jumbled demonstrations

Sasha D. Hafner

16 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.

```r
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`.

```r
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`.

```r
args(interpm)
```

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

```r
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
```

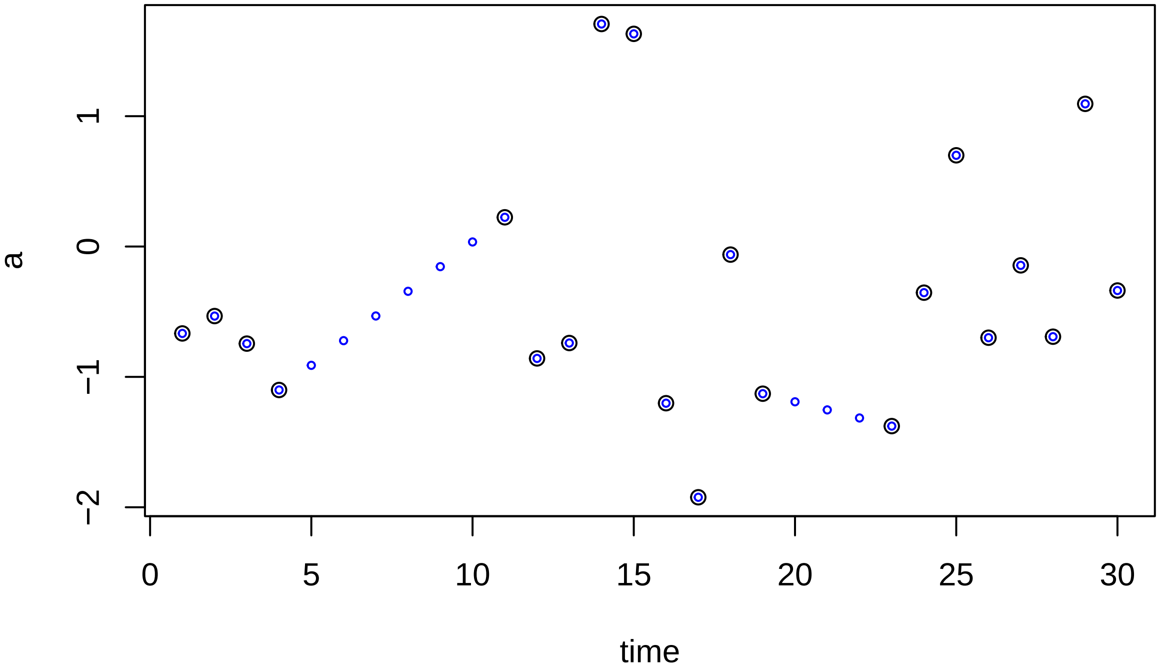
```
##    time           a          b           c
## 1     1 -0.66671630 -1.7291281  1.85090376
## 2     2 -0.53336990 -1.0466082 -1.00509813
## 3     3 -0.74463034 -1.3165467 -1.48993141
## 4     4 -1.10049129 -0.2752308 -1.99745104
## 5     5          NA         NA          NA
## 6     6          NA         NA          NA
## 7     7          NA         NA          NA
## 8     8          NA         NA          NA
## 9     9          NA         NA          NA
## 10   10          NA         NA          NA
## 11   11  0.22432514  2.7033049  0.08404299
## 12   12 -0.85829285  0.8100524  0.55702803
## 13   13 -0.74017336 -1.9740774 -0.69680774
## 14   14  1.70699411 -1.1068043  0.90220976
## 15   15  1.63153423  0.3065542  1.10807657
## 16   16 -1.20190128  0.1187166  0.94391512
## 17   17 -1.92338779 -0.2472745 -1.50702106
## 18   18 -0.06198481  0.2775730  0.33719316
## 19   19 -1.12883905  1.1516247 -1.66546789
## 20   20          NA  1.7005657 -0.40005695
## 21   21          NA -0.5520710  0.33911207
## 22   22          NA  0.5397536 -0.47573458
## 23   23 -1.37741466  0.1701469  1.10607264
## 24   24 -0.35393111 -1.0496532  0.71146710
## 25   25  0.69974446 -0.8860403  0.39024826
## 26   26 -0.69933606 -1.8931785 -0.11504205
## 27   27 -0.14438115  1.1699397  0.94992783
## 28   28 -0.69163148 -0.2050105 -0.33440964
## 29   29  1.09447986 -2.0800392 -0.61565369
## 30   30 -0.33732226  0.6823797 -1.03293116
```

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

dat2
```

```
##    time           a          b           c
## 1     1 -0.66671630 -1.7291281  1.85090376
## 2     2 -0.53336990 -1.0466082 -1.00509813
## 3     3 -0.74463034 -1.3165467 -1.48993141
## 4     4 -1.10049129 -0.2752308 -1.99745104
## 5     5 -0.91123180  0.1502743 -1.70009475
## 6     6 -0.72197231  0.5757794 -1.40273846
## 7     7 -0.53271282  1.0012845 -1.10538217
## 8     8 -0.34345333  1.4267896 -0.80802588
## 9     9 -0.15419384  1.8522947 -0.51066959
## 10   10  0.03506565  2.2777998 -0.21331330
## 11   11  0.22432514  2.7033049  0.08404299
## 12   12 -0.85829285  0.8100524  0.55702803
## 13   13 -0.74017336 -1.9740774 -0.69680774
## 14   14  1.70699411 -1.1068043  0.90220976
## 15   15  1.63153423  0.3065542  1.10807657
## 16   16 -1.20190128  0.1187166  0.94391512
## 17   17 -1.92338779 -0.2472745 -1.50702106
## 18   18 -0.06198481  0.2775730  0.33719316
## 19   19 -1.12883905  1.1516247 -1.66546789
## 20   20 -1.19098295  1.7005657 -0.40005695
## 21   21 -1.25312685 -0.5520710  0.33911207
## 22   22 -1.31527076  0.5397536 -0.47573458
## 23   23 -1.37741466  0.1701469  1.10607264
## 24   24 -0.35393111 -1.0496532  0.71146710
## 25   25  0.69974446 -0.8860403  0.39024826
## 26   26 -0.69933606 -1.8931785 -0.11504205
## 27   27 -0.14438115  1.1699397  0.94992783
## 28   28 -0.69163148 -0.2050105 -0.33440964
## 29   29  1.09447986 -2.0800392 -0.61565369
## 30   30 -0.33732226  0.6823797 -1.03293116
```

```
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  1.01813303  0.02171134 -0.14637138
## 2     2     a -0.06784548 -0.87305096 -1.21741828
## 3     3     a  0.43144428  0.83702156  1.40531918
## 4     4     a -1.20808519  0.83487796  0.25383383
## 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  1.23688987 -1.77740340  1.59620129
## 11    1     b -1.50767395  0.51931565 -1.18823939
## 12    2     b  0.46491387 -0.36865008 -0.03820847
## 13    3     b -0.84931211 -1.61552311 -0.33993709
## 14    4     b -0.66790206 -0.97549553 -0.75164275
## 15    5     b  0.54003802 -0.10848121 -0.35810822
## 16    6     b -1.14686884 -0.01716195 -0.04167782
## 17    7     b -0.24542080  0.54998430  0.10737836
## 18    8     b  0.25095241 -1.37233085 -1.75254525
## 19    9     b -1.30959573 -0.21965076  0.49000761
## 20   10     b          NA -0.73114596 -0.26752341
## 21    1     c  0.41435402  0.01330910 -1.13220027
## 22    2     c          NA  1.06562764 -0.62284144
## 23    3     c  0.93335431  0.18747459 -0.27358537
## 24    4     c -0.06847277 -2.46068837 -0.37174595
## 25    5     c  0.53122694  0.90463216 -2.24516695
## 26    6     c  0.80705415  1.02210921 -1.75463989
## 27    7     c  0.62624689  1.15494643 -0.50610749
## 28    8     c  0.14442198 -0.47901131  1.28354605
## 29    9     c  0.09351413 -0.96746411  0.48166129
## 30   10     c  0.28317088 -0.41423018 -0.57358035
```

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

```
##    time group           a           b           c
## 1     1     a  1.01813303  0.02171134 -0.14637138
## 2     2     a -0.06784548 -0.87305096 -1.21741828
## 3     3     a  0.43144428  0.83702156  1.40531918
## 4     4     a -1.20808519  0.83487796  0.25383383
## 5     5     a -0.80058934  0.39949774  0.47756174
## 6     6     a -0.39309350 -0.03588249  0.70128965
## 7     7     a  0.01440234 -0.47126272  0.92501756
## 8     8     a  0.42189819 -0.90664295  1.14874547
## 9     9     a  0.82939403 -1.34202317  1.37247338
## 10   10     a  1.23688987 -1.77740340  1.59620129
## 11    1     b -1.50767395  0.51931565 -1.18823939
## 12    2     b  0.46491387 -0.36865008 -0.03820847
## 13    3     b -0.84931211 -1.61552311 -0.33993709
## 14    4     b -0.66790206 -0.97549553 -0.75164275
## 15    5     b  0.54003802 -0.10848121 -0.35810822
## 16    6     b -1.14686884 -0.01716195 -0.04167782
## 17    7     b -0.24542080  0.54998430  0.10737836
## 18    8     b  0.25095241 -1.37233085 -1.75254525
## 19    9     b -1.30959573 -0.21965076  0.49000761
## 20   10     b          NA -0.73114596 -0.26752341
## 21    1     c  0.41435402  0.01330910 -1.13220027
## 22    2     c  0.67385416  1.06562764 -0.62284144
## 23    3     c  0.93335431  0.18747459 -0.27358537
## 24    4     c -0.06847277 -2.46068837 -0.37174595
## 25    5     c  0.53122694  0.90463216 -2.24516695
## 26    6     c  0.80705415  1.02210921 -1.75463989
## 27    7     c  0.62624689  1.15494643 -0.50610749
## 28    8     c  0.14442198 -0.47901131  1.28354605
## 29    9     c  0.09351413 -0.96746411  0.48166129
## 30   10     c  0.28317088 -0.41423018 -0.57358035
```

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

```
##    time group           a           b           c
## 1     1     a  1.01813303  0.02171134 -0.14637138
## 2     2     a -0.06784548 -0.87305096 -1.21741828
## 3     3     a  0.43144428  0.83702156  1.40531918
## 4     4     a -1.20808519  0.83487796  0.25383383
## 5     5     a -0.80058934  0.39949774  0.47756174
## 6     6     a -0.39309350 -0.03588249  0.70128965
## 7     7     a  0.01440234 -0.47126272  0.92501756
## 8     8     a  0.42189819 -0.90664295  1.14874547
## 9     9     a  0.82939403 -1.34202317  1.37247338
## 10   10     a  1.23688987 -1.77740340  1.59620129
## 11    1     b -1.50767395  0.51931565 -1.18823939
## 12    2     b  0.46491387 -0.36865008 -0.03820847
## 13    3     b -0.84931211 -1.61552311 -0.33993709
## 14    4     b -0.66790206 -0.97549553 -0.75164275
## 15    5     b  0.54003802 -0.10848121 -0.35810822
## 16    6     b -1.14686884 -0.01716195 -0.04167782
## 17    7     b -0.24542080  0.54998430  0.10737836
## 18    8     b  0.25095241 -1.37233085 -1.75254525
## 19    9     b -1.30959573 -0.21965076  0.49000761
## 20   10     b -1.30959573 -0.73114596 -0.26752341
## 21    1     c  0.41435402  0.01330910 -1.13220027
## 22    2     c  0.67385416  1.06562764 -0.62284144
## 23    3     c  0.93335431  0.18747459 -0.27358537
## 24    4     c -0.06847277 -2.46068837 -0.37174595
## 25    5     c  0.53122694  0.90463216 -2.24516695
## 26    6     c  0.80705415  1.02210921 -1.75463989
## 27    7     c  0.62624689  1.15494643 -0.50610749
## 28    8     c  0.14442198 -0.47901131  1.28354605
## 29    9     c  0.09351413 -0.96746411  0.48166129
## 30   10     c  0.28317088 -0.41423018 -0.57358035
```

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

```
##     time group           a           b           c
##  1:    1     a  1.01813303  0.02171134 -0.14637138
##  2:    2     a -0.06784548 -0.87305096 -1.21741828
##  3:    3     a  0.43144428  0.83702156  1.40531918
##  4:    4     a -1.20808519  0.83487796  0.25383383
##  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  1.23688987 -1.77740340  1.59620129
## 11:    1     b -1.50767395  0.51931565 -1.18823939
## 12:    2     b  0.46491387 -0.36865008 -0.03820847
## 13:    3     b -0.84931211 -1.61552311 -0.33993709
## 14:    4     b -0.66790206 -0.97549553 -0.75164275
## 15:    5     b  0.54003802 -0.10848121 -0.35810822
## 16:    6     b -1.14686884 -0.01716195 -0.04167782
## 17:    7     b -0.24542080  0.54998430  0.10737836
## 18:    8     b  0.25095241 -1.37233085 -1.75254525
## 19:    9     b -1.30959573 -0.21965076  0.49000761
## 20:   10     b          NA -0.73114596 -0.26752341
## 21:    1     c  0.41435402  0.01330910 -1.13220027
## 22:    2     c          NA  1.06562764 -0.62284144
## 23:    3     c  0.93335431  0.18747459 -0.27358537
## 24:    4     c -0.06847277 -2.46068837 -0.37174595
## 25:    5     c  0.53122694  0.90463216 -2.24516695
## 26:    6     c  0.80705415  1.02210921 -1.75463989
## 27:    7     c  0.62624689  1.15494643 -0.50610749
## 28:    8     c  0.14442198 -0.47901131  1.28354605
## 29:    9     c  0.09351413 -0.96746411  0.48166129
## 30:   10     c  0.28317088 -0.41423018 -0.57358035
##     time group           a           b           c
```

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

```
##     time group           a           b           c
##  1:    1     a  1.01813303  0.02171134 -0.14637138
##  2:    2     a -0.06784548 -0.87305096 -1.21741828
##  3:    3     a  0.43144428  0.83702156  1.40531918
##  4:    4     a -1.20808519  0.83487796  0.25383383
##  5:    5     a -0.80058934  0.39949774  0.47756174
##  6:    6     a -0.39309350 -0.03588249  0.70128965
##  7:    7     a  0.01440234 -0.47126272  0.92501756
##  8:    8     a  0.42189819 -0.90664295  1.14874547
##  9:    9     a  0.82939403 -1.34202317  1.37247338
## 10:   10     a  1.23688987 -1.77740340  1.59620129
## 11:    1     b -1.50767395  0.51931565 -1.18823939
## 12:    2     b  0.46491387 -0.36865008 -0.03820847
## 13:    3     b -0.84931211 -1.61552311 -0.33993709
## 14:    4     b -0.66790206 -0.97549553 -0.75164275
## 15:    5     b  0.54003802 -0.10848121 -0.35810822
## 16:    6     b -1.14686884 -0.01716195 -0.04167782
## 17:    7     b -0.24542080  0.54998430  0.10737836
## 18:    8     b  0.25095241 -1.37233085 -1.75254525
## 19:    9     b -1.30959573 -0.21965076  0.49000761
## 20:   10     b          NA -0.73114596 -0.26752341
## 21:    1     c  0.41435402  0.01330910 -1.13220027
## 22:    2     c  0.67385416  1.06562764 -0.62284144
## 23:    3     c  0.93335431  0.18747459 -0.27358537
## 24:    4     c -0.06847277 -2.46068837 -0.37174595
## 25:    5     c  0.53122694  0.90463216 -2.24516695
## 26:    6     c  0.80705415  1.02210921 -1.75463989
## 27:    7     c  0.62624689  1.15494643 -0.50610749
## 28:    8     c  0.14442198 -0.47901131  1.28354605
## 29:    9     c  0.09351413 -0.96746411  0.48166129
## 30:   10     c  0.28317088 -0.41423018 -0.57358035
##     time group           a           b           c
```

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

```
##     time group           a           b           c
##  1:    1     a  1.01813303  0.02171134 -0.14637138
##  2:    2     a -0.06784548 -0.87305096 -1.21741828
##  3:    3     a  0.43144428  0.83702156  1.40531918
##  4:    4     a -1.20808519  0.83487796  0.25383383
##  5:    5     a -0.80058934  0.39949774  0.47756174
##  6:    6     a -0.39309350 -0.03588249  0.70128965
##  7:    7     a  0.01440234 -0.47126272  0.92501756
##  8:    8     a  0.42189819 -0.90664295  1.14874547
##  9:    9     a  0.82939403 -1.34202317  1.37247338
## 10:   10     a  1.23688987 -1.77740340  1.59620129
## 11:    1     b -1.50767395  0.51931565 -1.18823939
## 12:    2     b  0.46491387 -0.36865008 -0.03820847
## 13:    3     b -0.84931211 -1.61552311 -0.33993709
## 14:    4     b -0.66790206 -0.97549553 -0.75164275
## 15:    5     b  0.54003802 -0.10848121 -0.35810822
## 16:    6     b -1.14686884 -0.01716195 -0.04167782
## 17:    7     b -0.24542080  0.54998430  0.10737836
## 18:    8     b  0.25095241 -1.37233085 -1.75254525
## 19:    9     b -1.30959573 -0.21965076  0.49000761
## 20:   10     b -1.30959573 -0.73114596 -0.26752341
## 21:    1     c  0.41435402  0.01330910 -1.13220027
## 22:    2     c  0.67385416  1.06562764 -0.62284144
## 23:    3     c  0.93335431  0.18747459 -0.27358537
## 24:    4     c -0.06847277 -2.46068837 -0.37174595
## 25:    5     c  0.53122694  0.90463216 -2.24516695
## 26:    6     c  0.80705415  1.02210921 -1.75463989
## 27:    7     c  0.62624689  1.15494643 -0.50610749
## 28:    8     c  0.14442198 -0.47901131  1.28354605
## 29:    9     c  0.09351413 -0.96746411  0.48166129
## 30:   10     c  0.28317088 -0.41423018 -0.57358035
##     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 -1.1058595 a
## 2   2  0.2940618 b
## 3   3 -1.3300553 c
## 4   4 -0.1231988 d
## 5   5  0.3438040 e
## 6   6 -1.1756627 f
## 7   7  0.0058997 g
## 8   8 -0.6808259 h
## 9   9 -1.9060219 i
## 10 10 -0.3251777 j
```

```
rounddf(dat)
```

```
##     a     b c
## 1   1 -1.11 a
## 2   2  0.29 b
## 3   3 -1.33 c
## 4   4 -0.12 d
## 5   5  0.34 e
## 6   6 -1.18 f
## 7   7  0.01 g
## 8   8 -0.68 h
## 9   9 -1.91 i
## 10 10 -0.33 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 -1.1059 a
## 2   2  0.2941 b
## 3   3 -1.3301 c
## 4   4 -0.1232 d
## 5   5  0.3438 e
## 6   6 -1.1757 f
## 7   7  0.0059 g
## 8   8 -0.6808 h
## 9   9 -1.9060 i
## 10 10 -0.3252 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 -1.1060 a
## 2   2  0.2941 b
## 3   3 -1.3300 c
## 4   4 -0.1232 d
## 5   5  0.3438 e
## 6   6 -1.1760 f
## 7   7  0.0059 g
## 8   8 -0.6808 h
## 9   9 -1.9060 i
## 10 10 -0.3252 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 -1.1000 a
## 2   2  0.2900 b
## 3   3 -1.3000 c
## 4   4 -0.1200 d
## 5   5  0.3400 e
## 6   6 -1.2000 f
## 7   7  0.0059 g
## 8   8 -0.6800 h
## 9   9 -1.9000 i
## 10 10 -0.3300 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
```

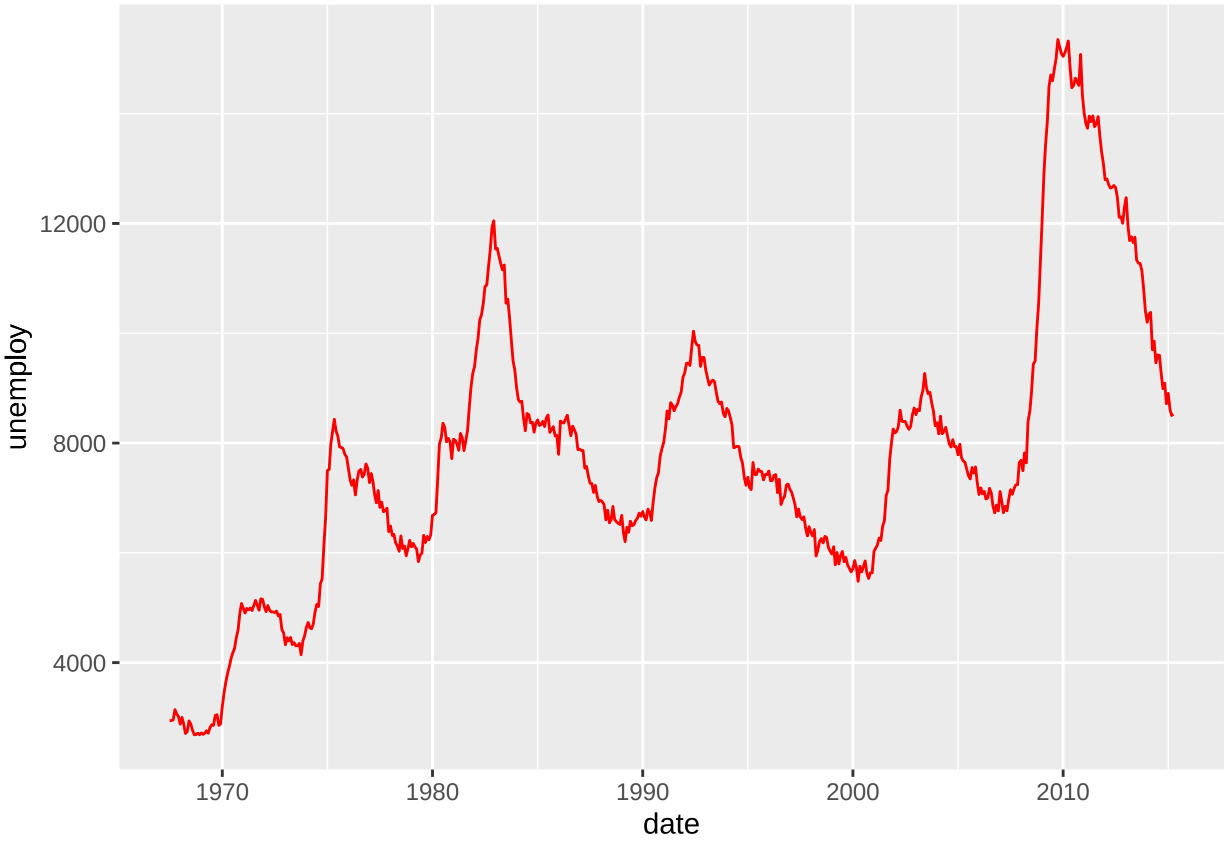
```
dat <- rounddf(dat, pad = TRUE)
summary(dat)
```

```
##        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.

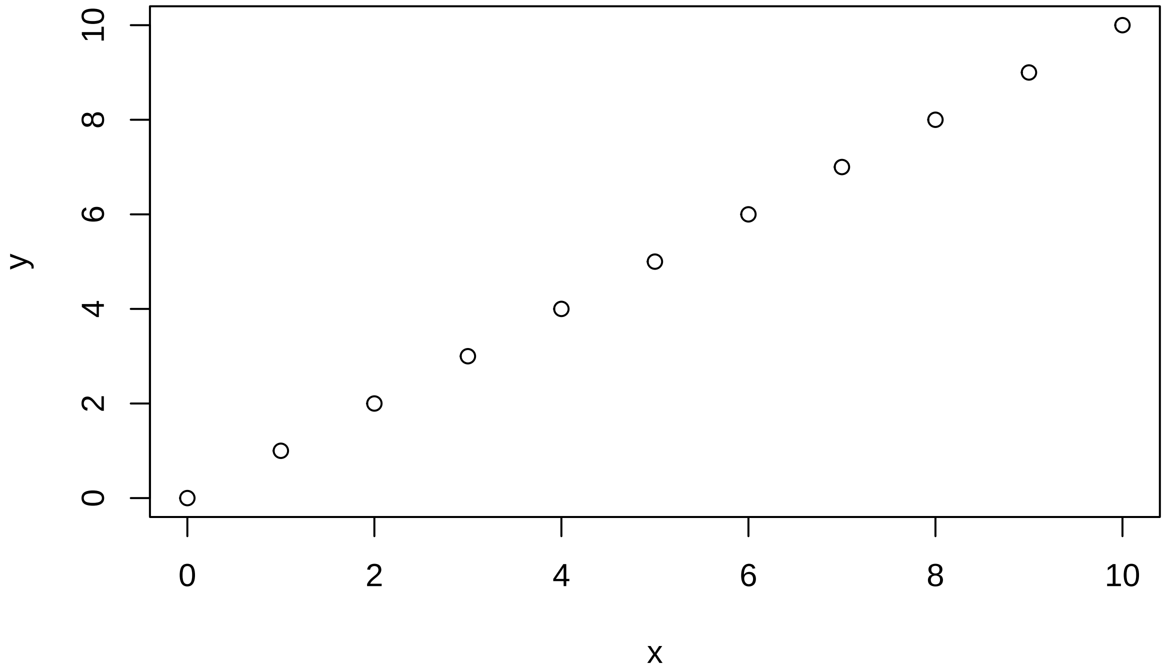
`mintegrate`

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')
```

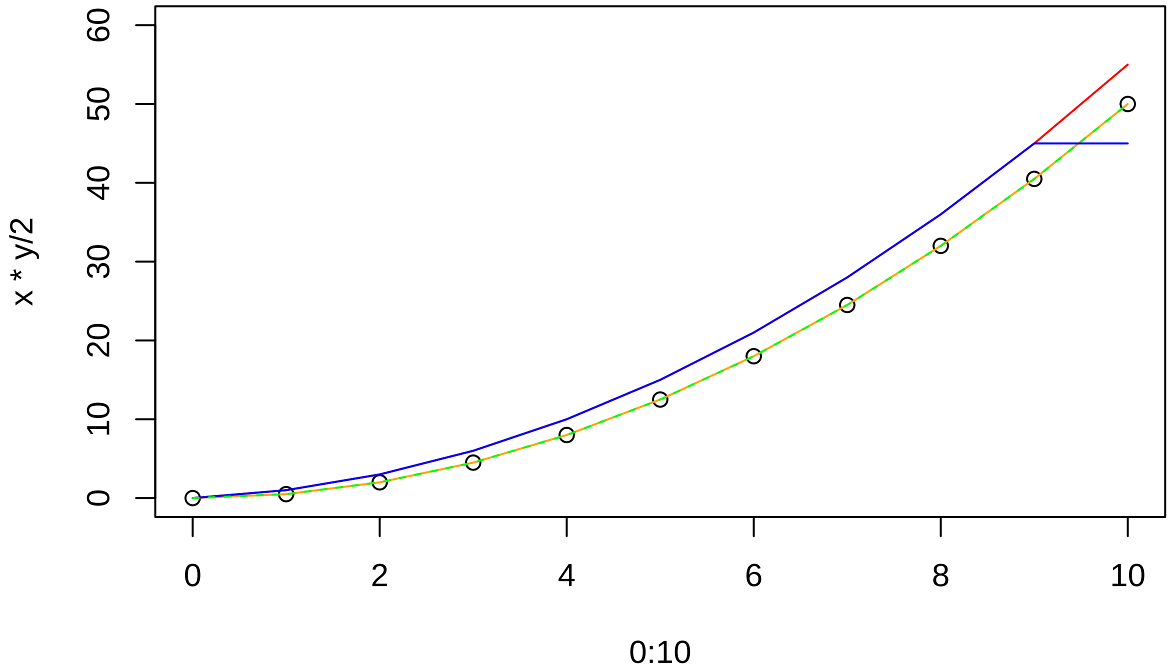
```
## [1]  0  1  3  6 10 15 21 28 36 45 45
```

```
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
```

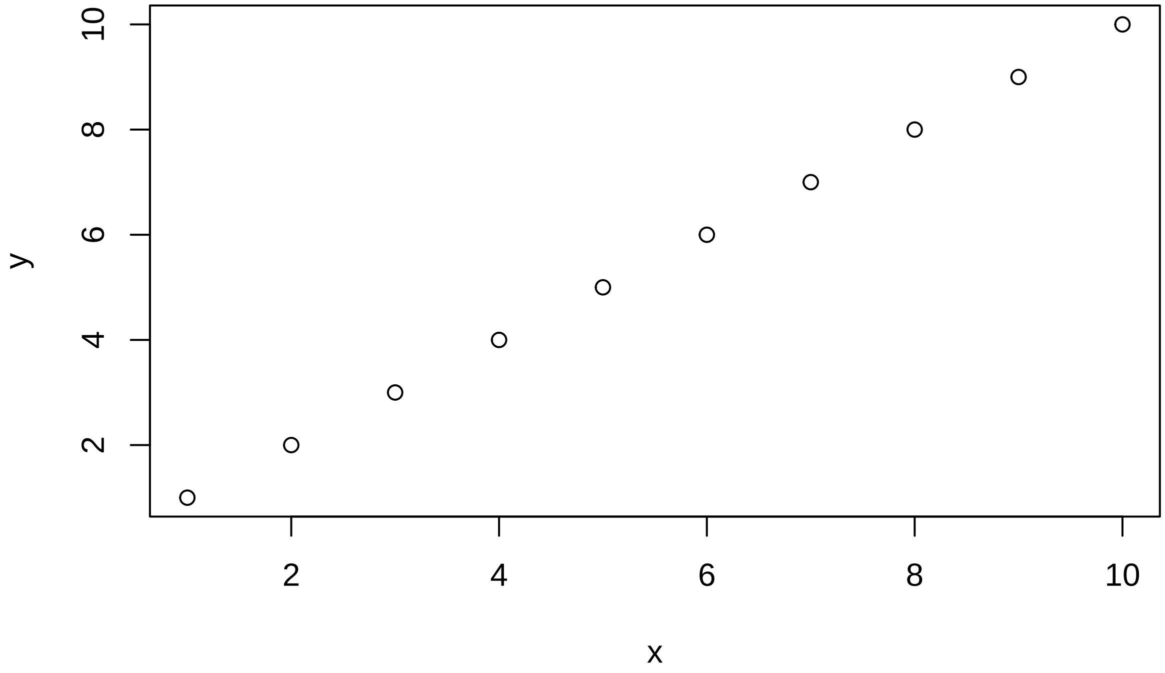
Note differences on the way up.

```
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).

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



Exact integral depends on what occurred before t = 1.

```r
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', lwr = 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', lwr = 0)
```

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

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

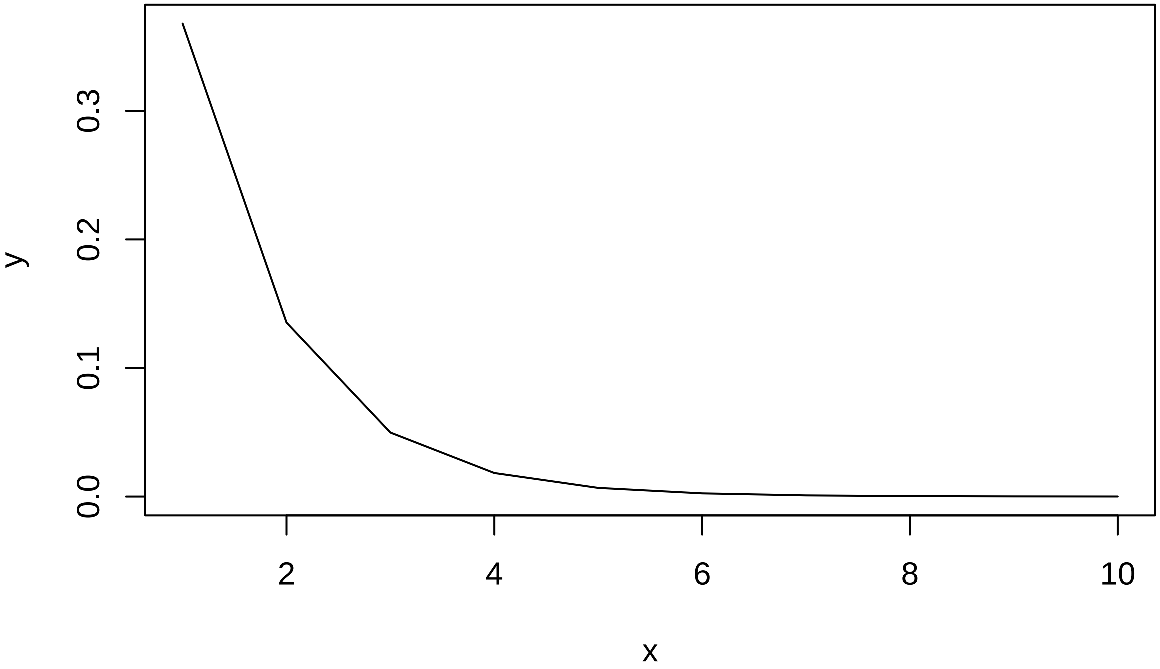
```
## [1]  1  3  6 10 15 21 28 36 45 45
```

```
mintegrate(x, y, 'trap', lwr = 0, ylwr = 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
```

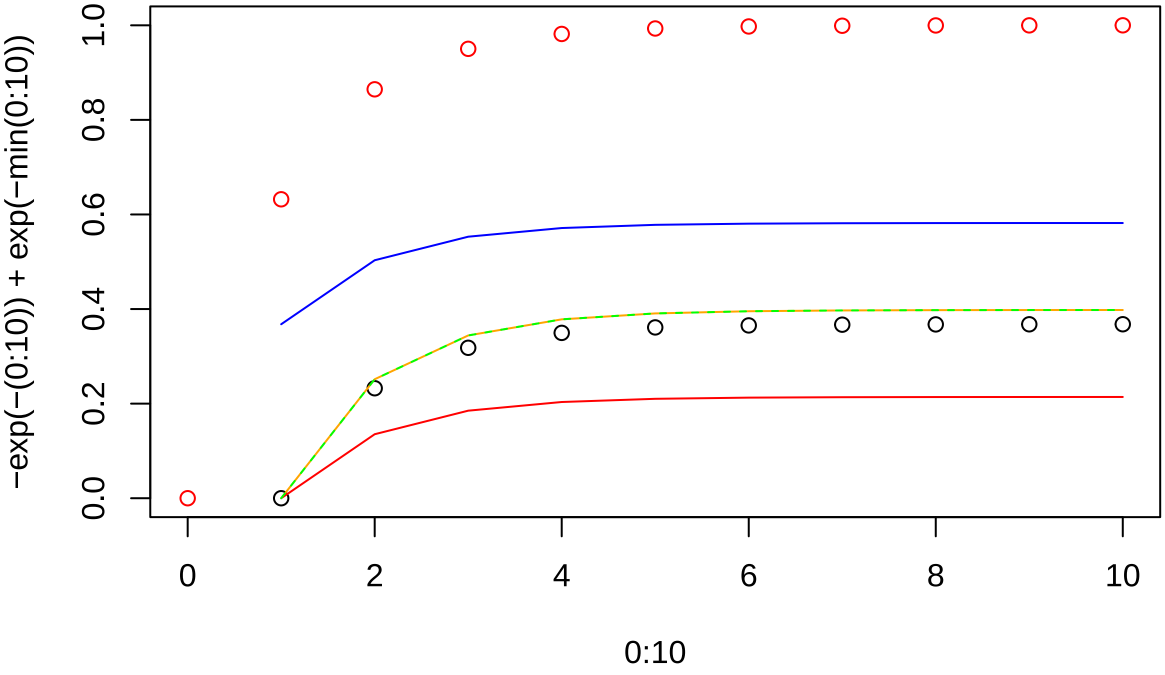
```
mintegrate(x, y, 'right', value = 'total')
```

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

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

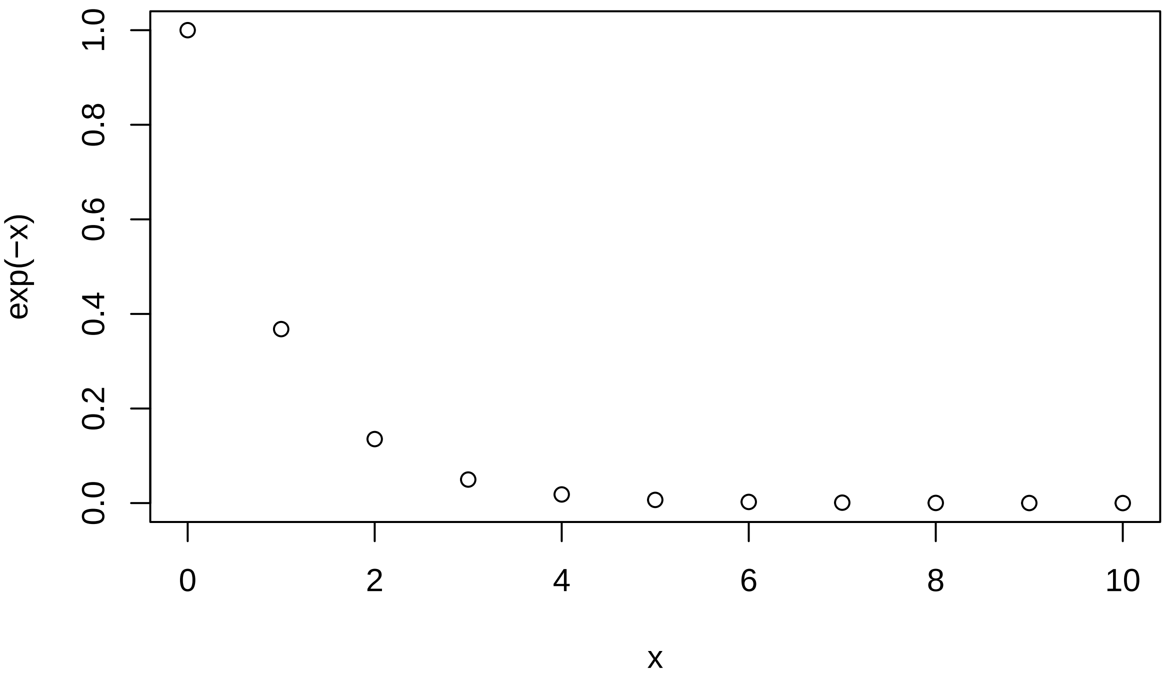
```
## [1] 0.3979879
```

```
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)
```



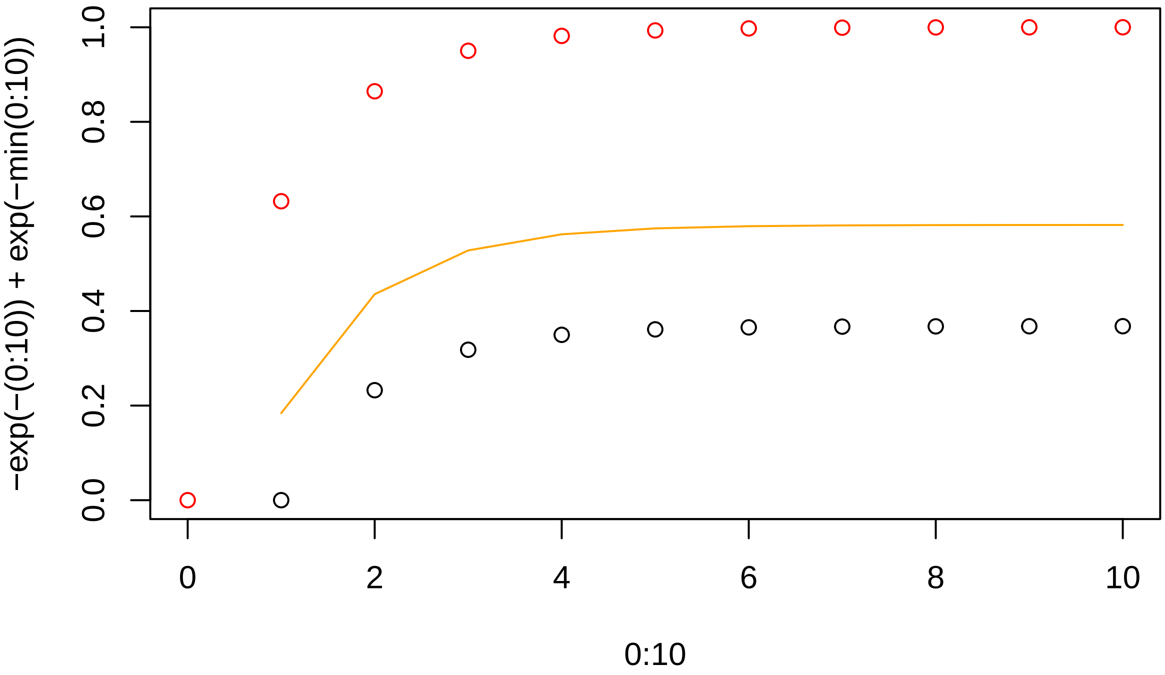
None is perfect, but midpoint and trapezoid (identical in this implementation) are the best, only slightly overestimating. Note that they all do poorly compared to a true integral that starts at 0 (red points). This cannot really be helped–how could we infer the true high values of y close to 0 from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

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



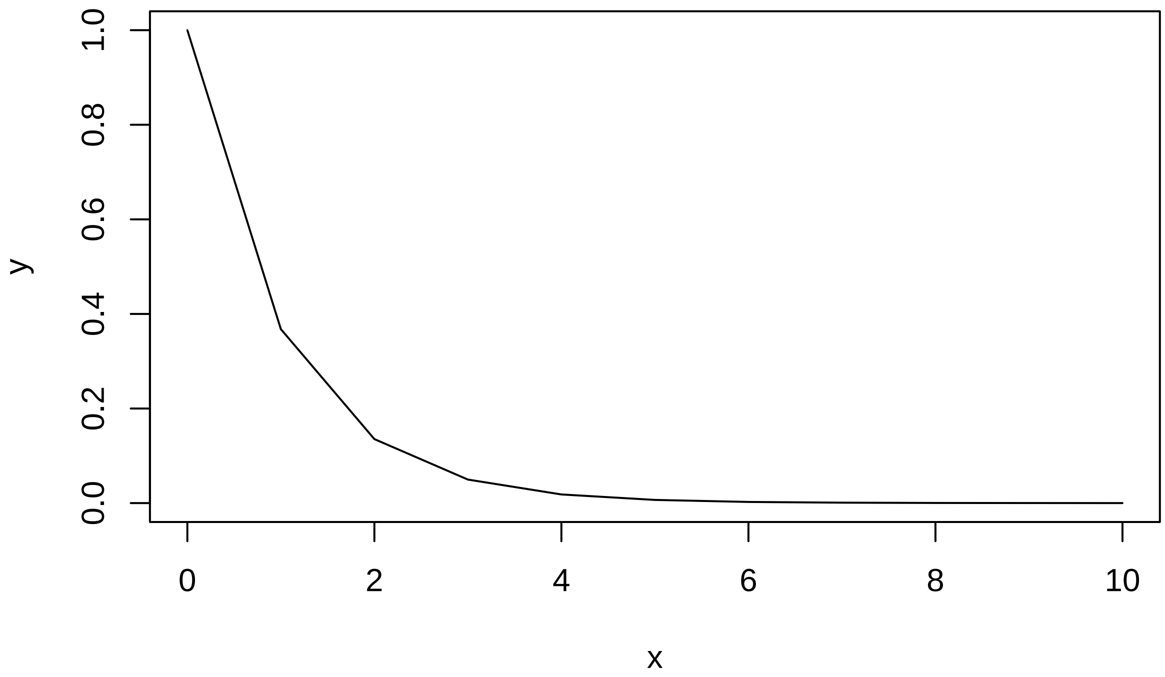
The `lwr` argument can extend the first rate back to 0 or any arbitrary starting point, which helps a bit.

```
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', lwr = 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
```

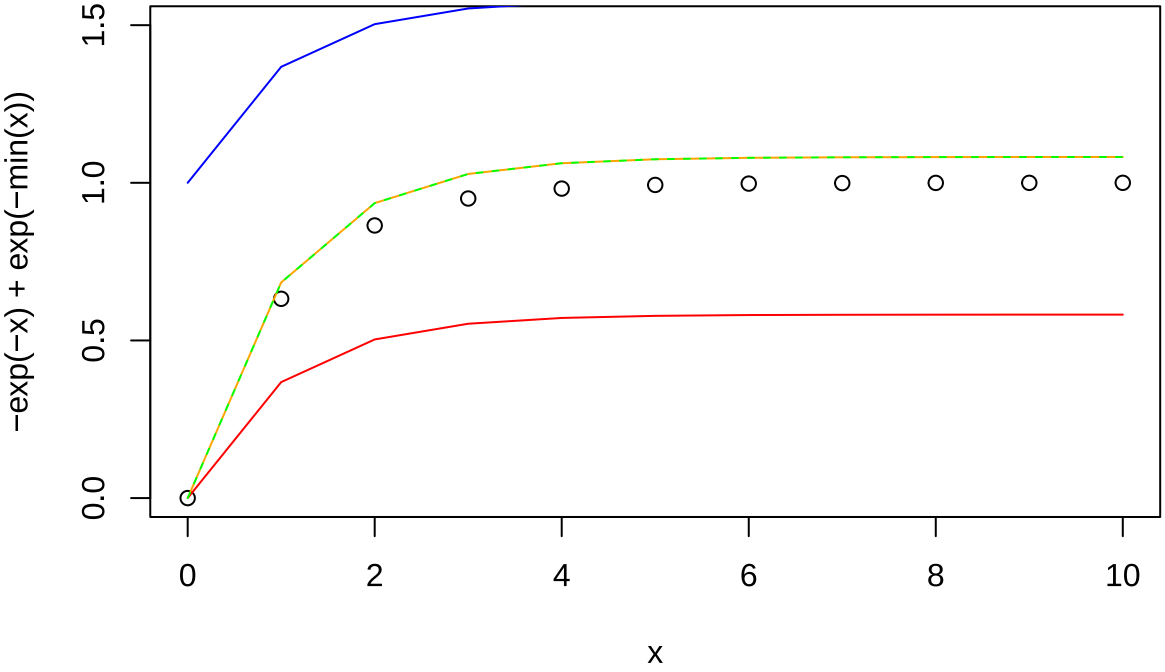
```
mintegrate(x, y, 'right', value = 'total')
```

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

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

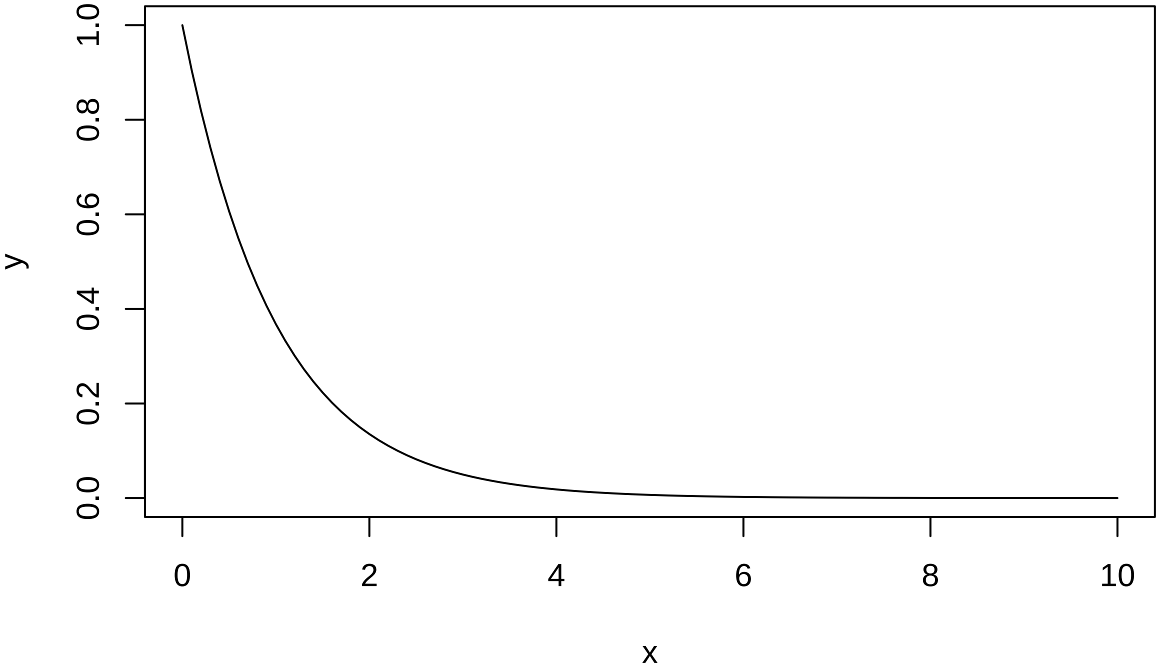
```
## [1] 1.081928
```

```
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
```

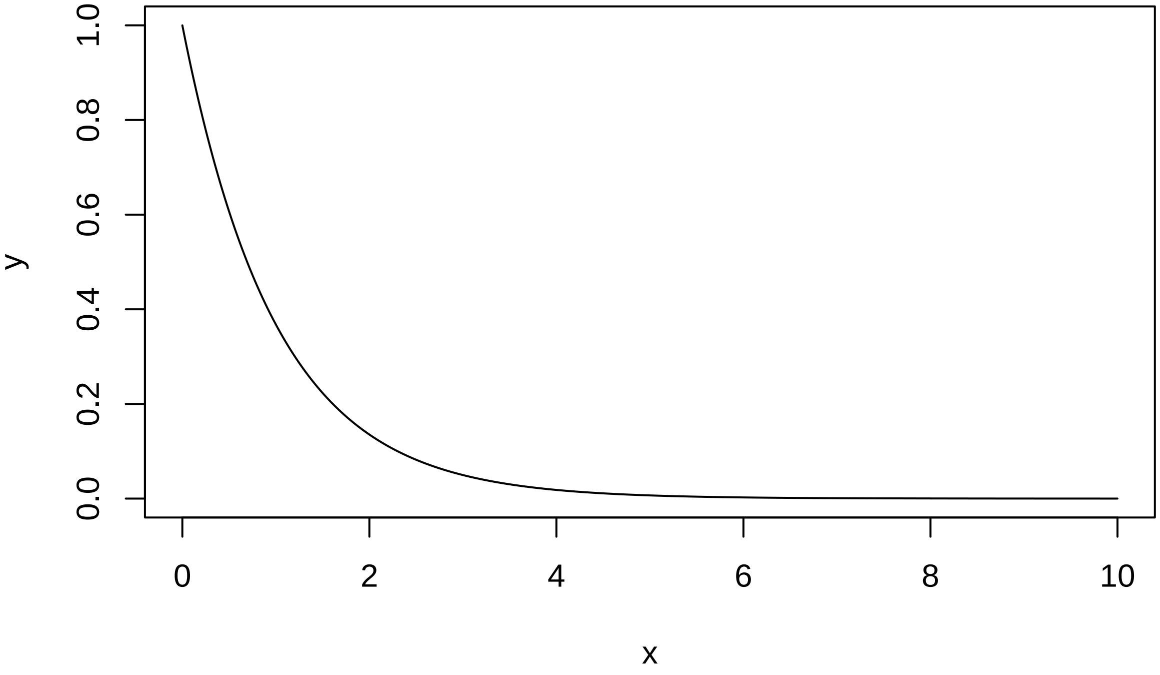
```r
mintegrate(x, y, 'right', value = 'total')
```

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

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

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

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



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

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

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

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

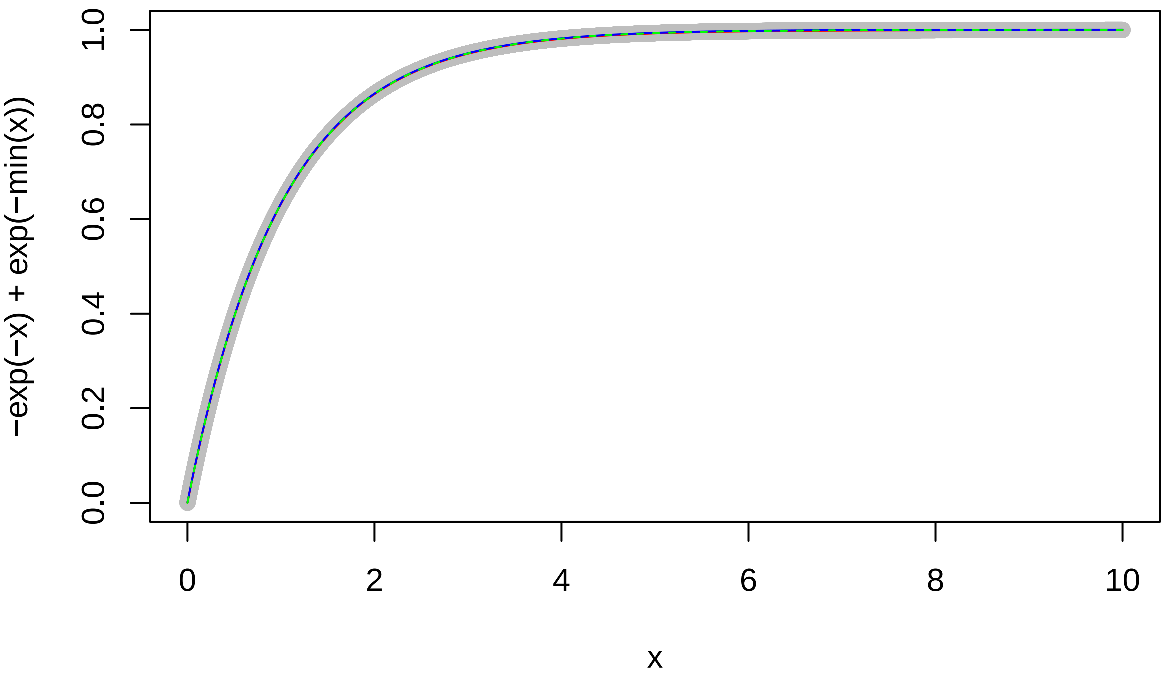
```r
mintegrate(x, y, 'right', value = 'total')
```

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

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

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

```r
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.

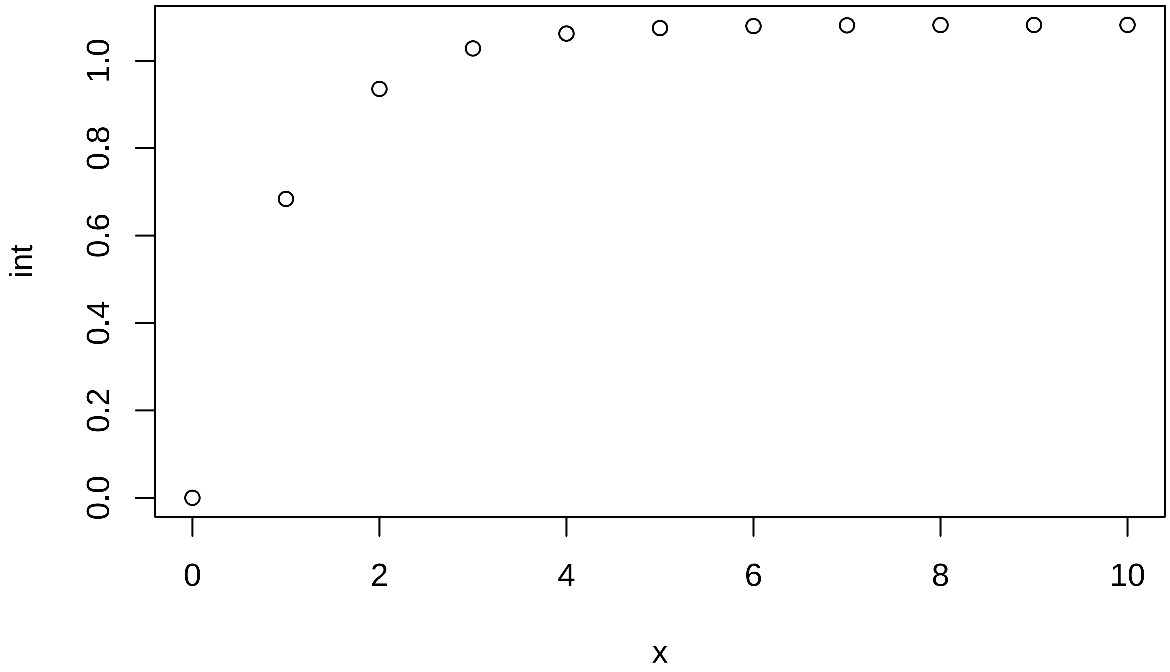
```
x <- 0:10
y <- exp(-x)
```

```
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)
```

```
int <- mintegrate(x, y, 'midpoint')
plot(x, int)
```



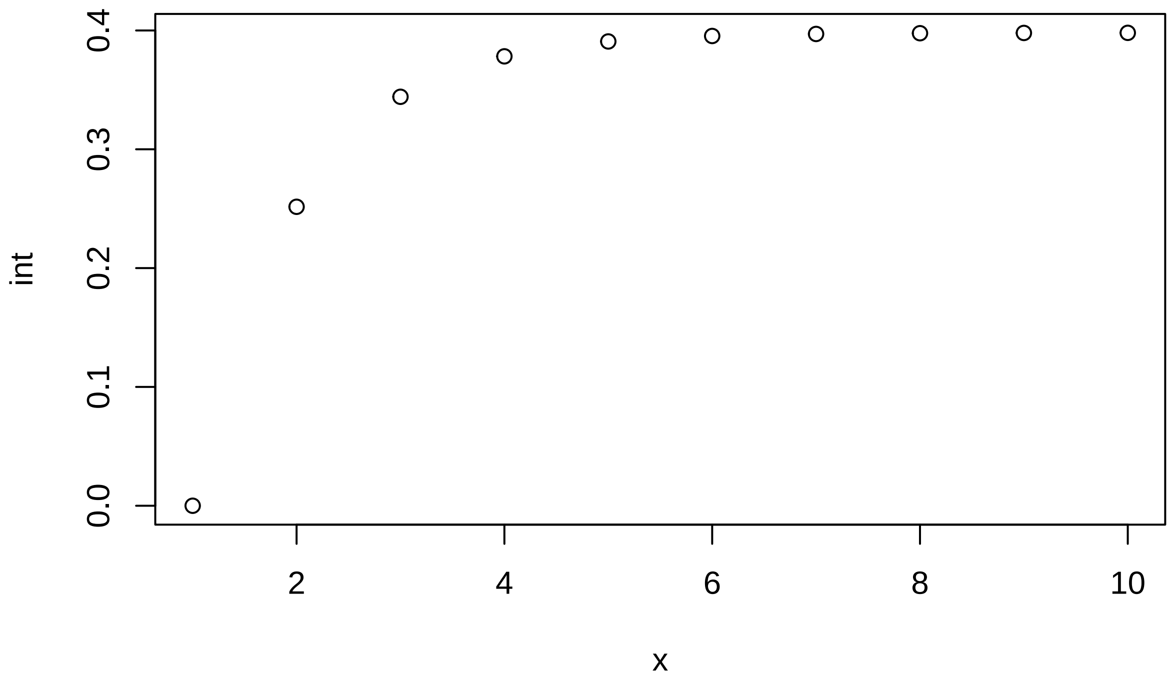
difftime

```
now <- Sys.time()
x <- difftime(now, now - 1:10)
y <- exp(-as.numeric(x))
```

```
int <- mintegrate(x, y)
```

```
## Warning in mintegrate(x, y): Converting x to numeric. Check values with value =
## "xy".
```

```
plot(x, int)
```



```
mintegrate(x, y, value = 'xy')
```

```
## Warning in mintegrate(x, y, value = "xy"): Converting x to numeric. Check values
## with value = "xy".
```

```
##       [,1]      [,2]
##  [1,]    1 0.0000000
##  [2,]    2 0.2516074
##  [3,]    3 0.3441685
##  [4,]    4 0.3782199
##  [5,]    5 0.3907467
##  [6,]    6 0.3953550
##  [7,]    7 0.3970504
##  [8,]    8 0.3976740
##  [9,]    9 0.3979035
## [10,]   10 0.3979879
```

With different units, result will differ. It is up to the user to make sure y and x have same time unit!

```
x <- difftime(now, now - 1:10, units = 'hours')
y <- exp(-as.numeric(x * 3600))
```

```
mintegrate(x, y, value = 'xy')
```

```
## Warning in mintegrate(x, y, value = "xy"): Converting x to numeric. Check values
## with value = "xy".
```

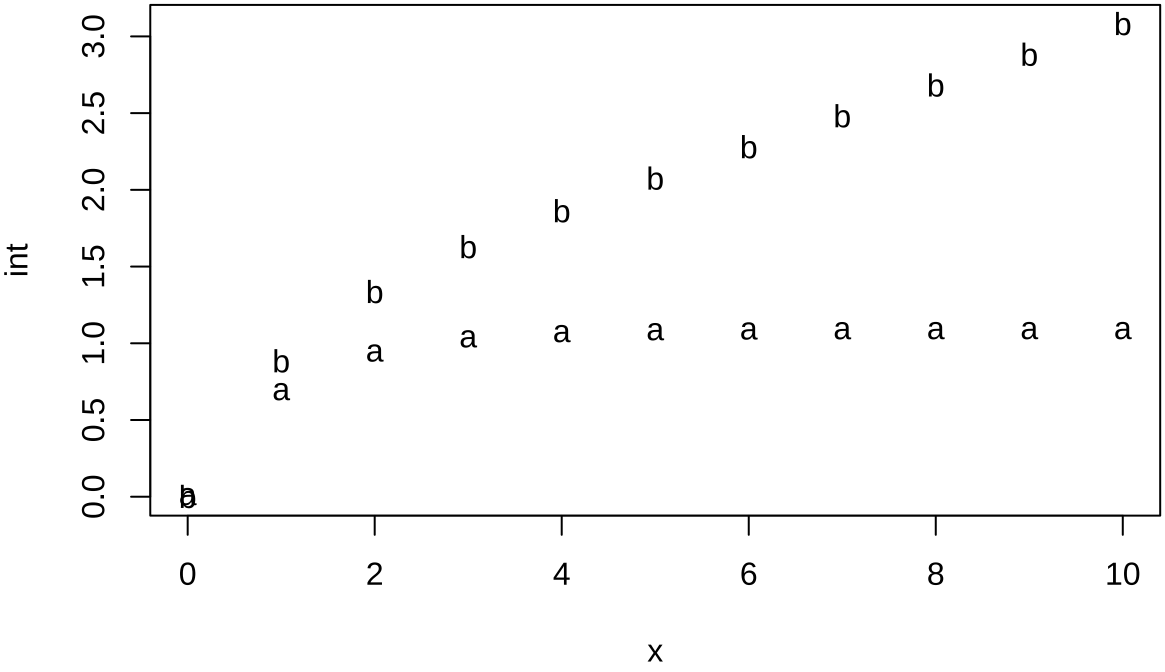
```
##               [,1]         [,2]
##  [1,] 0.0002777778 0.000000e+00
##  [2,] 0.0005555556 6.989093e-05
##  [3,] 0.0008333333 9.560237e-05
##  [4,] 0.0011111111 1.050611e-04
##  [5,] 0.0013888889 1.085407e-04
##  [6,] 0.0016666667 1.098208e-04
##  [7,] 0.0019444444 1.102918e-04
##  [8,] 0.0022222222 1.104650e-04
##  [9,] 0.0025000000 1.105287e-04
## [10,] 0.0027777778 1.105522e-04
```

Grouped

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

```
x <- c(x, x)
y <- c(y, y + 0.2)
g <- rep(c('a', 'b'), each = 11)
```

```
int <- mintegrate(x, y, by = g)
plot(x, int, pch = g)
```


int
x
a
b