jumbled demonstrations

Sasha D. Hafner

12 February, 2024

Overview

This document demonstrates usage of some of the function in the jumbled repo, available from github.com/sashahafner/jumbled.

Load functions

```
ff <- list.files(pattern = '\\.R$')
for(i in ff) source(i)
```

aggregate2

A wrapper for `aggregate` that accepts multiple functions and simpler arguments. Does not accept formula notation.

Example from `aggregate` help file:

```
aggregate(breaks ~ wool + tension, data = warpbreaks, mean)
```

```
##   wool tension   breaks
## 1    A       L 44.55556
## 2    B       L 28.22222
## 3    A       M 24.00000
## 4    B       M 28.77778
## 5    A       H 24.55556
## 6    B       H 18.77778
```

To include sd and n, use `aggregate2`:

```
aggregate2(warpbreaks, x = 'breaks', by = c('wool', 'tension'),
           FUN = list(mean = mean, sd = sd, n = length))
```

```
##   wool tension breaks.mean breaks.sd breaks.n
## 1    A       L    44.55556 18.097729        9
## 2    B       L    28.22222  9.858724        9
## 3    A       M    24.00000  8.660254        9
## 4    B       M    28.77778  9.431036        9
## 5    A       H    24.55556 10.272671        9
## 6    B       H    18.77778  4.893306        9
```

Accepts multiple variables (as in `aggregate`).

```
aggregate2(na.omit(airquality), x = c('Ozone', 'Temp'), by = 'Month',
       FUN = list(mean = mean, sd = sd, n = length))
```

```
##   Month Ozone.mean Temp.mean Ozone.sd  Temp.sd Ozone.n Temp.n
## 1     5   24.12500  66.45833 22.88594 6.633113      24     24
## 2     6   29.44444  78.22222 18.20790 7.838651       9      9
## 3     7   59.11538  83.88462 31.63584 4.439161      26     26
## 4     8   60.00000  83.69565 41.76776 7.054559      23     23
## 5     9   31.44828  76.89655 24.14182 8.503549      29     29
```

aggregate3

Similar, but uses formula notation. Example from `aggregate` help file:

```
aggregate(breaks ~ wool + tension, data = warpbreaks, mean)
```

```
##   wool tension   breaks
## 1    A       L 44.55556
## 2    B       L 28.22222
## 3    A       M 24.00000
## 4    B       M 28.77778
## 5    A       H 24.55556
## 6    B       H 18.77778
```

To include sd and n, use `aggregate3`:

```
aggregate3(warpbreaks, breaks ~ wool + tension,
       FUN = list(mean = mean, sd = sd, n = length))
```

```
##   wool tension breaks.mean breaks.sd breaks.n
## 1    A       L    44.55556 18.097729        9
## 2    B       L    28.22222  9.858724        9
## 3    A       M    24.00000  8.660254        9
## 4    B       M    28.77778  9.431036        9
## 5    A       H    24.55556 10.272671        9
## 6    B       H    18.77778  4.893306        9
```

For multiple response variables, use `cbind()`.

```
aggregate3(airquality, cbind(Ozone, Temp) ~ Month,
       FUN = list(mean = mean, sd = sd, n = length))
```

```
##   Month Ozone.mean Temp.mean Ozone.sd  Temp.sd Ozone.n Temp.n
## 1     5   23.61538  66.73077 22.22445 6.533346      26     26
## 2     6   29.44444  78.22222 18.20790 7.838651       9      9
## 3     7   59.11538  83.88462 31.63584 4.439161      26     26
## 4     8   59.96154  83.96154 39.68121 6.666218      26     26
## 5     9   31.44828  76.89655 24.14182 8.503549      29     29
```

So `Ozone + Temp ~ Month` doesn't work, because `aggregate()` can't handle it propertly. It would be nice to address this limitation in the future.

dfcombos

Something like `expand.grid` for data frames. Can accept vectors too, but resulting name is poor.

```r
d1 <- data.frame(name = letters[1:5], x = 1.1)
d2 <- data.frame(b = 1:3)
dfcombos(d1, d2)
```

```
##    name   x b
## 1     a 1.1 1
## 2     b 1.1 1
## 3     c 1.1 1
## 4     d 1.1 1
## 5     e 1.1 1
## 6     a 1.1 2
## 7     b 1.1 2
## 8     c 1.1 2
## 9     d 1.1 2
## 10    e 1.1 2
## 11    a 1.1 3
## 12    b 1.1 3
## 13    c 1.1 3
## 14    d 1.1 3
## 15    e 1.1 3
```

```r
v1 <- c(TRUE, FALSE)
dfcombos(d1, d2, v1)
```

```
##    name   x b X[[i]]
## 1     a 1.1 1   TRUE
## 2     b 1.1 1   TRUE
## 3     c 1.1 1   TRUE
## 4     d 1.1 1   TRUE
## 5     e 1.1 1   TRUE
## 6     a 1.1 2   TRUE
## 7     b 1.1 2   TRUE
## 8     c 1.1 2   TRUE
## 9     d 1.1 2   TRUE
## 10    e 1.1 2   TRUE
## 11    a 1.1 3   TRUE
## 12    b 1.1 3   TRUE
## 13    c 1.1 3   TRUE
## 14    d 1.1 3   TRUE
## 15    e 1.1 3   TRUE
## 16    a 1.1 1  FALSE
## 17    b 1.1 1  FALSE
## 18    c 1.1 1  FALSE
## 19    d 1.1 1  FALSE
## 20    e 1.1 1  FALSE
## 21    a 1.1 2  FALSE
## 22    b 1.1 2  FALSE
## 23    c 1.1 2  FALSE
## 24    d 1.1 2  FALSE
## 25    e 1.1 2  FALSE
## 26    a 1.1 3  FALSE
## 27    b 1.1 3  FALSE
## 28    c 1.1 3  FALSE
## 29    d 1.1 3  FALSE
```

```
## 30    e 1.1 3  FALSE
```

dfsumm

Generate a data frame summary more detailed and compact than `summary` output.

```
dfsumm(attenu)
```

```
##
##  182 rows and 5 columns
##  182 unique rows
##                       event     mag station    dist   accel
## Class              numeric numeric  factor numeric numeric
## Minimum                  1       5    1008     0.5   0.003
## Maximum                 23     7.7    c266     370    0.81
## Mean                  14.7    6.08     262    45.6   0.154
## Unique (excld. NA)      23      17     117     153     120
## Missing values           0       0      16       0       0
## Sorted                TRUE   FALSE   FALSE   FALSE   FALSE
##
```

Compare to `summary`.

```
summary(attenu)
```

```
##      event            mag           station         dist
##  Min.   : 1.00   Min.   :5.000   117    :  5   Min.   :  0.50
##  1st Qu.: 9.00   1st Qu.:5.300   1028   :  4   1st Qu.: 11.32
##  Median :18.00   Median :6.100   113    :  4   Median : 23.40
##  Mean   :14.74   Mean   :6.084   112    :  3   Mean   : 45.60
##  3rd Qu.:20.00   3rd Qu.:6.600   135    :  3   3rd Qu.: 47.55
##  Max.   :23.00   Max.   :7.700   (Other):147   Max.   :370.00
##                                  NA's   : 16
##      accel
##  Min.   :0.00300
##  1st Qu.:0.04425
##  Median :0.11300
##  Mean   :0.15422
##  3rd Qu.:0.21925
##  Max.   :0.81000
##
```

interpm

Fill in missing observations for multiple columns via interpolation. `interpm` calls `approx`.

```
args(interpm)
```

```
## function (dat, x, ys, by = NA, ...)
## NULL
```

```
dat <- data.frame(time = 1:30, a = rnorm(30), b = rnorm(30), c = rnorm(30))
dat[5:10, -1] <- NA
dat[20:22, 'a'] <- NA

dat
```

```
##    time           a           b           c
## 1     1  0.42602283 -0.83747766  0.56810401
## 2     2 -1.28166622 -1.14729446 -0.46049699
## 3     3  1.24570368  0.57220312 -0.02231355
## 4     4  1.36619497  1.28136221 -0.01197275
## 5     5          NA          NA          NA
## 6     6          NA          NA          NA
## 7     7          NA          NA          NA
## 8     8          NA          NA          NA
## 9     9          NA          NA          NA
## 10   10          NA          NA          NA
## 11   11 -0.42143784 -0.02074321  0.30763951
## 12   12 -0.68503482 -1.46543222 -0.61838258
## 13   13  0.62998211  0.80338166 -0.70249645
## 14   14  0.16632581 -1.72369795 -0.64764132
## 15   15 -0.54649013 -0.47985197 -0.24780292
## 16   16 -0.65645337  1.48771365  0.43599306
## 17   17  1.62567689 -0.26293628 -1.10097251
## 18   18  1.75805754  0.34573414  0.33833938
## 19   19 -1.00370661  0.08725073  0.24010344
## 20   20          NA  2.00856802  1.19421929
## 21   21          NA -1.41668539 -0.64250194
## 22   22          NA -0.32769628 -1.95181870
## 23   23 -0.61564940 -1.45811983  0.86473234
## 24   24 -0.35683196  0.63975764  0.05530019
## 25   25 -0.09595417  0.73920378 -2.93751468
## 26   26 -1.24642048  1.46266016 -0.62561441
## 27   27  0.98199107  1.45556390 -0.03645061
## 28   28  0.84079085  0.26250693  0.06506766
## 29   29 -0.50798089 -0.69510417 -0.63492162
## 30   30  0.56830151  0.82018341  1.24238818
```

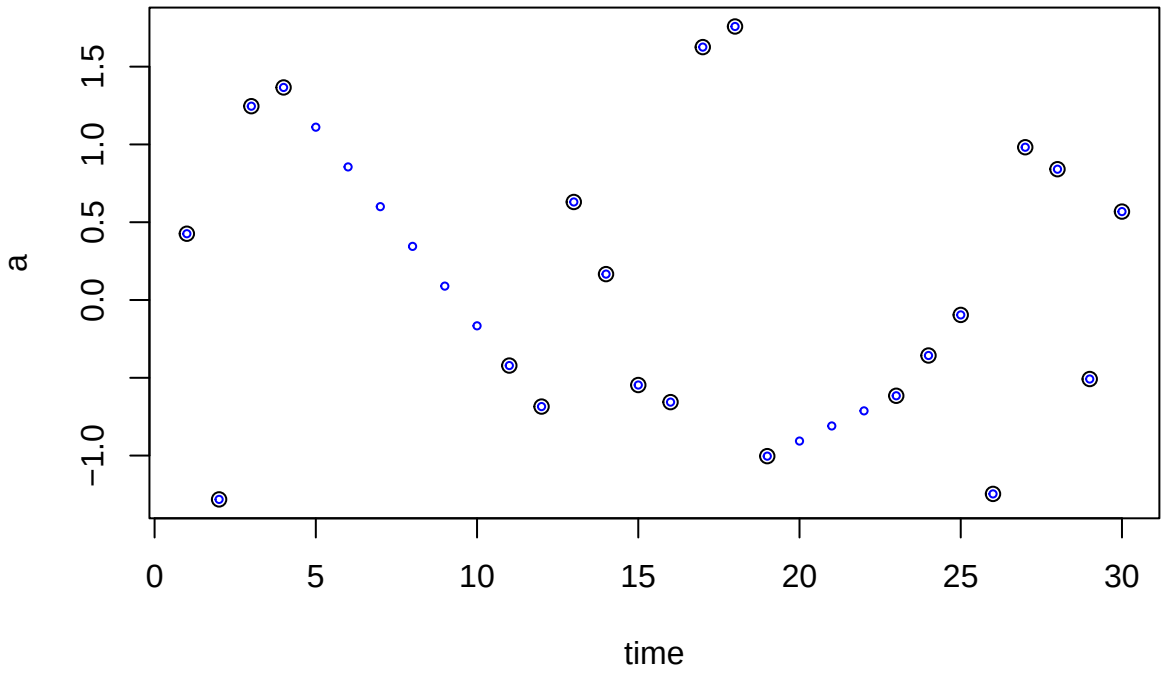
```
dat2 <- interpm(dat, 'time', c('a', 'b', 'c'))

dat2
```

```
##    time           a           b           c
## 1     1  0.42602283 -0.83747766  0.56810401
## 2     2 -1.28166622 -1.14729446 -0.46049699
## 3     3  1.24570368  0.57220312 -0.02231355
## 4     4  1.36619497  1.28136221 -0.01197275
## 5     5  1.11081886  1.09534715  0.03368615
## 6     6  0.85544274  0.90933209  0.07934504
## 7     7  0.60006662  0.72331703  0.12500394
## 8     8  0.34469051  0.53730197  0.17066283
## 9     9  0.08931439  0.35128691  0.21632173
## 10   10 -0.16606172  0.16527185  0.26198062
## 11   11 -0.42143784 -0.02074321  0.30763951
## 12   12 -0.68503482 -1.46543222 -0.61838258
## 13   13  0.62998211  0.80338166 -0.70249645
## 14   14  0.16632581 -1.72369795 -0.64764132
## 15   15 -0.54649013 -0.47985197 -0.24780292
## 16   16 -0.65645337  1.48771365  0.43599306
## 17   17  1.62567689 -0.26293628 -1.10097251
## 18   18  1.75805754  0.34573414  0.33833938
```

```
## 19   19 -1.00370661  0.08725073  0.24010344
## 20   20 -0.90669231  2.00856802  1.19421929
## 21   21 -0.80967800 -1.41668539 -0.64250194
## 22   22 -0.71266370 -0.32769628 -1.95181870
## 23   23 -0.61564940 -1.45811983  0.86473234
## 24   24 -0.35683196  0.63975764  0.05530019
## 25   25 -0.09595417  0.73920378 -2.93751468
## 26   26 -1.24642048  1.46266016 -0.62561441
## 27   27  0.98199107  1.45556390 -0.03645061
## 28   28  0.84079085  0.26250693  0.06506766
## 29   29 -0.50798089 -0.69510417 -0.63492162
## 30   30  0.56830151  0.82018341  1.24238818
```

```
plot(a ~ time, data = dat)
points(a ~ time, data = dat2, cex = 0.5, col = 'blue')
```



Now woks for data.tables too.

```
dat <- data.table::as.data.table(dat)
dat2 <- interpm(dat, 'time', c('a', 'b', 'c'))
```

```
dat <- data.frame(time = rep(1:10, 3), group = rep(c('a', 'b', 'c'), each = 10), a = rnorm(30), b = rno
dat[5:9, -1:-2] <- NA
dat[c(20, 22), 'a'] <- NA

dat
```

```
##    time group           a           b           c
## 1     1     a -0.09984584 -0.69126279  1.28253277
## 2     2     a  0.27929311  0.34841194 -2.27360921
## 3     3     a -2.29457451 -1.72321135 -0.91340095
## 4     4     a -0.39341071  0.24427701 -0.40928836
## 5     5     a          NA          NA          NA
## 6     6     a          NA          NA          NA
```

```
## 7     7     a           NA          NA          NA
## 8     8     a           NA          NA          NA
## 9     9     a           NA          NA          NA
## 10   10     a -0.715319608  1.78301197  0.19610528
## 11    1     b  0.977166708 -1.10936579  0.51072651
## 12    2     b -0.816234501 -0.99476832  0.40154101
## 13    3     b  0.679361993  1.73202874 -0.11063503
## 14    4     b -1.069573471 -0.35339266  1.29316820
## 15    5     b  0.148636155 -0.48146387 -0.29451216
## 16    6     b -0.109232658 -0.29087415  0.67510212
## 17    7     b -0.002795425  1.09003267 -0.03076447
## 18    8     b -1.682816869  0.35022395  0.55954100
## 19    9     b  0.781807143 -0.20013043 -0.75496529
## 20   10     b           NA -0.83180867  1.98032353
## 21    1     c -0.500382795 -0.18489119  0.30585021
## 22    2     c           NA -0.70307454 -2.75336550
## 23    3     c  2.335652854  0.95398577  0.27257391
## 24    4     c  0.024890692 -0.37968664 -1.76284659
## 25    5     c -0.568980751  0.19836043 -0.12572581
## 26    6     c -0.320961625  0.56736713 -1.43849442
## 27    7     c -0.042430969  0.49220374 -0.79726776
## 28    8     c  1.420622580 -0.06560438 -0.30698078
## 29    9     c -1.028488583 -0.22370923  1.05900547
## 30   10     c -0.862451716 -1.04173099 -0.68862816
```

```
interpm(dat, 'time', c('a', 'b', 'c'), by = 'group')
```

```
##    time group           a           b           c
## 1     1     a -0.099845844 -0.69126279  1.282532773
## 2     2     a  0.279293114  0.34841194 -2.273609212
## 3     3     a -2.294574512 -1.72321135 -0.913400955
## 4     4     a -0.393410713  0.24427701 -0.409288361
## 5     5     a -0.447062196  0.50073284 -0.308389421
## 6     6     a -0.500713678  0.75718866 -0.207490481
## 7     7     a -0.554365161  1.01364449 -0.106591541
## 8     8     a -0.608016643  1.27010031 -0.005692601
## 9     9     a -0.661668125  1.52655614  0.095206338
## 10   10     a -0.715319608  1.78301197  0.196105278
## 11    1     b  0.977166708 -1.10936579  0.510726510
## 12    2     b -0.816234501 -0.99476832  0.401541009
## 13    3     b  0.679361993  1.73202874 -0.110635032
## 14    4     b -1.069573471 -0.35339266  1.293168202
## 15    5     b  0.148636155 -0.48146387 -0.294512164
## 16    6     b -0.109232658 -0.29087415  0.675102117
## 17    7     b -0.002795425  1.09003267 -0.030764467
## 18    8     b -1.682816869  0.35022395  0.559540996
## 19    9     b  0.781807143 -0.20013043 -0.754965286
## 20   10     b           NA -0.83180867  1.980323533
## 21    1     c -0.500382795 -0.18489119  0.305850207
## 22    2     c  0.917635029 -0.70307454 -2.753365504
## 23    3     c  2.335652854  0.95398577  0.272573909
## 24    4     c  0.024890692 -0.37968664 -1.762846590
## 25    5     c -0.568980751  0.19836043 -0.125725809
## 26    6     c -0.320961625  0.56736713 -1.438494425
## 27    7     c -0.042430969  0.49220374 -0.797267764
```

```
## 28    8     c  1.420622580 -0.06560438 -0.306980780
## 29    9     c -1.028488583 -0.22370923  1.059005474
## 30   10     c -0.862451716 -1.04173099 -0.688628160
```

```
interpm(dat, 'time', c('a', 'b', 'c'), by = 'group', rule = 2)
```

```
##    time group           a           b           c
## 1     1     a -0.099845844 -0.69126279  1.282532773
## 2     2     a  0.279293114  0.34841194 -2.273609212
## 3     3     a -2.294574512 -1.72321135 -0.913400955
## 4     4     a -0.393410713  0.24427701 -0.409288361
## 5     5     a -0.447062196  0.50073284 -0.308389421
## 6     6     a -0.500713678  0.75718866 -0.207490481
## 7     7     a -0.554365161  1.01364449 -0.106591541
## 8     8     a -0.608016643  1.27010031 -0.005692601
## 9     9     a -0.661668125  1.52655614  0.095206338
## 10   10     a -0.715319608  1.78301197  0.196105278
## 11    1     b  0.977166708 -1.10936579  0.510726510
## 12    2     b -0.816234501 -0.99476832  0.401541009
## 13    3     b  0.679361993  1.73202874 -0.110635032
## 14    4     b -1.069573471 -0.35339266  1.293168202
## 15    5     b  0.148636155 -0.48146387 -0.294512164
## 16    6     b -0.109232658 -0.29087415  0.675102117
## 17    7     b -0.002795425  1.09003267 -0.030764467
## 18    8     b -1.682816869  0.35022395  0.559540996
## 19    9     b  0.781807143 -0.20013043 -0.754965286
## 20   10     b  0.781807143 -0.83180867  1.980323533
## 21    1     c -0.500382795 -0.18489119  0.305850207
## 22    2     c  0.917635029 -0.70307454 -2.753365504
## 23    3     c  2.335652854  0.95398577  0.272573909
## 24    4     c  0.024890692 -0.37968664 -1.762846590
## 25    5     c -0.568980751  0.19836043 -0.125725809
## 26    6     c -0.320961625  0.56736713 -1.438494425
## 27    7     c -0.042430969  0.49220374 -0.797267764
## 28    8     c  1.420622580 -0.06560438 -0.306980780
## 29    9     c -1.028488583 -0.22370923  1.059005474
## 30   10     c -0.862451716 -1.04173099 -0.688628160
```

```
dat <- data.table::as.data.table(dat)
dat
```

```
##     time group           a           b           c
##  1:    1     a -0.099845844 -0.69126279  1.28253277
##  2:    2     a  0.279293114  0.34841194 -2.27360921
##  3:    3     a -2.294574512 -1.72321135 -0.91340095
##  4:    4     a -0.393410713  0.24427701 -0.40928836
##  5:    5     a           NA          NA          NA
##  6:    6     a           NA          NA          NA
##  7:    7     a           NA          NA          NA
##  8:    8     a           NA          NA          NA
##  9:    9     a           NA          NA          NA
## 10:   10     a -0.715319608  1.78301197  0.19610528
## 11:    1     b  0.977166708 -1.10936579  0.51072651
## 12:    2     b -0.816234501 -0.99476832  0.40154101
## 13:    3     b  0.679361993  1.73202874 -0.11063503
## 14:    4     b -1.069573471 -0.35339266  1.29316820
```

```
## 15:    5     b  0.148636155 -0.48146387 -0.29451216
## 16:    6     b -0.109232658 -0.29087415  0.67510212
## 17:    7     b -0.002795425  1.09003267 -0.03076447
## 18:    8     b -1.682816869  0.35022395  0.55954100
## 19:    9     b  0.781807143 -0.20013043 -0.75496529
## 20:   10     b           NA -0.83180867  1.98032353
## 21:    1     c -0.500382795 -0.18489119  0.30585021
## 22:    2     c           NA -0.70307454 -2.75336550
## 23:    3     c  2.335652854  0.95398577  0.27257391
## 24:    4     c  0.024890692 -0.37968664 -1.76284659
## 25:    5     c -0.568980751  0.19836043 -0.12572581
## 26:    6     c -0.320961625  0.56736713 -1.43849442
## 27:    7     c -0.042430969  0.49220374 -0.79726776
## 28:    8     c  1.420622580 -0.06560438 -0.30698078
## 29:    9     c -1.028488583 -0.22370923  1.05900547
## 30:   10     c -0.862451716 -1.04173099 -0.68862816
##     time group            a           b           c
```

```
interpm(dat, 'time', c('a', 'b', 'c'), by = 'group')
```

```
##     time group            a           b           c
##  1:    1     a -0.099845844 -0.69126279  1.282532773
##  2:    2     a  0.279293114  0.34841194 -2.273609212
##  3:    3     a -2.294574512 -1.72321135 -0.913400955
##  4:    4     a -0.393410713  0.24427701 -0.409288361
##  5:    5     a -0.447062196  0.50073284 -0.308389421
##  6:    6     a -0.500713678  0.75718866 -0.207490481
##  7:    7     a -0.554365161  1.01364449 -0.106591541
##  8:    8     a -0.608016643  1.27010031 -0.005692601
##  9:    9     a -0.661668125  1.52655614  0.095206338
## 10:   10     a -0.715319608  1.78301197  0.196105278
## 11:    1     b  0.977166708 -1.10936579  0.510726510
## 12:    2     b -0.816234501 -0.99476832  0.401541009
## 13:    3     b  0.679361993  1.73202874 -0.110635032
## 14:    4     b -1.069573471 -0.35339266  1.293168202
## 15:    5     b  0.148636155 -0.48146387 -0.294512164
## 16:    6     b -0.109232658 -0.29087415  0.675102117
## 17:    7     b -0.002795425  1.09003267 -0.030764467
## 18:    8     b -1.682816869  0.35022395  0.559540996
## 19:    9     b  0.781807143 -0.20013043 -0.754965286
## 20:   10     b           NA -0.83180867  1.980323533
## 21:    1     c -0.500382795 -0.18489119  0.305850207
## 22:    2     c  0.917635029 -0.70307454 -2.753365504
## 23:    3     c  2.335652854  0.95398577  0.272573909
## 24:    4     c  0.024890692 -0.37968664 -1.762846590
## 25:    5     c -0.568980751  0.19836043 -0.125725809
## 26:    6     c -0.320961625  0.56736713 -1.438494425
## 27:    7     c -0.042430969  0.49220374 -0.797267764
## 28:    8     c  1.420622580 -0.06560438 -0.306980780
## 29:    9     c -1.028488583 -0.22370923  1.059005474
## 30:   10     c -0.862451716 -1.04173099 -0.688628160
##     time group            a           b           c
```

```
interpm(dat, 'time', c('a', 'b', 'c'), by = 'group', rule = 2)
```

```
##     time group           a           b           c
##  1:    1     a -0.099845844 -0.69126279  1.282532773
##  2:    2     a  0.279293114  0.34841194 -2.273609212
##  3:    3     a -2.294574512 -1.72321135 -0.913400955
##  4:    4     a -0.393410713  0.24427701 -0.409288361
##  5:    5     a -0.447062196  0.50073284 -0.308389421
##  6:    6     a -0.500713678  0.75718866 -0.207490481
##  7:    7     a -0.554365161  1.01364449 -0.106591541
##  8:    8     a -0.608016643  1.27010031 -0.005692601
##  9:    9     a -0.661668125  1.52655614  0.095206338
## 10:   10     a -0.715319608  1.78301197  0.196105278
## 11:    1     b  0.977166708 -1.10936579  0.510726510
## 12:    2     b -0.816234501 -0.99476832  0.401541009
## 13:    3     b  0.679361993  1.73202874 -0.110635032
## 14:    4     b -1.069573471 -0.35339266  1.293168202
## 15:    5     b  0.148636155 -0.48146387 -0.294512164
## 16:    6     b -0.109232658 -0.29087415  0.675102117
## 17:    7     b -0.002795425  1.09003267 -0.030764467
## 18:    8     b -1.682816869  0.35022395  0.559540996
## 19:    9     b  0.781807143 -0.20013043 -0.754965286
## 20:   10     b  0.781807143 -0.83180867  1.980323533
## 21:    1     c -0.500382795 -0.18489119  0.305850207
## 22:    2     c  0.917635029 -0.70307454 -2.753365504
## 23:    3     c  2.335652854  0.95398577  0.272573909
## 24:    4     c  0.024890692 -0.37968664 -1.762846590
## 25:    5     c -0.568980751  0.19836043 -0.125725809
## 26:    6     c -0.320961625  0.56736713 -1.438494425
## 27:    7     c -0.042430969  0.49220374 -0.797267764
## 28:    8     c  1.420622580 -0.06560438 -0.306980780
## 29:    9     c -1.028488583 -0.22370923  1.059005474
## 30:   10     c -0.862451716 -1.04173099 -0.688628160
##     time group           a           b           c
```

logaxis

Add log axis to base R plots.

logistic

The logistic function for transformations.

rbindf

Like `rbind` but data frame columns do not need to match. From monitoR package.

rounddf

Round complete data frames.

```
dat <- data.frame(a = 1:10, b = rnorm(10), c = letters[1:10])
dat
```

```
##     a          b c
## 1   1  0.3750802 a
## 2   2  0.6922055 b
## 3   3  0.5080144 c
## 4   4  1.2253288 d
## 5   5 -1.5011495 e
## 6   6  0.4150171 f
## 7   7 -1.5422410 g
## 8   8  2.3978156 h
## 9   9 -0.4108147 i
## 10 10 -0.4976349 j
```

```
rounddf(dat)
```

```
##     a     b c
## 1   1  0.38 a
## 2   2  0.69 b
## 3   3  0.51 c
## 4   4  1.23 d
## 5   5 -1.50 e
## 6   6  0.42 f
## 7   7 -1.54 g
## 8   8  2.40 h
## 9   9 -0.41 i
## 10 10 -0.50 j
```

```
rounddf(dat, digits = c(0, 4))
```

```
## Warning in rounddf(dat, digits = c(0, 4)): First value in digits repeated to
## match length.
```

```
##     a       b c
## 1   1  0.3751 a
## 2   2  0.6922 b
## 3   3  0.5080 c
## 4   4  1.2253 d
## 5   5 -1.5011 e
## 6   6  0.4150 f
## 7   7 -1.5422 g
## 8   8  2.3978 h
## 9   9 -0.4108 i
## 10 10 -0.4976 j
```

```
rounddf(dat, digits = c(0, 4), func = signif)
```

```
## Warning in rounddf(dat, digits = c(0, 4), func = signif): First value in digits
## repeated to match length.
```

```
##     a       b c
## 1   1  0.3751 a
## 2   2  0.6922 b
## 3   3  0.5080 c
## 4   4  1.2250 d
## 5   5 -1.5010 e
```

```
## 6   6  0.4150 f
## 7   7 -1.5420 g
## 8   8  2.3980 h
## 9   9 -0.4108 i
## 10 10 -0.4976 j
```

```
rounddf(dat, digits = c(2, 2), func = signif)
```

```
## Warning in rounddf(dat, digits = c(2, 2), func = signif): First value in digits
## repeated to match length.
```

```
##     a     b c
## 1   1  0.38 a
## 2   2  0.69 b
## 3   3  0.51 c
## 4   4  1.20 d
## 5   5 -1.50 e
## 6   6  0.42 f
## 7   7 -1.50 g
## 8   8  2.40 h
## 9   9 -0.41 i
## 10 10 -0.50 j
```

Trailing zeroes are dropped when written out (although this does not show up in R console). Avoid with `pad = TRUE`, which converts adds trailing zeroes and converts column to character.

```
set.seed(124)
dat <- data.frame(a = 1:10, b = rnorm(10), c = letters[1:10])
dat
```

```
##     a           b c
## 1   1 -1.38507062 a
## 2   2  0.03832318 b
## 3   3 -0.76303016 c
## 4   4  0.21230614 d
## 5   5  1.42553797 e
## 6   6  0.74447982 f
## 7   7  0.70022940 g
## 8   8 -0.22935461 h
## 9   9  0.19709386 i
## 10 10  1.20715377 j
```

```
summary(dat)
```

```
##        a               b                  c
##  Min.   : 1.00   Min.   :-1.3851   Length:10
##  1st Qu.: 3.25   1st Qu.:-0.1624   Class :character
##  Median : 5.50   Median : 0.2047   Mode  :character
##  Mean   : 5.50   Mean   : 0.2148
##  3rd Qu.: 7.75   3rd Qu.: 0.7334
##  Max.   :10.00   Max.   : 1.4255
```

```
rounddf(dat)
```

```
##     a     b c
## 1   1 -1.39 a
## 2   2  0.04 b
## 3   3 -0.76 c
```

```
## 4   4  0.21 d
## 5   5  1.43 e
## 6   6  0.74 f
## 7   7  0.70 g
## 8   8 -0.23 h
## 9   9  0.20 i
## 10 10  1.21 j
```

```
rounddf(dat, pad = TRUE)
```

```
##     a     b c
## 1   1 -1.39 a
## 2   2  0.04 b
## 3   3 -0.76 c
## 4   4  0.21 d
## 5   5  1.43 e
## 6   6  0.74 f
## 7   7  0.70 g
## 8   8 -0.23 h
## 9   9  0.20 i
## 10 10  1.21 j
```

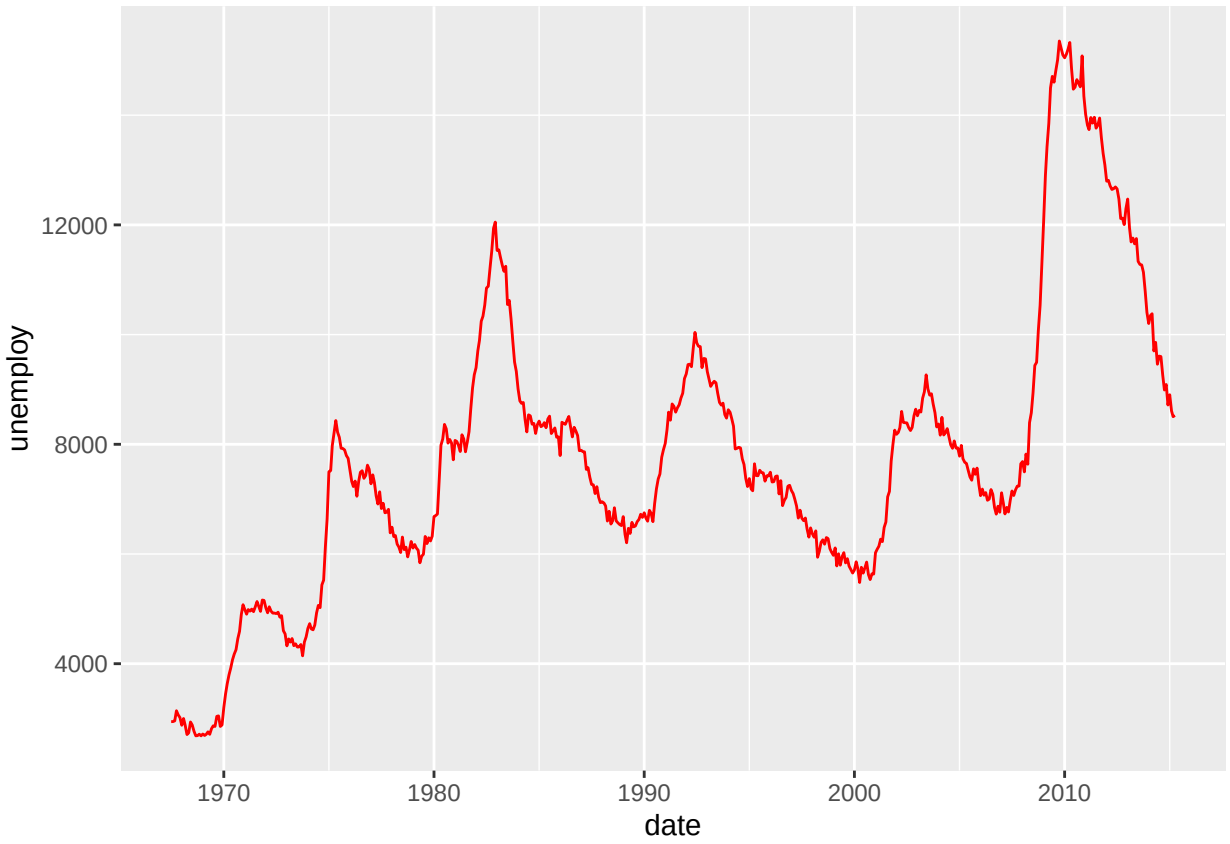
```
dat <- rounddf(dat, pad = TRUE)
summary(dat)
```

```
##        a              b                  c
##  Min.   : 1.00   Length:10          Length:10
##  1st Qu.: 3.25   Class :character   Class :character
##  Median : 5.50   Mode  :character   Mode  :character
##  Mean   : 5.50
##  3rd Qu.: 7.75
##  Max.   :10.00
```

ggsave2x

Save a ggplot2 figure in more than one format in a single call.

```
library(ggplot2)
ggplot(economics, aes(date, unemploy)) +
  geom_line(colour = "red")
```




```
ggsave2x('economics', width = 5, height = 5)
```

Saves png and pdf by default, add more with `type` argument. Use `...` optional arguments for more flexibility.

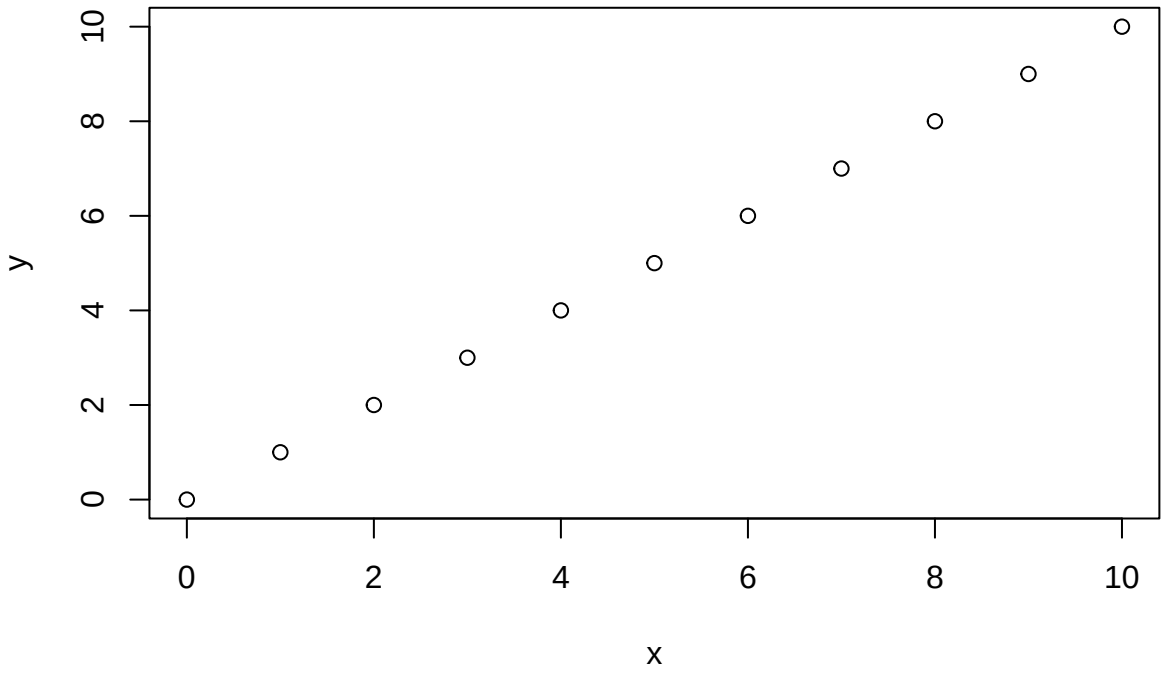
`mintegrate`

Integrate *f*lux measurements for emission.

```
source('mintegrate.R')
```

1. Linear

```
x <- 0:10
y <- 0:10
plot(x, y)
```



Exact integral is `10 * 10 / 2 = 50`.

```
mintegrate(x, y, 'midpoint')
```

```
##  [1]  0.0  0.5  2.0  4.5  8.0 12.5 18.0 24.5 32.0 40.5 50.0
```

```
mintegrate(x, y, 'left')
```

```
##  [1]  0  1  3  6 10 15 21 28 36 45 55
```

```
mintegrate(x, y, 'right')
```

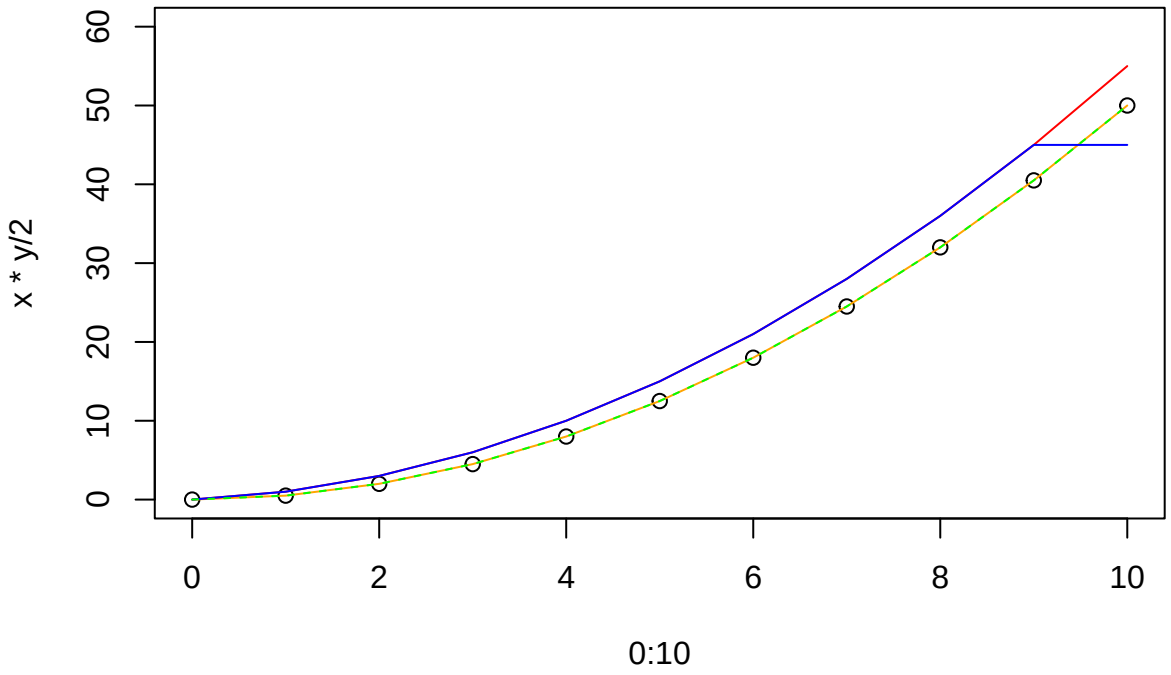
```
##  [1]  0  1  3  6 10 15 21 28 36 45 45
```

```
mintegrate(x, y, 'trap')
```

```
##  [1]  0.0  0.5  2.0  4.5  8.0 12.5 18.0 24.5 32.0 40.5 50.0
```

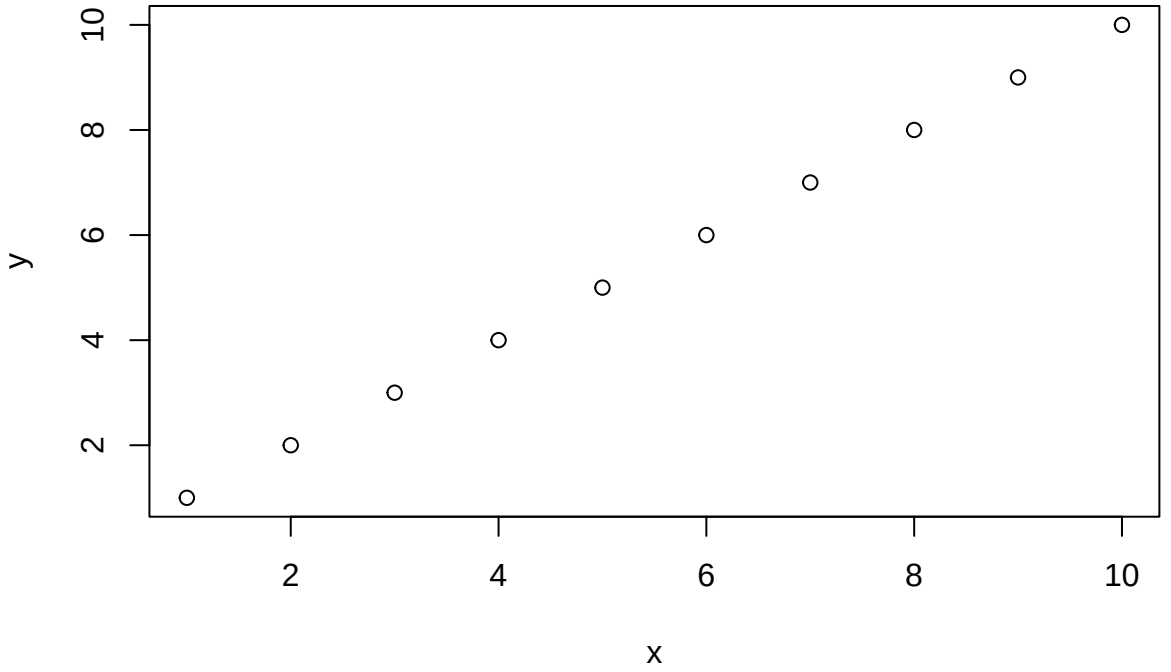
Note differences on the way up.

```
plot(0:10, x * y / 2, ylim = c(0, 60))
lines(0:10, mintegrate(x, y, 'midpoint'), col = 'orange')
lines(0:10, mintegrate(x, y, 'left'), col = 'red')
lines(0:10, mintegrate(x, y, 'right'), col = 'blue')
lines(0:10, mintegrate(x, y, 'trap'), col = 'green', lty = 2)
```



Leave out 0 (say first measurement is at time = 1).

```
x <- 1:10
y <- 1:10
plot(x, y)
```



Exact integral depends on what occurred before t = 1.

```
mintegrate(x, y, 'midpoint')
```

```
##  [1]  0.0  1.5  4.0  7.5 12.0 17.5 24.0 31.5 40.0 49.5
```

```
mintegrate(x, y, 'left')
```

```
##  [1]  0  2  5  9 14 20 27 35 44 54
```

```
mintegrate(x, y, 'right')
```

```
##  [1]  1  3  6 10 15 21 28 36 45 45
```

```
mintegrate(x, y, 'trap')
```

```
##  [1]  0.0  1.5  4.0  7.5 12.0 17.5 24.0 31.5 40.0 49.5
```

Can incorporate assumptions.

```
mintegrate(x, y, 'midpoint', start = 0)
```

```
##  [1]  0.5  2.0  4.5  8.0 12.5 18.0 24.5 32.0 40.5 50.0
```

```
mintegrate(x, y, 'left', start = 0)
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

```
mintegrate(x, y, 'right', start = 0)
```

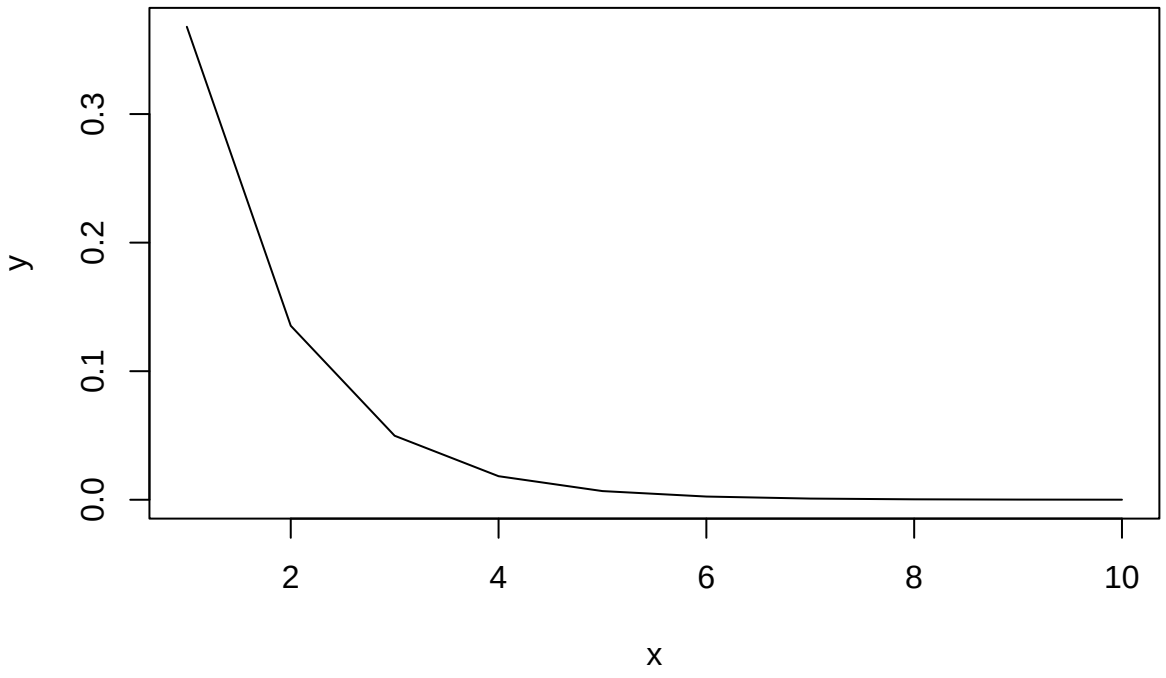
```
##  [1]  1  3  6 10 15 21 28 36 45 45
```

```
mintegrate(x, y, 'trap', start = 0, ystart = 0)
```

```
##  [1]  0.5  2.0  4.5  8.0 12.5 18.0 24.5 32.0 40.5 50.0
```

Nonlinear

```
x <- 1:10
y <- exp(-x)
plot(x, y, type = 'l')
```



Exact integral from 1:10 is `exp(-10) - exp(-1)` = 0.3678. From 0 it is 1.0.

```
mintegrate(x, y, 'midpoint', value = 'total')
```

```
## [1] 0.3979879
```

```
mintegrate(x, y, 'left', value = 'total')
```

```
## [1] 0.2140708
```

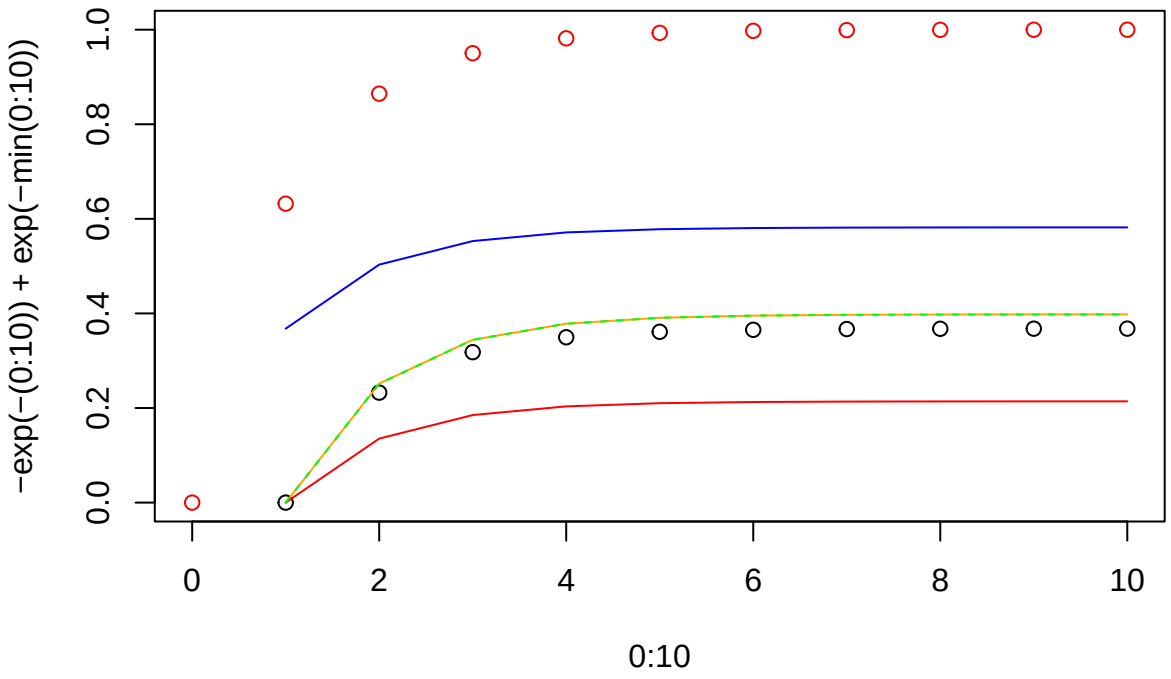
```
mintegrate(x, y, 'right', value = 'total')
```

```
## [1] 0.5819049
```

```
mintegrate(x, y, 'trap', value = 'total')
```

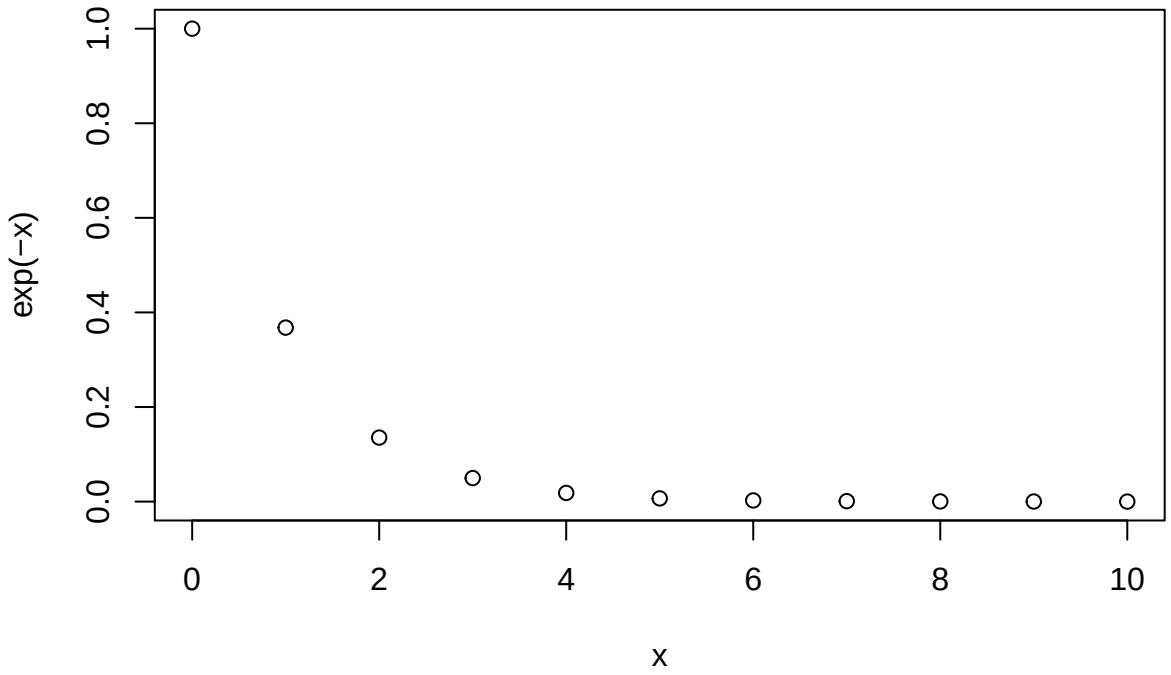
```
## [1] 0.3979879
```

```
plot(0:10, -exp(-(0:10)) + exp(-min(0:10)), col = 'red')
points(x, -exp(-x) + exp(-min(x)), ylim = c(0, 0.7))
lines(x, mintegrate(x, y, 'midpoint'), col = 'orange')
lines(x, mintegrate(x, y, 'left'), col = 'red')
lines(x, mintegrate(x, y, 'right'), col = 'blue')
lines(x, mintegrate(x, y, 'trap'), col = 'green', lty = 2)
```



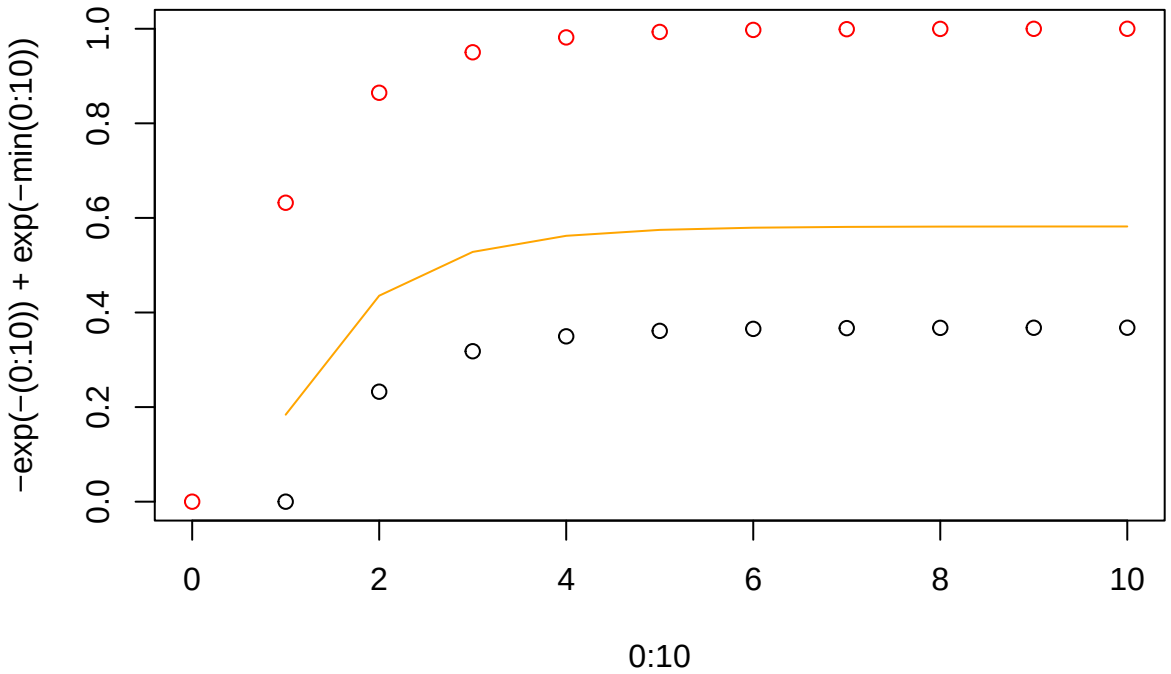
None is perfect, but midpoint and trapezoid (identical in this implementation) are the best, only slightly overestimating. Note that they all do poorly compared to a true integral that starts at 0 (red points). This cannot really be helped–how could we infer the true high values of y close to 0 from these limited measurements?

```
x <- 0:10
plot(x, exp(-x))
```



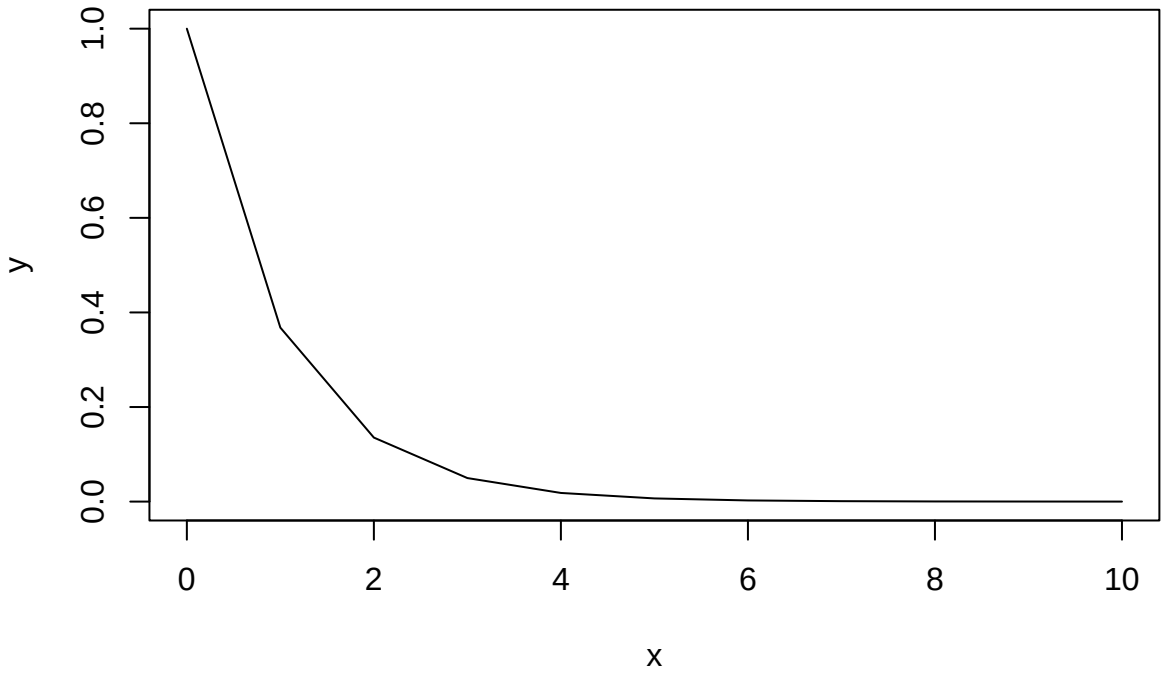
The `start` argument can extend the first rate back to 0 or any arbitrary starting point, which helps a bit.

```
x <- 1:10
plot(0:10, -exp(-(0:10)) + exp(-min(0:10)), col = 'red')
points(x, -exp(-x) + exp(-min(x)), ylim = c(0, 0.7))
lines(x, mintegrate(x, y, 'midpoint', start = 0), col = 'orange')
```



But measurements are needed at or closer to 0 to do really well with this function. Start at 0.

```
x <- 0:10
y <- exp(-x)
plot(x, y, type = 'l')
```




```
mintegrate(x, y, 'midpoint', value = 'total')
```

```
## [1] 1.081928
```

```
mintegrate(x, y, 'left', value = 'total')
```

```
## [1] 0.5819503
```

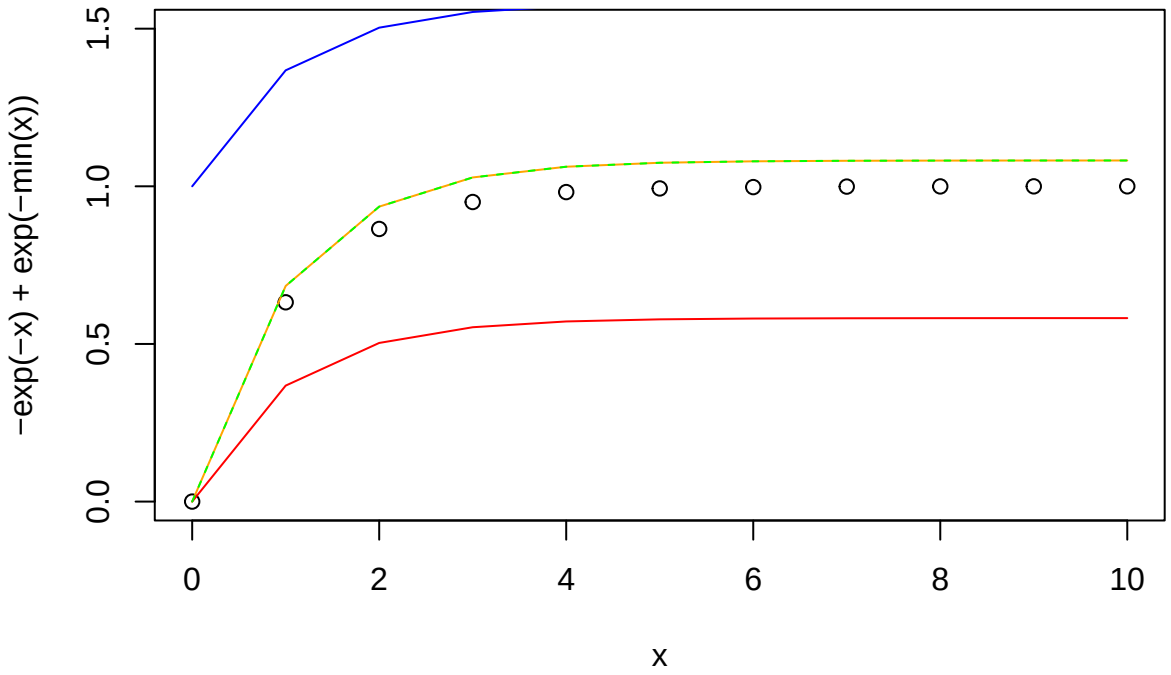
```
mintegrate(x, y, 'right', value = 'total')
```

```
## [1] 1.581905
```

```
mintegrate(x, y, 'trap', value = 'total')
```

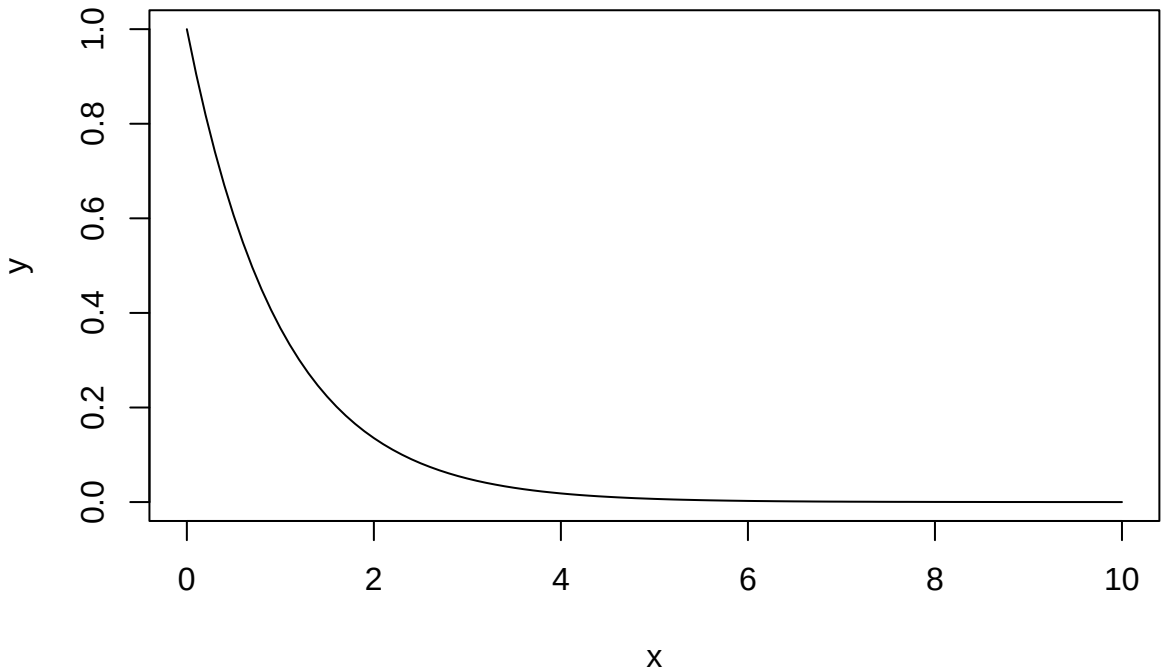
```
## [1] 1.081928
```

```
plot(x, -exp(-x) + exp(-min(x)), ylim = c(0, 1.5))
lines(x, mintegrate(x, y, 'midpoint'), col = 'orange')
lines(x, mintegrate(x, y, 'left'), col = 'red')
lines(x, mintegrate(x, y, 'right'), col = 'blue')
lines(x, mintegrate(x, y, 'trap'), col = 'green', lty = 2)
```



We can prove that all methods become accurate with very high resolution.

```
x <- 0:100 / 10
y <- exp(-x)
plot(x, y, type = 'l')
```




```
mintegrate(x, y, 'midpoint', value = 'total')
```

```
## [1] 1.000788
```

```
mintegrate(x, y, 'left', value = 'total')
```

```
## [1] 0.95079
```

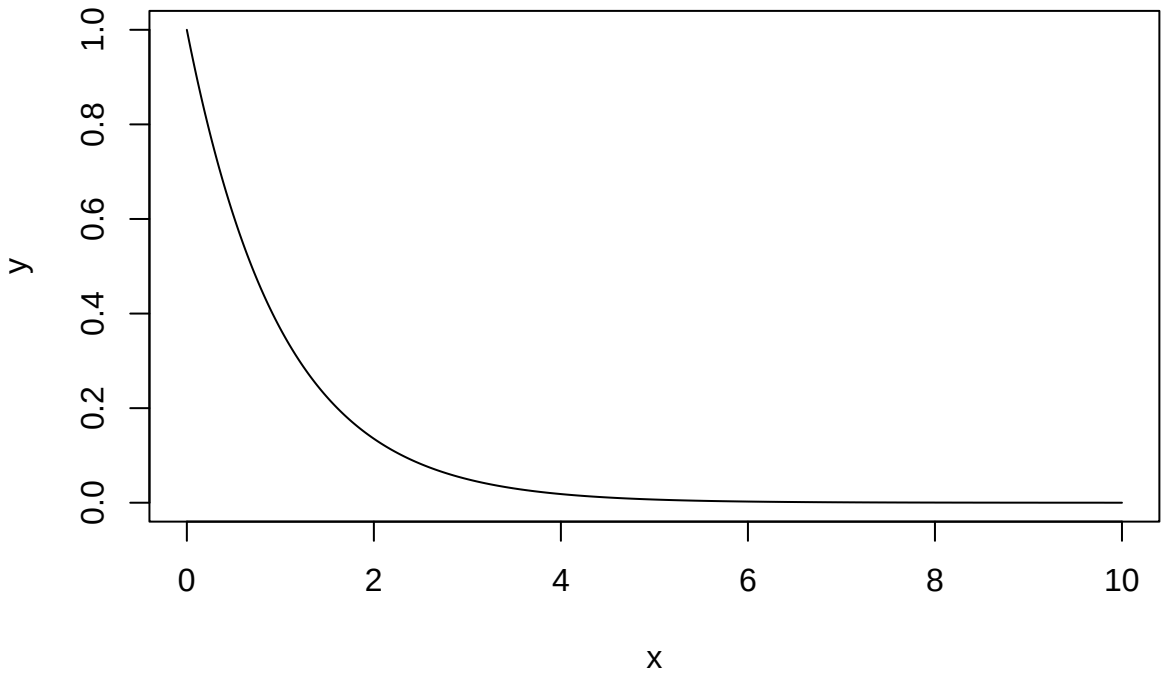
```
mintegrate(x, y, 'right', value = 'total')
```

```
## [1] 1.050785
```

```
mintegrate(x, y, 'trap', value = 'total')
```

```
## [1] 1.000788
```

```
x <- 0:10000 / 1000
y <- exp(-x)
plot(x, y, type = 'l')
```




```
mintegrate(x, y, 'midpoint', value = 'total')
```

```
## [1] 0.9999547
```

```
mintegrate(x, y, 'left', value = 'total')
```

```
## [1] 0.9994547
```

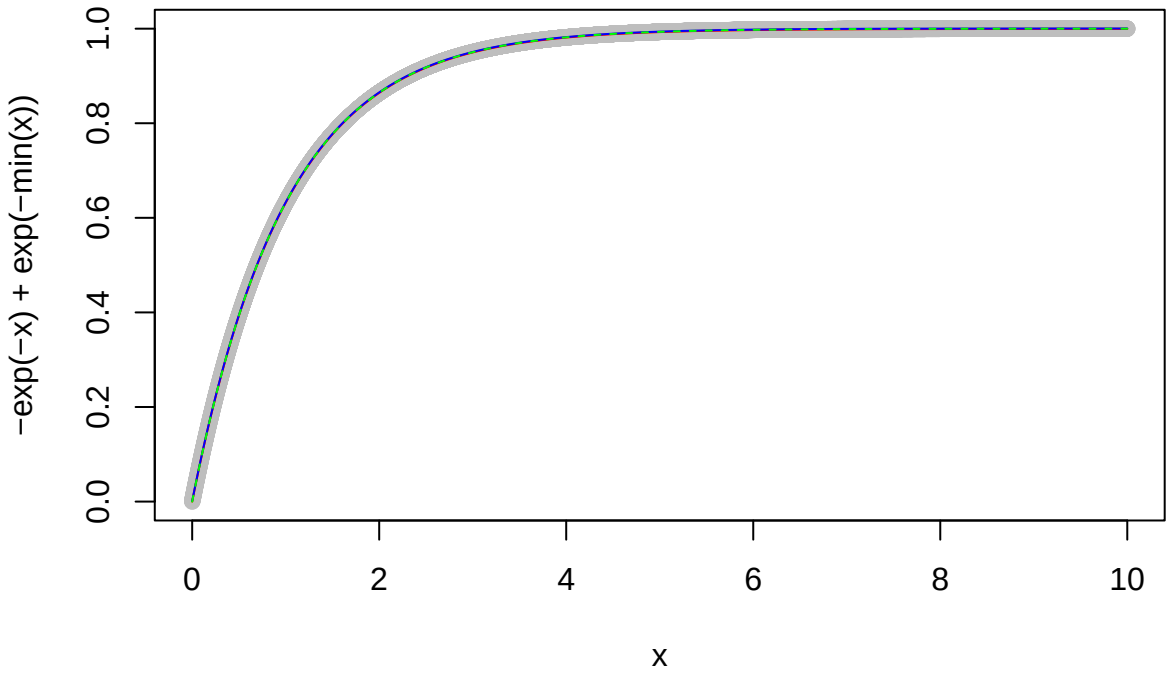
```
mintegrate(x, y, 'right', value = 'total')
```

```
## [1] 1.000455
```

```
mintegrate(x, y, 'trap', value = 'total')
```

```
## [1] 0.9999547
```

```
plot(x, -exp(-x) + exp(-min(x)), col = 'gray')
lines(x, mintegrate(x, y, 'midpoint'), col = 'orange')
lines(x, mintegrate(x, y, 'left'), col = 'red')
lines(x, mintegrate(x, y, 'right'), col = 'blue')
lines(x, mintegrate(x, y, 'trap'), col = 'green', lty = 2)
```



Note that data need not be sorted by x.

```
x <- 0:10
y <- exp(-x)
```

```
mintegrate(x, y, 'midpoint')
```

```
##  [1] 0.0000000 0.6839397 0.9355471 1.0281083 1.0621596 1.0746864 1.0792948
##  [8] 1.0809901 1.0816137 1.0818432 1.0819276
```

```
x[1] <- 4
x[5] <- 0
y <- exp(-x)
```

```
mintegrate(x, y, 'midpoint')
```

```
##  [1] 1.0621596 0.6839397 0.9355471 1.0281083 0.0000000 1.0746864 1.0792948
##  [8] 1.0809901 1.0816137 1.0818432 1.0819276
```