jumbled demonstrations

Sasha D. Hafner

02 December, 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`).

```r
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:

```r
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`:

```r
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()`.

```r
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
##
```

Add date to check R v4.3 problem.

```
attenu$date.time <- Sys.time()
```

```
dfsumm(attenu)
```

```
##
##   182 rows and 6 columns
##   182 unique rows
##                        event     mag station    dist   accel           date.time
## Class                numeric numeric  factor numeric numeric     POSIXct, POSIXt
## Minimum                    1       5    1008     0.5   0.003 2024-12-02 10:07:19
## Maximum                   23     7.7    c266     370    0.81 2024-12-02 10:07:19
## Mean                   14.7    6.08      262    45.6   0.154 2024-12-02 10:07:19
## Unique (excld. NA)       23      17      117     153     120                    1
## Missing values            0       0       16       0       0                    0
## Sorted                 TRUE   FALSE    FALSE   FALSE   FALSE                 TRUE
##
```

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            date.time
##  Min.   :0.00300   Min.   :2024-12-02 10:07:19.73
##  1st Qu.:0.04425   1st Qu.:2024-12-02 10:07:19.73
##  Median :0.11300   Median :2024-12-02 10:07:19.73
##  Mean   :0.15422   Mean   :2024-12-02 10:07:19.73
##  3rd Qu.:0.21925   3rd Qu.:2024-12-02 10:07:19.73
##  Max.   :0.81000   Max.   :2024-12-02 10:07:19.73
##
```

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
```

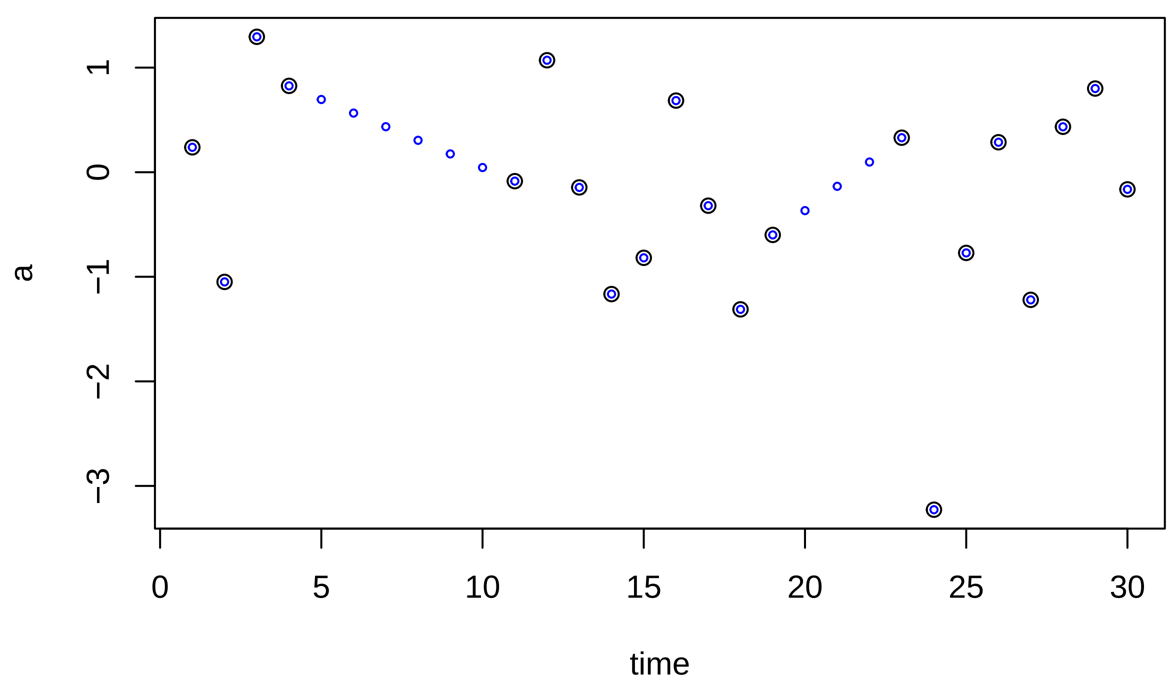
```
##    time           a          b            c
## 1     1  0.23821292  1.2429188 -1.427685784
## 2     2 -1.04889314 -0.9343851  0.619283535
## 3     3  1.29476325  0.3937087 -0.006198262
## 4     4  0.82553984  0.4036315 -0.685706846
## 5     5          NA         NA           NA
## 6     6          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 -0.08542326  0.2760235 -0.768473603
## 12    12  1.07061054 -1.0489755 -0.625910913
## 13    13 -0.14539355 -0.5208693 -0.900870855
## 14    14 -1.16554485  1.6232025  0.663728670
## 15    15 -0.81851572 -1.0700682  0.300279118
## 16    16  0.68493608  1.6858872  0.074856824
## 17    17 -0.32005642 -0.2416898  0.206372695
## 18    18 -1.31152241 -0.4682005 -0.488922835
## 19    19 -0.59960833 -0.7729782 -0.627951658
## 20    20          NA  2.1499193 -0.046916726
## 21    21          NA -1.3343536  0.162618115
## 22    22          NA  0.4958705  1.292305915
## 23    23  0.32979120  1.2339762 -0.463556502
## 24    24 -3.22732283  0.6343621  0.305463227
## 25    25 -0.77179177  0.4120223 -0.083988713
## 26    26  0.28654857  0.7935853  0.410363449
## 27    27 -1.22051198 -0.1524106  0.183678241
## 28    28  0.43455038 -0.2288958  1.778741618
## 29    29  0.80017687 -0.9007918  0.037682847
## 30    30 -0.16393097 -0.7350262  1.176220124
```

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

dat2
```

```
##    time           a          b            c
## 1     1  0.23821292  1.2429188 -1.427685784
## 2     2 -1.04889314 -0.9343851  0.619283535
## 3     3  1.29476325  0.3937087 -0.006198262
## 4     4  0.82553984  0.4036315 -0.685706846
## 5     5  0.69540226  0.3854017 -0.697530668
## 6     6  0.56526467  0.3671720 -0.709354491
## 7     7  0.43512708  0.3489423 -0.721178313
## 8     8  0.30498950  0.3307126 -0.733002136
## 9     9  0.17485191  0.3124829 -0.744825958
## 10   10  0.04471432  0.2942532 -0.756649780
## 11   11 -0.08542326  0.2760235 -0.768473603
## 12   12  1.07061054 -1.0489755 -0.625910913
## 13   13 -0.14539355 -0.5208693 -0.900870855
## 14   14 -1.16554485  1.6232025  0.663728670
## 15   15 -0.81851572 -1.0700682  0.300279118
## 16   16  0.68493608  1.6858872  0.074856824
## 17   17 -0.32005642 -0.2416898  0.206372695
## 18   18 -1.31152241 -0.4682005 -0.488922835
## 19   19 -0.59960833 -0.7729782 -0.627951658
## 20   20 -0.36725845  2.1499193 -0.046916726
## 21   21 -0.13490856 -1.3343536  0.162618115
## 22   22  0.09744132  0.4958705  1.292305915
## 23   23  0.32979120  1.2339762 -0.463556502
## 24   24 -3.22732283  0.6343621  0.305463227
## 25   25 -0.77179177  0.4120223 -0.083988713
## 26   26  0.28654857  0.7935853  0.410363449
## 27   27 -1.22051198 -0.1524106  0.183678241
## 28   28  0.43455038 -0.2288958  1.778741618
## 29   29  0.80017687 -0.9007918  0.037682847
## 30   30 -0.16393097 -0.7350262  1.176220124
```

```
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 -0.5585358 -0.392553554 -1.81353336
## 2     2     a -0.9456179  0.007051323  1.37261371
## 3     3     a -0.6651886 -2.494835444 -0.56426954
## 4     4     a  0.4520302 -0.977296362  0.97031123
## 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 -0.2657389 -0.263992919 -0.33128448
## 11    1     b  0.1516114 -0.752462697 -0.28370270
## 12    2     b  1.3766098  0.440692091  0.31415290
## 13    3     b -0.1803943 -1.277450766  1.84483867
## 14    4     b -1.5676751  1.177192046 -0.98191715
## 15    5     b -0.2607259  0.902505834  2.19600376
## 16    6     b  0.9618104 -1.261304177 -0.20466767
## 17    7     b  0.8538955  0.837455146  0.97514294
## 18    8     b  0.4187967 -2.348290308 -0.86756612
## 19    9     b  0.3399565  0.610971137 -0.50118759
## 20   10     b        NA -0.047867742  0.78559116
## 21    1     c  1.8714180 -2.399197708 -2.10224732
## 22    2     c        NA -0.019318956 -0.04220493
## 23    3     c -0.7661688 -0.088685452 -0.40480941
## 24    4     c -0.6203265 -1.595484898 -0.11276597
## 25    5     c  0.7901903  0.851709321  1.79714044
## 26    6     c -0.2419765 -0.713560806 -0.81032626
## 27    7     c  1.1174865  1.066430344  1.90090007
## 28    8     c  1.1849306 -0.536242590  0.70895444
## 29    9     c  1.6464748  0.535917056  0.73619477
## 30   10     c  0.1929962 -1.828626634  1.36577655
```

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

```
##    time group           a            b           c
## 1     1     a -0.55853581 -0.392553554 -1.81353336
## 2     2     a -0.94561794  0.007051323  1.37261371
## 3     3     a -0.66518864 -2.494835444 -0.56426954
## 4     4     a  0.45203019 -0.977296362  0.97031123
## 5     5     a  0.33240200 -0.858412455  0.75337861
## 6     6     a  0.21277381 -0.739528548  0.53644599
## 7     7     a  0.09314562 -0.620644640  0.31951337
## 8     8     a -0.02648257 -0.501760733  0.10258075
## 9     9     a -0.14611075 -0.382876826 -0.11435187
## 10   10     a -0.26573894 -0.263992919 -0.33128448
## 11    1     b  0.15161137 -0.752462697 -0.28370270
## 12    2     b  1.37660981  0.440692091  0.31415290
## 13    3     b -0.18039431 -1.277450766  1.84483867
## 14    4     b -1.56767513  1.177192046 -0.98191715
## 15    5     b -0.26072589  0.902505834  2.19600376
## 16    6     b  0.96181035 -1.261304177 -0.20466767
## 17    7     b  0.85389546  0.837455146  0.97514294
## 18    8     b  0.41879670 -2.348290308 -0.86756612
## 19    9     b  0.33995651  0.610971137 -0.50118759
## 20   10     b          NA -0.047867742  0.78559116
## 21    1     c  1.87141801 -2.399197708 -2.10224732
## 22    2     c  0.55262460 -0.019318956 -0.04220493
## 23    3     c -0.76616880 -0.088685452 -0.40480941
## 24    4     c -0.62032649 -1.595484898 -0.11276597
## 25    5      c  0.79019025  0.851709321  1.79714044
## 26    6      c -0.24197651 -0.713560806 -0.81032626
## 27    7      c  1.11748648  1.066430344  1.90090007
## 28    8      c  1.18493063 -0.536242590  0.70895444
## 29    9      c  1.64647484  0.535917056  0.73619477
## 30   10      c  0.19299619 -1.828626634  1.36577655
```

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

```
##    time group           a            b           c
## 1     1     a -0.55853581 -0.392553554 -1.81353336
## 2     2     a -0.94561794  0.007051323  1.37261371
## 3     3     a -0.66518864 -2.494835444 -0.56426954
## 4     4     a  0.45203019 -0.977296362  0.97031123
## 5     5     a  0.33240200 -0.858412455  0.75337861
## 6     6     a  0.21277381 -0.739528548  0.53644599
## 7     7     a  0.09314562 -0.620644640  0.31951337
## 8     8     a -0.02648257 -0.501760733  0.10258075
## 9     9     a -0.14611075 -0.382876826 -0.11435187
## 10   10     a -0.26573894 -0.263992919 -0.33128448
## 11    1     b  0.15161137 -0.752462697 -0.28370270
## 12    2     b  1.37660981  0.440692091  0.31415290
## 13    3     b -0.18039431 -1.277450766  1.84483867
## 14    4     b -1.56767513  1.177192046 -0.98191715
## 15    5     b -0.26072589  0.902505834  2.19600376
## 16    6     b  0.96181035 -1.261304177 -0.20466767
## 17    7     b  0.85389546  0.837455146  0.97514294
## 18    8     b  0.41879670 -2.348290308 -0.86756612
## 19    9     b  0.33995651  0.610971137 -0.50118759
## 20   10     b  0.33995651 -0.047867742  0.78559116
## 21    1     c  1.87141801 -2.399197708 -2.10224732
## 22    2     c  0.55262460 -0.019318956 -0.04220493
## 23    3     c -0.76616880 -0.088685452 -0.40480941
## 24    4     c -0.62032649 -1.595484898 -0.11276597
## 25    5     c  0.79019025  0.851709321  1.79714044
## 26    6     c -0.24197651 -0.713560806 -0.81032626
## 27    7     c  1.11748648  1.066430344  1.90090007
## 28    8     c  1.18493063 -0.536242590  0.70895444
## 29    9     c  1.64647484  0.535917056  0.73619477
## 30   10     c  0.19299619 -1.828626634  1.36577655
```

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

```
##      time  group          a            b           c
##     <int> <char>      <num>        <num>       <num>
##   1:     1      a -0.5585358 -0.392553554 -1.81353336
##   2:     2      a -0.9456179  0.007051323  1.37261371
##   3:     3      a -0.6651886 -2.494835444 -0.56426954
##   4:     4      a  0.4520302 -0.977296362  0.97031123
##   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 -0.2657389 -0.263992919 -0.33128448
## 11:     1      b  0.1516114 -0.752462697 -0.28370270
## 12:     2      b  1.3766098  0.440692091  0.31415290
## 13:     3      b -0.1803943 -1.277450766  1.84483867
## 14:     4      b -1.5676751  1.177192046 -0.98191715
## 15:     5      b -0.2607259  0.902505834  2.19600376
## 16:     6      b  0.9618104 -1.261304177 -0.20466767
## 17:     7      b  0.8538955  0.837455146  0.97514294
## 18:     8      b  0.4187967 -2.348290308 -0.86756612
## 19:     9      b  0.3399565  0.610971137 -0.50118759
## 20:    10      b         NA -0.047867742  0.78559116
## 21:     1      c  1.8714180 -2.399197708 -2.10224732
## 22:     2      c         NA -0.019318956 -0.04220493
## 23:     3      c -0.7661688 -0.088685452 -0.40480941
## 24:     4      c -0.6203265 -1.595484898 -0.11276597
## 25:     5      c  0.7901903  0.851709321  1.79714044
## 26:     6      c -0.2419765 -0.713560806 -0.81032626
## 27:     7      c  1.1174865  1.066430344  1.90090007
## 28:     8      c  1.1849306 -0.536242590  0.70895444
## 29:     9      c  1.6464748  0.535917056  0.73619477
## 30:    10      c  0.1929962 -1.828626634  1.36577655
##      time  group          a            b           c
```

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

```
##      time  group           a            b           c
##     <int> <char>       <num>        <num>       <num>
##  1:     1      a -0.55853581 -0.392553554 -1.81353336
##  2:     2      a -0.94561794  0.007051323  1.37261371
##  3:     3      a -0.66518864 -2.494835444 -0.56426954
##  4:     4      a  0.45203019 -0.977296362  0.97031123
##  5:     5      a  0.33240200 -0.858412455  0.75337861
##  6:     6      a  0.21277381 -0.739528548  0.53644599
##  7:     7      a  0.09314562 -0.620644640  0.31951337
##  8:     8      a -0.02648257 -0.501760733  0.10258075
##  9:     9      a -0.14611075 -0.382876826 -0.11435187
## 10:    10      a -0.26573894 -0.263992919 -0.33128448
## 11:     1      b  0.15161137 -0.752462697 -0.28370270
## 12:     2      b  1.37660981  0.440692091  0.31415290
## 13:     3      b -0.18039431 -1.277450766  1.84483867
## 14:     4      b -1.56767513  1.177192046 -0.98191715
## 15:     5      b -0.26072589  0.902505834  2.19600376
## 16:     6      b  0.96181035 -1.261304177 -0.20466767
## 17:     7      b  0.85389546  0.837455146  0.97514294
## 18:     8      b  0.41879670 -2.348290308 -0.86756612
## 19:     9      b  0.33995651  0.610971137 -0.50118759
## 20:    10      b          NA -0.047867742  0.78559116
## 21:     1      c  1.87141801 -2.399197708 -2.10224732
## 22:     2      c  0.55262460 -0.019318956 -0.04220493
## 23:     3      c -0.76616880 -0.088685452 -0.40480941
## 24:     4      c -0.62032649 -1.595484898 -0.11276597
## 25:     5      c  0.79019025  0.851709321  1.79714044
## 26:     6      c -0.24197651 -0.713560806 -0.81032626
## 27:     7      c  1.11748648  1.066430344  1.90090007
## 28:     8      c  1.18493063 -0.536242590  0.70895444
## 29:     9      c  1.64647484  0.535917056  0.73619477
## 30:    10      c  0.19299619 -1.828626634  1.36577655
##      time  group           a            b           c
```

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

```
##      time  group           a            b           c
##     <int> <char>       <num>        <num>       <num>
##  1:     1      a -0.55853581 -0.392553554 -1.81353336
##  2:     2      a -0.94561794  0.007051323  1.37261371
##  3:     3      a -0.66518864 -2.494835444 -0.56426954
##  4:     4      a  0.45203019 -0.977296362  0.97031123
##  5:     5      a  0.33240200 -0.858412455  0.75337861
##  6:     6      a  0.21277381 -0.739528548  0.53644599
##  7:     7      a  0.09314562 -0.620644640  0.31951337
##  8:     8      a -0.02648257 -0.501760733  0.10258075
##  9:     9      a -0.14611075 -0.382876826 -0.11435187
## 10:    10      a -0.26573894 -0.263992919 -0.33128448
## 11:     1      b  0.15161137 -0.752462697 -0.28370270
## 12:     2      b  1.37660981  0.440692091  0.31415290
## 13:     3      b -0.18039431 -1.277450766  1.84483867
## 14:     4      b -1.56767513  1.177192046 -0.98191715
## 15:     5      b -0.26072589  0.902505834  2.19600376
## 16:     6      b  0.96181035 -1.261304177 -0.20466767
## 17:     7      b  0.85389546  0.837455146  0.97514294
## 18:     8      b  0.41879670 -2.348290308 -0.86756612
## 19:     9      b  0.33995651  0.610971137 -0.50118759
## 20:    10      b  0.33995651 -0.047867742  0.78559116
## 21:     1      c  1.87141801 -2.399197708 -2.10224732
## 22:     2      c  0.55262460 -0.019318956 -0.04220493
## 23:     3      c -0.76616880 -0.088685452 -0.40480941
## 24:     4      c -0.62032649 -1.595484898 -0.11276597
## 25:     5      c  0.79019025  0.851709321  1.79714044
## 26:     6      c -0.24197651 -0.713560806 -0.81032626
## 27:     7      c  1.11748648  1.066430344  1.90090007
## 28:     8      c  1.18493063 -0.536242590  0.70895444
## 29:     9      c  1.64647484  0.535917056  0.73619477
## 30:    10      c  0.19299619 -1.828626634  1.36577655
##      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 -0.5762639 a
## 2   2 -0.8047323 b
## 3   3 -0.5350658 c
## 4   4  0.7915479 d
## 5   5 -0.7076072 e
## 6   6 -1.2755434 f
## 7   7  2.3754645 g
## 8   8 -1.0935897 h
## 9   9  0.1924416 i
## 10 10 -0.1261604 j
```

```
rounddf(dat)
```

```
##     a     b c
## 1   1 -0.58 a
## 2   2 -0.80 b
## 3   3 -0.54 c
## 4   4  0.79 d
## 5   5 -0.71 e
## 6   6 -1.28 f
## 7   7  2.38 g
## 8   8 -1.09 h
## 9   9  0.19 i
## 10 10 -0.13 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 -0.5763 a
## 2   2 -0.8047 b
## 3   3 -0.5351 c
## 4   4  0.7915 d
## 5   5 -0.7076 e
## 6   6 -1.2755 f
## 7   7  2.3755 g
## 8   8 -1.0936 h
## 9   9  0.1924 i
## 10 10 -0.1262 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 -0.5763 a
## 2   2 -0.8047 b
## 3   3 -0.5351 c
## 4   4  0.7915 d
## 5   5 -0.7076 e
## 6   6 -1.2760 f
## 7   7  2.3750 g
## 8   8 -1.0940 h
## 9   9  0.1924 i
## 10 10 -0.1262 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 -0.58 a
## 2   2 -0.80 b
## 3   3 -0.54 c
## 4   4  0.79 d
## 5   5 -0.71 e
## 6   6 -1.30 f
## 7   7  2.40 g
## 8   8 -1.10 h
## 9   9  0.19 i
## 10 10 -0.13 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
```

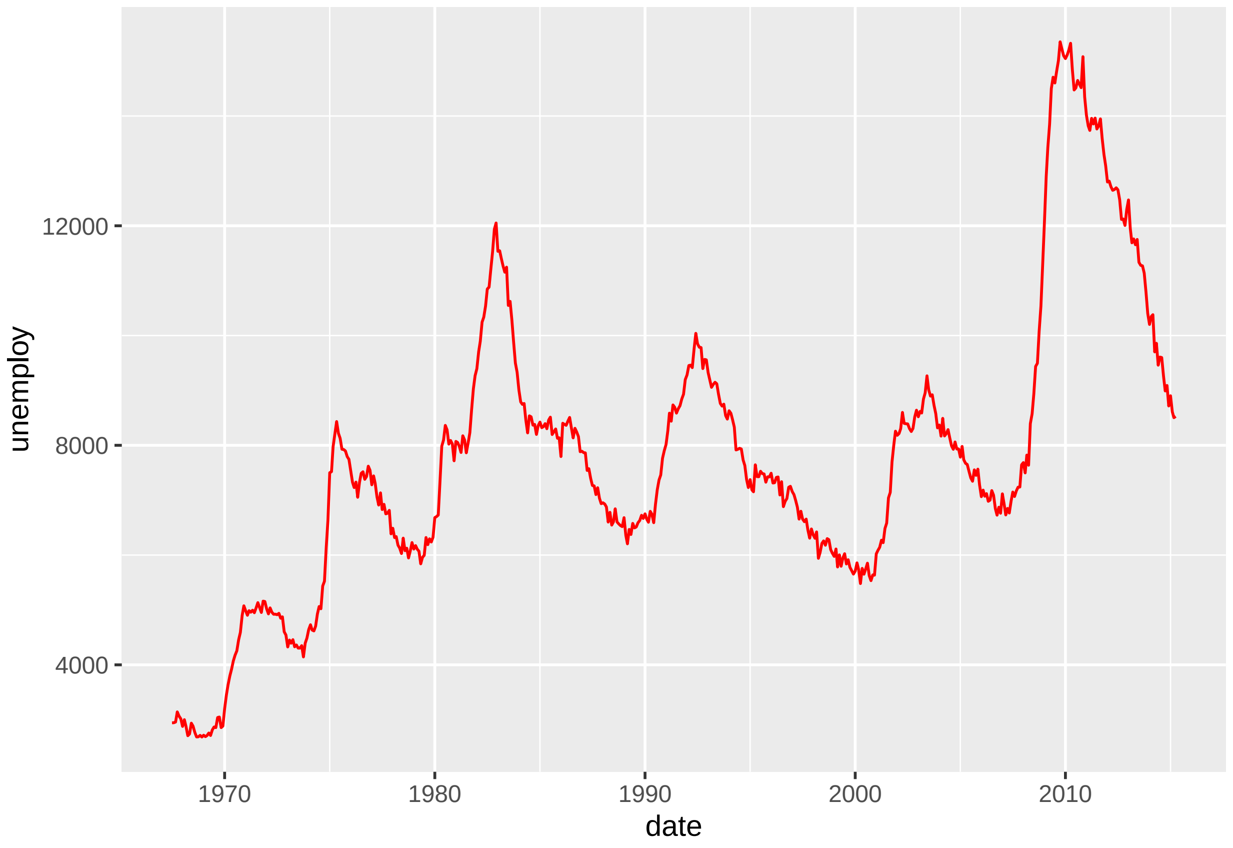
```
dat <- rounddf(dat, pad = TRUE)
summary(dat)
```

```
##        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.

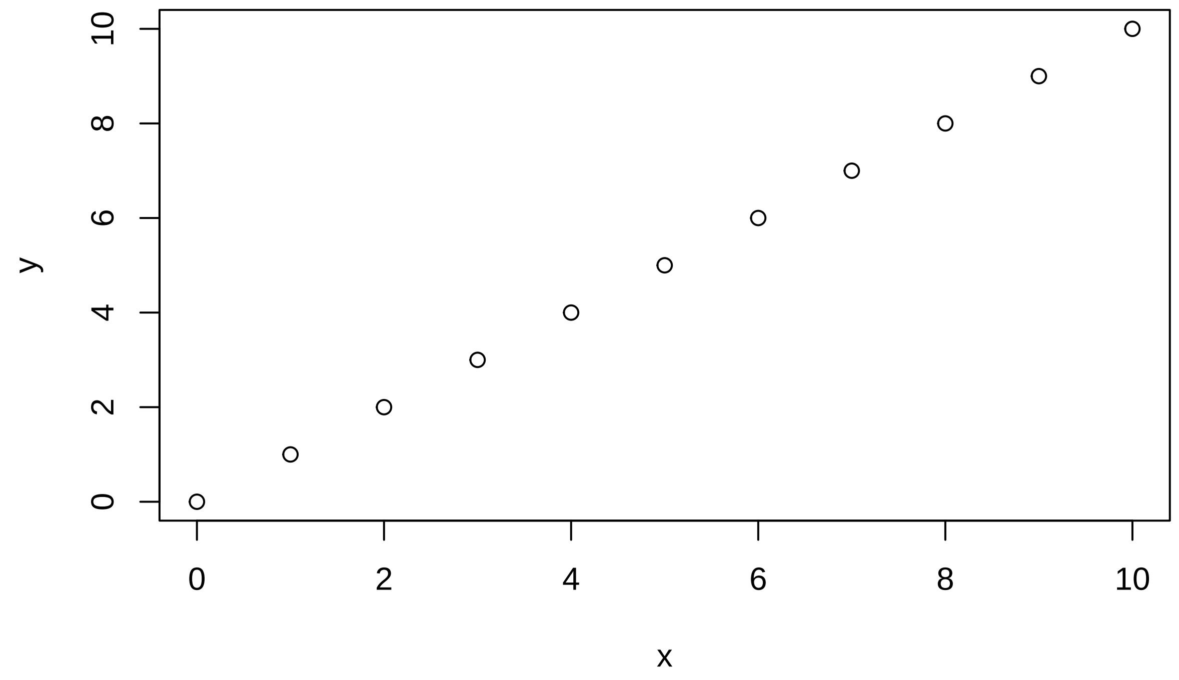
mintegrate

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`.

```r
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
```

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

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

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

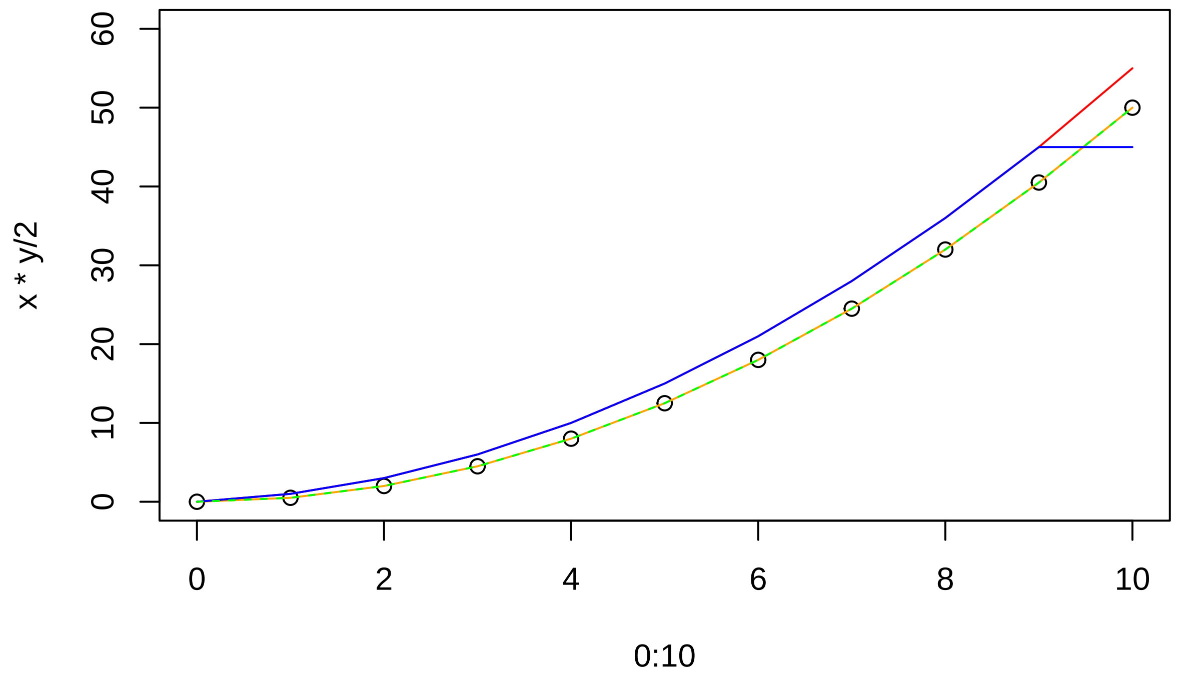
```
##  [1]  0  1  3  6 10 15 21 28 36 45 45
```

```r
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
```

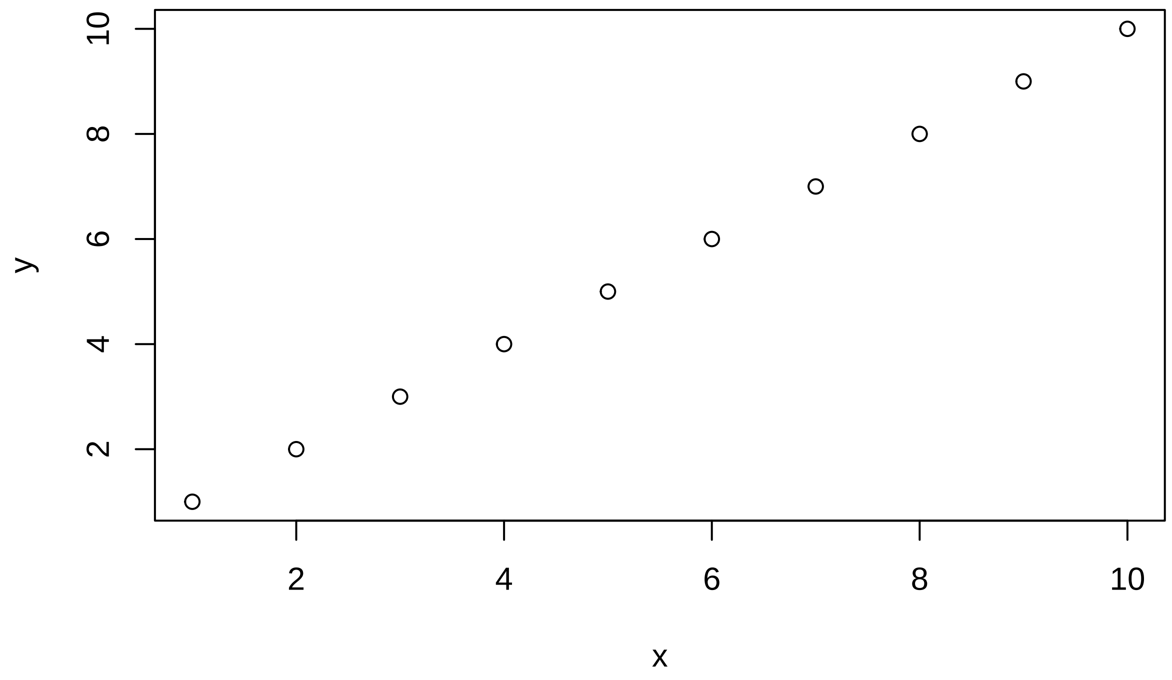
Note differences on the way up.

```r
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.

```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
```

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

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

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

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

```r
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.

```r
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
```

```r
mintegrate(x, y, 'left', lwr = 0)
```

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

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

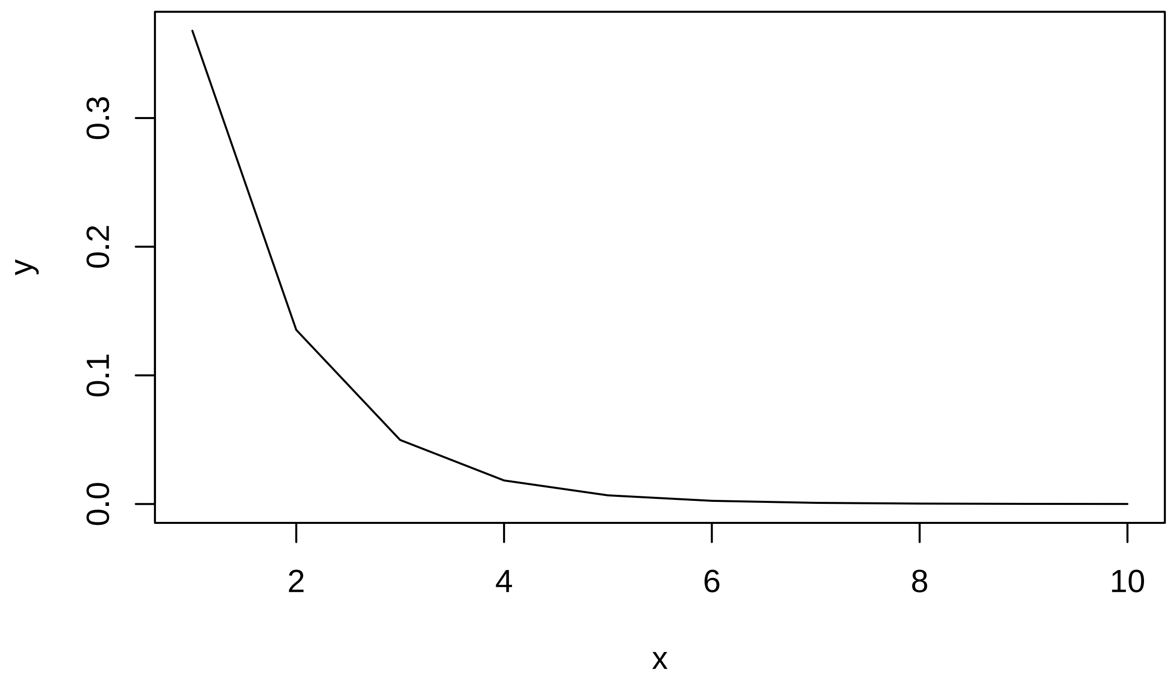
```
## [1]  1  3  6 10 15 21 28 36 45 45
```

```r
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

```r
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.

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

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

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

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

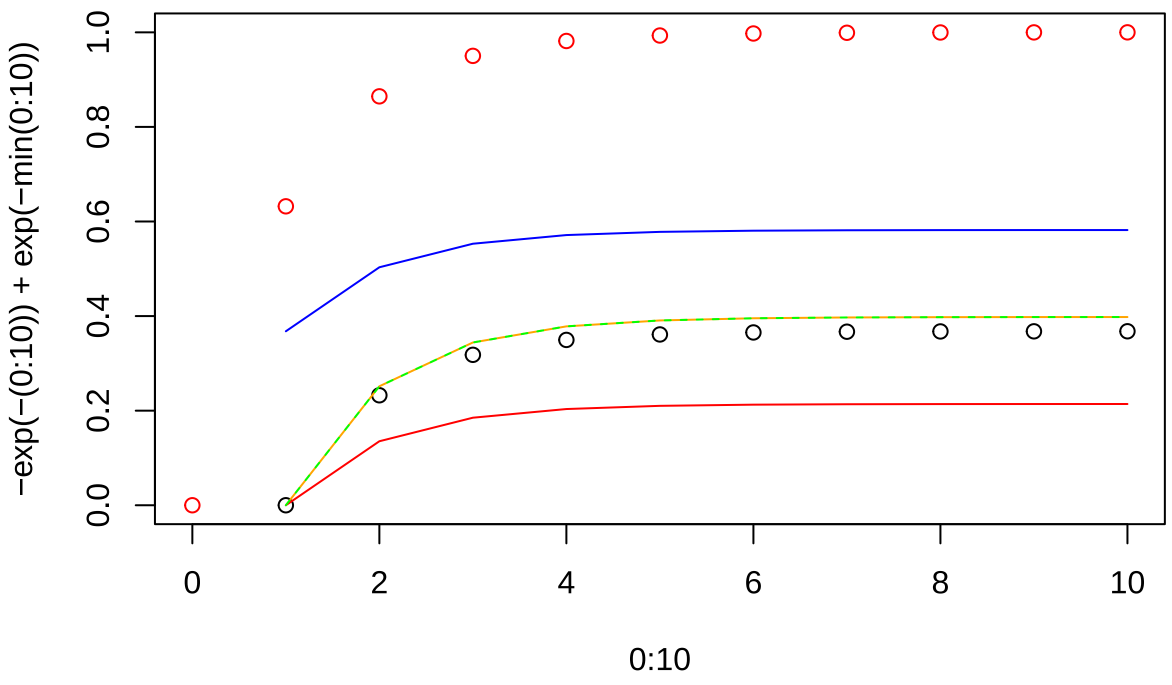
```r
mintegrate(x, y, 'right', value = 'total')
```

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

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

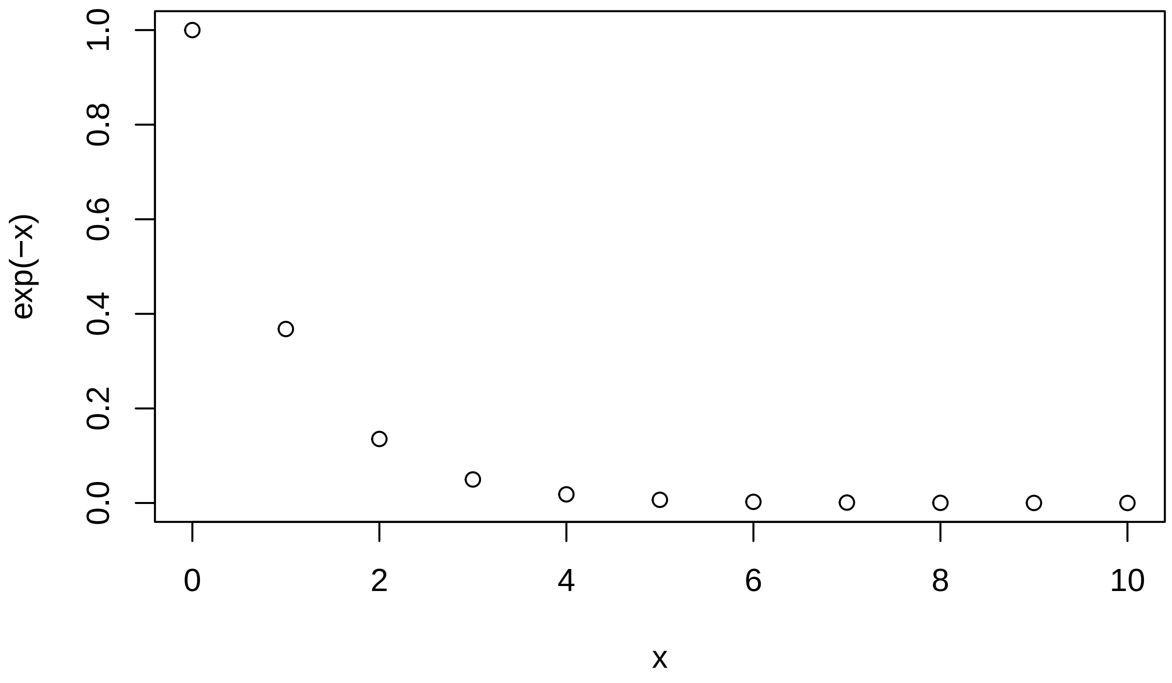
```
## [1] 0.3979879
```

```r
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)
```



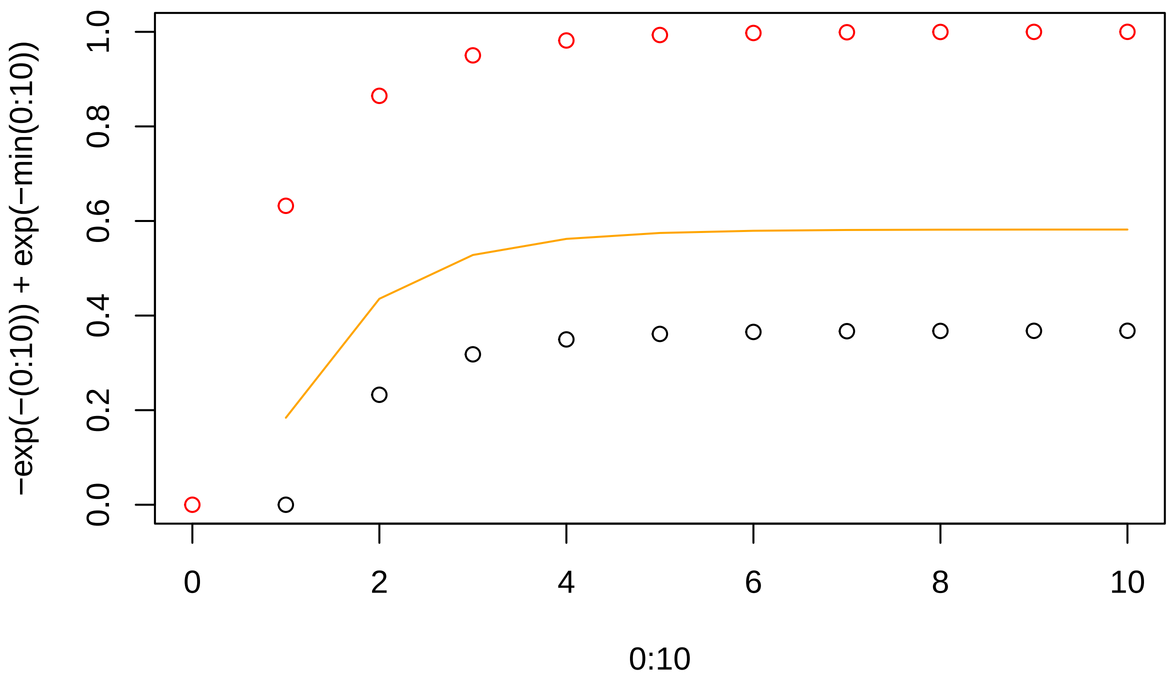
None is perfect, but midpoint and trapezoid (identical in this implementation) are the best, only slightly overestimating. Note that they all do poorly compared to a true integral that starts at 0 (red points). This cannot really be helped–how could we infer the true high values of y close to 0 from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

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



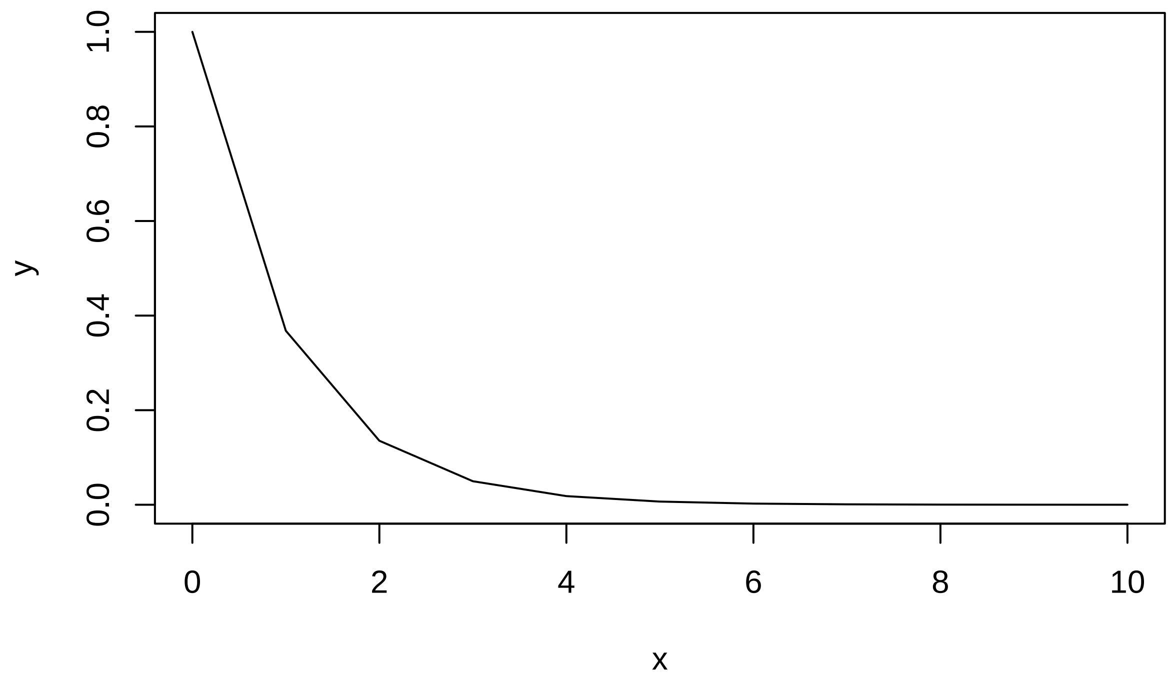
The `lwr` argument can extend the first rate back to 0 or any arbitrary starting point, which helps a bit.

```r
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')
```



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

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

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

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

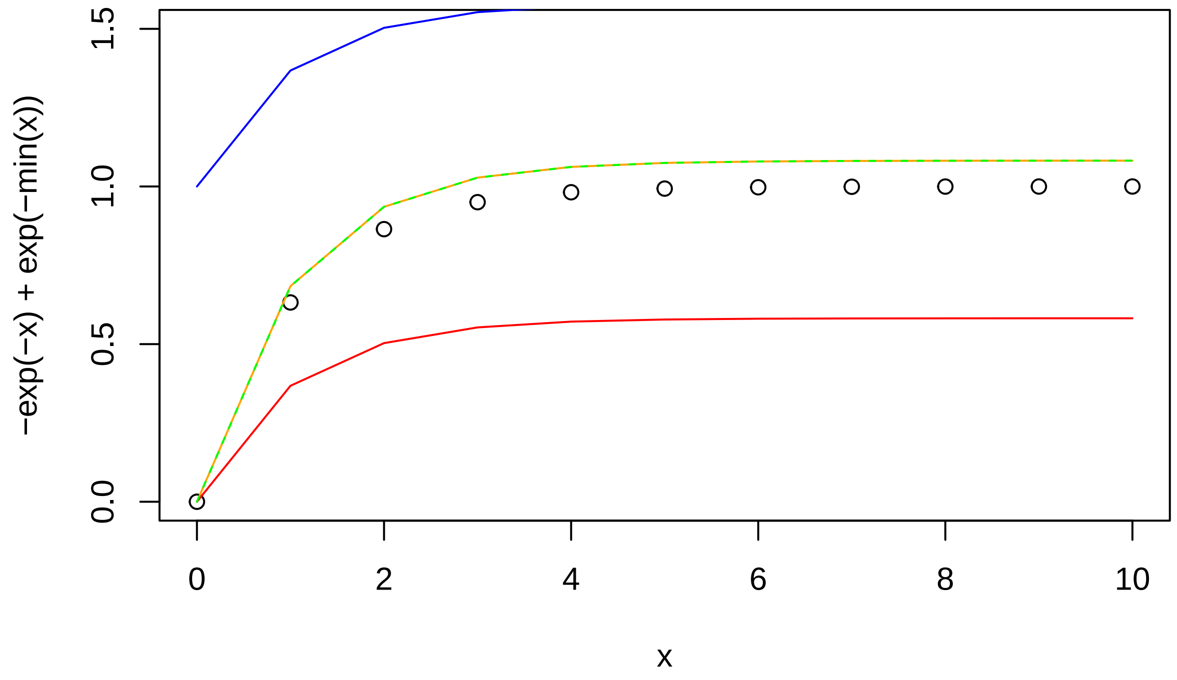
```r
mintegrate(x, y, 'right', value = 'total')
```

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

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

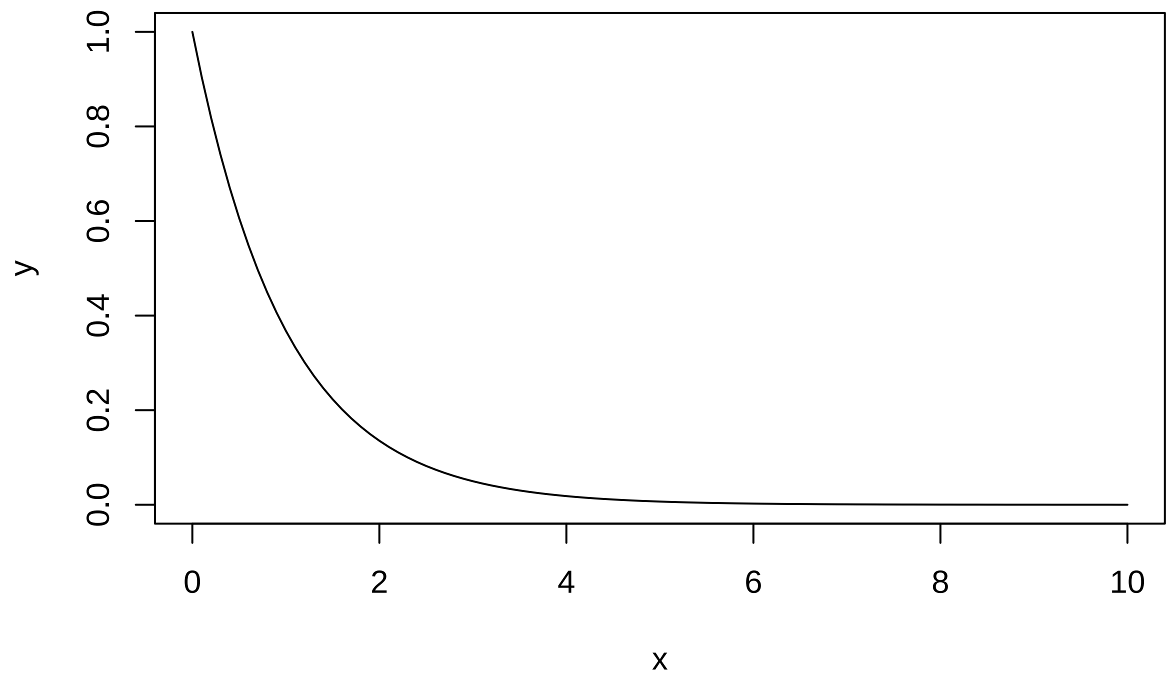
```
## [1] 1.081928
```

```r
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.

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



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

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

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

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

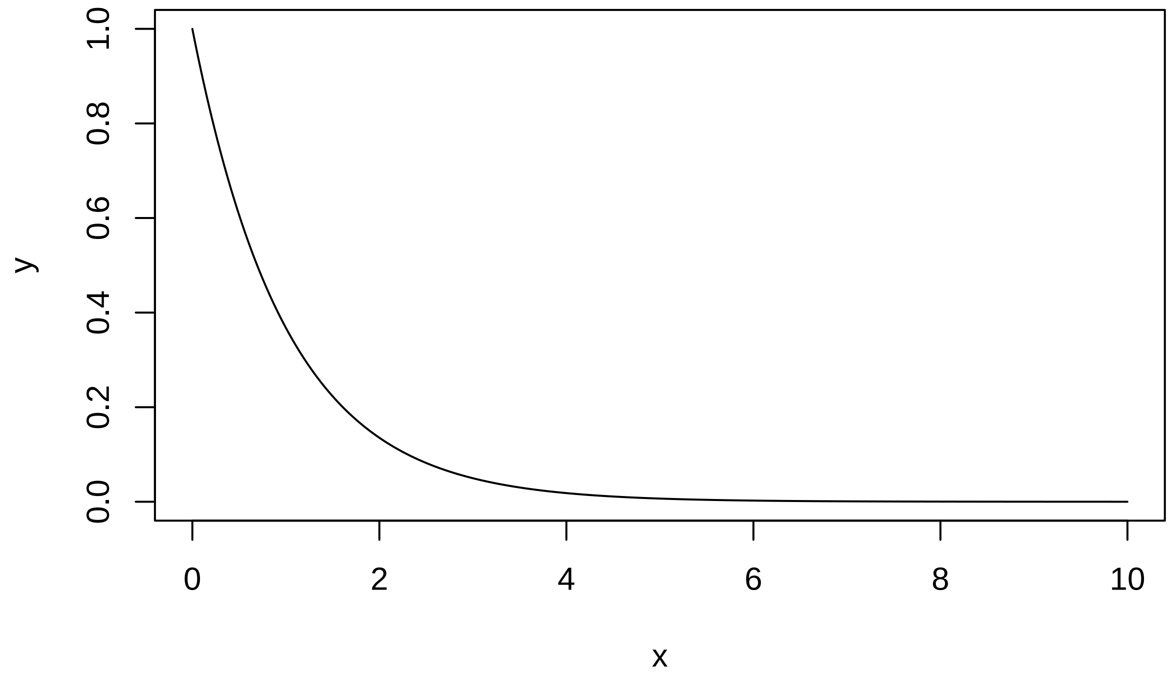
```r
mintegrate(x, y, 'right', value = 'total')
```

```
## [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')
```



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

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

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

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

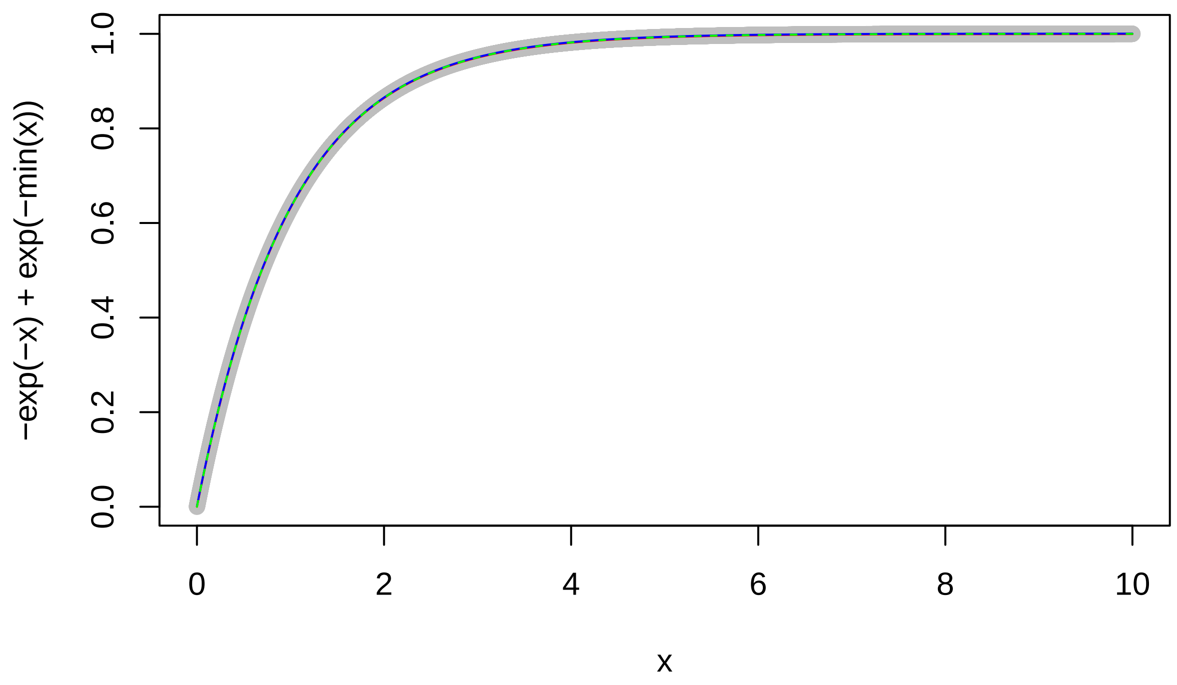
```
mintegrate(x, y, 'right', value = 'total')
```

```
## [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.

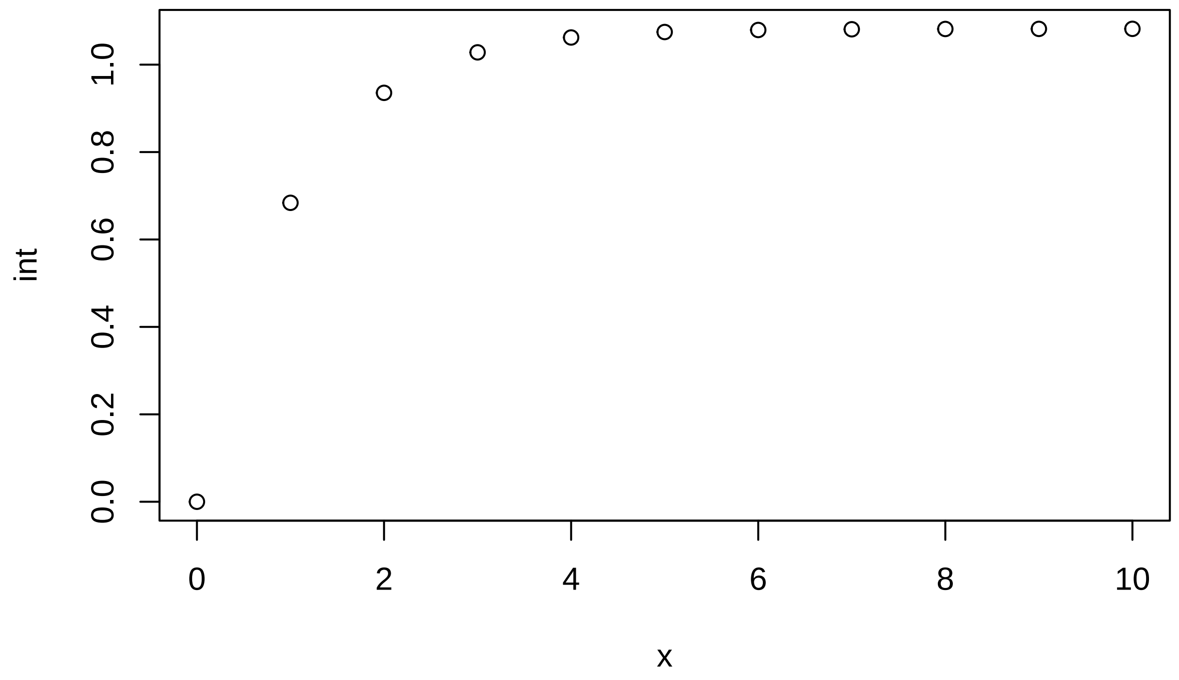
```r
x <- 0:10
y <- exp(-x)
```

```r
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
```

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

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



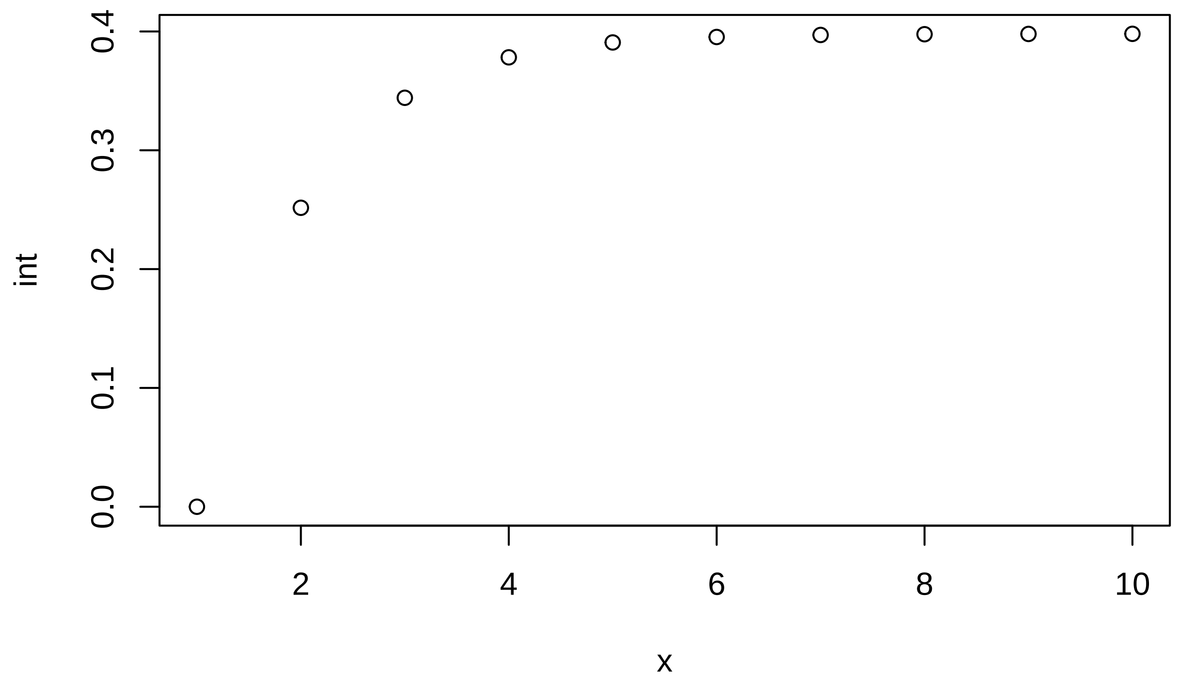
difftime

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

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

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

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



```r
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!

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

```r
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
```

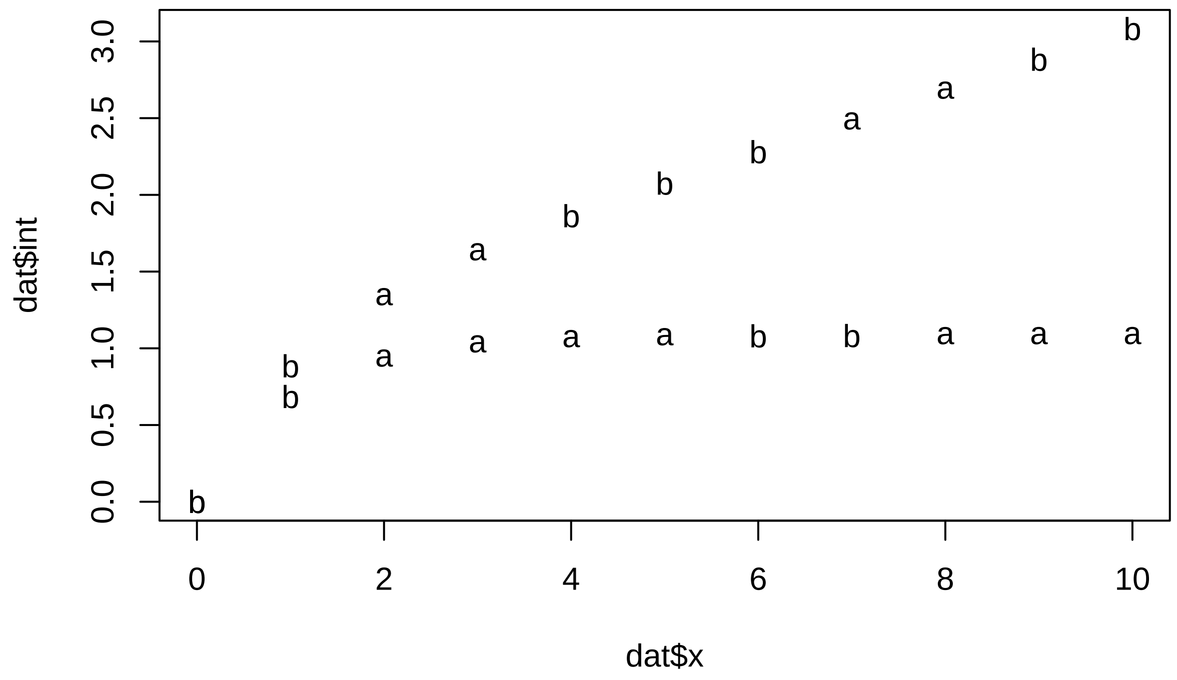
Grouped

```r
source('mintegrate.R')
x <- 0:10
y <- exp(-x)
```

Test below includes shuffling of order! First version couldn't handle this, oops.

```r
x <- c(x, x)
y <- c(y, y + 0.2)
g <- rep(c('a', 'b'), each = 11)
dat <- data.frame(x = x, y = y, g = g)
set.seed(123)
dat <- dat[sample(nrow(dat)), ]
```

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



consec

```
source('consec.R')

x <- data.frame(a = 1:20, b = c(1, 1, 1, 2, 2, 2, 4, 4, 5, 10, 10, 10, 10, 0, 0, 0, 0, 1, 1, 2))

x$group <- consec(x$b)
x
```

```
##     a  b group
## 1   1  1     1
## 2   2  1     1
## 3   3  1     1
## 4   4  2     2
## 5   5  2     2
## 6   6  2     2
## 7   7  4     3
## 8   8  4     3
## 9   9  5     4
## 10 10 10     5
## 11 11 10     5
## 12 12 10     5
## 13 13 10     5
## 14 14  0     6
## 15 15  0     6
## 16 16  0     6
## 17 17  0     6
## 18 18  1     7
## 19 19  1     7
## 20 20  2     8
```

```
x$group
```

```
##  [1] 1 1 1 2 2 2 3 3 4 5 5 5 5 6 6 6 6 7 7 8
```

```
x$group <- consec(x$b, value = 'f')
x
```

```
##     a  b group
## 1   1  1     1
## 2   2  1     1
## 3   3  1     1
## 4   4  2     2
## 5   5  2     2
## 6   6  2     2
## 7   7  4     3
## 8   8  4     3
## 9   9  5     4
## 10 10 10     5
## 11 11 10     5
## 12 12 10     5
## 13 13 10     5
## 14 14  0     6
## 15 15  0     6
## 16 16  0     6
## 17 17  0     6
## 18 18  1     7
## 19 19  1     7
## 20 20  2     8
```

```
x$group
```

```
##  [1] 1 1 1 2 2 2 3 3 4 5 5 5 5 6 6 6 6 7 7 8
## Levels: 1 2 3 4 5 6 7 8
```

With character variable.

```
x <- data.frame(a = 1:20, b = letters[c(1, 1, 1, 2, 2, 2, 4, 4, 5, 10, 10, 10, 10, 9, 9, 9, 9, 1, 1, 2)

x$group <- consec(x$b)
x
```

```
##     a b group
## 1   1 a     1
## 2   2 a     1
## 3   3 a     1
## 4   4 b     2
## 5   5 b     2
## 6   6 b     2
## 7   7 d     3
## 8   8 d     3
## 9   9 e     4
## 10 10 j     5
## 11 11 j     5
## 12 12 j     5
## 13 13 j     5
## 14 14 i     6
## 15 15 i     6
## 16 16 i     6
## 17 17 i     6
## 18 18 a     7
## 19 19 a     7
## 20 20 b     8
```

```
x$group
```

```
##  [1] 1 1 1 2 2 2 3 3 4 5 5 5 5 6 6 6 6 7 7 8
```

```
x$group <- consec(x$b, value = 'f')
x
```

```
##     a b group
## 1   1 a     1
## 2   2 a     1
## 3   3 a     1
## 4   4 b     2
## 5   5 b     2
## 6   6 b     2
## 7   7 d     3
## 8   8 d     3
## 9   9 e     4
## 10 10 j     5
## 11 11 j     5
## 12 12 j     5
## 13 13 j     5
## 14 14 i     6
## 15 15 i     6
## 16 16 i     6
## 17 17 i     6
## 18 18 a     7
## 19 19 a     7
## 20 20 b     8
```

```
x$group
```

```
##  [1] 1 1 1 2 2 2 3 3 4 5 5 5 5 6 6 6 6 7 7 8
## Levels: 1 2 3 4 5 6 7 8
```