Package 'phalen'

February 20, 2015

Type Package

Title Phalen Algorithms and Functions

Version 1.0

Date 2013-08-16

Author Robert P. Bronaugh

Maintainer Robert P. Bronaugh <robertbronaugh@gmail.com>

Description The phalen package contains (1) clustering and partitioning algorithms; (2) penalty functions for numeric vectors; (3) a ranking function; and (4) color palettes and functions.

License GPL-2

Depends graphics, grDevices, stats, utils

Imports sqldf

NeedsCompilation no

Repository CRAN

Date/Publication 2013-09-12 08:00:13

R topics documented:

basher *Basher Penalty*

Description

Growth and decay penalties for numeric vectors.

Usage

```
basher(X, A, K)
```

Arguments

X	A numeric vector.
A	The value at which the penalty starts.
K	The asymptotic ceiling or floor of penalized vector X.

Details

To create a growth penalty, where values greater than A are penalized, K must be greater than A. The growth penalty is K(1-exp(-r(X-M))) for all values of X greater than A.

To create a decay penalty, where values less than A are penalized, K must be less than A. The decay penalty is K(1+exp(r(X-M))) for all values of X less than A.

Value

basher returns an object of class "basher."

y	The numeric vector X with penalities applied.
A	The value at which the penalty starts.
K	The asymptotic ceiling or floor of penalized vector X (i.e., y).
r	The growth or decay rate of the penalty.
M	An extra parameter set so that y = X at A.
penalty	The type of penalty applied.

Author(s)

Robert P. Bronaugh

Examples

```
# get the inpatient cost per day, sorted
data(ipadmits)
attach(ipadmits)
ipc = sort(ipadmits$cost)
plot(ipc,type = "l",col = wash("gry",0.8),lwd=3)

# apply penalty starting 2000. Penalized value not to exceed 4500
ipc.bash = basher(X = ipc, A = 2000, K = 4500)
lines(ipc.bash$y,col = wash("blu1",1),lwd = 3)
plot(ipc,ipc,type = "l",col = wash("gry",0.8),lwd=3)
lines(ipc,ipc.bash$y,col = wash("blu1",1),lwd = 3)

# apply lower penalty ending at 1500.  Penalized value floor = 500
ipc.bash = basher(X = ipc, A = 1500, K = 500)
plot(ipc,type = "l",col = wash("gry",0.8),lwd=3)
lines(ipc.bash$y,col = wash("blu1",1),lwd = 3)
plot(ipc,ipc,type = "l",col = wash("gry",0.8),lwd=3)
lines(ipc,ipc.bash$y,col = wash("blu1",1),lwd = 3)

# combine above ceiling and floor penalties
ipc.bash = basher(X = ipc, A = 2000, K = 4500)
ipc.bash = basher(X = ipc.bash$y, A = 1500, K = 500)

plot(ipc,type = "l",col = wash("gry",0.8),lwd=3)
lines(ipc.bash$y,col = wash("blu1",1),lwd = 3)
plot(ipc,ipc,type = "l",col = wash("gry",0.8),lwd=3)
lines(ipc,ipc.bash$y,col = wash("blu1",1),lwd = 3)
detach(ipadmits)
```

---

epclaims *Episodic Claims*

Description

Fictitious claims data used to illustrate kdpec.

Format

`PatientID` Number to identify each patient.

`ClaimNumber` A number to identify each claim.

`Diagnosis` A character (`A` or `B`) identifying two fictitious diagnoses.

`ServiceStart` The claim's service start date.

`ServiceEnd` The claim's service end date.

`glpenalty` *Generalized Logistic Penalty*

Description

Flexible generalized logistic function.

Usage

```
glpenalty(X, x0 = NA, x1 = NA, p = NA, b = NA, type = NA,
          plotpenalty = TRUE, allowed.error = 0.005, invert = FALSE)
```

Arguments

`X`	A numeric vector.
`x0`	*optional*: The value(s) where `glpentalty` is within the `allowed.error` of `0`.
`x1`	*optional*: The value(s) where `glpenalty` is within the `allowed.error` of `1`.
`p`	*optional*: A percent of `X` over which the penalty is applied. `p` must be between `0` and `1` for single sided `glpenalty` and between `0` and `0.5` for double sided `glpenalty`.
`b`	*optional*: A positive valued rate at which `glpenalty` grows or decays.
`type`	*optional*: A character value accepting `"growth"`, `"decay"`, or `"both"`
`plotpenalty`	If `TRUE`, the `glpenalty` is plotted. The default is `TRUE`.
`allowed.error`	The allowed difference between `glpenalty` and `0` or `1` at `x0` or `x1` respectively. The default is `0.005`.
`invert`	If `TRUE`, the `glpenalty` is inverted. The default is `FALSE`.

Details

The *parameter arguments* (i.e., the arguements required to parametrize `glpenalty`) are `x0`, `x1`, `p`, `b`, and `side`. Only certian combinations of these parameter arguments are required. If more than the needed number of *parameter arguments* are given, then `glpenalty` will use only the first two supplied. The following combinations of parameter arguments are accepted:

1. `{x0,x1}`: The function will grow (if `x0 < x1`) or decay (if `x0 > x1`) between the supplied values. For both growth and decay, supply 2 values to both `x0` and `x1` where `x0[1] < x1[1] < x1[2] < x0[2]`.

2. `{x1,b,side}`: The function will grow or decay starting at `x1` and at rate `b`.

3. `{p,b,side}`: The function will grow or decay starting at `p` percent of the vector `X` and at rate `b`.

Value

Returns the numeric penalty vector.

Examples

```
# create a vector of numbers
x = seq.int(1,100)


# specify x near 0 (x0) and x near 1 (x1), growth is computed
glpenalty(x,x0 = 20,x1 = 50)

#How the plot might look when numbers aren't consecutive
glpenalty(sort(sample(x,40)),x0 = 20,x1 = 50)

# decay when x near 0 (x0) is greater than x near 1 (x1)
glpenalty(x,x0 = 50,x1 = 20)

# adjust shrink allowed.error at x0 and x1.  Smaller allowed.error
# makes for "steeper" rates of growth or decay.
glpenalty(x,x0 = 50,x1 = 20,allowed.error = 0.0001)

# combine growth and decay by specifying 2 x0 and 2 x1
# growth from 10 to 35 and decay from 50 to 90.
glpenalty(x,x0 = c(10,90),x1 = c(35,50))
# invert
glpenalty(x,x0 = c(10,90),x1 = c(35,50),invert=TRUE)

# specify x1 and growth rate.
glpenalty(x,x1 = 30,b = 0.4, type = "growth")
glpenalty(x,x1 = 20,b = 0.5, type = "decay")
glpenalty(x,x1 = 30,b = 0.3, type = "both")


# specify percent of vector to be penalized
# and growth rate.

glpenalty(x,p = 0.6,b = 0.4, type = "growth")
glpenalty(x,p = 0.6,b = 0.5, type = "decay")
glpenalty(x,p = 0.4,b = 0.3, type = "both")
```

---

ipadmits *Inpatient Admissions Dataset*

Description

A dataset of fictional inpatient admissions designed to demonstrate several functions in the `phalen` package.

Format

`HospID` Hospital unique identifier.

`Age` Age of patient.

`isReadmission` Readmissions are coded as `1`.

`cost` Per day cost of inpatient stay.

`kdpec` *K-Dimensional Partitioned Episodic Clustering*

Description

Overlapping or sudo-overlapping observation clustering within k-dimensional partitions.

Usage

```
kdpec(id, kdim, startdate, enddate,
      slack = 0, restartindex = FALSE)
```

Arguments

`id`	A vector or data.frame representing a unique identifier or key.
`kdim`	A vector or data.frame representing a "k-dimensional" index across which clusters cannot be formed.
`startdate`	A date vector of each observation's start date.
`enddate`	A date vector of each observation's end date.
`slack`	a positive numeric value representing the gap in days over which a cluster can be formed. the default is `0`.
`restartindex`	If `TRUE`, the squential cluster IDs will restart at `1` within each k-dimensional partition.

Value

`kdpec` returns a data.frame with the column or group of columns used to uniquely identify each observation along with the following:

`kdimidx`	K-Dimensional Index. A sequential ID indexing each k-dimensional set.
`episode`	A sequential ID indexing each episodic cluster.

Author(s)

Robert P. Bronaugh

Examples

```
## Not run:
  # merge a patient's claims for a specific diagnosis together:
  # use kdim to prevent episode clustering across patient and diagnosis
  # (i.e.,) the combination of PatientID and Diagnosis become a partition
  # across which episodic clusters cannot be formed).
  # restartindex = TRUE starts the episode index over at 1 for each k-dimensional partition
  data(epclaims)
  attach(epclaims)
  require(sqldf)
  kd = kdpec(id = epclaims$ClaimNumber,kdim = cbind(epclaims$PatientID,epclaims$Diagnosis)
            ,startdate = epclaims$ServiceStart,enddate = epclaims$ServiceEnd
            ,restartindex=TRUE)

  # print the id, k-dimensional partition index (kdimidx), and the episodes
  print(kd)

  # restartindex = FALSE
  kd = kdpec(epclaims$ClaimNumber,cbind(epclaims$PatientID,epclaims$Diagnosis),
             epclaims$ServiceStart,epclaims$ServiceEnd,restartindex=FALSE)
  print(kd)

  # merge episode indexes with original data
  ep.2 = sqldf("SELECT  ep.PatientID
               ,ep.ClaimNumber
               ,ep.Diagnosis
               ,ep.ServiceStart
               ,ep.ServiceEnd
               ,kd.kdimidx
               ,kd.episode
               FROM    epclaims ep
               INNER JOIN kd
               ON ep.ClaimNumber = kd.id")


  # plot time spans of original records
  washcol = wash("gry",0.8)
  for (i in 1:nrow(epclaims)) {
    if (i ==1) {
      plot(c(epclaims$ServiceStart[i],epclaims$ServiceEnd[i]),rep(i,2)
           ,type="l", col = washcol, lwd = 3
           ,xlim = c(min(epclaims$ServiceStart)-3
                     ,max(epclaims$ServiceStart)+3)
           ,ylim = c(0,15)
           ,xlab = "length of service"
           ,ylab = "claim record index")
    } else if (i < 6) {
      lines(c(epclaims$ServiceStart[i],epclaims$ServiceEnd[i])
            ,rep(i,2),col = washcol, lwd = 3)
    } else if ( i < 10) {
      lines(c(epclaims$ServiceStart[i],epclaims$ServiceEnd[i])
            ,rep(i,2),col = washcol, lwd = 3)
```

```
  } else if (i == 10) {
    lines(c(epclaims$ServiceStart[i],epclaims$ServiceEnd[i])
          ,rep(i,2),col = washcol, lwd = 3)
  } else {
    lines(c(epclaims$ServiceStart[i],epclaims$ServiceEnd[i])
          ,rep(i,2),col = washcol, lwd = 3)
  }
}

# plot time spans of original records.  Color by assigned k-dim index
washcol = c(wash("blu1",1),wash("grn2",1),wash("org",1),wash("red1",1))
for (i in 1:nrow(ep.2)) {
  if (i ==1) {
    plot(c(ep.2$ServiceStart[i],ep.2$ServiceEnd[i]),rep(i,2)
         ,type="l", col = washcol[ep.2$kdimidx[i]], lwd = 3
         ,xlim = c(min(ep.2$ServiceStart)-3,max(ep.2$ServiceStart)+3)
         ,ylim = c(0,15)
         ,xlab = "length of service"
         ,ylab = "claim record index")
  } else if (i < 6) {
    lines(c(ep.2$ServiceStart[i],ep.2$ServiceEnd[i]),rep(i,2)
          ,col = washcol[ep.2$kdimidx[i]], lwd = 3)
  } else if ( i < 10) {
    lines(c(ep.2$ServiceStart[i],ep.2$ServiceEnd[i]),rep(i,2)
          ,col = washcol[ep.2$kdimidx[i]], lwd = 3)
  } else if (i == 10) {
    lines(c(ep.2$ServiceStart[i],ep.2$ServiceEnd[i]),rep(i,2)
          ,col = washcol[ep.2$kdimidx[i]], lwd = 3)
  } else {
    lines(c(ep.2$ServiceStart[i],ep.2$ServiceEnd[i]),rep(i,2)
          ,col = washcol[ep.2$kdimidx[i]], lwd = 3)
  }
}

# merge records to get the full length of each episode
ep.episodes = data.frame("kdimidx" = tapply(ep.2$kdimidx,ep.2$episode,min),
                         "episodeStart" = as.Date(tapply(ep.2$ServiceStart
                            ,ep.2$episode,min),origin = "1970-01-01"),
                         "episodeEnd" = as.Date(tapply(ep.2$ServiceEnd
                            ,ep.2$episode,max),origin = "1970-01-01"))

# plot the length of service of each episode.  kdimidx, not claim
# records, are on the y axis colors represent each kdimidx
washcol = c(wash("blu1",1),wash("grn2",1),wash("org",1),wash("red1",1))
i = 1
for (i in 1:nrow(ep.episodes)) {
  if (i ==1) {
    plot(c(ep.episodes$episodeStart[i],ep.episodes$episodeEnd[i])
          ,rep(ep.episodes$kdimidx[i],2)
         ,type="l", col = washcol[ep.episodes$kdimidx[i]], lwd = 3
         ,xlim = c(min(ep.2$ServiceStart)-3,max(ep.2$ServiceStart)+3)
         ,ylim = c(0,4)
         ,xlab = "length of episode"
```

```
            ,ylab = "k-dimensional index")
    } else if (i < 6) {
      lines(c(ep.episodes$episodeStart[i],ep.episodes$episodeEnd[i])
            ,rep(ep.episodes$kdimidx[i],2)
            ,col = washcol[ep.episodes$kdimidx[i]], lwd = 3)
    } else if ( i < 10) {
      lines(c(ep.episodes$episodeStart[i],ep.episodes$episodeEnd[i])
            ,rep(ep.episodes$kdimidx[i],2)
            ,col = washcol[ep.episodes$kdimidx[i]], lwd = 3)
    } else if (i == 10) {
      lines(c(ep.episodes$episodeStart[i],ep.episodes$episodeEnd[i])
            ,rep(ep.episodes$kdimidx[i],2)
            ,col = washcol[ep.episodes$kdimidx[i]], lwd = 3)
    } else {
      lines(c(ep.episodes$episodeStart[i],ep.episodes$episodeEnd[i])
            ,rep(ep.episodes$kdimidx[i],2)
            ,col = washcol[ep.episodes$kdimidx[i]], lwd = 3)
    }
  }
  detach(epclaims)

## End(Not run)
```

---

kparts *K-Partitions Clustering*

Description

Unsupervised vector partitioning.

Usage

```
kparts(x, y, parts, maxiter = 50, trials = 3,
       nblind = FALSE, trialprint = TRUE, iterprint = FALSE)
```

Arguments

x	The numeric vector to be partitioned.
y	The numeric response variable vector used to partition vector x.
parts	The desired number of partitions.
maxiter	The maximum number of iterations allowed for each trial. If convergence does not occur, the trail will stop after the specified number of iterations is reached. The default is 50 iterations.
trials	The number of times the algorithm is run with new, randomly assigned partitions. The default number of trials is 3.
nblind	If TRUE, the algorithm will ignore the sum of squares within each unique value of x. The default is FALSE.

trialprint	If TRUE, the trial number and the sum of squares will print while the algorithm is running. The default is TRUE.
iterprint	If TRUE, the iteration number and sum of squares will print while the algorithm is running. The default is FALSE.

Details

kparts finds the best contiguous partitions for x by minimizing the sum of squares of y.

The sum of squares for a unique value of x cannot be partitioned, which has the effect of weighting unique values of x by the number observations at those values. Using nblind = "FALSE" cause kparts to ignore the number of observations and treat all x values as equally weighted.

kparts can take a long time to process datasets with large numbers of unique x values. To gain efficiency, pre-processing vector x by binning is recommended.

Value

partitions	A data frame naming the index of the partition and the range x over which the partition extends.
data	A data frame containing the partition index (parts), the unique values of x, the average of y and the range of the partition.

Note

In later versions, kparts will be updated to allow for a matrix of data as y input.

Author(s)

Robert P. Bronaugh

Examples

```
# plot readmission rates against age.
data(ipadmits)
attach(ipadmits)
ipadmits.summary = data.frame("AvgReadmission" = tapply(ipadmits$isReadmission
                                                        ,ipadmits$Age
                                                        ,mean)
                              ,"AvgCost" = tapply(ipadmits$cost
                                                  ,ipadmits$Age
                                                  ,mean))
plot(ipadmits.summary$AvgReadmission,xlab = "Age",ylab = "AvgReadmission")


# find the best partitions of age against readmission rate.
# run kparts with 4 trials with 5 partitions
kp = kparts(x = ipadmits$Age,y = ipadmits$isReadmission,parts = 5,trials = 4)
# list value range for each partition
kp$partitions
plot(kp)
# run with 7 partitions and ignore number of samples per age
```

```
# when computing error
kp = kparts(ipadmits$Age,ipadmits$isReadmission,parts = 7,trials = 5,nblind = TRUE)
kp$partitions
plot(kp)
detach(ipadmits)
```

---

qqrank *Load-Deviance Ranking*

Description

Rank by size and deviance from the hypothesized mean.

Usage

```
qqrank(X, INDEX, alternative = c("two.sided", "less", "greater"),
       absrank = TRUE, N = NA, b = NA, plotpenalty = TRUE,
       allowed.error = 0.005)
```

Arguments

X	A numeric vector.
INDEX	A factor of length X.
alternative	The alternative hypothesis. Accepts "two.sided", "less", or "greater".
absrank	If TRUE, INDEX means greater than or less than the population mean will produce a positive qqscore. If FALSE, INDEX means greater than the population mean will have a positive qqscore and INDEX means less than the population mean will have a negative qqscore. The default is TRUE.
N	The number of observations below which a growth penalty is applied. N is passed to x1 argument of glpenalty.
b	A positive numeric value representing the growth rate of the glpenalty.
plotpenalty	If TRUE, the glpenalty is plotted. The default is TRUE.
allowed.error	The allowed difference between glpenalty and 1 at N. The default is 0.005.

Details

qqrank ranks by size and deviance from the hypothesized mean using either the Binomial Test or Welch's t-Test. Restated, qqrank is a function of a size penalty, test statistic or variant thereof, and p-value.

Value

qqrankmatrix	A data frame containing the size, mean, standard deviation, and qqrank of each INDEX.
test.used	The statistical test used to measure deviance from the mean.
pop.mean	The mean of X.
pop.sd	The standard deviation of X.

Author(s)

Robert P. Bronaugh

See Also

glpenalty t.test binom.test

Examples

```
# which hospital has the "worst" readmissions? (note: the average
# readmission rate is 17.13%
data(ipadmits)
attach(ipadmits)
ip.ag = data.frame('sum' = tapply(ipadmits$isReadmission,ipadmits$HospID,sum),
                   'avg' = tapply(ipadmits$isReadmission,ipadmits$HospID,mean))

# hospital 9 has the most readmissions (1,094), but the percent of readmissions
# is low at 14%, less than the population average.
ip.ag[order(-ip.ag$sum),][1,]

# hostpital 80 has the highest percentage of readmissions 87.5%, but only
# 7 readmissions over all.
ip.ag[order(-ip.ag$avg),][1,]

# using qqrank and penalizing samples less than N = 250 at a growth
# rate of b = 0.05, Hospital 39 has 1606 readmissions with a readmission
# percent of 38%.
qqr = qqrank(ipadmits$isReadmission,ipadmits$HospID
             ,alternative = "greater",N = 250, b = 0.05)
round(qqr$rankmatrix,2)

# relax sample penalty and rank on both sides of the mean
# Hospital 21 has the "best" readmission track record.
qqr = qqrank(ipadmits$isReadmission,ipadmits$HospID
             ,alternative = "two.sided",absrank = FALSE,N = 30, b = 0.1)
round(qqr$rankmatrix,2)
detach(ipadmits)
```

---

wash *Wash Color Palette*

Description

Color Palette for R.

Usage

```
wash(color, grade)
```

Arguments

color	Name of the color or color vector to be returned. Run wash.showall() or see details for a complete list of accepted color arguments.
grade	A positive numeric value. For grade between 0 and 1, a single color of gradient grade will be returned. For grade between 1 and 61, a character vector of N equidistant grandients will be returned.

Details

Use wash.showall() to see the list of colors available and to return 15 of the 61 color gradients. Below is a list of all colors:

"grd1" green to red, vibrant

"grd2" green to red, pale

"blu1" blue

"blu2" dark blue

"grn1" lime green

"grn2" green

"ylw" yellow

"org" orange

"red1" orange to red

"red2" pink to red

"prp1" purple

"prp2" dark purple

"cyn" cyan

"grb" grey blue

"gry" grey

Value

A color or vector of colors.

See Also

washout

Examples

```
# show all colors
wash.showall()

# ---------------------------------------------
# Chose a single color:
# ---------------------------------------------
# Specify the color then specify a gradient 0-1.
```

```
# Here is purple2 at 30%
washcol = wash('prp2',0.3)
plot(1,1,col = washcol,cex = 21,pch=16,axes = FALSE,xlab = "",ylab = "")

# purple2 at 100%
washcol = wash('prp2',1)
plot(1,1,col = washcol,cex = 21,pch=16,axes = FALSE,xlab = "",ylab = "")


# ------------------------------------------
# Chose a vector of colors:
# ------------------------------------------
# Specify the color then specify the number of
# gradients (1-61) to include in the vector. Here
# are 5 shades of blue1.
washcol = wash("blu1",5)
plot(seq.int(1,5),rep(1,5),col = washcol,pch=15,cex = 10
     ,axes = FALSE,xlim = c(0.5,5.5),xlab = "",ylab = "")

# chose 5 different colors
washcol = c(wash("blu1",1),
            wash("grn2",1),
            wash("ylw",1),
            wash("org",1),
            wash("red2",1))
plot(seq.int(1,5),rep(1,5),col = washcol,pch=15,cex = 10
     ,axes = FALSE,xlim = c(0.5,5.5),xlab = "",ylab = "")

# 61 shades of greenred1 (for heat maps)
washcol = wash("grd1",61)
plot(seq.int(1,61),rep(1,61),col = washcol,pch=15,cex = 21
     ,xlim= c(-4,54),axes = FALSE,xlab = "",ylab = "")


# ------------------------------------------
# Expand a color vector to match data
# ------------------------------------------

# plot readmission by age, no color
data(ipadmits)
attach(ipadmits)
ipadmits.summary = data.frame("AvgReadmission" = tapply(ipadmits$isReadmission
                                                        ,ipadmits$Age
                                                        ,mean)
                              ,"AvgCost" = tapply(ipadmits$cost
                                                  ,ipadmits$Age
                                                  ,mean))

plot(ipadmits.summary$AvgReadmission
     ,xlab = "Age",ylab = "AvgReadmission")

# get vector of 9 greenred1 colors then expand color vector
# to match readmission data with color gradient increasing by
```

```
# value of avg readmission
washcol = wash("grd1",9)
washoutcol = washout(ipadmits.summary$AvgReadmission,washcol,method = "value")
plot(ipadmits.summary$AvgReadmission,col=washoutcol
     ,pch=16,xlab = "Age",ylab = "AvgReadmission")

# increase gradient by the index of the vector Age of value
washoutcol = washout(ipadmits.summary[,1],washcol,method = "index")
plot(ipadmits.summary$AvgReadmission,col=washoutcol
     ,pch=16,xlab = "Age",ylab = "AvgReadmission")

# increase gradient by average cost
washoutcol = washout(ipadmits.summary$AvgCost[1:60]
                     ,washcol, method = "value")
plot(ipadmits.summary$AvgReadmission[1:60], col=washoutcol
     , pch=16,xlab = "Age", ylab = "AvgReadmission")
detach(ipadmits)
```

`washout` *Wash Out*

Description

Fit color vector to data.

Usage

```
washout(x, washcol, method = "index")
```

Arguments

`x`	A vector. For `method = "value"`, a numeric vector must be supplied.
`washcol`	A vector of colors.
`method`	Accepts `"index"` or `"value"`. The default is `"index"`.

Details

`washout` assigns a `washcol` color to each element in vector `x`. For `method = "index"`, each color element in `washcol` is evenly applied to vector `x` based on the index of vector `x`. For `method = "value"`, each color element in `washcol` is evenly applied to vector `x` based on the value of vector `x`.

Value

`washout` returns a character color vector of length `x`.

See Also

`wash`

Examples

```
# show all colors
wash.showall()

# ------------------------------------------
# Chose a single color:
# ------------------------------------------
# Specify the color then specify a gradient 0-1.
# Here is purple2 at 30%
washcol = wash('prp2',0.3)
plot(1,1,col = washcol,cex = 21,pch=16,axes = FALSE,xlab = "",ylab = "")

# purple2 at 100%
washcol = wash('prp2',1)
plot(1,1,col = washcol,cex = 21,pch=16,axes = FALSE,xlab = "",ylab = "")


# ------------------------------------------
# Chose a vector of colors:
# ------------------------------------------
# Specify the color then specify the number of
# gradients (1-61) to include in the vector. Here
# are 5 shades of blue1.
washcol = wash("blu1",5)
plot(seq.int(1,5),rep(1,5),col = washcol,pch=15,cex = 10
     ,axes = FALSE,xlim = c(0.5,5.5),xlab = "",ylab = "")

# chose 5 different colors
washcol = c(wash("blu1",1),
            wash("grn2",1),
            wash("ylw",1),
            wash("org",1),
            wash("red2",1))
plot(seq.int(1,5),rep(1,5),col = washcol,pch=15,cex = 10
     ,axes = FALSE,xlim = c(0.5,5.5),xlab = "",ylab = "")

# 61 shades of greenred1 (for heat maps)
washcol = wash("grd1",61)
plot(seq.int(1,61),rep(1,61),col = washcol,pch=15,cex = 21
     ,xlim= c(-4,54),axes = FALSE,xlab = "",ylab = "")


# ------------------------------------------
# Expand a color vector to match data
# ------------------------------------------

# plot readmission by age, no color
data(ipadmits)
attach(ipadmits)
ipadmits.summary = data.frame("AvgReadmission" = tapply(ipadmits$isReadmission
                                                        ,ipadmits$Age
                                                        ,mean)
```

```
                              ,"AvgCost" = tapply(ipadmits$cost
                                                  ,ipadmits$Age
                                                  ,mean))

plot(ipadmits.summary$AvgReadmission
     ,xlab = "Age",ylab = "AvgReadmission")

# get vector of 9 greenred1 colors then expand color vector
# to match readmission data with color gradient increasing by
# value of avg readmission
washcol = wash("grd1",9)
washoutcol = washout(ipadmits.summary$AvgReadmission,washcol,method = "value")
plot(ipadmits.summary$AvgReadmission,col=washoutcol
     ,pch=16,xlab = "Age",ylab = "AvgReadmission")

# increase gradient by the index of the vector Age of value
washoutcol = washout(ipadmits.summary[,1],washcol,method = "index")
plot(ipadmits.summary$AvgReadmission,col=washoutcol
     ,pch=16,xlab = "Age",ylab = "AvgReadmission")

# increase gradient by average cost
washoutcol = washout(ipadmits.summary$AvgCost[1:60]
                     ,washcol, method = "value")
plot(ipadmits.summary$AvgReadmission[1:60], col=washoutcol
     , pch=16,xlab = "Age", ylab = "AvgReadmission")
detach(ipadmits)
```

Index