries")
signature(x = "ANY")
signature(x = "character")
signature(x = "Cyclic")
signature(x = "numeric")
signature(x = "POSIXt")

as_datetime-methods  Methods for as_datetime in package pcts

Description

Methods for as_datetime in package pcts.

Methods

signature(x = "PeriodicTimeSeries")
autocorrelations-methods

Compute autocorrelations and periodic autocorrelations

Description

Methods for computation of autocorrelations and periodic autocorrelations.

Methods

signature(x = "numeric", maxlag = "ANY", lag_0 = "missing")
signature(x = "PeriodicTimeSeries", maxlag = "ANY", lag_0 = "missing")
signature(x = "PeriodicAutocovariances", maxlag = "ANY", lag_0 = "missing")
signature(x = "SamplePeriodicAutocovariances", maxlag = "ANY", lag_0 = "missing")
signature(x = "VirtualPeriodicAutocovariances", maxlag = "ANY", lag_0 = "missing")
signature(x = "VirtualPeriodicAutocovarianceModel", maxlag = "ANY", lag_0 = "missing")

See Also

autocovariances-methods for autocovariances; autocorrelations in package "sarima" for further details.

Examples

```r
## periodic
autocorrelations(pcts(AirPassengers), maxlag = 10)

## for "ts" or "numeric" objects the default is non-periodic
## non-periodic acf
autocorrelations(AirPassengers, maxlag = 10)
autocorrelations(as.numeric(AirPassengers))
## argument 'nseasons' forces periodic acf
autocorrelations(AirPassengers, maxlag = 10, nseasons = 12)
autocorrelations(as.numeric(AirPassengers), maxlag = 10, nseasons = 12)
```

autocovariances-methods

Compute autocovariances and periodic autocovariances

Description

Methods for the generic function autocovariances(), which computes autocovariances meaningful for the first argument. For objects representing time series, it computes sample autocovariances (univariate, multivariate, periodic, as appropriate). For objects representing models, it computes the relevant theoretical autocovariances.
Methods

signature(x = "matrix", maxlag = "ANY")
signature(x = "numeric", maxlag = "ANY")
signature(x = "PeriodicArmaModel", maxlag = "ANY")
signature(x = "PeriodicArModel", maxlag = "ANY")
signature(x = "PeriodicAutocovarianceModel", maxlag = "ANY")
signature(x = "PeriodicTS", maxlag = "ANY")
signature(x = "VirtualPeriodicAutocovariances", maxlag = "ANY")

If maxlag is missing or equal to maxLag(x), x is returned unchanged. Otherwise the number of available lags is adjusted to maxlag.

See Also

autocorrelations-methods for autocorrelations; autocovariances in package "sarima" for further details.

---

**availStart**

*Time of first or last non-NA value*

Description

Time of first or last non-NA value.

Usage

```r
availStart(x, any = TRUE)
```

```r
availEnd(x, any = TRUE)
```

Arguments

- **x**: a time series or similar object
- **any**: logical flag for multivariate objects. The default TRUE requests the first/last index containing any non-NA value. FALSE requires that all values at the first/last index must be non-NA.

Details

The time is given as a cycle-season pair.

Argument any is meaningful only for multivariate objects. Its name is short for "the first/last index for which any of the values (ie at least one) is non-NA". any = FALSE is taken to mean that the index is the first/last for which all values are non-NA.

The functions can be used together with windows to trim NA’s from the beginning and/or end of the data. As an alternative we provide also methods for periodic time series methods for zoo:na.trim, see the examples below.
Value

numeric, length 2

See Also

window

Examples

tipi <- pcts(dataFranses1996[, "USTotalIPI"])
start(tipi)
end(tipi)
head(tipi)
tail(tipi)

tipi <- window(tipi, start = availStart(tipi), end = availEnd(tipi))
start(tipi)
end(tipi)
plot(tipi)

pcfr <- pcts(dataFranses1996)

pcfr2to4 <- pcfr[2:4]
head(pcfr2to4)
tail(pcfr2to4)

## time of first and last data, can be NA's
start(pcfr2to4) # 1955 Q1
dcnder(pcfr2to4) # 1991 Q4

## time of first nonNA:
availStart(pcfr[[2]]) # 1960 Q1
availStart(pcfr2to4) # 1960 Q1

## time of last nonNA:
availEnd(pcfr[[2]]) # 1991 Q4
availEnd(pcfr[[3]]) # 1987 Q4
availEnd(pcfr[[4]]) # 1990 Q4

## but at least one of them is available for 1991 Q4, so:
availEnd(pcfr2to4) # 1991 Q4

## this requests the time of the last full record:
availEnd(pcfr2to4, any = FALSE) # 1987 Q4

pcfr2to4a <- window(pcfr2to4, start = availStart(pcfr2to4), end = availEnd(pcfr2to4))
head(pcfr2to4a)
tail(pcfr2to4a, 20)

## trim NA's from both ends, up to the first last full record:
pcfr2to4b <- window(pcfr2to4, start = availStart(pcfr2to4, FALSE),
                     end = availEnd(pcfr2to4, FALSE))

## TODO: need a better example here since the first non-NA value for all
## ts in pcfr2to4 is at the same
## alternatively, use na.trim(), the default for is.na is "any"

```r
pcpres <- window(pcts(presidents), end = c(1972, 4))
```

```r
availStart(pcpres) # 1945 2
availEnd(pcpres) # 1972 2
both <- na.trim(pcpres) # same as "both"
identical(na.trim(pcpres), both) # TRUE
head(both, 7)
tail(both)
head(na.trim(pcpres, "left"), 7)
tail(na.trim(pcpres, "right"))
```

cguk <- pcfr[c("CanadaUnemployment", "GermanyGNP", "UKTotalInvestment")]

```r
availStart(cguk)
availEnd(cguk)
availStart(cguk, TRUE) # same
availEnd(cguk)
availEnd(cguk, TRUE) # same
```

```r
na.trim(cguk)
head( na.trim(cguk, sides = "left") )
tail( na.trim(cguk, sides = "right") )
```

```r
head( na.trim(cguk, sides = "left", is.na = "all") )
tail( na.trim(cguk, sides = "right", is.na = "all") )
```

### Description

Class BareCycle.

### Objects from the Class

Objects can be created by calls of the form pcCycle(nseasons) or new("BareCycle", nseasons).

Class "BareCycle" represents the number of seasons and is sufficient for many computations.

### Slots

- **nseasons**: Object of class "integer", the number of seasons.
BareCycle-class

Extends

Class "BasicCycle", directly.

Methods

coerce signature(from = "BuiltInCycle", to = "BareCycle"): ...
initialize signature(.Object = "BareCycle"): ...
nSeasons signature(object = "BareCycle"): ...
show signature(object = "BareCycle"): ...

Author(s)

Georgi N. Boshnakov

See Also

pcCycle for creation of cycle objects and extraction of cycle part of time series,

BuiltInCycle-class, SimpleCycle-class,

DayWeekCycle-class, MonthYearCycle-class, OpenCloseCycle-class, QuarterYearCycle-class
PartialCycle-class,

BasicCycle-class (virtual, for use in signatures)

Examples

pcCycle(5)
cycle <- new("BareCycle", 5)
identical(new("BareCycle", 5), pcCycle(5)) # TRUE

unitSeason(cycle)
unitCycle(cycle)
allSeasons(cycle)
seqSeasons(cycle)

cycle[]
cycle[3]

## if cycle represents 5-days week one may prefer:
BuiltInCycle(5)
BasicCycle-class

Class BasicCycle

Description

Class BasicCycle.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

[ signature(x = "BasicCycle",i = "ANY",j = "missing",drop = "ANY")
[ signature(x = "BasicCycle",i = "missing",j = "missing",drop = "ANY")
[< signature(x = "BasicCycle",i = "ANY",j = "missing",value = "ANY")
[< signature(x = "BasicCycle",i = "missing",j = "missing",value = "ANY")

allSeasons signature(x = "BasicCycle",abb = "ANY")

seqSeasons signature(x = "BasicCycle")

Author(s)

Georgi N. Boshnakov

See Also

BareCycle-class, SimpleCycle-class, BuiltinCycle-class, DayWeekCycle-class, MonthYearCycle-class, OpenCloseCycle-class, QuarterYearCycle-class, PartialCycle-class

Examples

showClass("BasicCycle")
Class "BuiltinCycle" and its subclasses in package 'pcts'

Description

Class "BuiltinCycle" and its subclasses in package 'pcts'.

Objects from the Class

Class "BuiltinCycle" is a virtual Class: no objects may be created from it. Class "BuiltinCycle" has several built-in cycle subclasses. Objects from the subclasses can be created by calls of the form `new("className", first,...)`, where "className" is the name of the subclass. The optional argument `first` can be used to designate a season to be considered first in the cycle, by default the first.

The function `BuiltinCycle` provides a more convenient way to generate objects from subclasses of class "BuiltinCycle". Its argument is the number of seasons.

These classes are effectively unmodifiable, but the user can convert them to other cycle classes, e.g. class "SimpleCycle", and adapt as needed.

The subclasses of "BuiltinCycle" have definitions for all methods promised by its superclass "BasicCycle".

Slots

The class "BuiltinCycle" and its subclasses have a single common slot:

`first`: Object of class "integer", the index of the season to be treated as the first in a cycle.

Extends

Class "BuiltinCycle" extends class "BasicCycle", directly.

Classes "DayWeekCycle", "Every30MinutesCycle", "FiveDayWeekCycle", "OpenCloseCycle" and "QuarterYearCycle" extend:

- Class "BuiltinCycle", directly.
- Class "BasicCycle", by class "BuiltinCycle", distance 2.

Methods

Functions with methods for this class:

- `coerce` signature(from = "BuiltinCycle", to = "BareCycle"): ...
- `coerce` signature(from = "BuiltinCycle", to = "SimpleCycle"): ...
- `initialize` signature(.Object = "BuiltinCycle"): ...
- `show` signature(object = "BuiltinCycle"): ...
- `allSeasons` signature(x = "DayWeekCycle", abbrev = "logical")
- `allSeasons` signature(x = "DayWeekCycle", abbrev = "missing")
- `nSeasons` signature(object = "DayWeekCycle")
unitCycle  signature(x = "DayWeekCycle")
unitSeason signature(x = "DayWeekCycle")
allSeasons signature(x = "Every30MinutesCycle", abb = "logical"): ...
allSeasons signature(x = "Every30MinutesCycle", abb = "missing"): ...
nSeasons signature(object = "Every30MinutesCycle"): ...
unitCycle signature(x = "Every30MinutesCycle"): ...
unitSeason signature(x = "Every30MinutesCycle"): ...
allSeasons signature(x = "FiveDayWeekCycle", abb = "logical"): ...
allSeasons signature(x = "FiveDayWeekCycle", abb = "missing"): ...
nSeasons signature(object = "FiveDayWeekCycle"): ...
unitCycle signature(x = "FiveDayWeekCycle"): ...
unitSeason signature(x = "FiveDayWeekCycle"): ...
allSeasons signature(x = "MonthYearCycle", abb = "logical"): ...
allSeasons signature(x = "MonthYearCycle", abb = "missing"): ...
nSeasons signature(object = "MonthYearCycle"): ...
unitCycle signature(x = "MonthYearCycle"): ...
unitSeason signature(x = "MonthYearCycle"): ...
allSeasons signature(x = "OpenCloseCycle", abb = "logical"): ...
allSeasons signature(x = "OpenCloseCycle", abb = "missing"): ...
nSeasons signature(object = "OpenCloseCycle"): ...
unitCycle signature(x = "OpenCloseCycle"): ...
unitSeason signature(x = "OpenCloseCycle"): ...
allSeasons signature(x = "QuarterYearCycle", abb = "logical"): ...
allSeasons signature(x = "QuarterYearCycle", abb = "missing"): ...
nSeasons signature(object = "QuarterYearCycle"): ...
unitCycle signature(x = "QuarterYearCycle"): ...
unitSeason signature(x = "QuarterYearCycle"): ...

Author(s)

Georgi N. Boshnakov

See Also

BuiltInCycle, pcCycle for creation of cycle objects and extraction of cycle part of time series,
BareCycle-class, SimpleCycle-class.
Examples

```r
## class "DayWeekCycle"
dwcycle <- BuiltinCycle(7) # new("DayWeekCycle")

unitSeason(dwcycle)
unitCycle(dwcycle)

allSeasons(dwcycle)
dwcycle[] # same

allSeasons(dwcycle, abb = TRUE)
dwcycle[, abb = TRUE] # same

dwcycle[2]
dwcycle[2, abb = TRUE]

seqSeasons(dwcycle)

## start the week on Sunday
dws <- BuiltinCycle(7, first = 7) # new("DayWeekCycle", first = 7)
dws[1] # "Sunday"
allSeasons(dws)

## class "Every30MinutesCycle"
cyc48 <- BuiltinCycle(48) # new("Every30MinutesCycle")
nSeasons(cyc48)
allSeasons(cyc48)

## class "FiveDayWeekCycle" is deprecated, use the equivalent:
fdcycle <- BuiltinCycle(5)

unitSeason(fdcycle)
unitCycle(fdcycle)

allSeasons(fdcycle)
fdcycle[] # same

allSeasons(fdcycle, abb = TRUE)
fdcycle[, abb = TRUE] # same

fdcycle[2]
fdcycle[2, abb = TRUE]

seqSeasons(fdcycle)

## class "MonthYearCycle"
mycycle <- BuiltinCycle(12) # new("MonthYearCycle")

unitSeason(mycycle)
unitCycle(mycycle)

allSeasons(mycycle)
```
Cyclic

Create objects from class Cyclic

Description

Create objects from class Cyclic.
Usage
Cyclic(cycle, start = NULL, ...)

## S3 method for class 'Cyclic'
as.Date(x, ...)

## S3 method for class 'Cyclic'
date(x)

## S3 method for class 'PeriodicTimeSeries'
as.Date(x, ...)

Arguments

cycle  a cycle object, a positive integer giving the number of seasons, or any other
        object that can be used to create a cycle with pcCycle(x, ...).

start  a cycle-season pair, a datetime object, a Date object or any object that can be
        converted to datetime with as_datetime(start).

... for Cyclic, arguments passed to pcCycle, used only if cycle is not from a cycle
        class.

x  a Cyclic object

Value
for Cyclic, an object from class "Cyclic"

See Also
BuiltinCycle, pcCycle for creation of cycle objects,
pcts importing and creating periodic time series

Examples

## bare bone Cyclic starting at Cycle 1, season 1
Cyclic(4)
Cyclic(4, c(1,1)) # same

## with quarter/year cycle
qu <- Cyclic(BuiltinCycle(4), start = c(2020, 1))
start(qu)
as_datetime(qu)

date(qu) <- c(2009, 2)
qu

ap <- pcts(AirPassengers)
as.Date(ap)
Cyclic-class

Class "Cyclic"

Description

Class "Cyclic"

Objects from the Class

Objects can be created by calls of the form new("Cyclic",...).

Slots

cycle: Object of class "BasicCycle" ~
pcstart: Object of class "ANY" ~

Methods

nSeasons signature(object = "Cyclic"): ...
pcCycle signature(x = "Cyclic",type = "ANY"): ...
allSeasons signature(x = "Cyclic",abb = "ANY"): ...
allSeasons<- signature(x = "Cyclic"): ...
seqSeasons signature(x = "Cyclic"): ...
unitCycle signature(x = "Cyclic"): ...
unitCycle<- signature(x = "Cyclic"): ...
unitSeason signature(x = "Cyclic"): ...
unitSeason<- signature(x = "Cyclic"): ...

Examples

showClass("Cyclic")

Example data from Franses (1996)

dataFranses1996

Description

A multivariate time series containing the data used in examples by Franses (1996).

Usage

data("dataFranses1996")
## Format

A multivariate quarterly time series.

## Details

Each column is a quarterly time series. The time series start and end at different times, so NA’s are used to align them in a single multivariate time series. Detailed account of the sources of the data is given by Franses (1996; Data Appendix, p. 214).

**year** (column 1)

The time formatted as yyyy.Q, where yyyy is the year and Q is the quarter (one of 1, 2, 3 or 4.). This column was part of the original data but is not really needed here since the time series object contains the time information.

**USTotalIPI** (column 2)


**CanadaUnemployment** (column 3)


**GermanyGNP** (column 4)

Real GNP in Germany, 1960.1 - 1990.4.

**UKTotalInvestment** (column 5)


**SA_USTotalIPI** (column 6) Seasonally adjusted USTotalIPI.

**SA_CanadaUnemployment** (column 7)

Seasonally adjusted CanadaUnemployment.

**SA_GermanyGNP** (column 8)

Seasonally adjusted GermanyGNP.

**UKGDP** (column 9)


**UKTotalConsumption** (column 10)


**UKNondurablesConsumption** (column 11)


**UKExport** (column 12)


**UKImport** (column 13)


**UKPublicInvestment** (column 14)


**UKWorkforce** (column 15)

SwedenNondurablesConsumption (column 16)

SwedenDisposableIncome (column 17)

SA_SwedenNondurablesConsumption (column 18)
Seasonally adjusted SwedenNondurablesConsumption with Census X-11 method, 1964.1–1988.1. (Using the approximate linear Census X-11 filter given in Table 4.1, p. 52 in Franses (1996) and generating the forecasts and backcasts as described in Ooms (1994)).

SA_SwedenDisposableIncome (column 19)
Seasonally adjusted SwedenDisposableIncome with Census X-11 method, 1964.1–1988.1. (Using the same method as above.)

More details on the individual time series are given by Franses (1996).

Note
Most of the time series in dataFranses1996 are available as separate datasets in package ‘partsm’. The numbers should be the same but note that, at the time of writing this, not all datasets there carry complete time information.

Source
The data were downloaded from http://people.few.eur.nl/franses/research/data/data1.txt, but this link is now broken.

References

Examples

data(dataFranses1996)
class(dataFranses1996)
colnames(dataFranses1996)
dim(dataFranses1996) # c(148, 19)
plot(dataFranses1996[, 2:11])

tipi <- dataFranses1996[, "USTotalIPI"]
plot(tipi)
## convert to PeriodicTS and remove NA's at the start and end
pctipi <- pcts(tipi)
pctipi <- window(pctipi, start = availStart(pctipi), end = availEnd(pctipi))
plot(pctipi)

## convert the whole dataset to class "PeriodicMTS"
pcfr <- pcts(dataFranses1996)
colnames(pcfr)[2:3] # "USTotalIPI" "CanadaUnemployment"
## subset as "PeriodicMTS"

```r
pcfr2to3 <- pcfr[2:3]
plot(pcfr2to3)
```

## ":["PeriodicMTS" even with length one arg.

```r
pcfr2to2 <- pcfr[2]
pcfr2to2a <- pcfr["USTotalIPI"] # same
```

## use "[[" or $ to get "PeriodicTS"

```r
pcfr2 <- pcfr[[2]]
pcfr2a <- pcfr["USTotalIPI"] # same
pcfr2b <- pcfr$USTotalIPI # same
identical(pcfr2, pcfr2a) # TRUE
identical(pcfr2, pcfr2b) # TRUE
```

cycle(pcfr)
frequency(pcfr)

date<--methods

Replace methods for date in package pcts

### Description

Replace methods for date in package pcts.

### Methods

- **signature(x = "BasicCycle")**
- **signature(x = "Cyclic")**

---

**ex1f**

An example PAR autocorrelation function

### Description

ex1f is the autocorrelation function used in the reference as an example when the solution of the periodic Yule-Walker system gives an invalid PAR model. This can happen only if Lambert-Lacroix’s condition on the PAR order is not satisfied, see `pdSafeParOrder`.

### Format

A function of two arguments

### Source

References


See Also

`pdSafeParOrder`

Examples

```r
data(ex1f)
## compute the first few autocorrelations
pc3 <- slMatrix(period = 2, maxlag = 5, f = ex1f, type = "tt")
## Fir a PAR(0,2) model
res0p2 <- alg1(pc3[,c(0,2)])
## model is invalid since a partial autocorrelation is larger than one:
res0p2$be
## Find a modified order:
pdSafeParOrder(c(0,2)) # PAR(1,2)
## now the parcor's are fine:
res1p2 <- alg1(pc3[,c(1,2)])
res1p2$be
```

`filterCoef-methods` Get the coefficients of a periodic filter

Description

Get the coefficients of a periodic filter.

Details

`filterCoef` is a generic function to extract the coefficients of periodic filters. Argument `convention` can be used to force a particular convention for the signs. The description here is for the methods defined in this package.

If `convention` is missing, the coefficient matrix is returned as stored in the object. Otherwise, if `convention` is one of the strings "BJ", "--" or "-", the coefficients returned have the opposite sign of those in the auxiliary polynomial (Box-Jenkins' convention). if `convention` is one of "SP", "++" or "+", the coefficients are as in the auxilliary polynomial (convention used in signal processing).

Value

a matrix

Methods

signature(object = "PeriodicBJFilter", convention = "character")
signature(object = "PeriodicSPFilter", convention = "character")
fitPM

Fit periodic time series models

Description

Generic function with methods for fitting periodic time series models.

Usage

fitPM( model, x, ...)

Arguments

x      the time series.
model  a periodic model, see Details.
...    further arguments to be passed on to individual methods.

Details

This is a generic function.

model provides the specification of the model. In particular, the class of model determines what
model is fitted. Specific values of the parameters are generally ignored by non-iterative methods
but some methods can handle more detailed specifications, see the individual methods.

Value

the fitted model, typically an object of class class(model)

Methods

signature(model = "ANY", x = "ANY") This is the default method. It simply exits with an error
message stating that fitPM does not have a method for the model specified by model.
signature(model = "numeric", x = "ANY") Fits a PAR model to x. model should be a vector of
non-negative integers giving the PAR order. The length of this vector is taken to be the number
of seasons.
   This is a convenience method. It constructs a PAR model and calls the method for model =
   "PeriodicArModel".
signature(model = "PeriodicArModel", x = "ANY") Fits a PAR model.
signature(model = "mcSpec", x = "ANY") Fits a periodic model according to the specification
given by model.
   Currently this method uses mC.ss to set up the optimisation environment and then calls one of
   the optimisation functions in that environment as specified by argument optim.method, see
   below.
   Additional arguments may be specified to control the optimisation.
Argument `init` can be used to give initial values. It is passed on to `mC.ss` (and so has the format required by it).

`optim.method` is the name of an optimisation function in the environment returned by `mC.ss`. The default is `optim.method = "minim"`, which is based on the standard R function `optim`. Alternatives are "minimBB" or "minimBBLU". All this needs to be documented but see `mC.ss` and `xx.ss` for details.

Further arguments are passed on to the optimisation method. A typical argument supported by most optimisation functions is `control`.

```
signature(model = "PiPeriodicArModel", x = "ANY") Fits a periodically integrated PAR model using the parameters of model as initial values. Calls `pclspiar` to do the actual work.
signature(model = "SiPeriodicArModel", x = "ANY") Fits a seasonally integrated PAR model.
signature(model = "PeriodicArModel", x = "PeriodicMTS")
signature(model = "PeriodicArModel", x = "PeriodicTS")
```

**Author(s)**

Georgi N. Boshnakov

**References**


**Examples**

```
## newm1 <- list(phi = matrix(1:12, nrow=4), p=rep(3,4), period=4, si2 = rep(1,4))
## new_pfm1 <- PeriodicFilterModel(newm1, intercept=0)

## generate some data;
set.seed(1234)
simts1 <- pcts(rnorm(1024), nseasons = 4)
fitPM(c(3,3,3,3), simts1)
fitPM(3, simts1)
## the fit on the underlying data is equivalent.
fitPM(c(3,3,3,3), as.numeric(simts1))

## equivalently, use a PAR(3,3,3,3) model for argument 'model'
## here the coefficients of pfm1 are ignored, since the estimation is linear.
pfm1 <- PeriodicArModel(matrix(1:12, nrow = 4), order = rep(3,4), sigma2 = 1)
pfm1
## these give same results as above
fitPM(pfm1, simts1)
fitPM(pfm1, as.numeric(simts1))

fitPM(c(1,1,1,1), simts1)
```
fitPM(c(3,2,2,1), simts1)
fitPM(c(3,2,2,2), simts1)

pdSafeParOrder(c(3,2,2,1))
pdSafeParOrder(rev(c(3,2,2,1)))

x <- arima.sim(list(ar = 0.9), n = 960)
pcx <- pcts(x, nseasons = 4)
mx <- matrix(x, nrow = 4)

##pc.acf(mx)
##pc.acf(mx, maxlag=10)
## TODO: avoid the warning when length ot the time series is not multiple
autocovariances(t(mx), maxlag = 6, nseasons = 4)
autocovariances(t(mx))

##It is an error to have more columns than rows.
## autocovariances(mx, maxlag = 6, nseasons = 4)
## autocovariances(mx)

num2pcpar(mx, c(1,1,1,1), period = 4)
num2pcpar(mx, c(3,3,3,3), period = 4)

sipfm1 <- new("SiPeriodicArModel", iorder = 1, siorder = 1, pcmodel = pfm1)
sipfm1
fitPM(sipfm1, mx)
pfm1

## experiments and testing
fit1 <- fitPM(c(3,3,3,3), simts1)
fit1_mf <- new("MultiFilter", coef = fit1$ar@coef)
vs <- mcompanion::mf_VSform(fit1_mf, form = "I")
tmp <- mcompanion::VAR2pcfilter(vs$Phi[, -4],
                               Phi0inv = vs$Phi0inv, D = fit1@sigma2, what = "")

names(tmp) # "pcfilter" "var" "Uform"
tmp$var
zapsmall(tmp$pcfilter)
fit1$ar@coef
all.equal(tmp$pcfilter[, 1:3], fit1$ar@coef, check.attributes = FALSE) # TRUE
tmp$Uform
fit1@sigma2

## both give the matrix Sigma for the "I" form
identical(
    vs$Phi0inv %*% diag(fit1@sigma2) %*% t(vs$Phi0inv),
    tmp$Uform$U0inv %*% diag(tmp$Uform$Sigma) %*% t(tmp$Uform$U0inv)
) # TRUE

## no, this is a different matrix
var1.mat <- cbind(vs$Phi0, # identity matrix
                  - vs$Phi) # drop trailing zero columns?
```r
var1_mat <- mcompanion::mCompanion(var1_mat)
var1_Sigma <- vs$Phi0inv %*% diag(fit1@sigma2) %*% t(vs$Phi0inv)
abs(eigen(diag(nrow(var1_mat)) - var1_mat)$values)
```

---

**FittedPeriodicArmaModel-class**

**Class** FittedPeriodicArmaModel

**Description**

Class FittedPeriodicArmaModel in package pcts

**Objects from the Class**

Objects can be created by calls of the form `new("FittedPeriodicArmaModel",...,mean)`.

**Slots**

- ar: Object of class "PeriodicBJFilter"
- ma: Object of class "PeriodicSPFilter"
- modelCycle: Object of class "BasicCycle"
- center: Object of class "numeric"
- intercept: Object of class "numeric"
- sigma2: Object of class "numeric"
- theTS: Object of class "PeriodicTS"
- asyCov: Object of class "ANY"
- ns: Object of class "numeric"

**Extends**


**Methods**

- `as_pcarma_list` signature(object = "FittedPeriodicArmaModel"): ...
- `show` signature(object = "FittedPeriodicArmaModel"): ...
Class FittedPeriodicArModel

Description

Class FittedPeriodicArModel.

Objects from the Class

Objects can be created by calls of the form new("FittedPeriodicArModel", ar, ma, sigma2,...).

Slots

- asyCov: Object of class "ANY"
- sigma2: Object of class "numeric"
- ar: Object of class "PeriodicArFilter"
- ma: Object of class "PeriodicMaFilter"
- center: Object of class "numeric"
- intercept: Object of class "numeric"
- theTS: Object of class "PeriodicTS"
- ns: Object of class "numeric"
- modelCycle: Object of class "BasicCycle"

Extends


Methods

No methods defined with class "FittedPeriodicArModel" in the signature.

- show signature(object = "FittedPeriodicArModel"): ...
- summary signature(object = "FittedPeriodicArModel"): ...
fit_trigPAR_optim

Fit a subset trigonometric PAR model

Description
Fit a subset PAR model with trigonometric parameterisation.

Usage

fit_trigPAR_optim(x, order, nseasons, seasonof1st = 1, maxiter = 200,
harmonics = NULL, sintercept = FALSE, tol = 1e-07,
type = c("vecbyrow", "bylag"), verbose = TRUE)

Arguments

x          time series.
order      order, an integer number.
nseasons   number of seasons, an integer number.
seasonof1st season of the first observation.
maxiter    max number of iterations.
harmonics  the harmonics to include in the model, vector of non-negative integers.
sintercept if TRUE include seasonal intercept.
tol        when to stop the iterations.
type       type of parameterisation, currently one of "vecbyrow" or "bylag".
verbose    if TRUE print more details during estimation.

Details
Fits a subset PAR model using trigonometric parameterisation, i.e. Fourier series written in terms of sines and cosines.

If argument type is bylag, the parameters for each lag are parameterised independently from other lags. If sintercept is TRUE, it has its own trigonometric representation.

If argument type is vecbyrow (“Vec operation by row”), the PAR parameters are stacked in a vector with all parameters for the first season, followed by all parameters for the second, and so on. The trigonometric parameterisation for this vector is used. So the fundamental frequency is 1/(nseasons * order). If sintercept is TRUE when type = vecbyrow, then then the intercept for each season is put before the PAR parameters and the fundamental frequency becomes 1/(nseasons * (order + 1)). Putting together the intercepts and the PAR parameters may not be very useful for parsimonious trigonometric parameterisation, so to have a separate set of coefficients for the intercepts set attribute “merge” of sintercept to FALSE.

Value

an object from class SubsetPM
Note
This function may change.

Author(s)
Georgi N. Boshnakov

Examples
## see examples for class "SubsetPM"

data(four_stocks_since2016_01_01)
DELL <- four_stocks_since2016_01_01$DELL
head(DELL)
tail(DELL)
dell <- pcts(DELL)

Description
Data for four stocks since 2016-01-01.

Usage
data("four_stocks_since2016_01_01")

Format
A list with components "DELL","MSFT", "INTC", "IBM". Each component is a time series from class "xts" "zoo".

Details
Note: This dataset is temporary for use in testing, although it may appear in some examples. Eventually it may be renamed and/or replaced by other data.
Stock market data for Dell, Microsoft, Intel and IBM, from 2016-01-01 to 2020-04-17. The Dell data start from 2016-08-17. All data were downloaded from Yahoo Finance on 2020-04-18.

Source
https://finance.yahoo.com/

Examples
data(four_stocks_since2016_01_01)
DELL <- four_stocks_since2016_01_01$DELL
head(DELL)
tail(DELL)
dell <- pcts(DELL)
Fraser2017

head(as_datetime(dell))
head(Pctime(dell))

## Weekends are totally absent from the data,
## so a Monday-Friday sub-cycle is created:
nSeasons(dell)
dell@cyle

## there are some NA's in the data, due to Bank holidays
Pctime(c(2624, 5), pcCycle(dell)) # "W2624 Fri"
as_datetime(Pctime(c(2624, 5), pcCycle(dell))) # "2020-04-10 UTC"

## dell["2020-04-10 UTC"]
head(cycle(dell))
tail(Pctime(dell))
tail(as.Date(Pctime(dell)))

---

Fraser2017

*Fraser River at Hope, mean monthly flow*

**Description**

Mean monthly flow (cms) of Fraser River From March 1912 to December 2017, recorded by Fraser River at Hope station.

**Usage**

data("Fraser2017")

**Format**

A time series (class "ts") with frequency 12, starting from January 1912 (the first two data values are NA) to December 2017.

**Details**

Dataset Fraser2017 is an extention of dataset "Fraser" in package "pear". The latter runs upto December 1990 (not the documented December 1991). At the time of writing this package "pear" is archived on CRAN, which is the main reason to include the dataset (with the added benefit of almost 30 years of additional data).

**Source**

[https://wateroffice.ec.gc.ca/](https://wateroffice.ec.gc.ca/)
Examples

data(Fraser2017)

fr <- window(Fraser2017, start = c(1912, 3), end = c(1990, 12))
## all.equal(as.numeric(fr), as.numeric(pear::Fraser)) # TRUE
## all.equal(tsp(fr), tsp(pear::Fraser)) # TRUE

maxLag-methods

Methods for function maxLag() in package 'pcts'

Methods

signature(object = "PeriodicArmaFilter")

Examples

## non-periodic autocovariances
maxLag(autocovariances(AirPassengers))

## periodic
pcts_exdata() # creates ap, ap7to9, pcfr, pcfr2to4,
maxLag(autocovariances(ap, maxlag = 6))

## pcarma filter
m <- rbind(c(0.81, 0), c(0.4972376, 0.4972376))
ar_filt3 <- new("PeriodicBJFilter", coef = m, order = c(1,2))
arma_filt3 <- new("PeriodicArmaFilter", ar = ar_filt3)
maxLag(arma_filt3)

mC.ss

Create environment for mc-fitting

Description

Creates an environment for mc-fitting. These functions are transitory, hence the strange names.

Usage

mC.ss(spec, ...)

xx.ss(period, type.eigval, n.root, eigabs, eigsign, co_r, co_arg,
  init = NULL, len.block = NULL, mo.col, generators = NULL)
Arguments

spec a model, an object of class mcSpec.

... further arguments to be passed on to xx.ss.

period the number of seasons.

type.eigval types of the eigenvalues, a character vector with elements "r" or "cp", see Details.

n.root number of roots. Currently the dimension of the matrix is set to this.

eigabs The absolute values/moduli of the eigenvalues, numeric vector.

eigsign The signs/moduli of the eigenvalues.

c0_r similar to eigabs but for the c0 parameters.

c0.arg similar to eigsign but for the c0 parameters.

init initial values, see Details.

len.block lengths of Jordan blocks.

mo.col last non-zero column in the top of the matrix.

generators ~~ TODO: describe this argument. ~~

Details

mC.ss takes the specification of the model as an object of class mcSpec and calls xx.ss.

Basically, the value returned by these functions is an extended model specification together with an environment which can be used for fitting the model, exploring the results and trying various things. This may be used for getting better understanding of the model and the optimisation routines.

The result of both functions is a list, containing several functions and an environment. The environment (element env) is the most important element since it allows access to everything in the model environment. The function elements of the list are simply a convenience.

Several functions in env are available for fitting the model. Currently these are minim, minimBB and minimBBlu. The first argument of all these functions is a time series to which the model is to be fitted. By default, a conditional likelihood is being optimised. To base the optimisation on conditional sum of squares, set argument CONDLIK to FALSE. The remaining arguments in a call to any of the above functions are passed on to the corresponding optimisation routine (whose help page should be consulted for details).

minim uses the core R function optim. minimBB and minimBBlu use BBoptim from package BB. They result is a list, as returned by the corresponding optimisation function with the optimal parameters in element par. The elements of this vector are named to help somewhat in its interpretation but complete information about the fitted model can be obtained from the environment.

Firstly, at the end of the optimisation, the optimal parameters and other information are stored in env. If the same call (maybe with modified instructions for the optimisation) is repeated, these parameters will be used as initial values for a new optimisation run. This may be useful, for example, if the previous run didn't converge.

Secondly, properties of the fitted model and more useful representations can be obtained using functions in the environment or the convinience functions in the list returned by xx.ss.

optparam2mcparam converts a vector of parameters into the more familiar filter representation, where the i-th row contains the coefficients for the i-th season. This function takes one argument the
vector of parameters, e.g. the one returned by the fitting functions. It updates a number of variables
in env, computes the filter representation of the model and stores it in wrkmodel. It returns NULL.
This function may be used for exploratory purposes or to set new values for the parameters, e.g. to
be used as starting values for a new optimisation run.

mcparam2optparam does the opposite. It converts the current model in env to a vector of parameters.
This function does not have arguments.

mclik computes the value of the conditional likelihood for given parameters. Its first argument is
a time series, the second is a vector of parameters and the third is a vector of innovations. Only
the first argument is compulsory. If param is not supplied, the current parameters in env are used.
Otherwise, they are updated with the new parameters and then used. The innovations default to the
zero vector. mcss is similar but computes the conditional sum of squares.

**Argument** init can be used to provide initial values. If it is missing or NULL, random initial
values are generated for the free parameters. init may also be a numeric vector suitable for the
call optparam2mcparam(init), see above. This vector would typically come from a previous
optimisation run.

init may also be a list with elements "eigabs", "eigsign", "co_r", "co_abs". These compo-
nents have the same meaning as the corresponding arguments of xx.ss.

**TODO: more is needed here!**

**Value**

A list with the following components:

- fmcss
- fparamvec
- fmcparam
- env

**Author(s)**

Georgi N. Boshnakov

**References**

Boshnakov GN, Iqelan BM (2009). “Generation of time series models with given spectral proper-

**See Also**

xx.ss which is called by mC.ss

**Examples**

```r
# test0 roots
spec.coz2 <- mcompanion::mcSpec(dim = 5, mo = 4, root1 = c(1,1), order = rep(2,4))
spec.coz2
xxcoz2a <- mC.ss(spec.coz2)
```
meanvarcheck

```r
## test0 roots
spec.coz4 <- mcompanion::mcSpec(dim = 5, mo = 4, root1 = c(1,1), order = rep(3,4))
xccoz4a <- mC.ss(spec.coz4)
```

---

**Description**

Asymptotic covariance matrix of periodic mean.

**Usage**

```r
meanvarcheck(parmodel, n)
```

```r
meancovmat(parmodel, n, cor = FALSE, result = "var")
```

**Arguments**

- `parmodel`: a periodic model.
- `n`: number of observations (TODO: need clarification here).
- `cor`: if TRUE, return correlations
- `result`: if "var", return the diagonal of the covariance matrix, otherwise return the matrix.

**Details**

Computes asymptotic covariance or correlation matrix of the periodic means.

**Value**

If `result = "var"` a matrix, otherwise a vector

**Author(s)**

Georgi N. Boshnakov

**See Also**

- `parcovmatlist`
Examples

```r
x <- arima.sim(list(ar=0.9), n=1000)
proba1 <- fitPM(c(3,2,2,2), x)
meancovmat(proba1, 100)
meancovmat(proba1, 100, cor = TRUE)
meancovmat(proba1, 100, result = "")
meancovmat(proba1, 100, cor = TRUE, result = "")
meanvarcheck(proba1, 100)
```

---

**modelCycle**

*Get the cycle of a periodic object*

**Description**

Get the cycle of a periodic object, a generic function.

**Usage**

```r
modelCycle(object)
```

```r
modelCycle(object, ... ) <- value
```

**Arguments**

- `object` an object.
- `value` the new value for the cycle, an object inheriting from "BasicCycle".
- `...` not used.

**Details**

`modelCycle` is essentially internal, for programming. The user level function to get the cycle of an object is `pcCycle`.

`modelCycle` returns the Cycle object (in the sense of package `pcts`), associated with object.

`modelCycle` is a generic function which makes it possible to associate a cycle with objects from a class, without inheriting from the cycle classes.

By definition, NULL represents the model cycle of objects from classes with no (inherited) method for `modelCycle`.

The default method of `modelCycle` returns NULL. The default method for its replacement version throws error.

**Value**

for `modelCycle`, an object inheriting from class "BasicCycle" or NULL;

"modelCycle<-" is used for the side effect of changing the cycle of object.
ModelCycleSpec-class

Methods

signature(object = "ANY")
signature(object = "ModelCycleSpec")

Slots

modelCycle: Object of class "BasicCycle"

Methods

modelCycle  signature(object = "ModelCycleSpec"): ...
modelCycle<- signature(object = "ModelCycleSpec"): ...

nCycles  Basic time information about periodic time series

Description

Basic time information about periodic time series.

Usage

nCycles(x, ...)
nTicks(x)
nVariables(x, ...)

Arguments

x  an object from a periodic time series class.
... further arguments for methods.
Details

nTicks gives the number of time points, i.e. number of rows in the matrix representation.

nVariables gives the number of variables in the time series.

nCycles gives the number of cycles available in the data, e.g. number of years for monthly data. It always gives an integer number. Currently, if the result is not an integer an error is raised. TODO: There is a case to round up or give the number of full cycles available but this seems somewhat dangerous if done quietly. A good alternative is to provide argument for control of this.

Value

integer number

Author(s)

Georgi N. Boshnakov

Examples

```r
ap <- pcts(AirPassengers)
nVariables(ap)
nTicks(ap)
nCycles(ap)

monthplot(ap)
boxplot(ap)
```

---

nSeasons-methods

Number of seasons for a periodic object

Description

Number of seasons for a periodic object.

Arguments

object an object for which the notion of number of seasons makes sense.

Details

nSeasons is a generic function. This page gives its methods for the periodic classes in package "pcts".

Value

an integer number
Methods

signature(object = "DayWeekCycle")
signature(object = "FiveDayWeekCycle")
signature(object = "MonthYearCycle")
signature(object = "PeriodicIntegratedArmaSpec")
signature(object = "QuarterYearCycle")
signature(object = "PeriodicMonicFilterSpec")
signature(object = "Periodic Intercept Spec")
signature(object = "Cyclic")
signature(object = "BareCycle")
signature(object = "OpenCloseCycle")
signature(object = "Every30MinutesCycle")
signature(object = "PartialCycle")
signature(object = "VirtualPeriodicModel")

Author(s)

Georgi N. Boshnakov

Examples

ap <- pcts(AirPassengers)
nSeasons(ap) # 12

pcfr <- pcts(dataFranses1996)
nSeasons(pcfr) # 4

num2pcpar  Fit PAR model using sample autocorrelations

Description

Fit PAR model using sample autocorrelations.

Usage

num2pcpar(x, order, result = NULL, ...)

Arguments

x time series, a numeric vector.
order PAR order, a single number or a vector with one entry for each season.
result what to return, the default is to return the full model, see Details.
... passed on to calc_peracf.
num2pcpar

Details

Computes the periodic autocorrelations and fits a PAR model using the Periodic Levinson-Durbin algorithm.

The order is a vector of non-negative integers, specifying the autoregressive orders for each season. If order is a single number, then all seasons have that order.

mean controls centering in the computation of the autocorrelations. If mean is numeric, then subtract the supplied mean before computing the autocovariances. If mean is TRUE, the default, compute and subtract the sample periodic mean before computing the autocovariances. If mean is FALSE, do not centre the series, i.e. assume that the mean is zero.

If result is NULL, the default, returns the full model. If result = "coef", returns the PAR coefficients only (currently any value of result other than NULL has this effect).

Value

The coefficients of the fitted model or a list with components:

- mean: the mean, set as described in Details.
- coef: forward prediction coefficients.
- scale: standard deviations of the innovations.

Author(s)

Georgi N. Boshnakov

See Also

fitPM which uses num2pcpar for calculations

Examples

```r
## Not run:
simts1 <- matrix(rnorm(100), nrow = 4)

num2pcpar(simts1, order = c(3, 2, 2, 2), period = 4)
num2pcpar(simts1, order = c(3, 2, 1, 2), period = 4)
pdSafeParOrder(c(3, 2, 1, 2))
pdSafeParOrder(c(3, 2, 1, 1))
num2pcpar(simts1, order = c(3, 2, 1, 1), period = 4)
num2pcpar(simts1, order = pdSafeParOrder(c(3, 2, 1, 1)), period = 4)

num2pcpar(simts1, order = c(3, 2, 1, 2), period = 4)
num2pcpar(simts1, order = c(3, 2, 1, 2), period = 4, mean = rep(0,4))
num2pcpar(simts1, order = c(3, 2, 1, 2), period = 4, mean = FALSE)
num2pcpar(simts1, order = c(3, 2, 1, 2), period = 4, mean = FALSE)$coef@m -
num2pcpar(simts1, order = c(3, 2, 1, 2), period = 4)$coef@m

## End(Not run)
```
parcovmatlist  

Compute asymptotic covariance matrix for PAR model

Description
Compute asymptotic covariance matrix for PAR model

Usage
parcovmatlist(parmodel, n, cor = FALSE, result = "list")

Arguments
- `parmodel` : PAR model, object of class `parModel`
- `n` : length of the series or a vector with one element for each season.
- `cor` : If TRUE return correlation matrix.
- `result` : if "list", the default, return a list, if "Matrix" return a Matrix object, otherwise return an ordinary matrix, see Details.

Details
Uses eq. (3.3) in the reference.

If `result = "list"`, `parcovmatlist` returns a list whose s-th element is the covariance matrix of the PAR parameters for the s-th season. Otherwise, if `result = "Matrix"` it returns a block-diagonal matrix created by `.bdiag()` from package "Matrix". If `result = "matrix"` it returns an ordinary matrix (with the current implementation this is returned for any value other than "list" or "Matrix").

Value
a list, matrix or block-diagonal matrix, as described in Details

Author(s)
Georgi N. Boshnakov

References

See Also
`pcacfMat, pc.acf.parModel`
Examples

```r
x <- arima.sim(list(ar=0.9), n=1000)
proba1 <- fitPM(c(3,2,2,2), x)
parcovmatlist(proba1, 100)
parcovmatlist(proba1, 100, cor = TRUE)
sqrt(diag(parcovmatlist(proba1, 100, cor = TRUE)[[1]]))
meanvarcheck(proba1, 100)
```

---

PartialCycle-class

Class PartialCycle

Description

Class PartialCycle

Objects from the Class

Objects can be created by calls of the form `new("PartialCycle",...)`.

Partial cycles are often created implicitly when subsetting time series using `window()` with argument `seasons`, see the examples.

Slots

- `orig`: the parent class of the partial cycle, an object inheriting from class "BasicCycle".
- `subindex`: an integer vector specifying the seasons to include in the partial cycle.

Extends

Class "BasicCycle", directly.

Methods

- `allSeasons` signature(x = "PartialCycle", abb = "logical"): ...
- `allSeasons` signature(x = "PartialCycle", abb = "missing"): ...
- `nSeasons` signature(object = "PartialCycle"): ...
- `unitCycle` signature(x = "PartialCycle"): ...
- `unitSeason` signature(x = "PartialCycle"): ...

See Also

`BuiltinCycle`
Examples

dwc <- new("DayWeekCycle")
dwc
allSeasons(dwc)

## a five day week cycle
dwc5 <- new("PartialCycle", orig = dwc, subindex = 1:5)
dwc5
allSeasons(dwc5)

weekend <- new("PartialCycle", orig = dwc, subindex = 6:7)
weekend
allSeasons(weekend)

ap <- pcts(AirPassengers)

## take data for the summer months (in Northern hemisphere)
ap7to9 <- window(ap, seasons = 7:9)
## the above implicitly creates a partial cycle
ap7to9
allSeasons(ap7to9)

Description

Class PartialPeriodicAutocorrelations.

Objects from the Class

Objects can be created by calls of the form new("PartialPeriodicAutocorrelations",...,data).

Slots

modelCycle: Object of class "BasicCycle" ~~
data: Object of class "Lagged" ~~

Extends


Methods

show signature(object = "PartialPeriodicAutocorrelations"): ...
Applies a periodic ARMA filter to a time series

Description

Filter time series with a periodic arma filter. If \texttt{whiten} is \texttt{FALSE} (default) the function applies the given ARMA filter to \texttt{eps} (\texttt{eps} is often periodic white noise). If \texttt{whiten} is \texttt{TRUE} the function applies the “inverse filter” to \texttt{x}, effectively computing residuals.

Usage

\begin{verbatim}
pc.filter(model, x, eps, seasonof1st = 1, from = NA, whiten = FALSE,
          nmean = NULL, nintercept = NULL)
\end{verbatim}

Arguments

\begin{itemize}
\item \texttt{x} \hfill the time series to be filtered, a vector.
\item \texttt{eps} \hfill residuals, a vector or NULL.
\item \texttt{model} \hfill the model parameters, a list with components "phi", "theta", "p", "q", "period", "mean" and "intercept", see Details.
\item \texttt{seasonof1st} \hfill the season of the first observation (i.e., of \texttt{x}[1]).
\item \texttt{from} \hfill the index from which to start filtering.
\item \texttt{whiten} \hfill if \texttt{TRUE} use \texttt{x} as input and apply the inverse filter to produce \texttt{eps} ("whiten" \texttt{x}),
              if \texttt{FALSE} use \texttt{eps} as input and generate \texttt{x} ("colour" \texttt{eps}).
\item \texttt{nmean} \hfill a vector of means having the length of the series, see Details.
\item \texttt{nintercept} \hfill a vector of intercepts having the length of the series, see details.
\end{itemize}

Details

The model is specified by argument \texttt{model}, which is a list with the following components:

\begin{itemize}
\item \texttt{phi} \hfill the autoregression parameters,
\item \texttt{theta} \hfill the moving average parameters,
\item \texttt{p} \hfill the autoregression orders, a single number or a vector with one element for each season,
\item \texttt{q} \hfill the moving average orders, a single number or a vector with one element for each season,
\item \texttt{period} \hfill number of seasons in a cycle,
\item \texttt{mean} \hfill means of the seasons,
\item \texttt{intercept} \hfill intercepts of the seasons.
\end{itemize}

The relation between \texttt{x} and \texttt{eps} is assumed to be the following. Let

\[ y_t = x_t - \mu_t \]
be the mean corrected series, where $\mu_t$ is the mean, see below. The equation relating the mean corrected series, $y_t = x_t - \mu_t$, and $\varepsilon$ is the following:

$$y_t = c_t + \sum_{i=1}^{p_t} \phi_t(i)y_{t-i} + \sum_{i=1}^{q_t} \theta_t(i)\varepsilon_{t-i} + \varepsilon_t$$

where $c_t$ is the intercept, $\text{inintercept}$. The inverse filter is obtained by writing this as an equation expressing $\varepsilon_t$ through the remaining quantities.

If $\text{whiten} = \text{TRUE}$, \text{pc.filter} uses the above formula to compute the filtered values of $x$ for $t = \text{from}, \ldots, n$, i.e. whitening the time series if $\varepsilon$ is white noise. If $\text{whiten} = \text{FALSE}$, $\varepsilon$ is computed, i.e. the inverse filter is applied to $x$ from $\varepsilon$, i.e. “colouring” $x$. In both cases the first few values in $x$ and/or $\varepsilon$ are used as initial values.

Essentially, the mean is subtracted from the series to obtain the mean-corrected series, say $y$. Then either $y$ is filtered to obtain $\varepsilon$ or the inverse filter is applied to obtain $y$ from $\varepsilon$ finally the mean is added back to $y$ and the result returned.

The mean is formed by $\text{model}\$mean and argument $\text{nmean}$. If $\text{model}\$mean is supplied it is recycled periodically to the length of the series $x$ and subtracted from $x$. If argument $\text{nmean}$ is supplied, it is subtracted from $x$. If both $\text{model}\$mean and $\text{nmean}$ are supplied their sum is subtracted from $x$.

The above gives a vector $y$, $y_t = x_t - \mu_t$, which is then filtered. If the mean is zero, $y_t = x_t$ in the formulas below.

Finally, the mean is added back, $x_t = y_t + \mu_t$, and the new $x$ is returned.

The above gives a vector $y$ which is used in the filtering. If the mean is zero, $y_t = x_t$ in the formulae below.

\text{pc.filter} can be used to simulate \text{pc-arma} series with the default value of $\text{whiten}=\text{FALSE}$. In this case $\varepsilon$ is the input series and $y$ the output.

$$y_t = c_t + \sum_{i=1}^{p_t} \phi_t(i)y_{t-i} + \sum_{i=1}^{q_t} \theta_t(i)\varepsilon_{t-i} + \varepsilon_t$$

Then $\text{model}\$mean or $\text{nmean}$ are added to $y$ to form the output vector $x$.

Residuals corresponding to a series $y$ can be obtained by setting $\text{whiten}=\text{TRUE}$. In this case $y$ is the input series. The elements of the output vector $\varepsilon$ are calculated by the formula:

$$\varepsilon_t = -c_t - \sum_{i=1}^{q_t} \theta_t(i)\varepsilon_{t-i} - \sum_{i=1}^{p_t} \phi_t(i)y_{t-i} + y_t$$

There is no need in this case to restore $x$ since $\varepsilon$ is returned.

In both cases any necessary initial values are assumed to be already in the vectors. If from is not supplied it is chosen as the smallest $i$ such that for all $t \geq i$, $t-p[t]>0$ and $t-q[t]>0$, i.e. the filter will not require negative indices for $x$ or $\varepsilon$.

\text{pc.filter} calls the lower level function \text{pc.filter arma} to do the computation.

**Value**

The filtered series: the modified $x$ if $\text{whiten}=\text{FALSE}$, the modified $\varepsilon$ if $\text{whiten}=\text{TRUE}$. 
pc.filter.xarma

Filter time series with periodic arma filters

Description

Filter time series with periodic arma filters with or options for periodic and non-periodic intercepts.

Usage

pc.filter.xarma(x, eps, phi, theta, period, p, q, n, from, seasonof1st = 1, intercept = NULL, nintercept = NULL)

Arguments

x the time series to be filtered, a vector.
eps the innovations, a vector.
phi the autoregression parameters, a matrix.
theta the moving average parameters, a matrix.
period the period (number of seasons in a year).
p the autoregression orders, recycled to period if length(p)=1.
q the moving average orders, recycled to period if length(q)=1.
n a positive integer, the time index of the last observation to be filtered.
from a positive integer, the time index of the first observation to be filtered.
seasonof1st a positive integer, the season of the time index of x[1], see Details.
intercept the intercepts of the seasons, a vector of length period.
nintercept intercepts, a vector of the same length as x.
Details

`pc.filter.xarma` is somewhat lower level. The user level function is `pc.filter` which uses `pc.filter.xarma` to do the computations.

`pc.filter.xarma` filters the time series `x` by the following formula (for `t=from,...,n`):

\[ x_t = c_t + \sum_{i=1}^{p_t} \phi_t(i)x_{t-i} + \sum_{i=1}^{q_t} \theta_t(i)\varepsilon_{t-i} + \varepsilon_t, \]

where `c_t` is the overall intercept at time `t`, see below. Values of `x[t]` for `t` outside the range `from,...,n`, if any, are left unchanged. Values for `t<from` are used as initial values when needed.

Two intercepts are provided for convenience and some flexibility. The periodic intercept, `intercept`, is a vector of length `period`. It is replicated to length `n`, taking care to ensure that the first element of the resulting vector, say `a`, starts with `intercept[seasonof1st]`. `nintercept` can be an arbitrary vector of length `n`. It can be used to represent trend or contributions from covariates. `nintercept` is not necessarily periodic and argument `seasonof1st` does not affect its use. The overall intercept is obtained as the sum `c = a + nintercept`.

Usually `x` is a numeric vector but it can also be a matrix in which each column represents the data for one “year”. Also, the length of `x` is typically, but not necessarily, equal to `n`. It is prudent to ensure that `length(x) >= n` and this must be done if `x` is a matrix.

Argument `phi` is ignored if `p==0`, argument `theta` is ignored if `q==0`.

`pc.filter.xarma` is meant to be called by other functions whose task is to prepare the arguments with proper checks. It does not make much sense to repeat the checks in `pc.filter.xarma`. In particular, no check is made to ensure that `from` and `n` are correctly specified.

This is a low level function meant to be used with basic vectors and matrices. TODO: Implement in C/C++. In the current implementation, it accesses the elements of the arguments with straightforward indexing, so objects from classes may be used as well, provided that `x[t]`, `eps[t]`, `phi[t,i]`, `theta[t,i]`, as well as assignment to `x[t]`, are defined for scalar indices.

Value

Returns `x` with `x[from]` to `x[n]` filled with the filtered values and values outside the interval `from,...,n` left unchanged.

The mode of `x` is left unchanged. In particular, `x` may be a matrix with each row representing the data for a season. This is convenient since periodic time series are often more easily processed in this form.

Level

0 (base)

Author(s)

Georgi N. Boshnakov

See Also

`pc.filter`
*pc.hat.h*

---

**pc.hat.h**

*function to compute estimates of the h weights*

---

**Description**

The h coefficients are scaled cross-covariances between the time series and the innovations. This function computes estimates for h using as input the observed series, a series of estimated innovations, and an estimate of the variance of the innovations.

**Usage**

```r
pc.hat.h(x, eps, maxlag, si2hat)
```

**Arguments**

- `x` the observed time series x(t)
- `eps` a series of estimated innovations
- `maxlag` maximum lag
- `si2hat` estimate of the variance of the innovations

**Details**

If missing, the variance of the innovations is estimated from `eps`.

**Value**

A matrix of the coefficient up to lag `maxlag` with one row for each season.

**Author(s)**

Georgi N. Boshnakov

**References**

Description
Compute PAR autocovariance matrix

Usage
pc.acf.parModel(parmodel, maxlag = NULL)

pcacfMat(parmodel)

Arguments
parmmodel       PAR model, an object of class parModel.
maxlag           maximum lag

Details
pc.acf.parModel returns the autocovariances of a PAR model in season-lag form with maximum lag equal to maxlag. If maxlag is larger than the available precomputed autocovariances, they missing ones are computed using the Yule-Walker relations. Note that pc.acf.parModel assumes that there are enough precomputed autocovariances to use the Yule-Walker recursions directly.

TODO: pc.acf.parModel is tied to the old classes since it accesses their slots. Could be used as a template to streamline the method for autocovariances for class "PeriodicAutocovariance".

The season-lag form can be easily converted to other forms with the powerful indexing operator, see the examples and slMatrix-class.

pcacfMat is a convenience function for statistical inference. It creates a covariance matrix with dimension chosen automatically. This covariance matrix is such that the asymptotic covariance matrix of the estimated parameters can be obtained by dividing sub-blocks by innovation variances and inverting them. See, eq. (3.3) in the reference.

Value
for pcacfMat, a matrix
for pc.acf.parModel, an slMatrix

Author(s)
Georgi N. Boshnakov

References
See Also

slMatrix-class

Examples

```r
x <- arima.sim(list(ar = 0.9), n = 1000)
proba1 <- fitPM(c(3,2,2,2), x)

acfb <- pc.acf.parModel(proba1, maxlag = 8)
acfb[4:(-2), 4:(-2), type = "tt"]

cacfMat(proba1)
```

pcacf_pwn_var

Variances of sample periodic autocorrelations

Description

Computes the variances of sample periodic autocorrelations from periodic white noise.

Usage

```r
pcacf_pwn_var(nepoch, period, lag, season)
```

Arguments

- **lag**: desired lags, a vector of positive integers.
- **season**: desired seasons.
- **nepoch**: number of epochs.
- **period**: number of seasons.

Details

These are given by McLeod (1994), see the reference, eq. (4.3).

Value

A matrix whose (i,j)th entry contains the variance of the autocorrelation coefficient for season `season[i]` and lag `lag[j]`.

Author(s)

Georgi N. Boshnakov
References


Examples

`pcacf_pwn_var(79, 12, 0:16, 1:12)`

---

**pcalg1**

*Periodic Levinson-Durbin algorithm*

**Description**

Calculate partial periodic autocorrelations, forward and backward prediction coefficients and error variances using the periodic Levinson-Durbin algorithm.

**Usage**

`alg1(r, p)`

**Arguments**

- `r` periodic autocovariances, a matrix, see `Details`.
- `p` autoregressive orders, numeric vector.

**Details**

`alg1(r, p)` calculates the partial periodic correlations from autocovariances `r` and autoregression orders `p`. The matrix `r` has the same format as that of the `r` slot of `pcAcvf` objects. The periodicity, `d`, is set equal to the number of rows in `r`. If the length of `p` is not equal to the periodicity, all autoregressive orders are set to the first element of `p`. This last feature is really meant to be used only with a scalar `p`.

The convention for the signs of the coefficients is the one from Boshnakov(1996) and is consistent with other R time series functions.

`pmax` below stands for the maximal element of `p`, i.e. the maximal AR order.

As in the non-periodic case, the periodic Levinson-Durbin algorithm fits recursively models of order 0, 1, ..., `pmax`. Namely, at step `i` the AR orders for all seasons are set to `i`. This is done in a way that correctly handles the case when not all elements of `p` are equal, see the references.

The essential quantities calculated by the periodic Levinson-Durbin algorithm are returned as matrices, whose `i`th rows contain values for season `i`. The complete details depend on the quantities, as described below.

The partial autocorrelations, the forward innovation variances and the backward innovation variances are returned as matrices with `d` rows and 1+`pmax` columns, whose `j`-th columns contain the quantities for order `j-1` (partial autocorrelations, forward innovation variances and backward innovation variances, respectively). Note that the lag-0 partial autocorrelations are the autocovariances for lag 0, see the references for details.
The forward autoregression parameters are returned as a list whose $j$th element is a matrix containing the coefficients for order $j$. Similarly for the backward autoregression parameters.

One often is interested in the model of order $p$ only. Its coefficients are given by $af[[pmax]]$, while the innovation variances are in the last column of $fv$.

**Value**

A list with the following elements.

- `orders`: autoregression orders
- `be`: partial autocorrelations, a matrix with $d$ rows
- `fv`: forward innovation variances, a matrix with $d$ rows
- `bv`: backward innovation variances, a matrix with $d$ rows
- `af`: forward autoregression parameters, a list with one element for the parameters for each order.
- `ab`: backward autoregression parameters, a list with one element for the parameters for each order.

**Note**

The autoregression orders of the output are not necessarily the same as those specified in the call. There may be no PAR model with the requested orders, see the references.

**Author(s)**

Georgi N. Boshnakov

**References**


**See Also**

- `pdSafeParOrder`

**Examples**

```r
r1 <- rbind(c(1,0.81,0.729),c(1,0.90,0.900))
alg1(r1,2)
## pc2 <- pcAcvf(init=r1)
## pc2a <- pcAcvf(init=r1,seasonnames=c("am","pm"), periodunit="day")
```
# example of Lambert-Lacroix
data(ex1f)
pc3 <- slMatrix(period=2,maxlag=5,f=ex1f,type="tt")
res0p2 <- alg1(pc3[,],c(0,2))
res1p2 <- alg1(pc3[,],c(1,2))
res3p3 <- alg1(pc3[,],c(3,3))

paramsys1 <- pcarma_param_system(pc3, NULL, NULL, 2, 0, 2)
t1 <- solve(paramsys1$A,paramsys1$b)

# this is from tests.r but I have lost t1
# set it to pc3 below
# note: t1 is not the t1 computed above and in other examples!
t1 <- pc3
t1
t1[]
alg1(t1[,],c(1,1))
alg1(t1[,],c(1,0))
alg1(t1[,],c(0,1))
alg1(t1[,],c(5,5))
alg1(t1[,],c(2,2))
alg1(t1[,],c(2,3))
alg1(t1[,],c(3,3))
alg1(t1[,],c(4,4))
alg1(t1[,],c(5,5))
Details

This function is a wrapper for \texttt{alg1()}. It calls \texttt{alg1}, to do the computations and returns the requested element as an object from class \texttt{slMatrix}. The model order is set to the maximal lag available in \(x\).

If \(\text{at0}\) is the character string "var", then the lag zero values in the result are set to the lag zero autocovariances, otherwise they are set to 1. This is mainly relevant for the periodic partial autocorrelations (s="be"), since the setting \(\text{at0}="\text{var}"\) ensures that they are in one to one correspondence with the autocovariances.

Value

the requested quantity as an object of type \texttt{slMatrix}

Author(s)

Georgi N. Boshnakov

References


See Also

\texttt{pdSafeParOrder, alg1}

Examples

\begin{verbatim}
r1 <- rbind(c(1,0.81,0.729),c(1,0.90,0.900))
\# example of Lambert-Lacroix
data(ex1f)
pc3 <- slMatrix(period=2,maxlag=5,f=ex1f,type="tt")
res0p2 <- alg1(pc3[,],c(0,2))
res1p2 <- alg1(pc3[,],c(1,2))
res3p3 <- alg1(pc3[,],c(3,3))
\end{verbatim}
Usage

pcApply(object, ...)

## S4 method for signature 'numeric'
pcApply(object, nseasons, FUN, ...)

## S4 method for signature 'matrix'
pcApply(object, nseasons, FUN, ...)

## S4 method for signature 'PeriodicTS'
pcApply(object, FUN, ...)

## S4 method for signature 'PeriodicMTS'
pcApply(object, FUN, ...)

Arguments

object an object for which periodic mean makes sense.
nseasons number of seasons.
FUN a function, as for apply.
... further arguments for FUN.

Details

For univariate periodic time series, pcApply applies FUN to the data for each season. For multivariate periodic time series, this is done for each variable.

The methods for "numeric" and "matrix" are equivalent to those for "PeriodicTS" and "PeriodicMTS", respectively. The difference is that the latter two don’t need argument nseasons and take the names of the seasons from object.

Argument "..." is for further arguments to FUN. In particular, with many standard R functions argument na.rm = TRUE can be used to omit NA's, see the examples.

In the univariate case, when length(object) is an integer multiple of the number of seasons the periodic mean is equivalent to apply(matrix(object,nrow = nseasons),1,FUN,...).

Value

numeric or matrix for the methods described here, see section Details.

Methods

signature(object = "matrix")
signature(object = "numeric")
signature(object = "PeriodicMTS")
signature(object = "PeriodicTS")
pcAr.ss

Compute the sum of squares for a given PAR model

Description

Compute the sum of squares for a given PAR model.

Usage

pcAr.ss(x, model, eps = numeric(length(x)))

Arguments

x time series, a numeric vector.
model a model.
eps residuals, defaults to a vector of zeroes. This may be used for models with moving average terms, for example.

details
todo:

Value

a number

Author(s)

Georgi N. Boshnakov

Examples

pcApply(pcts(presidents), mean, na.rm = TRUE)
pcMean(pcts(presidents), na.rm = TRUE) # same

pcApply(pcts(presidents), median, na.rm = TRUE)
pcApply(pcts(presidents), var, na.rm = TRUE)
pcApply(pcts(presidents), sd, na.rm = TRUE)

pcfr2to4 <- pcts(dataFranses1996)[2:4]
pcApply(pcfr2to4, median, na.rm = TRUE)
pcApply(pcfr2to4, sd, na.rm = TRUE)
**pcAR2acf**

*Compute periodic autocorrelations from PAR coefficients*

**Description**
Compute periodic autocorrelations from PAR coefficients. This effectively solves the inverse problem to that solved by the periodic Levinson-Durbin algorithm but does not use a recursion.

**Usage**

```
pcAR2acf(coef, sigma2, p, maxlag = 10)
```

**Arguments**

- `coef`: PAR coefficients, a matrix, see Details.
- `sigma2`: innovations variances.
- `p`: PAR order.
- `maxlag`: How many lags to compute.

**Details**

`coef` is a matrix with the coefficients for season *i* in the *i*-th row. The coefficients start from lag 1. The first few autocorrelations are computed by solving a linear system, see the references. The rest, are generated using the periodic Yule-Walker equations.

**Value**

A matrix, in which row *s* contains the acf’s for season *s* for lags 0, 1, ..., maxlag (in this order).

**Author(s)**

Georgi N. Boshnakov

**References**


**See Also**

`pcarma_acvf_lazy`, which does the main computation, but note that the coefficients for it start from lag zero.
Examples

```r
m <- rbind( c(0.81, 0), c(0.4972376, 0.4972376) )
si2 <- PeriodicVector(c(0.3439000, 0.1049724))

pcAR2acf(m)
pCAR2acf(m, si2)
pCAR2acf(m, si2, 2)
pCAR2acf(m, si2, 2, maxlag = 10)

# same using pcarma_acvf_lazy directly
m1 <- rbind( c(1, 0.81, 0), c(1, 0.4972376, 0.4972376) )

testphi <- slMatrix(init = m1)
myf <- pcarma_acvf_lazy(testphi, testtheta, si2, 2, 0, 2, maxlag = 10)
myf(1:2, 0:9)  # get a matrix of values

all(myf(1:2, 0:9) == pcAR2acf(m, si2, 2, maxlag = 9))  # TRUE
```

pcarma_acvf2model  

*Fit a PC-ARMA model to a periodic autocovariance function*

Description

Fit a PC-ARMA model to a periodic autocovariance function.

Usage

```r
pcarma_acvf2model(acf, model, maxlag)
```

Arguments

- `acf`: a periodic autocovariance function, an object of class `pcAcvf`.
- `model`: a pc-arma model, an object of class `pcARMApq`.
- `maxlag`: not used. (todo: check!)

Value

- Describe the value returned If it is a LIST, use

```
comp1 Description of `comp1`
comp2 Description of `comp2`
...
```

Author(s)

Georgi N. Boshnakov
References


Examples

```r
data(ex1f)
pc3 <- slMatrix(period=2,maxlag=5,f=ex1f,type="tt")
# pcarma_param_system(pc3, NULL, NULL, 2, 0, 2)
parsys <- pcarma_param_system(pc3, NULL, NULL, c(2,2), 0, 2)
param <- solve(parsys$A,parsys$b)

# res <- pcarma_acvf2model(pc3, list(p=c(1,2),q=0,period=2))
# res <- pcarma_acvf2model(pc3, list(p=c(1,2),q=0))
# res <- pcarma_acvf2model(pc3, list(p=c(1,2),period=2))
res <- pcarma_acvf2model(pc3, list(p=c(1,2)))

print(param)
print(res)
```

pcarma_solve

*Functions to compute various characteristics of a PCARMA model*

Description

Given a PCARMA model, create a function for computing autocovariances or coefficients of the corresponding infinite moving average representation or prepare the linear system whose solution provides the first few autocovariances of the model.

Usage

- `pcarma_acvf_lazy(phi, theta, sigma2, p, q, period, maxlag = 100)`
- `pcarma_h_lazy(phi, theta, p, q, period, maxlag = 200)`
- `pcarma_acvf_system(phi, theta, sigma2, p, q, period)`
- `pcarma_param_system(acf, h, sigma2, p, q, period)`
- `pcarma_h(h, na = NA)`

Arguments

- `phi` the autoregression parameters, an object of class "slMatrix"
- `theta` the moving average parameters, an object of class "slMatrix"
- `sigma2` the innovation variances, an object of class "PeriodicVector" or a vector of size `period`. Details.
- `p` the (maximal) autoregression order or the autoregression orders.
- `q` the (maximal) moving average order or the moving average orders.
period
maxlag
acf

h

na

Details

Compute acvf from parameters:
pcarma_acvf_lazy creates a function that will compute (on demand) values of the acf by a recursive formula. Computed values are stored internally for lags up to maxlag.

System for acvf from parameters:
pcarma_acvf_system forms a linear system for the calculation of autocovariances from the parameters of a pc-arma model. The argument theta is not used if q = 0 and phi is not used if p = 0.

System for parameters from acvf: pcarma_param_system takes the periodic autocovariances of a pc-arma model and computes a matrix and a vector representing the linear system whose solution provides the parameters of the model.
Scalar p specifies the same autoregression order for each season, similarly for q. p and q may be vectors of length period specifying the order for each season individually. In the latter case the solution of the system may not be a proper model or, if it is, its autocovariances may not be the ones used here! See the references for details.
The class of acf is not required to be one of those explicitly listed above, but it should understand their indexing conventions, similarly for sigma2.
For pure autoregression, q = 0, the arguments h and sigma2 are ignored. TODO: add sigma2 (if supplied) to the returned list?

Compute h from parameters:
pcarma_h_lazy: h(t,i) are the coefficients in infinite the moving average representation of the pc.arma model. The calculations use formula (4.4) from my paper (or elsewhere) with internal storage (in an slMatrix) of calculated results (for i<maxlag) and recursive calls to itself. So, it is not necessary to compute h(t,i) in any particular order.

Infinite MA coefficients(h):
pcarma_h Function to create a function for lazy computation of h(t,i) in pc.arma models
Takes a matrix of h(t,i) coefficients and returns a function that calculates h(t,i) from my paper xxx.
The returned value can be used in the same way as that of pcarma_h_lazy.

Value

for pcarma_acvf_lazy:
a function taking two arguments t and k such that for scalar t and k the call f(t,k) will return EX(t)X(t-k). If either of the arguments is a vector, then f(t,k) returns a matrix of size (length(t),length(k)) containing the respective autocovariances.
for `pcarma_h_lazy`:
a function, say $h$. In calls to $h$, if both arguments are scalars $h(t,i)$ returns $h_{t,i}$. If at least one of the arguments is a vector a matrix of values of $h$ is returned.

for `pcarma_acvf_system`:
a list with two components representing the linear system:

`A` The $(p + 1)\times (p + 1)$ period matrix of the system, an object of class "matrix".
`b` The right-hand side of the system, a vector of length $(p + 1)$ period, an object of class "vector".

$A^{-1}b$ can be used to get a vector of the autocovariances in the following order (d is the period, p is the maximal AR order):

$$K(1, 0), ..., K(d, 0), K(1, 1), ..., K(d, 1), ..., K(1, p), ..., K(d, p).$$

for `pcarma_param_system`:
A list with components representing the linear system and the AR and MA orders:

`A` The matrix of the system
`b` The right-hand side of the system
`p` The AR order
`q` The MA order

$A^{-1}b$ will return a vector of the parameters of the pc-arma model: all parameters for the first season, followed by all parameters for the second seasons and so on. For each season the parameters are in the following order (s is the current season, d is the period, $p[s]$ and $q[s]$ are the corresponding AR and MA orders):

$$\sigma^2(s), \phi(s, 1), ..., \phi(s, p[s]), \theta(s, 1), ..., \theta(s, q[s]).$$

for `pcarma_h`:
a function, say $h$. In calls to $h$, if both arguments are scalars $h(t,i)$ returns $h_{t,i}$. If at least one of the arguments is a vector a matrix of values of $h$ is returned. Analogous to `pcarma_h_lazy`.

Note

for `pcarma_acvf_lazy`: The recursion may become extremely slow for lags greater than `maxlag`. If large lags are likely to be needed the argument `maxlag` should be used to increase the internal storage. The default for `maxlag` currently is 100.

Author(s)

Georgi N. Boshnakov

References


See Also

`pcarma_h`, `pcarma_param_system`
Examples

```r
## periodic acf of Lambert-Lacroix
data(ex1f)
(pc3 <- slMatrix(period = 2, maxlag = 5, f = ex1f, type = "tt"))
## find the parameters
s3 <- pcarma_param_system(pc3, NULL, NULL, 2, 0, 2)
coef3 <- solve(s3$A, s3$b)
pcarma_unvec(list(p = 2, q = 0, period = 2, param = coef3))

## actually, the model is PAR(1,2):
s3a <- pcarma_param_system(pc3, NULL, NULL, c(1, 2), 0, 2)
coef3a <- solve(s3a$A, s3a$b)
pcarma_unvec(list(p = c(1, 2), q = 0, period = 2, param = coef3a))

## prepare test parameters for a PAR(2) model with period=2.
## (rounded to 6 digits from the above example.
m1 <- rbind(c(1, 0.81, 0), c(1, 0.4972376, 0.4972376))
m2 <- rbind(c(1, 0, 0), c(1, 0, 0))
testphi <- slMatrix(init = m1)
testtheta <- slMatrix(init = m2)
si2 <- PeriodicVector(c(0.3439000, 0.1049724)) # or si2 <- c(1,1)

## acf from parameters
myf <- pcarma_acvf_lazy(testphi, testtheta, si2, 2, 0, 2, maxlag = 110)
myf(1,4)  # compute a value
a1 <- myf(1:2,0:9)  # get a matrix of values

## h from parameters
h <- pcarma_h_lazy(testphi, testtheta, 2, 2, 2)
h(3, 2)   # a scalar
h1 <- h(1:2, 1:4) # a matrix

## compute acvf from parameters
( acfsys <- pcarma_acvf_system(testphi, testtheta, si2, 2, 0, 2) )
acfv <- solve(acfsys$A, acfsys$b)
acf1 <- slMatrix(acfv, period = 2)

## TODO: examples wirh q != 0
```

Description

Handle a simple list specification of pcarma models. Functions to convert to and from a representation appropriate for handing on to optimisation functions.
Usage

pcarma_prepare(model, type)
pcarma_unvec(model)
pcarma_tovec(model)

Arguments

model specification of a pcarma model, a list, see Details.
type not used.

Details

These functions work with a specification of a pcarma model as a list with components period, p, q, param, phi, theta and si2, see also section ‘Values’. The functions do not necessarily need or examine all these components.

Argument model is a list with components as accepted by pcarma_prepare. Details are below but the guiding rule is that there are sensible defaults for absent components.

pcarma_prepare gives a standard representation of model, in the sense that it ensures that the model has components period, p and q, such that p and q are vectors of length period. pcarma_prepare does not examine any other components of the model. (TODO: do the same for the innovation variance?)

If model$period is NULL, pcarma_prepare sets it to the length of the longer of model$p and model$q. If model$p is a scalar it is extended with rep(model$p, period). Missing or NULL model$p is equivalent to model$p = 0. model$q is processed analogously.

The net effect is that period, p and q will be set as expected as long as period is given or at least one of the other two is of length equal to the period. A warning is issued if period <= 1 (it is all too easy to give scalar values for p and q and forget to set the period, in which case period will be deduced to be one).

A number of functions (including pcarma_tovec and pcarma_unvec) dealing with the list representation of pcarma models start by calling pcarma_prepare to avoid the need for handling all possible cases.

pcarma_tovec returns a list with components p, q and param, where param is a numeric vector containing the pcarma parameters and the innovations variances and thus is suitable for optimisation functions. Notice that it is component param that is a vector. The reason that pcarma_tovec returns a list, is that the caller may need to do further work before calling a generic optimisation function. For example, it may wish to dop the variances from the vector.

pcarma_unvec(model) performs the inverse operation. It takes a list like that produced by pcarma_tovec and converts it to a detailed list containing the components of the model.

Value

for pcarma_unvec, a list with components:

p autoregressive orders, numeric vector
q moving average orders, numeric vector
si2 innovation variances
phi  autoregressive parameters
theta moving average parameters

for pcarma_tovec, a list with components:
p  autoregressive order
q  moving average order
param parameters of the model, a numeric vector. TODO: give the order of the parameters in the vector!

for pcarma_prepare, a list as pcarma_unvec, see also Details.

Note
The specification and the functions were created ad hoc to get the computations going and are not always consistent with other parts of the package.

Author(s)
Georgi N. Boshnakov

---

pcCycle-methods  
*Create or extract Cycle objects*

Description
pcCycle() is a generic function with methods for creating, converting, modifying, and extracting cycle objects. BuiltinCycle() is a function to create cycle objects from the builtin cycle classes.

Usage
pcCycle(x, type, ...)

BuiltinCycle(n, coerce = FALSE, first = 1, stop = TRUE)

Arguments

- `x`  an object, methods include numeric, character and cyclic objects, see Details.
- `type`  class of the result. If equal to "auto", the default, the class is determined by the argument(s), otherwise should be the name of a cycle class.
- `...`  further arguments for methods.
- `n`  number of seasons, an integer.
- `coerce`  if TRUE coerce the objects to a modifiable cycle class, currently "SimpleCycle".
- `first`  which season is first for this object.
- `stop`  if TRUE, the default, throw error if there is no builtin class with n seasons, otherwise create a "BareCycle" object.
**Details**

`pcCycle` serves as both a constructor and extractor of cycle objects. It is meant to just do the right thing, relieving the user from the burden of specifying a particular cycle class.

If `x` is numeric it constructs a cycle object with period `x` and additional properties as specified by the other arguments. If `x` is a character string, it is taken to be the name of one of the built-in cycles. `pcCycle` can be used to create a modified version of a cycle object and/or convert it to another cycle type. This is done by providing a cycle object as argument `x`, i.e. one inheriting from "BasicCycle".

If `x` inherits from "Cyclic", `pcCycle` returns its cycle component.

Argument `type` should be rarely needed, except maybe to conveniently force conversion of the built-in type to an ordinary type.

The descriptions of the individual methods in section Methods give some further specific details.

`BuiltinCycle` is a convenience function to create objects from built-in cycle classes by specifying the number of seasons. The built-in cycle classes are essentially fixed, except that which season is considered first can be changed using argument `first`. If other modifications are desired, convert the returned built-in cycle object to class "SimpleCycle". This can be done also in the call to `BuiltinCycle()` by specifying `coerce = TRUE`.

By default, `BuiltinCycle` throws an error if there is no built-in class with the requested number of seasons. Set argument `stop` to `FALSE` to create an object from class "BareCycle" instead (and it will be converted to "SimpleCycle" if `coerce = TRUE`). Argument `stop` is mainly for programming.

**Value**

for `pcCycle`, an object from one of the cycle classes;

for `BuiltinCycle`, an object from one of the built-in classes, coerced if requested.

**Methods**

signature(x = "numeric", type = "missing") creates a cycle object with period `x`. If `x` is the only argument, a "BareCycle" object is created, otherwise the constructor of "SimpleCycle" is invoked with all arguments except `type` passed on to it.

signature(x = "character", type = "missing") creates an object from the class specified by `x`. Currently this is equivalent to `new(x,...)` but somewhat more portable. Future amendments may use a more suitable class for some combinations of the arguments. Also, if a class is renamed, a code will be inserted here to create an equivalent object.

signature(x = "numeric", type = "character")

signature(x = "character", type = "character") first call the method with `type = "missing"`, then convert the result to class `type`.

signature(x = "Cyclic", type = "ANY") extracts the cycle component of `x` (`x@cycle`). Currently ignores the remaining arguments.

signature(x = "BasicCycle", type = "missing") convert an object from any cycle class to class "SimpleCycle". This is like `as(x, "SimpleCycle")` but can have further arguments.

signature(x = "BasicCycle", type = "character") convert an object from any cycle class to class `type`.
signature(x = "ts", type = "missing")
signature(x = "ts", type = "character") when x is of class "ts", extract the frequency and convert it to a cycle class. Just as for "ts", certain frequencies are taken to correspond to specific classes. While base R treats periodicities 4 and 12 specially, pcCycle extends this to all builtin classes in pcts. Argument type can be used to overwrite this default behaviour by requesting a specific class. In particular, type = "BareCycle" and type = "" cause the result to be "BareCycle".

signature(x = "PeriodicTimeSeries", type = "missing")
signature(x = "PeriodicTimeSeries", type = "character") extract the cycle part of an object inheriting from "PeriodicTimeSeries", currently "PeriodicTS" or "PeriodicMTS". Argument type can be used to force the result to be from a specific cycle class, as in the methods for "ts".

Author(s)
Georgi N. Boshnakov

See Also
allSeasons for further examples,
class BuiltinCycle for the available builtin classes and more examples

Examples

## pcCycle
pcCycle(4)
pcCycle(4, seasons = c("Spring", "Summer", "Autumn", "Winter"))

pcCycle("QuarterYearCycle")
BuiltinCycle(4) # same, recommended

pcCycle("QuarterYearCycle", type = "BareCycle")
pcCycle("QuarterYearCycle", type = "SimpleCycle")

## BuiltinCycle
BuiltinCycle(2) # "OpenCloseCycle"
BuiltinCycle(4) # "QuarterYearCycle"
BuiltinCycle(5) # five day week cycle
BuiltinCycle(7) # "DayWeekCycle"
BuiltinCycle(12) # "MonthYearCycle"
BuiltinCycle(48) # "Every30MinutesCycle"

## error, since there is no builtin cycle with 19 seasons:
## BuiltinCycle(19)

## use stop = FALSE to reate a default cycle in this case
BuiltinCycle(19, stop = FALSE)
BuiltinCycle(19, coerce = TRUE, stop = FALSE)
Fit PAR models using least squares. The model may contain intercepts and linear trends, seasonal or non-seasonal.

Usage

\[ \text{pclsdf}(x, d, \text{lags} = \text{integer}(0), \text{sintercept} = \text{TRUE}, \text{sslope} = \text{FALSE}, \text{intercept} = \text{FALSE}, \text{slope} = \text{FALSE}, xreg, \text{contrasts} = \text{NULL}, \text{seasonof1st} = \text{NULL}, \text{coefonly} = \text{FALSE}) \]

Arguments

- **x**: time series, a numeric vector
- **d**: period, an integer
- **lags**: an integer vector, typically \(1:p\), where \(p\) is the order of the autoregression. The same lags are used for all seasons.
- **sintercept**: if TRUE include seasonal intercepts.
- **sslope**: if TRUE include seasonal linear trend.
- **intercept**: if TRUE include non-seasonal intercept.
- **slope**: if TRUE include non-seasonal linear trend.
- **xreg**: additional regressors, not used currently.
- **contrasts**: contrasts to use for the seasons factor variable.
- **seasonof1st**: season of the first observation in the time series, see Details.
- **coefonly**: if TRUE, return only the parameters of the fitted model, otherwise include also the object returned by \(\text{lm}\).

Details

This function fits PAR models by the method of least squares. Seasonal intercepts are included by default. Non-seasonal intercepts are available, as well as seasonal and non-seasonal linear trend. Separate arguments are provided, so that any combination of seasonal and non-seasonal intercepts and slopes can be specified.

If coefonly is TRUE, pclsdf returns only the estimated parameters, otherwise it includes additional statistical information, see section Note for the current details.
Value

A list with the components listed below. Some components are present only if included in the model specification.

- **par**: the PAR coefficients, a matrix with a row for each season.
- **sintercept**: (if specified) seasonal intercepts, a numeric vector.
- **sigma2hat**: innovation variances.
- **formula.char**: the formula used in the call of `lm`, a character string.
- **fit**: (if `coefonly = FALSE`) the fitted object obtained from `lm`.

Note

Currently, `pclsdf` prepares a model formula according to the specification and calls `lm` to do the fitting. Component "fit" in the result (available when `coefonly = FALSE`) contains the raw fitted object returned by `lm`. Statistical inference based on this object would, in general, not be justified for correlated data.

todo: currently some of the parameters are returned only via the fitted object from `lm`.

Author(s)

Georgi N. Boshnakov

See Also

- `pclspiar`

Examples

```r
## data(dataFranses1996)
cu <- pcts(dataFranses1996[, "CanadaUnemployment"])
cu <- window(cu, start = availStart(cu), end = availEnd(cu))
pclsdf(cu, 4, 1:2, sintercept = TRUE)

pclsdf(austres, 4, lags = 1:3)
pclsdf(austres, 4, lags = 1:3, sintercept = TRUE)
pclsdf(austres, 4, lags = 1:3, sintercept = TRUE, sslope = TRUE)

x <- rep(1:4,10)
pclsdf(x, 4, lags = 1:3, sintercept = TRUE, sslope = TRUE)

## this is for the version when contrasts arg. was passed on directly to lm.
## tmp1 <- pclsdf(austres, 4, lags = 1, sintercept = FALSE, sslope = TRUE,
##                 contrasts = list(Season = "contr.sum" ))
```
**Description**

Fit a periodically integrated autoregressive model.

**Usage**

```r
pclspiar(x, d, p, icoef = NULL, parcoef = NULL, sintercept = FALSE, seasonof1st = 1, weights = TRUE, itol = 1e-07, maxniter = 1000)
```

**Arguments**

- `x` time series.
- `d` period.
- `p` order of the model, a positive integer, see Details.
- `icoef` initial values for the periodic integration coefficients.
- `parcoef` not used currently.
- `sintercept` if `TRUE` include seasonal intercepts.
- `seasonof1st` season of the first observation.
- `weights` if `TRUE`, use periodic weights in the nonlinear least squares, see Details.
- `itol` threshold value for the stopping criterion.
- `maxniter` maximum number of iterations.

**Details**

This function fits a periodically integrated autoregressive model using non-linear least squares. The order of integration is one and the order of the periodically correlated part is \( p - 1 \). So, \( p \) must be greater than or equal to one.

If `weights = TRUE` the non-linear optimisation is done with weights inversely proportional to the innovation variances for the seasons, otherwise the unweighted sum of squared residuals is minimised.

**Value**

A list currently containing the following elements:

- `icoef` coefficients of the periodic integration filter.
- `parcoef` coefficients of the PAR filter.
- `sintercept` seasonal intercepts.
- `sigma2hat` innovation variances.
Author(s)

Georgi N. Boshnakov

References


See Also

*pclsdf*, *test_piar*, *fitPM*

Examples

```r
## see also the examples for fitPM()

ts1 <- window(dataFranses1996[, "CanadaUnemployment"],
               start = c(1960, 1), end = c(1987, 4))
pclspiar(ts1, 4, p = 1, sintercept = TRUE)
pclspiar(ts1, 4, p = 2, sintercept = TRUE)
```

Description

Compute periodic mean, generic function.

Usage

```r
pcMean(object, ...)
```

```r
# S4 method for signature 'numeric'
pcMean(object, nseasons, ...)
```

```r
# S4 method for signature 'matrix'
pcMean(object, nseasons, ...)
```

```r
# S4 method for signature 'PeriodicTS'
pcMean(object, ...)
```

```r
# S4 method for signature 'PeriodicMTS'
pcMean(object, ...)
```
Arguments

- **object**: an object for which periodic mean makes sense.
- **nseasons**: number of seasons.
- **...**: further arguments for methods.

Details

For univariate periodic time series, `pcMean` computes the mean for each season and returns a named vector. For multivariate periodic time series, the result is a matrix with one column for each variable.

The methods for "numeric" and "matrix" are equivalent to those for "PeriodicTS" and "PeriodicMTS", respectively. The difference is that the latter two don’t need argument `nseasons` and take the names of the seasons from `object`.

Argument `na.rm = TRUE` can be used to omit NA’s.

In the univariate case, when `length(object)` is an integer multiple of the number of seasons the periodic mean is equivalent to computing the row means of `matrix(object,nrow = nseasons)`.

Value

numeric or matrix for the methods described here, see section Details.

Methods

```r
signature(object = "matrix")
signature(object = "numeric")
signature(object = "PeriodicMTS")
signature(object = "PeriodicTS")
signature(object = "VirtualPeriodicArmaModel")
```

Author(s)

Georgi N. Boshnakov

See Also

`pcApply`

Examples

```r
pcMean(pcts(presidents))
pcMean(pcts(presidents), na.rm = TRUE)

pcMean(pcts(dataFranses1996)[2:5], na.rm = TRUE)

pcMean(1:20, nseasons = 4)
m <- matrix(1:20, nrow = 4)
all(apply(m, 1, mean) == pcMean(1:20, nseasons = 4)) # TRUE
```
Description

Plot periodic time series.

Usage

## S3 method for class 'PeriodicTimeSeries'
boxplot(x, ...)

## S3 method for class 'PeriodicTimeSeries'
monthplot(x, ylab = deparse(substitute(x)), base, ...)

Arguments

x

a periodic time series object.

... further arguments to be passed to the plotting function.

ylab

label for the y-axis, only used for univariate time series.

base

a function for use for computing reference lines.

Details

Functions for periodic/seasional plots and boxplots.

The S3 method for `monthplot()` just calls `pcPlot()`.

Author(s)

Georgi N. Boshnakov

Examples

ap <- pcts(AirPassengers)
monthplot(ap)
boxplot(ap)

fr23 <- pcts(dataFranses1996[, 2:3])
monthplot(fr23)
boxplot(fr23)
Test for periodicity

Usage

pcTest(x, nullmodel, nseasons, ...)

Arguments

- **x**: the object to be tested, e.g. a time series or a periodic acf
- **nullmodel**: specification of the test to be performed
- **nseasons**: number of seasons
- **...**: additional arguments to be passed on to methods

Details

This is a generic function which acts as a despatcher for various tests for periodicity and periodic correlation.

- **x**: is typically a time series but conceptually it is an object containing the statistics needed for carrying out the requested test. For example, **x** may be the periodic autocovariance function for tests based on sample autocorrelations and autocovariances.

The method with signature `(x = "ANY", nullmodel = "character")` may be considered as default for `pcTest`. The “real” default method simply prints an error message.

Value

- a list containing the results of the requested test, see the individual methods for details

Methods

signature(x = "ANY", nullmodel = "character") Argument nullmodel specifies the test to be performed. It should be a single character string. If it is one of the strings recognised by this method, the test specified below is carried out. Otherwise nullmodel is taken to be the name of a function which is called with arguments (x, ...).

Currently, the following character strings are recognised:

- "wn" Box test for (non-periodic) white noise, simply calls Box.test.

signature(x = "slMatrix", nullmodel = "character") x here is the periodic autocovariance function. This method works similarly to the method for signature `(x = "ANY", nullmodel = "character")`, see its description.

Currently, the following character strings are recognised:
"pwn"  Ljung-Box test for periodic white noise,
"periodicity"  McLeod test for periodic correlation.
signature(x = "numeric", nullmodel = "character")
signature(x = "PeriodicTimeSeries", nullmodel = "character")

Note
TODO: critical values

Author(s)
Georgi N. Boshnakov

See Also
test_piar, pwn_McLeodLjungBox_test periodic_acf1_test

Examples

```r
cu <- pcts(dataFranses1996[, "CanadaUnemployment"])
cu <- window(cu, start = availStart(cu), end = availEnd(cu))

test_piar(cu, 4, 1, sintercept = TRUE)
pcTest(cu, "piar", 4, 1, sintercept = TRUE)

if(require(partsm)){
  # same with LRurpar.test from partsm
  LRurpar.test(cu, list(regular = c(0,0,0), seasonal = c(1,0), regvar = 0), p = 1)
}
```

Pctime

### Petime objects

Description

Class "Pctime" is an S3 class inheriting from the base R datetime class "POSIXct". It has methods for conversion between datetimes and the pcts cycle-season pairs, as well as convenience methods for a few other functions.

Usage

```r
Pctime(x, cycle, ...)

as_Pctime(x, ...)
```

## S3 method for class 'Pctime'
x[i, j, drop = TRUE]
### Details

Pctime represents periodic times with cycle specification contained in attribute "cycle". It is basically datetime (inheriting from "POSIXct") with additional attribute(s).

For printing Pctime objects are shown as cycle-season pairs. To print in other formats, just convert them using `as_datetime` or other suitable function. Note though that some cycles in pcts do not have natural datetime representation. For them, Pctime sets it arbitrarily as the number of seconds from a origin.

The seasons in cycle-season pairs are numbered from one to the number of seasons. Names and abbreviations are used when available and this is the case for all builtin cycles and partial cycles obtained from them.

The cycles in cycle-season pairs are numbered from a starting point. For years, it is what is expected. For cycles representing weeks, week 1 is the first ISO week of 1970, so `c(1,1)` corresponds to 1969-12-29. For some other cycle classes `c(1,1)` also corresponds to the first time in the first ISO week of 1970.

Subsetting with "[]" keeps the Pctime class, while "[[" returns a datetime object. Other standard functions work with Ptime objects, as well, including seq.

A common source of frustration is the accidental use of `as.Date` or `as_date`, instead of `as.POSIXlt` or `as_datetime`. These four are often equivalent, most notably for monthly, quarterly and daily observations but, in general, conversion to dates drops the fractional day part of a datetime.

The default time zone is UTC. Other time zones can be used since the calculations use standard datetime and date functions from base R and package lubridate (Grolemund and Wickham 2011), but currently this has not been tested.

### Value

for Pctime, an object from S3 class Pctime
References


Examples

```r
## a bare bone date for four seasons
pct4 <- Pctime(c(2020, 2), pcCycle(4))
pct4

## quarterly cycle
Pctime("2020-04-01", BuiltinCycle(4))
pctQ <- Pctime(c(2020, 2), BuiltinCycle(4))  # same
pctQ

## day-in-week cycle
## c(1, 1) is the start of the first ISO week of 1970
weekW1S1 <- Pctime(c(1, 1), BuiltinCycle(7))  # W1 Mon
as_datetime(weekW1S1)

Pctime("1970-01-01", BuiltinCycle(7))  # W1 Thu
pctW1Th <- Pctime(c(1, 4), BuiltinCycle(7))  # same
pctW1Th

Pctime("2020-04-01", BuiltinCycle(7))
pctW2623Wed <- Pctime(c(2623, 3), BuiltinCycle(7))  # same
pctW2623Wed
as_datetime(pctW2623Wed)

## Monday-Friday week - a partial cycle derived from DayOfWeekCycle
BuiltinCycle(5)
pctMF <- Pctime("2020-04-03", BuiltinCycle(5))  # Fri
seq(pctMF, length.out = 10)  # note: Sat, Sun are skipped

Pctime("2020-04-04", BuiltinCycle(5))  # Sat, not in the cycle

## monthly cycle
Pctime("2020-04-01", BuiltinCycle(12))
pctY2020Apr <- Pctime(c(50, 4), BuiltinCycle(12))  # same
pctY2020Apr
as_datetime(pctY2020Apr)

## Pctime can hold a vector of times
ap <- pcts(AirPassengers)
aptime <- Pctime(ap)  # as_Pctime(ap)
aptime[1:12]  # keep Pctime class
aptime[1]

aptime[[1]]  # drop Pctime class

head(aptime)
```
pcts

Description

Create objects from periodic time series classes.

Usage

pcts(x, nseasons, start, ..., keep = FALSE)

Arguments

x
  a time series.

nseasons
  number of seasons. This argument is ignored by some methods.

start
  the starting time of the time series, can be a (cycle, season) pair or any object
  that can be converted to datetime.

keep
  if TRUE and x is from class "ts", "mts", "zoo", or "zooreg", create a periodic
  object inheriting from that class.

... further arguments to be passed on to methods.

Details

pcts creates periodic time series objects inheriting from "PeriodicTimeSeries". The particular
class depends on arguments x and, in some cases, keep. The idea is that in normal use the user does
not care about the particular class.

There are also methods for as for conversion to and from the time series classes defined in package
pcts.

Value

an object inheriting from "PeriodicTimeSeries", the defaults are "PeriodicTS" and "PeriodicMTS"

Methods

signature(x = "numeric", nseasons = "missing")
signature(x = "numeric", nseasons = "numeric")
signature(x = "numeric", nseasons = "BasicCycle") Creates an object of class "PeriodicTS",
  the native class for univariate periodic time series in package "pcts".
signature(x = "matrix", nseasons = "missing")
signature(x = "matrix", nseasons = "numeric")
signature(x = "matrix", nseasons = "BasicCycle") Creates an object of class "PeriodicMTS", the native class for multivariate periodic time series in package "pcts".
signature(x = "data.frame", nseasons = "ANY") Currently this just calls pcts(as.data.frame(x), nseasons,...).
signature(x = "ts", nseasons = "missing")
signature(x = "ts", nseasons = "numeric") If keep = TRUE creates an object of class "PeriodicTS_ts", otherwise the result is from "PeriodicTS". The number of seasons is taken from the "ts" object.
signature(x = "mts", nseasons = "missing")
signature(x = "mts", nseasons = "numeric") If keep = TRUE creates an object of class "PeriodicMTS_ts", otherwise the result is from "PeriodicMTS". The number of seasons is taken from the "ts" object.
signature(x = "xtsORzoo", nseasons = "missing") x needs to a regular time series, possibly with missing values for some times (technically, zoo::is.regular(x) should give TRUE). For daily time series, the cycle is taken to be day of week or a subcycle of it, most commonly Monday-Friday.
The implementation is incomplete but for daily data should work as described.

Author(s)
Georgi N. Boshnakov

See Also
PeriodicTS,
dataFranses1996 for further examples

Examples

## convert a ts object, no need for further info
pcts(AirPassengers, 12)

## numeric
v24 <- rnorm(24)
pcts(v24, nseasons = 4) # generic seasons
pcts(v24, nseasons = BuiltinCycle(4)) # Quarter/Year
ts1 <- pcts(v24, nseasons = BuiltinCycle(4), c(2006, 1)) # Quarter/Year with dates

## select subset of the seasons
window(ts1, seasons = 3:4)

## matrix, multivariate pcts
m24 <- matrix(v24, ncol = 3)
colnames(m24) <- c("A", "B", "C")
pcts(m24, nseasons = 4) # generic seasons
pcts(m24, nseasons = BuiltinCycle(4)) # Quarter/Year
mts1 <- pcts(m24, nseasons = BuiltinCycle(4), c(2006, 1)) # Quarter/Year with dates
mts1
## select subset of the seasons for mutivariate
window(mts1, seasons = 3:4)

### Deprecated Functions in Package `pcts`

#### Description

These functions are marked for removal and are provided temporarily for compatibility with older versions of package `pcts` only. Use the recommended renamed or new functions instead.

#### Details

- **mCpar** has been renamed to **sim_parCoef**
- **sim_arAcf** has been renamed to **sim_parAcf**

### pcts_exdata

#### Periodic time series objects for examples

#### Description

Periodic time series objects for examples and tests. These objects are from classes defined in package “pcts” and as a consequence are not suitable for access with `data()`.

#### Usage

```r
pcts_exdata(x, envir = parent.frame())
```

#### Arguments

- **x**: a character vector giving the names of objects. If missing, all available objects will be created. Can also be `NA`. In that case no objects are created and the names of all available objects are returned.
- **envir**: environment where the objects are put, the default is the environment of the caller.

#### Details

The requested objects are created and put in `envir`. Its default is the environment of the caller, which should be sufficient in most use cases.

The following objects are currently available: `ap`, `ap7to9`, `pcfr`, `pcfr2to4`. 
Value

if \( x \) is NA, the names of the available objects. Otherwise the function is called for the side effect of creating objects in \texttt{envir} and the return value (the names of the created objects) is usually discarded.

See Also

\texttt{dataFranses1996}

Examples

```r
## the objects are created with something like:
ap <- pcts(AirPassengers)
ap7to9 <- window(ap, seasons = 7:9)

pcfr <- pcts(dataFranses1996)
pcfr2to4 <- pcfr[2:4]
```

---

### Description

Compute a matrix of factors such that elementwise division of the periodic autocovariance matrix by it will give the periodic autocorrelations.

### Usage

```
pc_sdfactor(sd, maxlag)
```

### Arguments

- `sd`: standard deviations of the seasons numeric.
- `maxlag`: maximal lag, a number.

### Value

a matrix of coefficients of size \texttt{period x (maxlag+1)}. The length of \texttt{sd} is taken to be the period.

### Author(s)

Georgi N. Boshnakov

### See Also

\texttt{autocorrelations}
pdSafeParOrder

Functions for some basic operations with seasons

Description

Functions for some basic operations with seasons.

Usage

pdSafeParOrder(p)

Arguments

p

autoregression order, a vector of integers.

Details

pdSafeParOrder(p) modifies the periodic AR order specified by vector p. The modified order is such that the correspondence between autocovariances and partial autocorrelations is one-to-one, see the references for details.

Value

a vector of integers

Author(s)

Georgi N. Boshnakov

References


PeriodicArmaModel-class

Description

Class PeriodicArmaModel.

Objects from the Class

Objects can be created by calls of the form new("PeriodicArmaModel", ar, ma, sigma2, ...).

PeriodicArmaFilter-class

Class "PeriodicArmaFilter"

Description

Class PeriodicArmaFilter.

Objects from the Class

Objects can be created by calls of the form new("PeriodicArmaFilter", ..., ar, ma, nseasons).

Slots

ar: Object of class "PeriodicBJFilter" ~
ma: Object of class "PeriodicSPFilter" ~

Extends

Class "VirtualArmaFilter", directly. Class "VirtualMonicFilter", by class "VirtualArmaFilter", distance 2.

Methods

No methods defined with class "PeriodicArmaFilter" in the signature.
Slots

sigma2: Object of class "numeric" ~
ar: Object of class "PeriodicArFilter" ~
ma: Object of class "PeriodicMaFilter" ~
center: Object of class "numeric" ~
intercept: Object of class "numeric" ~

Extends


Methods

autocovariances signature(x = "PeriodicArmaModel"): ...
PeriodicArModel-class  

Class PeriodicArModel.

Description

Class PeriodicArModel.

Objects from the Class

Objects can be created by calls of the form `new("PeriodicArModel", ar, ma, sigma2, ...)`. 

Slots

- `sigma2`: Object of class "numeric"
- `ar`: Object of class "PeriodicArFilter"
- `ma`: Object of class "PeriodicMaFilter"
- `center`: Object of class "numeric"
- `intercept`: Object of class "numeric"

Extends


Methods

- `autocovariances` signature(x = "PeriodicArModel"): ...
- `fitPM` signature(x = "ANY", model = "PeriodicArModel"): ...

Create objects from class PeriodicArModel

Description

Create objects from class PeriodicArModel

Usage

`PeriodicArModel(object, ...)"
Arguments

object  an object, can have one of a number of classes.
...  further arguments for methods.

Details

PeriodicArModel creates objects from class PeriodicArModel. This is a generic function with
despach methods on the first argument.

Value

an object from class "PeriodicArModel"

Methods

signature(object = "matrix")  "object" gives the coefficients, one row per season.
signature(object = "numeric")  "object" gives the model order. Its length is taken to be the
number of seasons. The coefficients are set to NA.
signature(object = "PeriodicMonicFilterSpec")
signature(object = "VirtualPeriodicArmaModel")
signature(object = "PeriodicMonicFilterSpec")
signature(object = "VirtualPeriodicArmaModel")

Description

Class PeriodicAutocorrelations.

Objects from the Class

Objects can be created by calls of the form new("PeriodicAutocorrelations",...,data).

Slots

modelCycle:  Object of class "BasicCycle" ~~
data:  Object of class "Lagged" ~~

Extends

Class "ModelCycleSpec", directly. Class "FlexibleLagged", directly. Class "VirtualPeriodicAutocorrelations",
directly. Class "Lagged", by class "FlexibleLagged", distance 2. Class "VirtualPeriodicModel",
by class "VirtualPeriodicAutocorrelations", distance 2.
Methods

plot signature(x = "PeriodicAutocorrelations", y = "missing"): ...

PeriodicAutocovarianceModel-class

Class PeriodicAutocovarianceModel

Description

Class PeriodicAutocovarianceModel.

Objects from the Class

Objects can be created by calls of the form new("PeriodicAutocovarianceModel", ...).

Slots

pcmatrix: Object of class "ANY" ~

Extends

Class "VirtualPeriodicAutocovarianceModel", directly. Class "PeriodicAutocovarianceSpec", directly.

Methods

autocovariances signature(x = "PeriodicAutocovarianceModel"): ...

[ signature(x = "PeriodicAutocovarianceModel", i = "missing", j = "missing", drop = "ANY"): ...

[ signature(x = "PeriodicAutocovarianceModel", i = "missing", j = "numeric", drop = "ANY"): ...

[ signature(x = "PeriodicAutocovarianceModel", i = "numeric", j = "missing", drop = "ANY"): ...

[ signature(x = "PeriodicAutocovarianceModel", i = "numeric", j = "numeric", drop = "ANY"): ...

<- signature(x = "PeriodicAutocovarianceModel", i = "ANY", j = "ANY", value = "ANY"): ...

autocovariances signature(x = "PeriodicAutocovarianceModel", maxlag = "ANY"): ...

backwardPartialCoefficients signature(x = "PeriodicAutocovarianceModel"): ...

backwardPartialVariances signature(x = "PeriodicAutocovarianceModel"): ...

maxLag signature(object = "PeriodicAutocovarianceModel"): ...

partialAutocorrelations signature(x = "PeriodicAutocovarianceModel", maxlag = "ANY", lag_0 = "ANY"): ...

partialAutocovariances signature(x = "PeriodicAutocovarianceModel"): ...
### Class PeriodicAutocovariances

**Description**

Class PeriodicAutocovariances.

**Objects from the Class**

Objects can be created by calls of the form `new("PeriodicAutocovariances",...,data)`.

**Slots**

- `modelCycle`: Object of class "BasicCycle" ~
- `data`: Object of class "Lagged" ~

**Extends**


**Methods**

- **autocorrelations** signature(x = "PeriodicAutocovariances", maxlag = "ANY", lag_0 = "missing"): ...

- **partialAutocorrelations** signature(x = "PeriodicAutocovariances", maxlag = "ANY", lag_0 = "missing"): ...
PeriodicAutocovarianceSpec-class

Class PeriodicAutocovarianceSpec

Description

Class PeriodicAutocovarianceSpec.

Objects from the Class

Objects can be created by calls of the form new("PeriodicAutocovarianceSpec", ...).

Slots

pmatrix: Object of class "ANY" ~

Methods

No methods defined with class "PeriodicAutocovarianceSpec" in the signature.

PeriodicBJFilter-class

Class PeriodicBJFilter

Description

A class for filters following the Box-Jenkins sign convention

Objects from the Class

Objects can be created by calls of the form new("PeriodicBJFilter", coef, order, ...).

Slots

coef: Object of class "matrix" ~

order: Object of class "numeric" ~

Extends

Methods

filterCoef signature(object = "PeriodicBJFilter", convention = "character"): ...
coerce signature(from = "matrix", to = "PeriodicBJFilter"): ...
coerce signature(from = "PeriodicBJFilter", to = "PeriodicSPFilter"): ...
coerce signature(from = "PeriodicSPFilter", to = "PeriodicBJFilter"): ...
filterPoly signature(object = "PeriodicBJFilter"): ...
filterPolyCoef signature(object = "PeriodicBJFilter"): ...
show signature(object = "PeriodicBJFilter"): ...

Author(s)

Georgi N. Boshnakov

See Also

PeriodicSPFilter

Examples

## a toy filter of order c(3, 3, 3, 3) and 4 seasons
co <- matrix(c(1, 1, 0,
               2, 2, 2,
               3, 0, 0,
               4, 4, 4), nrow = 4, ncol = 3)

## these are equivalent:
bj1 <- new("PeriodicBJFilter", coef = co)
bj1b <- new("PeriodicBJFilter", coef = co, order = 3)
bj1c <- new("PeriodicBJFilter", coef = co, order = c(3, 3, 3, 3))
identical(bj1b, bj1c) # TRUE
identical(bj1, bj1b) # FALSE but only because classbj1@order is "integer"

## a more refined spec. for the order:
show( new("PeriodicBJFilter", coef = co, order = c(2, 3, 1, 3)) )

## as()
show( as(co, "PeriodicBJFilter") )
show( as(co, "PeriodicSPFilter") )

## change the sign convention:
spl <- as(bj1, "PeriodicSPFilter")

## bj1 and spl represent the same filter
filterPolyCoef(bj1)
filterPolyCoef(spl)

identical(filterPolyCoef(bj1), filterPolyCoef(spl)) # TRUE
filterOrder(bj1)
nSeasons(bj1)

## filterPoly(bj1)
## filterPoly(sp1)

---

PeriodicFilterModel-class

Class PeriodicFilterModel

Description

Class PeriodicFilterModel.

Objects from the Class

Objects can be created by calls of the form `new("PeriodicFilterModel", pcmean, pciintercept,...)`.

Slots

center: Object of class "numeric" ~~
intercept: Object of class "numeric" ~~
sigma2: Object of class "numeric" ~~
ar: Object of class "PeriodicArFilter" ~~
ma: Object of class "PeriodicMaFilter" ~~

Extends


---

PeriodicIntegratedArmaSpec-class

Class PeriodicIntegratedArmaSpec

Description

Class PeriodicIntegratedArmaSpec.

Objects from the Class

Objects can be created by calls of the form `new("PeriodicIntegratedArmaSpec",...)`.

Slots

pcmodel: Object of class "PeriodicArmaModel" ~~
**PeriodicInterceptSpec-class**

Class PeriodicInterceptSpec

**Description**

Class PeriodicInterceptSpec.

**Objects from the Class**

Objects can be created by calls of the form `new("PeriodicInterceptSpec", ...)`.

**Slots**

- `center`: Object of class "numeric"
- `intercept`: Object of class "numeric"
- `sigma2`: Object of class "numeric"
- `modelCycle`: Object of class "BasicCycle"

**Extends**


Class "numLike", by class "numeric", distance 2. Class "number", by class "numeric", distance 2.

Class "atomicVector", by class "numeric", distance 2.

**Methods**

No methods defined with class "PeriodicInterceptSpec" in the signature.

- `sigmaSq` signature(object = "PeriodicInterceptSpec"): ...
- `allSeasons` signature(x = "PeriodicInterceptSpec", abb = "ANY"): ...
- `initialize` signature(.Object = "PeriodicInterceptSpec"): ...
- `nSeasons` signature(object = "PeriodicInterceptSpec"): ...
- `show` signature(object = "PeriodicInterceptSpec"): ...
**PeriodicMaModel-class**

*Class PeriodicMaModel*

**Description**

Class PeriodicMaModel.

**Objects from the Class**

Objects can be created by calls of the form `new("PeriodicMaModel",ar,ma,sigma2,...)`.

**Slots**

- `sigma2`: Object of class "numeric"
- `ar`: Object of class "PeriodicArFilter"
- `ma`: Object of class "PeriodicMaFilter"
- `center`: Object of class "numeric"
- `intercept`: Object of class "numeric"

**Extends**


**Methods**

No methods defined with class "PeriodicMaModel" in the signature.

---

**PeriodicMTS-class**

*Class "PeriodicMTS"

**Description**

Class "PeriodicMTS" is the main class for multivariate periodic time series in package "pcts".

**Objects from the Class**

Objects can be created by calls of the form `new("PeriodicMTS",...)` but it is recommended to use the function `pcts` in most cases.
Slots

.Data: Object of class "matrix" ~
cycle: Object of class "BasicCycle" ~
pcstart: Object of class "ANY" ~

Extends


Methods

$ signature(x = "PeriodicMTS"): ...
[ signature(x = "PeriodicMTS", i = "ANY", j = "missing", drop = "ANY"): ...
[ signature(x = "PeriodicMTS", i = "missing", j = "missing", drop = "ANY"): ...
[[ signature(x = "PeriodicMTS", i = "ANY"): ...
coerce signature(from = "mts", to = "PeriodicMTS"): ...
coerce signature(from = "PeriodicMTS", to = "ts"): ...
coerce signature(from = "ts", to = "PeriodicMTS"): ...
plot signature(x = "PeriodicMTS", y = "missing"): ...
show signature(object = "PeriodicMTS"): ...
summary signature(object = "PeriodicMTS"): ...
[ signature(x = "PeriodicMTS", i = "ANY", j = "ANY", drop = "ANY"): ...
[ signature(x = "PeriodicMTS", i = "AnyDateTime", j = "ANY", drop = "ANY"): ...
[ signature(x = "PeriodicMTS", i = "AnyDateTime", j = "missing", drop = "ANY"): ...
[[ signature(x = "PeriodicMTS", i = "ANY", j = "ANY"): ...
fitPM signature(model = "PeriodicArModel", x = "PeriodicMTS"): ...
pcApply signature(object = "PeriodicMTS"): ...
pcMean signature(object = "PeriodicMTS"): ...

See Also

pcts which is the recommended way to create periodic time series,
class PeriodicTS for the univariate case,
dataFranses1996 and pcts-package for examples
Examples

```r
pcfr <- pcts(dataFranses1996)
colnames(pcfr)[4] # "GermanyGNP"

## extracting single time series as univariate
class(pcfr[[4]]) # "PeriodicTS"
identical(pcfr[[4]], pcfr$GermanyGNP ) # TRUE
identical(pcfr[[4]], pcfr[["GermanyGNP"]]) # TRUE
plot(pcfr[[4]])

## ... and as multivariate
pcfr[4] # "PeriodicMTS"
plot(pcfr[4])

## extracting more than one time series
plot(pcfr[2:4])
summary(pcfr[2:4])

pcfr2 <- pcfr[[2]]
plot(pcfr2)
```

---

### PeriodicMTS_ts-class

**Class** "PeriodicMTS_ts"

---

### Description

Class "PeriodicMTS_ts" is a periodic class holding multivariate "ts" objects.

### Objects from the Class

Objects can be created by calls of the form `new("PeriodicMTS_ts", x,...).`

### Slots

- `.Data`: Object of class "vector"
- `cycle`: Object of class "BasicCycle"
- `pcstart`: Object of class "ANY"
- `tsp`: Object of class "numeric"
- `.S3Class`: Object of class "character"

### Extends

PeriodicMTS_zooreg-class

Methods

- **coerce** signature(from = "ts", to = "PeriodicMTS_ts"): ...
- **initialize** signature(.Object = "PeriodicMTS_ts"): ...

Description

Class "PeriodicMTS_zooreg" is a periodic class holding multivariate "zooreg" objects.

Objects from the Class

Objects can be created by calls of the form new("PeriodicMTS_zooreg",...).

Slots

- `cycle`: Object of class "BasicCycle" ~~
- `.S3Class`: Object of class "character" ~~
- `pcstart`: Object of class "ANY" ~~

Extends


Methods

No methods defined with class "PeriodicMTS_zooreg" in the signature.

PeriodicSPFilter-class

Description

A class for filters following the signal processing sign convention.

Objects from the Class

Objects can be created by calls of the form new("PeriodicSPFilter", coef, order,...).
Slots

   coef: Object of class "matrix" ~~
   order: Object of class "numeric" ~~

Extends


Methods

   filterCoef signature(object = "PeriodicSPFilter", convention = "character"):

Author(s)

Georgi N. Boshnakov

See Also

   PeriodicBJFilter

Examples

   ## see "PeriodicBJFilter-class" for examples

---

Class PeriodicTimeSeries

Class PeriodicTimeSeries-

Description

Class PeriodicTimeSeries.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

   cycle: Object of class "BasicCycle".
   pcstart: Object of class "ANY".

Extends

Class "Cyclic", directly.
PeriodicTS-class

Methods

    pcTest signature(x = "PeriodicTimeSeries", nullmodel = "character"): ...
    head signature(x = "PeriodicTimeSeries"): ...
    pcCycle signature(x = "PeriodicTimeSeries", type = "ANY"): ...
    tail signature(x = "PeriodicTimeSeries"): ...

---

PeriodicTS-class  Class "PeriodicTS"

Description

Class "PeriodicTS" is the main class for univariate periodic time series in package "pcts".

Objects from the Class

Objects can be created by calls of the form new("PeriodicTS", ...).

Slots

  .Data: Object of class "numeric" ~~
  cycle: Object of class "BasicCycle" ~~
  pcstart: Object of class "ANY" ~~

Extends


Methods

    coerce signature(from = "mts", to = "PeriodicTS"): ...
    coerce signature(from = "PeriodicTS", to = "ts"): ...
    coerce signature(from = "ts", to = "PeriodicTS"): ...
    plot signature(x = "PeriodicTS", y = "missing"): ...
    show signature(object = "PeriodicTS"): ...
    summary signature(object = "PeriodicTS"): ...
PeriodicTS_ts-class

Class "PeriodicTS_ts"

Description

Class "PeriodicTS_ts" is a periodic class holding "ts" objects.

Objects from the Class

Objects can be created by calls of the form new("PeriodicTS_zooreg",...).

Objects from the Class

Objects can be created by calls of the form new("PeriodicTS_ts",x,...).

Slots

>Data: Object of class "vector" ~
cycle: Object of class "BasicCycle" ~
tsp: Object of class "numeric" ~
.S3Class: Object of class "character" ~
pcstart: Object of class "ANY" ~

Extends


Methods

coerce signature(from = "ts",to = "PeriodicTS_ts"): ...
initialize signature(.Object = "PeriodicTS_ts"): ...

PeriodicTS_zooreg-class

Class "PeriodicTS_zooreg"

Description

Class "PeriodicTS_zooreg" is a periodic class holding "zooreg" objects.

Objects from the Class

Objects can be created by calls of the form new("PeriodicTS_zooreg",...).
Slots

cycle: Object of class "BasicCycle" ~~
.S3Class: Object of class "character" ~~
pcstart: Object of class "ANY" ~~

Extends


Methods

No methods defined with class "PeriodicTS_zooreg" in the signature.

See Also

classes PeriodicTS and PeriodicMTS

Description

Objects and methods for class PeriodicVector.

Usage

PeriodicVector(x, period = length(x))

Arguments

x the values for indices from 1 to period, numeric.
period the period, defaults to length(x).

Details

A $p$-periodic vector, $X$, is such that $X_{i+pk} = X_i$ for any integers $i, k$.

Class PeriodicVector stores the values of $X_1, \ldots, X_p$ and provides indexing methods for extracting and setting its elements.

Value

an object from class "PeriodicVector"
Objects from the Class

Objects can be created by calls of the form `new("PeriodicVector",...)` or more conveniently by using `"PeriodicVector()"`.

Slots

.Data: Object of class "numeric"
.period: Object of class "numeric"

Extends

Class "numLike", by class "numeric", distance 2.
Class "number", by class "numeric", distance 2. Class "replValue", by class "numeric", distance 2.

Methods

"PeriodicVector" methods are defined for "[" and "[<-". Arithmetic operations just inherit the recycling rules from "numeric".

[ signature(x = "PeriodicVector",i = "ANY",j = "ANY",drop = "ANY"): ...
[ signature(x = "PeriodicVector",i = "ANY",j = "missing",drop = "ANY"): ...
[ signature(x = "PeriodicVector",i = "missing",j = "ANY",drop = "ANY"): ...
[ signature(x = "PeriodicVector",i = "missing",j = "missing",drop = "ANY"): ...
[<- signature(x = "PeriodicVector",i = "ANY",j = "ANY",value = "ANY"): ...
[<- signature(x = "PeriodicVector",i = "missing",j = "ANY",value = "ANY"): ...

Author(s)

Georgi N. Boshnakov

See Also

PeriodicVector

Examples

```r
PeriodicVector(1:4, period = 4)
PeriodicVector(1:4) ## same
new("PeriodicVector", 1:4, period = 4)

## if period is given but x is missing, the vector is filled with NA's
PeriodicVector(period = 4)

## this throws error, since length(x) != period:
##   PeriodicVector(1:3, period = 4)
```
## extract
x <- PeriodicVector(1:4)
x[3:12]
x[c(3, 7, 11, 15)]

# any indices in (-Inf, Inf) work
x[0]
x[-3:0]

## "[<-" works on the underling vector
x[1] <- 11; x

## modulo indexing works also in assignments:
x[5] <- 21; x

## empty index returns the underlying vector
x[]

## the recycling rule applies on assignment
x[] <- 9; x
x[] <- 1:2; x

## this gives warning, as for numeric vectors
##   x[] <- 8:1
## compare:
##   x <- 1:4
##   x[] <- 8:1

## arithmetic works as usual:
2 * x
x + 1:4
## x + 1:3 # warning - '... a multiple ...'

---

### periodic_acf1_test

**McLeod's test for periodic autocorrelation**

**Description**

Performs McLeod's test for periodic autocorrelation.

**Usage**

periodic_acf1_test(acf, nepochs)

**Arguments**

- **acf**: sample periodic autocorrelation function
- **nepochs**: the number of epochs used to get the acf
Details
The test statistic is a scaled sum of squares of lag 1 sample periodic autocorrelation coefficients, see McLeod (1993), eq. (5). The distribution is approximately chi-square under the null hypothesis of no periodic autocorrelation.

Value
A list containing the following components:

- statistic: the value of the test statistic.
- pvalue: the p-value associated with the test statistic.

Author(s)
Georgi N. Boshnakov

References


permean2intercept

Convert between periodic centering and intercepts

Description
Convert a periodic mean to periodic intercept and vice versa.

Usage

```
permean2intercept(mean, coef, order, nseasons = nrow(coef))
```

```
intercept2permean(intercept, coef, order, nseasons = nrow(coef))
```

Arguments

- **mean**: periodic mean, numeric.
- **coef**: PAR coefficients, matrix.
- **order**: PAR order, vector of positive integers.
- **nseasons**: number of seasons, a.k.a. period.
- **intercept**: periodic intercepts, numeric.
**Details**

A PAR model can be written in mean corrected or intercept form. `permean2intercept` calculates the intercepts from the means, while `intercept2permean` does the inverse (means from intercepts).

No check is made for periodic stationarity of the model. Converting from mean corrected to intercept form always succeeds and in fact the means do not need to be means. In the opposite direction there may be problems due to unit roots and similar features.

**Value**

a numeric vector

**Author(s)**

Georgi N. Boshnakov

**Examples**

```r
mu <- c(1, 2)
pm1 <- PeriodicArModel(matrix(c(0.5, 0.5), nrow = 2), order = rep(1, 2), sigma2 = 1, mean = mu)

cc <- permean2intercept(mu, pm1@ar@coef, c(1,1))
cc

intercept2permean(cc, pm1@ar@coef, c(1,1))

d <- 4
mu <- 1:d
c0 <- rep(0.5, d)
pm1 <- PeriodicArModel(matrix(c0, nrow = d), order = rep(1, d), sigma2 = 1, mean = mu)

cc <- permean2intercept(mu, pm1@ar@coef, order = rep(1, d))
cc

intercept2permean(cc, pm1@ar@coef, order = rep(1, d) )
```

---

**permodelmf**

*Compute the multi-companion form of a per model*

**Description**

Compute the multi-companion form of a per model.

**Usage**

`permodelmf(permodel, update = TRUE)`
Arguments

permodel a model.
update If TRUE store the multi-companion form in permodel and return the whole model, otherwise simply return the multi-companion form.

Details
todo:

Value

the multi-companion form of the model or the updated model, as described in Details.

Author(s)

Georgi N. Boshnakov

References


pi1ar2par Convert PIAR coefficients to PAR coefficients

Description

Convert PIAR coefficients to PAR coefficients

Usage

pi1ar2par(picoef, parcoef)
piar2par(picoef, parcoef)

Arguments

picoef coefficients of the periodic integration filter, a numeric vector or matrix, see Details.
parcoef coefficients of the periodically correlated part of the model.
**PiPeriodicArmaModel-class**

**Description**

Class PiPeriodicArmaModel.

**Objects from the Class**

Objects can be created by calls of the form `new("PiPeriodicArmaModel",...)`. 

**Details**

`pi1ar2par` converts PIAR(1) model coefficients specified as a set of coefficients corresponding to a periodic unit root and PAR coefficients to coefficients for a single filter.

`piar2par` does the same but admits higher order periodic integration.

`picoef` is a matrix, specifying one or more first order periodic unit root filters. Each column contains the coefficients of one filter. If there is only one filter, its coefficients can be given as a numeric vector.

The filters are applied from right to left, in the sense that first the PAR filter is applied to the time series, then the filter specified by the last column and so on.

Note that unlike the stationary case, in general, this is not equivalent to multiplication of the corresponding polynomials. This is rather different from the stationary case, where the analogous operation boils down just to multiplying the corresponding polynomials.

**Value**

a matrix, whose i-th row contains the coefficients for the i-th season.

**Author(s)**

Georgi N. Boshnakov

**Examples**

```r
## Lina's example
parcoef <- rbind(c(0.5, -0.06), c(0.6, -0.08), 
                 c(0.7, -0.1), c(0.2, 0.15) )
picoef1 <- c(0.8, 1.25, 2, 0.5)
parcoef2 <- pi1ar2par(picoef1, parcoef)
picoef2 <- c(4, 0.25, 5, 0.2)
coefper2I2 <- pi1ar2par(picoef2, parcoef2)
```
Slots

piorder: Object of class "numeric" 

picoef: Object of class "matrix" 

pcmodel: Object of class "PeriodicArmaModel" 

Extends

Class "VirtualPeriodicFilterModel", directly. Class "PeriodicIntegratedArmaSpec", directly.

Methods

No methods defined with class "PiPeriodicArmaModel" in the signature.

---

### PiPeriodicArModel-class

**Class PiPeriodicArModel**

Description

Class PiPeriodicArModel.

Objects from the Class

Objects can be created by calls of the form `new("PiPeriodicArModel",...).`

Slots

piorder: Object of class "numeric" 

picoef: Object of class "matrix" 

pcmodel: Object of class "PeriodicArmaModel" 

Extends


Methods

`fitPM` signature(`x = "ANY",model = "PiPeriodicArModel": ...`)
Class PiPeriodicMaModel

Description

Class PiPeriodicMaModel.

Objects from the Class

Objects can be created by calls of the form new("PiPeriodicMaModel",...).

Slots

piorder: Object of class "numeric" ~-
picoef: Object of class "matrix" ~-
pcmodel: Object of class "PeriodicArmaModel" ~-

Extends


Methods

No methods defined with class "PiPeriodicMaModel" in the signature.

pwn_McLeodLjungBox_test

McLeod-Ljung-Box test for periodic white noise

Description

Compute the McLeod-Ljung-Box test statistic for examining the null hypothesis of periodic white noise.

Usage

pwn_McLeodLjungBox_test(acf, nepoch, use = 1:maxlag,
                         maxlag = ncol(as.matrix(acf)) - 1,
                         period = nrow(as.matrix(acf)), fitdf = numeric(period))
Arguments

acf    the sample periodic autocorrelation function of the time series.
nepoch number of cycles used in computing the acf.
use    number of lags to use, may be a vector.
maxlag maximal lag.
period number of seasons in a cycle.
fitdf  degrees of freedom corrections for the number of estimated parameters, see Details.

Details

The McLeod-Ljung-Box test can be used to test the null hypothesis of periodic white noise. If acf contains sample autocorrelations of residuals from a fitted model, a correction of the degrees of freedom is strongly recommended.

Argument fitdf is a vector specifying how may degrees of freedom to subtract for each season. In the case of PAR models fitdf can be set to the PAR orders.

The value of the statistic is set to NA where the correction for degrees of freedom results in negative numbers.

Value

A list containing the following components:

  statistic the value of the test statistic for each lag specified by use.
  df         the corresponding degrees of freedom

Note

TODO: Consolidate this and similar tests!

Author(s)

Georgi N. Boshnakov

References


See Also

Box.test for the non-periodic case
Class SamplePeriodicAutocorrelations

Description

Class SamplePeriodicAutocorrelations.

Objects from the Class

Objects can be created by calls of the form `new("SamplePeriodicAutocorrelations",...,data)`.

Slots

- `modelCycle`: Object of class "BasicCycle" ~
- `data`: Object of class "Lagged" ~
- `n`: Object of class "numeric" ~
- `varnames`: Object of class "character" ~
- `objectname`: Object of class "character" ~

Extends


Methods

No methods defined with class "SamplePeriodicAutocorrelations" in the signature.

Class SamplePeriodicAutocovariances

Description

Class SamplePeriodicAutocovariances.

Objects from the Class

Objects can be created by calls of the form `new("SamplePeriodicAutocovariances",...,data)`.
seqSeasons-methods

Slots

modelCycle: Object of class "BasicCycle"

data: Object of class "Lagged"

n: Object of class "numeric"

varnames: Object of class "character"

objectname: Object of class "character"

Extends


Methods

autocorrelations signature(x = "SamplePeriodicAutocovariances", maxlag = "ANY", lag_0 = "missing"):

Description

Methods for seqSeasons() in package pcts.

Methods

signature(x = "BasicCycle")

signature(x = "Cyclic")

signature(x = "VirtualPeriodicModel")

See Also

allSeasons for related functions and examples
Methods for `sigmaSq` in package `pcts`.

**Methods**

```r
signature(object = "PeriodicIntegratedArmaSpec")
signature(object = "PeriodicInterceptSpec")
```

**SimpleCycle-class**

Class `SimpleCycle`.

**Description**

Class `SimpleCycle`.

**Objects from the Class**

Objects can be created by calls of the form `new("SimpleCycle", nseasons, seasons, first)`.

In addition to number of seasons, class `"SimpleCycle"` holds also seasons’ names and the index of the season to be treated as the first in a cycle.

**Slots**

- `seasons`: Object of class "character", the names of the seasons.
- `nseasons`: Object of class "integer", number of seasons.
- `cycle`: Object of class "character" ~~
- `season`: Object of class "character" ~~
- `abbreviated`: Object of class "character" ~~

**Extends**

Class "BareCycle", directly. Class "BasicCycle", directly.

**Methods**

No methods defined with class "SimpleCycle" in the signature.

**Author(s)**

Georgi N. Boshnakov
**See Also**

- `pcCycle` for creation of cycle objects and extraction of cycle part of time series,
- `BuiltinCycle-class`, `SimpleCycle-class`,
- `DayWeekCycle-class`, `MonthYearCycle-class`, `OpenCloseCycle-class`, `QuarterYearCycle-class`,
- `PartialCycle-class`

**Examples**

```r
showClass("SimpleCycle")
```

---

### sim_parAcvf

*Create a random periodic autocovariance function*

**Description**

Select randomly a periodic autoregression model and return the periodic autocovariances associated with it.

**Usage**

```r
sim_parAcvf(period, order, sigma2)
```

**Arguments**

- `period` the period, a positive integer.
- `order` the AR order, a vector of non-negative integers.
- `sigma2` the variances of the innovations, a numeric vector of length `period` (todo: or one?).

**Details**

Uses `sim_parCoef()` to generate a random PAR model.

**Value**

An object of class "matrix". In addition, the specification of the model is in attribute "model" which is a list with the following components:

- `ar` a matrix, the coefficients of the PAR model,
- `sigma2` numeric, the innovation variances,
- `order` the PAR order.

**Author(s)**

Georgi N. Boshnakov
References


Examples

```r
sim_parAcvf(2, 5)
sim_parAcvf(3, 5)

res <- sim_parAcvf(2, 6)
res
slMatrix(res)[3, 4, type = "tt"]

res <- sim_parAcvf(2, 4)
attr(res, "model")
acv <- res[, ] # drop attributes

acv[2, 1 + 0]
acv[2, 1 + 1]
slMatrix(acv)[2, 0]
slMatrix(acv)[2, 1]
slMatrix(acv)[3, 4, type = "tt"]
slMatrix(acv)[1:2, 1:2, type = "tt"]
slMatrix(acv)[1:4, 1:4, type = "tt"]

## TODO: need method for autocorrelation()
## pc.acrf(acv)

## TODO: these need changing, after the change of the return values of sim_parAcvf
## pc.fcoeffs(acv, 2)
## pc.fcoeffs(acv, 3)
## pc.fcoeffs(acv, 4)
pcts:::calc_predictionCoefficients(acv, c(2, 2))
pcts:::calc_predictionCoefficients(acv, c(3, 3))
pcts:::calc_predictionCoefficients(acv, c(4, 4))
```

---

**sim_parCoef**

*Generate a periodic autoregression model*

**Description**

Generate a periodic autoregression model, possibly integrated.

**Usage**

```r
sim_parCoef(period, n.root, sigma2 = rep(1, period), ...)
```
Arguments

- **period**: number of seasons in a cycle.
- **n.root**: number of non-zero roots, see details.
- **sigma2**: variances of the innovations.
- ...: additional arguments to be passed down to `sim_pcfilter`

Details

`sim_parCoef` uses the multi-companion method to generate the model. The function is essentially a wrapper for `sim_pcfilter`.

The order of the filter is set to `n.root` for each season. Part of the spectral information may be specified with the "..." arguments, see `sim_pcfilter` and `sim_mc` for a discussion of this.

Value

A periodic autoregression model as a list with elements:

- **ar**: a matrix whose $i$th row contains the coefficients for the $i$ season,
- **sigma2**: the innovation variances, a numeric vector.

Author(s)

Georgi N. Boshnakov

References


See Also

`sim_pcfilter`

Examples

```r
sim_parCoef(2, 4)  # 2 seasons
sim_parCoef(2, 4, sigma2 = c(2, 4))
sim_parCoef(2, 1)
sim_parCoef(4, 2)  # 4 seasons

sim_parCoef(period = 4, n.root = 6,
             eigabs = c(1, 1, 1, 0.036568887, 0.001968887),
             type.eigval = c("cp", "r", "r", "r", "r"),
             eigsign = c(pi/2, 1, -1, 1, -1))
```
**sim_pc**  
*Simulate Periodically Correlated ARMA Series*

**Description**

Simulate a realization of a periodically correlated arma model or a continuation of an existing series. Initial values may be given too.

**Usage**

```r
sim_pc(model, n = NA, randgen = rnorm, seasonof1st = 1, nepochs = NA,  
n.start = NA, x, eps, nmean = NULL, nintercept = NULL, ...)
```

**Arguments**

- `model`: a list with elements `phi`, `theta`, `p`, `q`, `period`, `mean`, `intercept`, specifying the model.
- `n`: length of the series.
- `randgen`: random number generator as required by `sim_pwn`.
- `seasonof1st`: season of the first value.
- `nepochs`: number of epochs; if `nepochs` is given, then `n` is computed as `nepochs * period`.
- `n.start`: burn-in number; generate `n.start + n` observations and discard the first `n.start` of them, see Details.
- `x`: initial or before values, see Details.
- `eps`: innovations, see Details.
- `nmean`: a vector of length `n` of means, see Details.
- `nintercept`: a vector of length `n` of intercepts, see Details.
- `...`: any additional arguments to be passed on to `sim_pwn`.

**Details**

Argument `x` can be used to specify two types of initialisation values - ‘before’ and ‘init’. They are used similarly in computations but ‘before’ values are not included in the result, while ‘init’ values are (unless dropped due to `n.start`). ‘Before’ values provide a convenient way to simulate continuation trajectories for a time series, for example for simulation based prediction intervals.

If `x` is "numeric", it represents ‘before’ values. Alternatively, `x` can be a list with components "before" and "init".

Innovations are usually generated with the random number generator specified by `randgen` (with default `rnorm`) and the `...` parameters by a call to the function `sim_pwn`, see the documentation for `sim_pwn` for various ways to control the distribution of the generated sequence.

The innovations can also be generated in advance and supplied using argument `eps`. If `eps` is numeric, it is taken to represent the innovations. Alternatively, `eps` can be a list with the innovations.
in component "main". This list may also contain components "before" and/or "init" specifying 'before' or 'initial' values, with interpretation as for x.

nintercept can be used to specify trend representing the effect of time and/or covariates. As for eps, if it is numeric it is taken to represent the main values. It can also be a list with components before, init, and main.

To avoid ambiguity, let's reiterate that before values are past values of the corresponding quantity (before the start of the simulated series), while init values are "initial" values. In particular, if initial values are specified for x, these will form the start of the generated series (unless n.start leads to them being discarded).

If before values are specified for the series and the innovations, then they play a role analogous to that of initial values, so it does not make much sense to supply also initial values.

The function effectively does the following. innov is generated if not supplied, a vector of innovations is created eps <- c(innovbefore, innovinit, innov), a vector x is created of the same length as eps, and initialised with xbefore and xinit. If there are no initial or before values, these are assumed to be 0. The remaining values of x are filled using the pc-arma equations. Finally, the xbefore values are discarded as well as the first n.start values.

n.start should usually be a multiple of the period since otherwise the first observation in the returned vector will not correspond to seasonof1st.

sim_pc deals mainly with the interpretation of the parameters. The actual computations are done by pc.filter. Moreover, sim_pc does not look at the model. It knows only about model$period and uses it to compute n if n is not specified. (It probably should not care even about this.)

Value

numeric, the simulated time series

To do

option to return the innovation sequence; option to include the before values.

option to return the season of the first value in the returned series (it may be different from seasonof1st due to n.start).

Author(s)

Georgi N. Boshnakov

See Also

sim_pwn, pc.filter

Examples

m1 <- rbind( c(1, 0.81, 0), c(1, 0.4972376, 0.4972376) )
testphi <- slMatrix( init = m1 )

m2 <- rbind( c(1, 0, 0), c(1, 0, 0) )
testtheta <- slMatrix( init = m2 )
## phi and theta are slMatrix here.
m1 <- list(phi = testphi, theta = testtheta, p = 2, q = 2, period = 2)
set.seed(1234)
a1 <- sim_pc(m1, 100)

## phi and theta are ordinary matrices here.
m2 <- list(phi = m1[ , 2:ncol(m1)], theta = m2[ , 2:ncol(m2)], p = 2, q = 2, period = 2)
set.seed(1234)
a2 <- sim_pc(m2, 100)
identical(a1, a2)

## Lina's PAR model
parcoef <- rbind(c(0.5, -0.06), c(0.6, -0.08),
                 c(0.7, -0.11), c(0.2, 0.15))
picoef1 <- c(0.8, 1.25, 2, 0.5)
parcoef2 <- pilar2par(picoef1, parcoef)
picoef2 <- c(4, 0.25, 5, 0.2)
co2I2 <- eigen(m2I2)$vectors
m2I2 <- mcompanion::mc_from_filter(co2I2)

#### specify the model using multi-companion approach
m2I2 <- mcompanion::mc_from_filter(co2I2)
co2I2 <- eigen(m2I2)$vectors
m2I2 <- mcompanion::mc_from_filter(period = 4, n.root = 4,
                                   eigabs = c(1.036588887, 0.001968887),
                                   eigsign = c(1, 1, -1),
                                   len.block = c(2, 1, 1),
                                   type.eigval = c("r", "r", "r"),
                                   co = cbind(co2I2[,1], rep(NA, 4), co2I2[,3:4]))

# todo: give example with sigmat^2 !!!

---

**sim_pwn**

*Simulate periodic white noise*

### Description

Generates a sample from periodic white noise.

### Usage

```r
sim_pwn(n = 100, period = NA, seasonof1st = 1, scale = NULL,
         shift = NULL, f = rnorm, ...)
```
Arguments

- **n**: length of the generated sample.
- **period**: number of seasons in an epoch.
- **seasonof1st**: season of the first observation in the result.
- **scale**: scale the series by this amount, a vector of length `period` or 1.
- **shift**: shift the series by this amount, a vector of length `period` or 1.
- **f**: a function or list of functions to generate random numbers.
- **...**: arguments for the random number generator(s) specified by `f`.

Details

First a series, say \( x \), of random numbers is generated as requested by the argument `f`. Then, if `shift` and/or `scale` are supplied, the values are modified as follows:

\[
y_t = shift_k + scale_k x_t
\]

where \( k \) is the season corresponding to time \( t \). The vector \( y \) is returned.

If `f` is a single a function (or name of a function), then the series is generated (effectively) by the call \( f(n, \ldots) \).

The argument `f` may also be a list whose \( k \)th element is itself a list specifying the random number generator for the \( k \)th season. The first element being the function (such as `rnorm`) and the remaining elements being parameters for that function. Parameters common to all seasons may be supplied through the `...` argument.

The argument `period` may be omitted. In that case it is inferred from `f` and/or the lengths of `shift` and `scale`. Currently there is no check for consistency here.

The arguments `shift` and `scale` may be used to specify simple linear transformations of the generated values, possibly different for the different seasons. Each of them should be a vector of length `period` or one.

`seasonof1st` can be used to request the simulated time series to start from a season other than the first one. Note that whatever the value of `seasonof1st`, the first elements of `scale`, `shift` and `f` (if a list) are taken to refer to season one.

Value

A vector of length \( n \) representing a realization of a periodic white noise series. The season of the first observation is `seasonof1st`.

Level

0 (base)

Author(s)

Georgi N. Boshnakov
Examples

```r
## three equivalent ways to specify periodic white noise with
## normal innovations, 2 seasons, s.d. = 0.5 for season 1, and 2 for season 2
sim_pwn(100, f = rnorm, scale = c(0.5, 2))
sim_pwn(n = 100, scale = c(0.5, 2))  # rnorm is the default generator
sim_pwn(100, f = list(c(rnorm, 0, 0.5), c(rnorm, 0, 2)))
```

SiPeriodicArmaModel-class

Class SiPeriodicArmaModel

Description

Class SiPeriodicArmaModel.

Objects from the Class

Objects can be created by calls of the form new("SiPeriodicArmaModel",...).

Slots

- `iorder`: Object of class "numeric"
- `siorder`: Object of class "numeric"
- `pcmodel`: Object of class "PeriodicArmaModel"

Extends

Class "VirtualPeriodicFilterModel", directly. Class "PeriodicIntegratedArmaSpec", directly.

Methods

No methods defined with class "SiPeriodicArmaModel" in the signature.

SiPeriodicArModel-class

Class SiPeriodicArModel

Description

Class SiPeriodicArModel.

Objects from the Class

Objects can be created by calls of the form new("SiPeriodicArModel",...).
SiPeriodicMaModel-class

Slots

iorder: Object of class "numeric" ~~
siorder: Object of class "numeric" ~~
pcmodel: Object of class "PeriodicArmaModel" ~~

Extends


Methods

fitPM signature(model = "SiPeriodicArModel", x = "ANY"): ...

---

Class SiPeriodicMaModel

Description

Class SiPeriodicMaModel.

Objects from the Class

Objects can be created by calls of the form new("SiPeriodicMaModel", ...).

Slots

iorder: Object of class "numeric" ~~
siorder: Object of class "numeric" ~~
pcmodel: Object of class "PeriodicArmaModel" ~~

Extends


Methods

No methods defined with class "SiPeriodicMaModel" in the signature.
Description

Functions for some basic operations with seasons.

Usage

toSeason(t, period, t1 = 1, from = 1)
toSeasonPair(t, s, period, ...)
ttTosl(r, period)
ttmatToslPairs(i, j, period)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>covariance matrix, see ‘Details’.</td>
</tr>
<tr>
<td>t</td>
<td>a vector of integers, representing times.</td>
</tr>
<tr>
<td>s</td>
<td>a vector of integers, representing times.</td>
</tr>
<tr>
<td>period</td>
<td>the number of seasons.</td>
</tr>
<tr>
<td>t1</td>
<td>time corresponding to the first season, an integer number.</td>
</tr>
<tr>
<td>from</td>
<td>1 or 0, depending on whether the season numbers start from 1 or 0.</td>
</tr>
<tr>
<td>i</td>
<td>a vector of integers.</td>
</tr>
<tr>
<td>j</td>
<td>a vector of integers.</td>
</tr>
<tr>
<td>...</td>
<td>todo: describe!</td>
</tr>
</tbody>
</table>

Details

ttmatToslPairs(i, j, period) transforms time-time pairs to season-lag pairs. The time pairs are obtained by pairing each element of i with each element of j. A four column matrix is created with one row for each pair (t, s), such that t = i[m] and s = j[n] for some m and n. The row is m, n, s, l, where (s, l) is the season-lag pair corresponding to (t, s).

ttTosl(r, period) converts autocovariances given in a covariance matrix (i.e. in “tt” form) to the “sl” form. The result is a period x (maxlag+1) matrix, where maxlag is the maximal lag available in r. Entries for which no values are available are filled with NA’s. Warning is given if contradictory entries are found (i.e. if r is not from a periodically correlated process with the given period).

toSeason(t, period, t1 = 1, from = 1) returns the season corresponding to t. t1 is a time (integer) whose season is the first season, from is 1 if the numbering of seasons is 1, 2, ..., period, or 0 if the numbering of seasons is 0, 1, ..., period-1. Other values for from are not admissible (but not checked).

Note: some of the functions in this package implicitly assume that t1 = 1 and from = 1.

toSeasonPair(t, s, period) converts the “tt” pair t, s to “sl” pair and returns the result in the form of a list with elements season and lag. Currently t and s must be scalars.
pc.omitneg helps to implement dropping of negative indices in season-lag objects. It returns its first argument, lags, if all of its elements are non-negative. Otherwise, all elements of lags must be non-positive. In this case the function creates the vector 0:maxlag and drops the elements specified by lags. Note that the default indexing will not work properly since zero elements in an index are omitted (and there are such indices in season-lag objects).

Value

for ttmatToslPairs, a matrix with four columns;
for ttTosl, a matrix with period rows;
for toSeason(t,period,t1=1,from=1), a vector of integers;
for toSeasonPair(t,s,period), a list with elements season and lag;
for pc.omitneg, a vector of lags (non-negative integers).

Note

2013-10-24 - Corrected the description of the return value of ttmatToslPairs. It incorrectly stated that the first two columns are "tt" pair (they are actually indices in i and j).

Author(s)

Georgi N. Boshnakov

References


Examples

```r
# ttmatToslPairs
ttmatToslPairs(3, 3, 4)  # 1, 1, 3, 0
ttmatToslPairs(3, 2, 4)  # 1, 1, 3, 1
ttmatToslPairs(1:4, 1:4, 4)
ttmatToslPairs(3:4, 3:4, 4)

# ttTosl - :todo:

toSeason((1:10, 4)  # 1 2 3 4 1 2 3 4 1 2
toSeason((1:10, 4, from = 0))  # 0 1 2 3 0 1 2 3 0 1

## first data is for 3rd quarter
toSeason((1:10, 4, t1 = 3))  # 3 4 1 2 3 4 1 2 3 4

# toSeasonPair
toSeasonPair(3, 3, period=4)  # season=3, lag = 0
toSeasonPair(8, 8, period=4)  # season=4, lag = 0
```
toSeasonPair(3, 2, period=4) # season=3, lag = 1
toSeasonPair(7, 6, period=4) # same

#### # pc.omitneg
#### pc.omitneg(0:5,10) # 0:5, unchanged since all values >= 0
#### pc.omitneg(-(0:5),10) # 6:10, works like
#### (0:10)[-c(0:5 +1)] # same
#### # don't mix positive and negative numbers in pc.omitneg
#### \dont{pc.omitneg(c(0,2,3,-4,5), 10)}

### SubsetPM-class

#### Class SubsetPM

**Description**

Class "SubsetPM" - subset PAR models with trigonometric parameterisation.

**Objects from the Class**

Objects can be created by calls of the form new("SubsetPM",...) but they are typically created by model fitting functions, see the examples.

**Slots**

- `theTS`: "ANY", the time series to which the model is fitted.
- `period`: "integer", the period.
- `order`: "integer", the order.
- `findex`: "function".
- `harmonics`: "integer", Fourier harmonics to include in the model.
- `call`: "call", the call used to fit the model.
- `other`: "namedList".

**Methods**

- `coef` signature(object = "SubsetPM"): ...
- `fitted` signature(object = "SubsetPM"): ...
- `residuals` signature(object = "SubsetPM"): ...
- `show` signature(object = "SubsetPM"): ...
- `vcov` signature(object = "SubsetPM"): ...
Examples

```r
pcfr4 <- pcts(dataFranses1996)[[4]]
x4 <- as.numeric(window(pcfr4, start = availStart(pcfr4), end = availEnd(pcfr4)))

## without 'harmonics' these models are equivalent
tmpfit <- fit_trigPAR_optim(x4, 2, 4, tol = 1e-14, verbose = FALSE)
tmpfitL <- fit_trigPAR_optim(x4, 2, 4, tol = 1e-14, type = "bylag", verbose = FALSE)

## for comparison
tmpfitP <- pclsdf(x4, 4, 1:2, sintercept = FALSE)

## with intercept
tmpfitc <- fit_trigPAR_optim(x4, 2, 4, tol = 1e-14, verbose = FALSE, sintercept = TRUE)
tmpfitcn <- fit_trigPAR_optim(x4, 2, 4, tol = 1e-14, verbose = FALSE, sintercept = structure(TRUE, merge = TRUE))
tmpfitLc <- fit_trigPAR_optim(x4, 2, 4, tol = 1e-14, type = "bylag", verbose = FALSE, sintercept = TRUE)

coef(tmpfitc, matrix = TRUE)
coef(tmpfitcn, matrix = TRUE)
coef(tmpfitLc, matrix = TRUE)

coef(tmpfitc)
coef(tmpfitcn)
coef(tmpfitLc)

coef(tmpfit)
coef(tmpfitL)

predict(tmpfitc, n.ahead = 4)
predict(tmpfitcn, n.ahead = 4)

sqrt(diag((vcov(tmpfitL))))
e <- residuals(tmpfitL)
fi <- fitted(tmpfitL)
```

---

**test_piar**

Test for periodic integration

**Description**

Test if a time series is periodically integrated.

**Usage**

```r
test_piar(x, d, p, sintercept = FALSE, sslope = FALSE, homoschedastic = FALSE)
```
Arguments

- x: time series
- d: period
- p: autoregressive order, a positive integer
- sintercept: if TRUE, include seasonal intercept
- sslope: if TRUE, include seasonal slope
- homoschedastic: if TRUE, assume the innovations variance is the same for all seasons

Details

Computes test statistics for Franses (1996) test for periodic integration of order 1. The test is based on periodic autoregression of order p, where p can be any positive integer.

Value

A list with the following components:

- p: autoregressive order.
- spec: values of sintercept, sslope, and homoschedastic, a named logical vector.
- statistics: a matrix containing the test statistics (first row) and the corresponding p-values (second row). “LR” is not normalised, so its p-value is NA.

Note

Currently only the case p = 1 is handled, for p > 1 the statistics are set to NA. :TODO: handle this.

All statistics are computed but some p-values are not computed yet.

Author(s)

Georgi N. Boshnakov

References


See Also

pclspiar, pclsdf
Examples

ts1 <- window(dataFranses1996[, "CanadaUnemployment"],
  start = c(1960, 1), end = c(1987, 4))
test_piar(ts1, 4, 1, sintercept = TRUE)
pcTest(ts1, "piar", 4, 1, sintercept = TRUE) # same

  test_piar(ts1, 4, 1, sintercept = TRUE, sslope = TRUE)
test_piar(ts1, 4, 1)
test_piar(ts1, 4, 1, homoschedastic = TRUE)

Description

Methods for unitCycle() in package pcts.

Methods

signature(x = "ANY")
signature(x = "DayWeekCycle")
signature(x = "FiveDayWeekCycle")
signature(x = "MonthYearCycle")
signature(x = "OpenCloseCycle")
signature(x = "QuarterYearCycle")
signature(x = "SimpleCycle")
signature(x = "Cyclic")
signature(x = "Every30MinutesCycle")
signature(x = "VirtualPeriodicModel")

See Also

allSeasons for related functions and examples.

Description

Methods for ‘unitCycle<-‘ in package pcts.

Methods

signature(x = "Cyclic")
signature(x = "SimpleCycle")
Methods for \texttt{unitSeason()} in package \texttt{pcts}

\section*{Description}

Methods for \texttt{unitSeason()} in package \texttt{pcts}.

\section*{Methods}

\begin{verbatim}
signature(x = "ANY")
signature(x = "DayWeekCycle")
signature(x = "FiveDayWeekCycle")
signature(x = "MonthYearCycle")
signature(x = "OpenCloseCycle")
signature(x = "QuarterYearCycle")
signature(x = "SimpleCycle")
signature(x = "Cyclic")
signature(x = "Every30MinutesCycle")
signature(x = "VirtualPeriodicModel")
\end{verbatim}

\section*{See Also}

\texttt{allSeasons} for related functions and examples

Methods for \texttt{unitSeason<-} in package \texttt{pcts}

\section*{Description}

Methods for \texttt{\`unitSeason<-} in package \texttt{pcts}.

\section*{Methods}

\begin{verbatim}
signature(x = "Cyclic")
signature(x = "SimpleCycle")
\end{verbatim}
Core data of periodic time series

Description
Core data of periodic time series.

Usage
Vec(x, ...)

tsMatrix(x, ...)

tsVector(x, ...)

tsVec(x, ...)

cMatrix(x, ...)

cArray(x, ndim = 3, ...)

cpArray(x, ndim = 3, ...)

cpctArray(x, ndim = 3, ...)

Arguments
x an object.
... further arguments for methods.
ndim currently not used.

Details
These functions give the core data in various common forms.
The data values can be extracted as a vector from a periodic time series object, say x, with as.vector(x) or as(x,"vector"). Similarly, as.matrix() and as(x,"matrix") extract the data as a matrix containing one column per variable. For multivariate time series the vector returned by as.vector(x) (or as(x,"vector")) is equivalent to as.vector(as.matrix(x)).

Vec() is like as.vector() but returns the result as a matrix with one column (column vector), as is usual in matrix calculus. The default does literally this:
The most common representation of data in statistics is matrix-like with one column per variable. The descriptions of algorithms for multivariate time series however usually define the vector of observations at a given time to be a column vector. In particular, implementations of the Kalman filter often require precisely this arrangement.
In that case the data matrix is the transposed of the more common one and the vectorising operation stacks the observations, not the variables.
The functions \texttt{tsMatrix()}, \texttt{tsVector()} and \texttt{tsVec()} provide the analogues of \texttt{as.vector()}, \texttt{as.matrix()} and \texttt{Vec()} for the “transposed” arrangement.

These functions may look redundant since they are simple combinations of the above and transpose operations. Having functions makes for more readable programming. They may be more efficient, as well, for example if the underlying time series class stores the data in the transposed format.

\texttt{pcMatrix()} and \texttt{pcArray()} also give the core data. Effectively, they give an additional dimension to the seasons. The season becomes the first dimension since for column oriented data the season changes fastest. \texttt{pcMatrix} is most suitable for univariate time series, \texttt{pcArray()} for multivariate. Note that \texttt{pcArray()} easily extends to multiple periodicities although currently (2019-04-19) there are no methods that exploit this.

For univariate time series, in the matrix returned by \texttt{pcMatrix()} each row represents the data for one season and each column for one cycle. For multivariate time series, the matrices for each variable are put next to each other.

\texttt{pcArray()} returns the data as an array, whose last dimension corresponds to variables. In the default case the array is 3-dimensional with dimensions (season, year, variable).

\texttt{pctsArray()} is a variant of \texttt{pcArray()} corresponding to the arrangement of \texttt{tsMatrix()}. The ordering of the dimensions here is (variable, season, cycle).

**Value**

vector, matrix or array, as indicated by the name of the function and described in Details.

**Author(s)**

Georgi N. Boshnakov

**Examples**

```r
## window to make number of years different from number of months
ap <- pcts(window(AirPassengers, start = c(1956, 1)))
class( as.vector(ap) )
class( as(ap, "vector") )
dim( as.matrix(ap) )
dim( as(ap, "matrix") )
dim( tsMatrix(ap) )
class( tsVector(ap) )
dim( tsVec(ap) )
dim( pcMatrix(ap) )
dim( pcArray(ap) )
dim( pctsArray(ap) )
dfr <- pcts(dataFranses1996)
dim(dfr) # c(148, 19)
nSeasons(dfr) # 4
```
window

Periodic methods for base R functions

Description

Periodic methods for base R functions.

Usage

```r
## S3 method for class 'PeriodicTS'
window(x, start = NULL, end = NULL, seasons = NULL, ...)

## S3 method for class 'PeriodicMTS'
window(x, start = NULL, end = NULL, seasons = NULL, ...)
```

Arguments

- `x`, `object`: an object from the indicated periodic class.
- `start`: numeric(2), start time.
- `end`: numeric(2), end time.
- `seasons`: numeric, a subset of 1:nSeasons(x).
- `...`: Not used by these methods.
sides: which side to trim: start ("left"), end ("right"), or both ("both").

is.na: for multivariate time series: if "all", the observation at time t will be considered missing only if all variables are NA at that time. Otherwise, if "any", any variable with value NA will cause the observation at time t to be considered missing.

Details

Periodic methods for base R and other common functions for manipulation of time series.

window: takes a part of x, preserving the class of the object. Argument seasons selects a subset of the seasons.

na.trim is a function defined in package zoo and re-exported by pcts. It trims NAs from one or both ends of the time series, as requested by the arguments. The arguments of the methods defined by pcts have the same meaning as those in zoo.

Value

an object from the same class as the original, representing the requested part of the time series.

See Also

window, na.trim, availStart and availEnd which give the times of the first and last non-NA observations.

Examples

```r
pres <- pcts(presidents)
head(pres, 8)
availStart(pres)

tail(pres, 12)
availEnd(pres)

## Q3 and Q4 only
presQ3Q4 <- window(pres, seasons = 3:4)
head(presQ3Q4)

identical(na.trim(pres),
          window(pres, start = availStart(pres), end = availEnd(pres)))
## TRUE
```

zoo-class

Class zoo made S4

Description

Class zoo made S4
Objects from the Class

A virtual Class: No objects may be created from it.

S4 Class "zoo" is derived from its namesake in package zoo for use as a super class for periodic time series classes and in S4 method signatures.

Slots

.S3Class: Object of class "character" ~~

Extends

Class "oldClass", directly.

Methods

No methods defined with class "zoo" in the signature.

See Also

PeriodicTS_zooreg, zooreg and package zoo

Examples

showClass("zoo")

zooreg-class

Virtual S4 class zooreg

Description

Virtual S4 class zooreg.

Objects from the Class

A virtual Class: No objects may be created from it.

S4 Class "zooreg" is derived from its namesake in package zoo for use as a super class for periodic time series classes and in S4 method signatures.

Slots

.S3Class: Object of class "character" ~~

Extends

Class "zoo", directly. Class "oldClass", by class "zoo", distance 2.
Methods

No methods defined with class "zooreg" in the signature.

See Also

PeriodicTS_zooreg, zoo and package zoo

Examples

showClass("zooreg")

-------------------------------------------------------------------

Description

Indexing of objects from classes in package pcts.

Methods

signature(x = "BasicCycle", i = "ANY", j = "missing", drop = "ANY")
signature(x = "BasicCycle", i = "missing", j = "missing", drop = "ANY")
signature(x = "PeriodicMTS", i = "ANY", j = "missing", drop = "ANY")
signature(x = "PeriodicMTS", i = "missing", j = "missing", drop = "ANY")
signature(x = "PeriodicVector", i = "ANY", j = "ANY", drop = "ANY")
signature(x = "PeriodicVector", i = "ANY", j = "missing", drop = "ANY")
signature(x = "PeriodicVector", i = "missing", j = "ANY", drop = "ANY")
signature(x = "PeriodicVector", i = "missing", j = "missing", drop = "ANY")
signature(x = "VirtualPeriodicAutocovarianceModel", i = "missing", j = "missing", drop = "ANY")
signature(x = "VirtualPeriodicAutocovarianceModel", i = "missing", j = "numeric", drop = "ANY")
signature(x = "VirtualPeriodicAutocovarianceModel", i = "numeric", j = "missing", drop = "ANY")
signature(x = "VirtualPeriodicAutocovarianceModel", i = "numeric", j = "numeric", drop = "ANY")
signature(x = "PeriodicTS", i = "missing", j = "missing", drop = "ANY")
signature(x = "PeriodicMTS", i = "ANY", j = "ANY", drop = "ANY")
signature(x = "PeriodicMTS", i = "AnyDateTime", j = "ANY", drop = "ANY")
signature(x = "PeriodicMTS", i = "AnyDateTime", j = "missing", drop = "ANY")
signature(x = "PeriodicTS", i = "AnyDateTime", j = "missing", drop = "ANY")

Author(s)

Georgi N. Boshnakov
Methods for function `[<-` in package `pcts`

Description

Index assignments for objects from classes in package pcts.

Methods

signature(x = "PeriodicVector", i = "ANY", j = "ANY", value = "ANY")
signature(x = "PeriodicVector", i = "missing", j = "ANY", value = "ANY")
signature(x = "BasicCycle", i = "ANY", j = "missing", value = "ANY")
signature(x = "BasicCycle", i = "missing", j = "missing", value = "ANY")
signature(x = "PeriodicAutocovarianceModel", i = "ANY", j = "ANY", value = "ANY")

Author(s)

Georgi N. Boshnakov

Methods for function `[[` in package `pcts`

Description

Methods for function `[[` in package `pcts`.

Methods

signature(x = "PeriodicMTS", i = "ANY", j = "ANY")
signature(x = "VirtualPeriodicAutocovarianceModel", i = "numeric", j = "ANY")

Methods for function `$` in package `pcts`

Description

Methods for function `$` in package `pcts`.

Methods

signature(x = "PeriodicMTS")
Index

* **acf**
  autocorrelations-methods, 12
  autocovariances-methods, 12
* **chron**
  as_datetime-methods, 11
  Pctime, 78
* **classes**
  BareCycle-class, 15
  BasicCycle-class, 17
  BuiltInCycle-class, 18
  Cyclic-class, 23
  FittedPeriodicArmaModel-class, 31
  FittedPeriodicArModel-class, 32
  ModelCycleSpec-class, 41
  PartialCycle-class, 46
  PartialPeriodicAutocorrelations-class, 47
  PeriodicArmaFilter-class, 86
  PeriodicArmaModel-class, 86
  PeriodicArModel-class, 88
  PeriodicAutocorrelations-class, 89
  PeriodicAutocovarianceModel-class, 90
  PeriodicAutocovariances-class, 91
  PeriodicAutocovarianceSpec-class, 92
  PeriodicBJFilter-class, 92
  PeriodicFilterModel-class, 94
  PeriodicIntegratedArmaSpec-class, 94
  PeriodicInterceptSpec-class, 95
  PeriodicMaModel-class, 96
  PeriodicMTS-class, 96
  PeriodicMTS_ts-class, 98
  PeriodicMTS_zooreg-class, 99
  PeriodicSPFilter-class, 99
  PeriodicTimeSeries-class, 100
  PeriodicTS-class, 101
  PeriodicTS_ts-class, 102
  PeriodicTS_zooreg-class, 102
  PeriodicVector-class, 103
  PiPeriodicArmaModel-class, 109
  PiPeriodicArModel-class, 110
  PiPeriodicMaModel-class, 111
  SamplePeriodicAutocorrelations-class, 113
  SamplePeriodicAutocovariances-class, 113
  SimpleCycle-class, 115
  SiPeriodicArmaModel-class, 123
  SiPeriodicArModel-class, 123
  SiPeriodicMaModel-class, 124
  SubsetPM-class, 127
  zoo-class, 135
  zooreg-class, 136
* **datagen**
  sim_parAcvf, 116
  sim_pc, 119
  sim_pwn, 121
* **datasets**
  dataFranses1996, 23
  four_stocks_since2016_01_01, 34
  Fraser2017, 35
  pcts_exdata, 83
* **fitmodel**
  num2pcpar, 43
* **graphics**
  pcPlot, 76
* **methods**
  [-methods, 137
  <<-methods, 138
  <<-methods, 138
  $-methods, 138
  allSeasons, 8
  as_date-methods, 11
  as_datetime-methods, 11
  autocorrelations-methods, 12
autocovariances-methods, 12
date<--methods, 26
filterCoef-methods, 27
fitPM, 28
maxLag-methods, 36
modelCycle, 40
nSeasons-methods, 42
pcApply-methods, 58
cycle-methods, 68
pcTest-methods, 77
pcts, 81
PeriodicArModel-methods, 88
seqSeasons-methods, 114
sigmaSq-methods, 115
unitCycle-methods, 130
unitCycle<--methods, 130
unitSeason-methods, 131
unitSeason<--methods, 131
package
pcts-package, 4
par
ex1f, 26
pcarithmetic
pc_sdfactor, 84
pcalg1, 55
pcalg1util, 57
pcarma
pc.hat.h, 52
pcarma_solve, 63
pcarma_unvec, 66
sim_pc, 119
pcstat
parcovmatlist, 45
pcacf_pwn_var, 54
pcTest-methods, 77
periodic_acfl_test, 105
pwn_McLeodLjungBox_test, 111
test_piar, 128
pcts
allSeasons, 8
fitPM, 28
mC.ss, 36
nCycles, 41
nSeasons-methods, 42
pc.filter, 48
pc.filter.xarma, 50
pcAr.ss, 60
pclsdf, 71
pclspiar, 73
Pctime, 78
pcts, 81
Vec, 132
periodic autoregression
sim_parCoef, 117
permodel
pcacfhMat, 53
pcAR2acf, 61
pcarma_acvf2model, 62
pcarma_solve, 63
pdSafeParOrder, 85
permean2intercept, 106
permodelmf, 107
pilar2par, 108
sim_parAcvf, 116
sim_parCoef, 117
simulation
sim_parCoef, 117
torevise
mC.ss, 36
ts
autocovariances-methods, 12
autocovariances-methods, 12
availStart, 13
pcMean-methods, 74
pcPlot, 76
[,BasicCycle,ANY,missing,ANY-method
([-methods), 137
[,BasicCycle,missing,missing,ANY-method
([-methods), 137
[,PeriodicMTS,ANY,ANY,ANY-method
([-methods), 137
[,PeriodicMTS,ANY,missing,ANY-method
([-methods), 137
[,PeriodicMTS,AnyDateTime,ANY,ANY-method
([-methods), 137
[,PeriodicMTS,AnyDateTime,missing,ANY-method
([-methods), 137
[,PeriodicMTS,missing,missing,ANY-method
([-methods), 137
[,PeriodicVector,ANY,ANY,ANY-method
([-methods), 137
INDEX

[,PeriodicVector, ANY, missing, ANY-method
\[-methods, 137
[,PeriodicVector, missing, ANY, ANY-method
\[-methods, 137
[,PeriodicVector, missing, missing, ANY-method
\[-methods, 137
[,VirtualPeriodicAutocovarianceModel, missing, missing, ANY-method
\[-methods, 137
[,VirtualPeriodicAutocovarianceModel, numeric, missing, ANY-method
\[-methods, 137
[,VirtualPeriodicAutocovarianceModel, numeric, numeric, ANY-method
\[-methods, 137
\[methods, 137
\.Date (Pctime), 78
\.Pctime (Pctime), 78
\.ts (Pctime), 78
<-, ANY, ANY, ANY, ANY-method
\[-->methods, 138
<-, BasicCycle, ANY, missing, ANY-method
\[-->methods, 138
<-, BasicCycle, missing, missing, ANY-method
\[-->methods, 138
<-, PeriodicAutocovarianceModel, ANY, ANY, ANY-method
\[-->methods, 138
<-, PeriodicVector, ANY, ANY, ANY-method
\[-->methods, 138
<-, PeriodicVector, missing, ANY, ANY-method
\[-->methods, 138
<-, pc armaPQ, ANY, ANY, ANY-method
\[-->methods, 138
<-, slMatrix, ANY, ANY, ANY-method
\[-->methods, 138
<-, POSIXlt (Pctime), 78
][, PeriodicAutocovarianceModel, numeric-method
\[-->methods, 138
[, PeriodicMTS, ANY-method
\[-->methods, 138
[, PeriodicMTS, ANY-method (\[methods, 138
[, VirtualPeriodicAutocovarianceModel, numeric-method
\[-->methods, 138
\[methods, 138
\.Date (Pctime), 78
\.Pctime (Pctime), 78
\$, PeriodicMTS-method (\$-methods, 138

alg1, 57, 58
alg1 (pcalg1), 55
algutil (pcalgutil), 57
allSeasons, 5, 8, 70, 114, 130, 131
allSeasons, BasicCycle, ANY-method
(allSeasons), 8
allSeasons, BuiltinCycle-method
(BuiltinCycle-class), 18
allSeasons, Cyclic, ANY-method
(allSeasons), 8
allSeasons, DayWeekCycle, logical-method
(allSeasons), 8
allSeasons, Every30MinutesCycle, logical-method
(allSeasons), 8
allSeasons, Every30MinutesCycle, missing-method
(allSeasons), 8
allSeasons, FiveDayWeekCycle, logical-method
(allSeasons), 8
allSeasons, FiveDayWeekCycle, missing-method
(allSeasons), 8
allSeasons, MonthYearCycle, logical-method
(allSeasons), 8
allSeasons, MonthYearCycle, missing-method
(allSeasons), 8
allSeasons, OpenCloseCycle, logical-method
(allSeasons), 8
allSeasons, OpenCloseCycle, missing-method
(allSeasons), 8
allSeasons, PartialCycle, logical-method
(PartialCycle-class), 46
allSeasons, PartialCycle, missing-method
(PartialCycle-class), 46
allSeasons, QuarterYearCycle, logical-method
(allSeasons), 8
allSeasons, QuarterYearCycle, missing-method
(allSeasons), 8
allSeasons, SimpleCycle, ANY-method
(allSeasons), 8
allSeasons-Methods (allSeasons), 8
allSeasons<-> (allSeasons), 8
allSeasons<-, Cyclic-method
(allSeasons), 8
allSeasons<-, SimpleCycle-method
   (allSeasons), 8
apply, 59, 60
array, 97
as.Date.Cyclic (Cyclic), 21
as.Date.PeriodicTimeSeries (Cyclic), 21
as_date, ANY-method (as_date-methods), 11
as_date, character-method
   (as_date-methods), 11
as_date, Cyclic-method
   (as_date-methods), 11
as_date, numeric-method
   (as_date-methods), 11
as_date, PeriodicTimeSeries-method
   (as_date-methods), 11
as_date, POSIXt-method
   (as_date-methods), 11
as_date-methods, 11
as_datetime, PeriodicTimeSeries-method
   (as_datetime-methods), 11
as_datetime-methods, 11
as_pcarma_list, FittedPeriodicArmaModel-method
   (FittedPeriodicArmaModel-class), 31
as_Pctime (Pctime), 78
atomicVector, 95, 101, 104
autocorrelations, numeric, ANY, missing-method
   (autocorrelations-methods), 12
autocorrelations, PeriodicAutocovariances, ANY, missing-method
   (autocorrelations-methods), 12
autocorrelations, PeriodicArmaModel, ANY-method
   (autocovariances-methods), 12
autocorrelations, PeriodicArmaModel-method
   (autocovariances-methods), 12
autocovariances, PeriodicArmaModel, ANY-method
   (autocovariances-methods), 12
autocovariances, PeriodicArmaModel-method
   (autocovariances-methods), 12
autocovariances, PeriodicArModel, ANY-method
   (autocovariances-methods), 12
autocovariances, PeriodicArModel-method
   (autocovariances-methods), 12
autocovariances, PeriodicTS, ANY-method
   (autocovariances-methods), 12
autocovariances, VirtualPeriodicAutocovariances, ANY-method
   (autocovariances-methods), 12
availEnd, 135
availEnd (availStart), 13
availStart, 13, 135
BareCycle, 115
BareCycle-class, 15
BasicCycle, 9, 16, 18, 46, 115
BasicCycle-class, 17
Box.test, 112
boxplot, PeriodicTimeSeries (pcPlot), 76
BuiltInCycle, 5, 18, 19, 22, 46, 70
BuiltInCycle (pcCycle-methods), 68
BuiltInCycle-class, 18
dataFranses1996, 23, 82, 84, 97
data.Cyclic (Cyclic), 21
data<--methods, 26
data<-- (Cyclic), 21
data<--, BasicCycle-method
   (date<--methods), 26
data<--, Cyclic-method (date<--methods), 26
DayWeekCycle-class
   (BuiltInCycle-class), 18
Every30MinutesCycle-class
(BuiltinCycle-class), 18
exif, 26
filterCoef, PeriodicBJFilter, character-method
(filterCoef-methods), 27
filterCoef, PeriodicSPFilter, character-method
(filterCoef-methods), 27
filterCoef-methods, 27
fit_trigPAR_optim, 33
fitPM, 28, 44, 74
fitPM, ANY, ANY-method (fitPM), 28
fitPM, mcSpec, ANY-method (fitPM), 28
fitPM, numeric, ANY-method (fitPM), 28
fitPM, PeriodicArModel, ANY-method
(fitPM), 28
fitPM, PeriodicArModel, PeriodicMTS-method
(fitPM), 28
fitPM, PeriodicArModel, PeriodicTS-method
(fitPM), 28
fitPM, PiPeriodicArModel, ANY-method
(fitPM), 28
fitPM, SiPeriodicArModel, ANY-method
(fitPM), 28
fitPM-methods (fitPM), 28
Fitted, 113, 114
fitted, SubsetPM-method
(SubsetPM-class), 127
FittedPeriodicArmaModel-class, 31
FittedPeriodicArModel-class, 32
FiveDayWeekCycle-class
(BuiltinCycle-class), 18
FlexibleLagged, 47, 89, 91, 113, 114
four_stocks_since2016_01_01, 34
Fraser2017, 35
index, 95, 101, 104
initialize, BuiltinCycle-method
(BuiltinCycle-class), 18
innovationVariances, PeriodicArmaSpec-method
(PeriodicArmaSpec-class), 87
intercept2permean (permean2intercept), 106
InterceptSpec, 31
Lagged, 47, 89, 91, 113, 114
matrix, 97
maxLag, PeriodicArmaFilter-method
(maxLag-methods), 36
maxLag-methods, 36
m.C. ss, 28, 36
mC.par(pcts-deprecated), 83
meanCovMat (meanvarcheck), 39
meanvarcheck, 39
modelCycle, 40
modelCycle, ANY-method (modelCycle), 40
modelCycle, ModelCycleSpec-method
(modelCycle), 40
modelCycle-methods (modelCycle), 40
modelCycle <- (modelCycle), 40
modelCycle <- , ANY-method (modelCycle), 40
modelCycle <- , ModelCycleSpec-method
(modelCycle), 40
modelCycle <- methods (modelCycle), 40
ModelCycleSpec, 31, 47, 89, 91, 113, 114
ModelCycleSpec-class, 41
monthplot.PeriodicTimeSeries (pcPlot),
76
MonthYearCycle-class
(BuiltinCycle-class), 18
na.trim, 135
na.trim (window), 134
nCycles, 41
nSeasons, BareCycle-method
(nSeasons-methods), 42
nSeasons, Cyclic-method
(nSeasons-methods), 42
nSeasons, DayWeekCycle-method
(nSeasons-methods), 42
nSeasons, Every30MinutesCycle-method
(nSeasons-methods), 42
nSeasons, FiveDayWeekCycle-method
(nSeasons-methods), 42
nSeasons, MonthYearCycle-method
(nSeasons-methods), 42
nSeasons, OpenCloseCycle-method
(nSeasons-methods), 42
nSeasons, PartialCloseCycle-method
(nSeasons-methods), 42
nSeasons, PartialCycle-method
(nSeasons-methods), 42
nSeasons, PeriodicFilterModel-method
(nSeasons-methods), 42
nSeasons, PeriodicIntegratedArmaSpec-method
(nSeasons-methods), 42
nSeasons, PeriodicInterceptSpec-method
(nSeasons-methods), 42
nSeasons, PeriodicMonicFilterSpec-method
  (nSeasons-methods), 42
nSeasons, QuarterYearCycle-method
  (nSeasons-methods), 42
nSeasons, VirtualPeriodicArmaFilter-method
  (nSeasons-methods), 42
nSeasons, VirtualPeriodicModel-method
  (nSeasons-methods), 42
nSeasons-methods, 42
nTicks (nCycles), 41
num2pcpar, 43
number, 95, 101, 104
numeric, 95, 101, 104
nVariables (nCycles), 41
oldClass, 98, 99, 102, 103, 136
OpenCloseCycle-class
  (BuiltInCycle-class), 18
optionalMatrix, 97
parcovmatlist, 39, 45
PartialCycle-class, 46
PartialPeriodicAutocorrelations-class, 47
pc.acf.parModel, 45
pc.acf.parModel (pcacfMat), 53
pc.armafilter (pc.filter), 48
pc.filter, 48, 51, 120
pc.filter.xarma, 50, 50
pc.hat.h, 52
pc.sdfactor (pc_sdfactor), 84
pc_sdfactor, 84
pcacf_pwn_var, 54
pcacfMat, 45, 53
pcalg1, 55
pcalg1util, 57
pcApply, 75
pcApply (pcApply-methods), 58
pcApply, matrix-method
  (pcApply-methods), 58
pcApply, numeric-method
  (pcApply-methods), 58
pcApply, PeriodicMTS-method
  (pcApply-methods), 58
pcApply, PeriodicTS-method
  (pcApply-methods), 58
pcApply-methods, 58
pcAr.ss, 60
pcAR2acf, 61
pcarma_acvf2model, 62
pcarma_acvf_lazy, 61
pcarma_acvf_lazy (pcarma_solve), 63
pcarma_acvf_system (pcarma_solve), 63
pcarma_h, 65
pcarma_h (pcarma_solve), 63
pcarma_h_lazy (pcarma_solve), 63
pcarma_param_system, 65
pcarma_param_system (pcarma_solve), 63
pcarma_prepare (pcarma_unvec), 66
pcarma_solve, 63
pcarma_tovec (pcarma_unvec), 66
pcarma_unvec, 66
pcArray (Vec), 132
pcCycle, 5, 16, 19, 22, 40, 116
pcCycle (pcCycle-methods), 68
pcCycle, BasicCycle, character-method
  (pcCycle-methods), 68
pcCycle, BasicCycle, missing-method
  (pcCycle-methods), 68
pcCycle, character, ANY-method
  (pcCycle-methods), 68
pcCycle, character, character-method
  (pcCycle-methods), 68
pcCycle, character, missing-method
  (pcCycle-methods), 68
pcCycle, Cyclic, ANY-method
  (pcCycle-methods), 68
pcCycle, numeric, character-method
  (pcCycle-methods), 68
pcCycle, numeric, missing-method
  (pcCycle-methods), 68
pcCycle, periodicTimeSeries, character-method
  (pcCycle-methods), 68
pcCycle, periodicTimeSeries, missing-method
  (pcCycle-methods), 68
pcCycle, ts, character-method
  (pcCycle-methods), 68
pcCycle, ts, missing-method
  (pcCycle-methods), 68
pcCycle-methods, 68
pclsdf, 71, 74, 129
pclspsiar, 29, 72, 73, 129
pcMatrix (Vec), 132
pcMean, 60
pcMean (pcMean-methods), 74
pcMean, matrix-method (pcMean-methods), 74
pcMean, numeric-method (pcMean-methods), 74
pcMean, PeriodicMTS-method (pcMean-methods), 74
pcMean, PeriodicTS-method (pcMean-methods), 74
pcMean, VirtualPeriodicArmaModel-method (pcMean-methods), 74
pcMean-methods, 74
pcPlot, 76
pcTest (pcTest-methods), 77
pcTest, ANY, ANY-method (pcTest-methods), 77
pcTest, ANY, character-method (pcTest-methods), 77
pcTest, numeric, character-method (pcTest-methods), 77
pcTest, PeriodicTimeSeries, character-method (pcTest-methods), 77
pcTest, slMatrix, character-method (pcTest-methods), 77
pcTest-methods, 77
Pctime, 78
pcts, 5, 22, 81, 96, 97
pcts, ANY-method (pcts), 81
pcts, data.frame, ANY-method (pcts), 81
pcts, matrix, BasicCycle-method (pcts), 81
pcts, matrix, missing-method (pcts), 81
pcts, matrix, numeric-method (pcts), 81
pcts, mts, missing-method (pcts), 81
pcts, mts, numeric-method (pcts), 81
pcts, numeric, BasicCycle-method (pcts), 81
pcts, numeric, missing-method (pcts), 81
pcts, numeric, numeric-method (pcts), 81
pcts, ts, missing-method (pcts), 81
pcts, ts, numeric-method (pcts), 81
pcts, xtsORzoo, missing-method (pcts), 81
pcts-deprecated, 83
pcts-methods (pcts), 81
pcts-package, 4
pcts.exdata, 83
pctsArray (Vec), 132
pdSafeParOrder, 27, 56, 58, 85
periodic_acf1_test, 78, 105
PeriodicArFilter-class (PeriodicArmaFilter-class), 86
PeriodicArmaFilter-class, 86
PeriodicArmaModel, 31, 32, 88, 96
PeriodicArmaModel-class, 86
PeriodicArmaSpec, 31, 32, 87, 88, 94, 96
PeriodicArmaSpec-class, 87
PeriodicArModel, 32
PeriodicArModel
(PeriodicArModel-methods), 88
PeriodicArModel, matrix-method (PeriodicArModel-methods), 88
PeriodicArModel, numeric-method (PeriodicArModel-methods), 88
PeriodicArModel, PeriodicArmaModel-method (PeriodicArModel-methods), 88
PeriodicArModel, PeriodicMonicFilterSpec-method (PeriodicArModel-methods), 88
PeriodicArModel, VirtualPeriodicArmaModel-method (PeriodicArModel-methods), 88
PeriodicArModel-class, 88
PeriodicArModel-methods, 88
PeriodicAutocorrelations, 113
PeriodicAutocorrelations-class, 89
PeriodicAutocovarianceModel-class, 90
PeriodicAutocovariances, 114
PeriodicAutocovariances-class, 91
PeriodicAutocovarianceSpec, 90
PeriodicAutocovarianceSpec-class, 92
PeriodicBJFilter, 100
PeriodicBJFilter-class, 92
PeriodicFilterModel-class, 94
PeriodicIntegratedArmaSpec, 110, 111, 123, 124
PeriodicIntegratedArmaSpec-class, 94
PeriodicInterceptSpec, 31
PeriodicInterceptSpec-class, 95
PeriodicMaFilter-class (PeriodicArmaFilter-class), 86
PeriodicMaModel-class, 96
PeriodicMonicFilterSpec, 92, 100
PeriodicMonicFilterSpec-class (PeriodicBJFilter-class), 92
PeriodicMTS, 103
PeriodicMTS-class, 96
PeriodicMTS_ts-class, 98
PeriodicMTS_zooreg-class, 99
PeriodicSPFilter, 93
PeriodicSPFilter-class, 99
PeriodicTimeSeries, 97–99, 101–103
PeriodicTimeSeries-class, 100
PeriodicTS, 82, 97, 103
PeriodicTS-class, 101
PeriodicTS_ts-class, 102
PeriodicTS_zooreg, 136, 137
PeriodicTS_zooreg-class, 102
PeriodicVector, 104
PeriodicVector (PeriodicVector-class), 103
PeriodicVector-class, 103
permean2intercept, 106
permodelmf, 107
piar2par (piar2par), 108
PiPeriodicArmaModel, 110, 111
PiPeriodicArmaModel-class, 109
PiPeriodicArModel-class, 110
PiPeriodicMaModel-class, 111
plot, PeriodicAutocorrelations, missing-method
(PeriodicAutocorrelations-class), 89
plot, PeriodicMTS, missing-method
(PeriodicMTS-class), 96
plot, PeriodicTS, missing-method
(PeriodicTS-class), 101
ptildeorders (pcts-deprecated), 83
pwn_McLeodLjungBox_test, 78, 111
QuarterYearCycle-class
(BuiltinCycle-class), 18
replValue, 95, 101, 104
residuals, SubsetPM-method
(SubsetPM-class), 127
SamplePeriodicAutocorrelations-class, 113
SamplePeriodicAutocovariances-class, 113
seqSeasons (allSeasons), 8
seqSeasons, BasicCycle-method
(seqSeasons-methods), 114
seqSeasons, Cyclic-method
(seqSeasons-methods), 114
seqSeasons, VirtualPeriodicModel-method
(seqSeasons-methods), 114
seqSeasons-methods, 114
show, FittedPeriodicArmaModel-method
(FittedPeriodicArmaModel-class), 31
show, PeriodicTS-method
(PeriodicTS-class), 101
show, SubsetPM-method (SubsetPM-class), 127
sigmaSq, PeriodicIntegratedArmaSpec-method
(sigmaSq-methods), 115
sigmaSq, PeriodicInterceptSpec-method
(sigmaSq-methods), 115
sigmaSq-methods, 115
sim_arAcf (pcts-deprecated), 83
sim_mc, 118
sim_parAcvf, 116
sim_parCoef, 117
sim_pc, 119
sim_pcfilter, 118
sim_pwn, 119, 120, 121
SimpleCycle-class, 115
SiPeriodicArmaModel, 124
SiPeriodicArmaModel-class, 123
SiPeriodicArModel-class, 123
SiPeriodicMaModel-class, 124
sl_utils, 125
start, 5
structure, 97–99, 102, 103
SubsetPM, 33
SubsetPM-class, 127
summary, PeriodicTS-method
(PeriodicTS-class), 101
test_piar, 74, 78, 128
toSeason (sl_utils), 125
toSeasonPair (sl_utils), 125
ts, 98, 99, 102, 103
tsMatrix (Vec), 132
tsVec (Vec), 132
tsVector (Vec), 132
ttmatToslPairs (sl_utils), 125
ttTosl (sl_utils), 125
unitCycle (allSeasons), 8
unitCycle, ANY-method
(unitCycle-methods), 130
unitCycle, BuiltinCycle-method
(BuiltinCycle-class), 18
unitCycle, Cyclic-method
(unitCycle-methods), 130
unitCycle, DayWeekCycle-method
(unitCycle-methods), 130
unitCycle, Every30MinutesCycle-method
  (unitCycle-methods), 130
unitCycle, FiveDayWeekCycle-method
  (unitCycle-methods), 130
unitCycle, MonthYearCycle-method
  (unitCycle-methods), 130
unitCycle, OpenCloseCycle-method
  (unitCycle-methods), 130
unitCycle, PartialCycle-method
  (PartialCycle-class), 46
unitCycle, QuarterYearCycle-method
  (unitCycle-methods), 130
unitCycle, SimpleCycle-method
  (unitCycle-methods), 130
unitCycle, VirtualPeriodicModel-method
  (unitCycle-methods), 130
unitCycle-methods, 130
unitCycle<--methods, 130
unitCycle<-- (allSeasons), 8
unitCycle<-- (Cyclic-method
  (unitCycle<--methods), 130
unitCycle<-- (SimpleCycle-method
  (unitCycle<--methods), 130
unitSeason (allSeasons), 8
unitSeason, ANY-method
  (unitSeason-methods), 131
unitSeason, BuiltinCycle-method
  (BuiltinCycle-class), 18
unitSeason, Cyclic-method
  (unitSeason-methods), 131
unitSeason, DayWeekCycle-method
  (unitSeason-methods), 131
unitSeason, Every30MinutesCycle-method
  (unitSeason-methods), 131
unitSeason, FiveDayWeekCycle-method
  (unitSeason-methods), 131
unitSeason, MonthYearCycle-method
  (unitSeason-methods), 131
unitSeason, OpenCloseCycle-method
  (unitSeason-methods), 131
unitSeason, PartialCycle-method
  (PartialCycle-class), 46
unitSeason, QuarterYearCycle-method
  (unitSeason-methods), 131
unitSeason, SimpleCycle-method
  (unitSeason-methods), 131
unitSeason, VirtualPeriodicModel-method
  (unitSeason-methods), 131
unitSeason-methods, 131
unitSeason<--methods, 131
unitSeason<-- (allSeasons), 8
unitSeason<-- (Cyclic-method
  (unitSeason<--methods), 131
unitSeason<-- (SimpleCycle-method
  (unitSeason<--methods), 131
vcov, SubsetPM-method (SubsetPM-class), 127
Vec, 132
vector, 95, 97–99, 101–104
VirtualArmaFilter, 31, 86
VirtualBJFilter, 92
VirtualMonicFilter, 31, 86, 92, 100
VirtualMonicFilterSpec, 92, 100
VirtualPeriodicArmaModel, 31, 32, 87, 88, 96
VirtualPeriodicAutocorrelations, 47, 89,
  113
VirtualPeriodicAutocovarianceModel, 31,
  32, 87, 88, 90, 96
VirtualPeriodicAutocovariances, 91, 114
VirtualPeriodicFilterModel, 31, 32, 87,
  88, 94, 96, 110, 111, 123, 124
VirtualPeriodicMeanModel, 31, 32, 87, 88,
  96
VirtualPeriodicModel, 31, 47, 89, 91, 113,
  114
VirtualPeriodicStationaryModel, 31, 32,
  87, 88, 96
VirtualSPFilter, 100
window, 5, 14, 134, 135
xx.ss, 38
xx.ss (mc.ss), 36
zoo, 136, 137
zoo-class, 135
zooreg, 136
zooreg-class, 136