jumbled demonstrations

Sasha D. Hafner

19 January, 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:

```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.

```
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
```

```
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
```

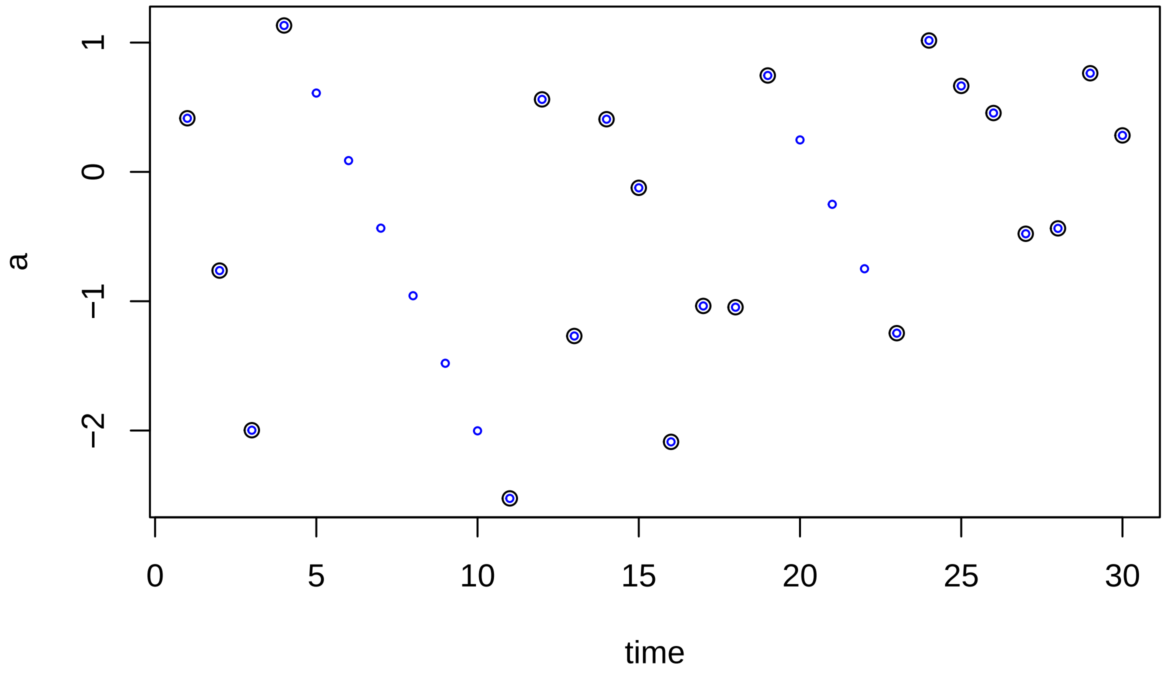
```
##    time          a           b           c
## 1     1  0.4146739 -0.32030487 -1.60814794
## 2     2 -0.7631388  1.53127649 -1.88420105
## 3     3 -1.9975089 -1.27387008  0.41793837
## 4     4  1.1320922 -0.07250209  1.96384173
## 5     5         NA          NA          NA
## 6     6         NA          NA          NA
## 7     7         NA          NA          NA
## 8     8         NA          NA          NA
## 9     9         NA          NA          NA
## 10   10         NA          NA          NA
## 11   11 -2.5246582 -0.97677404 -0.50853489
## 12   12  0.5607908  1.70668373  0.16508252
## 13   13 -1.2688344  1.65194645  0.08004954
## 14   14  0.4073851  2.60189129  0.96359976
## 15   15 -0.1231234  1.64926165  0.94231012
## 16   16 -2.0875150  0.07211064 -1.05845552
## 17   17 -1.0365061  2.21621815 -1.69738137
## 18   18 -1.0465087 -0.12141766  1.96930743
## 19   19  0.7455631 -0.38528916 -0.24729279
## 20   20         NA -0.14958971  0.32959681
## 21   21         NA  0.54821486  0.27203470
## 22   22         NA -2.40906325  2.15335261
## 23   23 -1.2469862 -1.84450964  0.74468509
## 24   24  1.0161096  1.45476108  0.26431943
## 25   25  0.6649798  1.01869988  0.90361380
## 26   26  0.4552556  0.65323398 -1.98231454
## 27   27 -0.4782209  0.78409495 -1.33163285
## 28   28 -0.4365978 -0.53368395 -1.99060491
## 29   29  0.7628191 -0.30224424  2.14882877
## 30   30  0.2821344 -1.19825252  2.36853801
```

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

dat2
```

```
##    time           a           b           c
## 1     1  0.41467386 -0.32030487 -1.60814794
## 2     2 -0.76313885  1.53127649 -1.88420105
## 3     3 -1.99750886 -1.27387008  0.41793837
## 4     4  1.13209219 -0.07250209  1.96384173
## 5     5  0.60969928 -0.20168379  1.61064507
## 6     6  0.08730637 -0.33086550  1.25744841
## 7     7 -0.43508654 -0.46004721  0.90425175
## 8     8 -0.95747945 -0.58922892  0.55105509
## 9     9 -1.47987236 -0.71841063  0.19785843
## 10   10 -2.00226527 -0.84759233 -0.15533823
## 11   11 -2.52465819 -0.97677404 -0.50853489
## 12   12  0.56079084  1.70668373  0.16508252
## 13   13 -1.26883437  1.65194645  0.08004954
## 14   14  0.40738509  2.60189129  0.96359976
## 15   15 -0.12312342  1.64926165  0.94231012
## 16   16 -2.08751504  0.07211064 -1.05845552
## 17   17 -1.03650609  2.21621815 -1.69738137
## 18   18 -1.04650871 -0.12141766  1.96930743
## 19   19  0.74556310 -0.38528916 -0.24729279
## 20   20  0.24742578 -0.14958971  0.32959681
## 21   21 -0.25071153  0.54821486  0.27203470
## 22   22 -0.74884885 -2.40906325  2.15335261
## 23   23 -1.24698617 -1.84450964  0.74468509
## 24   24  1.01610964  1.45476108  0.26431943
## 25   25  0.66497975  1.01869988  0.90361380
## 26   26  0.45525562  0.65323398 -1.98231454
## 27   27 -0.47822089  0.78409495 -1.33163285
## 28   28 -0.43659782 -0.53368395 -1.99060491
## 29   29  0.76281915 -0.30224424  2.14882877
## 30   30  0.28213438 -1.19825252  2.36853801
```

```
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  0.2967333 -1.42587814 -0.32878710
## 2     2     a  0.7695155 -1.60323073  0.16911639
## 3     3     a  0.7914987 -0.09754449 -1.48821616
## 4     4     a -1.6497331  0.13508043  1.89529384
## 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  1.5888979 -2.62431940  1.64423589
## 11    1     b -0.8582577 -1.33304129  1.59765925
## 12    2     b -1.6816094  1.52983965 -1.08174391
## 13    3     b -0.6522340 -2.03491001  0.06655042
## 14    4     b -1.1640318 -0.81412636  1.17031669
## 15    5     b  0.3499664  0.67303911  0.79412086
## 16    6     b  0.5183973  1.33718108 -0.03366862
## 17    7     b -0.4623732  0.76354637  1.02392965
## 18    8     b -1.3600316 -0.89690782  1.80116536
## 19    9     b -0.8352634  0.96410430 -0.84215418
## 20   10     b         NA  0.20760306  0.67035382
## 21    1     c -0.5865541 -0.69715853  1.02604741
## 22    2     c         NA -0.42522934  0.60000430
## 23    3     c -0.2839742  0.36348994 -0.03185524
## 24    4     c -0.8781523  0.19495920 -0.24331214
## 25    5     c  1.7873842 -1.27292526  1.57949232
## 26    6     c  0.6331555  0.02510162  0.22826516
## 27    7     c -1.3973835 -0.10401462 -0.34183738
## 28    8     c  0.8114896 -0.88733501 -0.81816525
## 29    9     c -0.6420098  0.50286180 -0.50142871
## 30   10     c -0.8587650  1.65451395 -0.70650384
```

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

```
##    time group           a           b           c
## 1     1     a  0.29673326 -1.42587814 -0.32878710
## 2     2     a  0.76951554 -1.60323073  0.16911639
## 3     3     a  0.79149868 -0.09754449 -1.48821616
## 4     4     a -1.64973311  0.13508043  1.89529384
## 5     5     a -1.10996128 -0.32481954  1.85345085
## 6     6     a -0.57018945 -0.78471951  1.81160786
## 7     7     a -0.03041761 -1.24461949  1.76976487
## 8     8     a  0.50935422 -1.70451946  1.72792187
## 9     9     a  1.04912605 -2.16441943  1.68607888
## 10   10     a  1.58889788 -2.62431940  1.64423589
## 11    1     b -0.85825773 -1.33304129  1.59765925
## 12    2     b -1.68160945  1.52983965 -1.08174391
## 13    3     b -0.65223401 -2.03491001  0.06655042
## 14    4     b -1.16403176 -0.81412636  1.17031669
## 15    5     b  0.34996642  0.67303911  0.79412086
## 16    6     b  0.51839729  1.33718108 -0.03366862
## 17    7     b -0.46237318  0.76354637  1.02392965
## 18    8     b -1.36003159 -0.89690782  1.80116536
## 19    9     b -0.83526336  0.96410430 -0.84215418
## 20   10     b          NA  0.20760306  0.67035382
## 21    1     c -0.58655411 -0.69715853  1.02604741
## 22    2     c -0.43526418 -0.42522934  0.60000430
## 23    3     c -0.28397425  0.36348994 -0.03185524
## 24    4     c -0.87815231  0.19495920 -0.24331214
## 25    5     c  1.78738423 -1.27292526  1.57949232
## 26    6     c  0.63315546  0.02510162  0.22826516
## 27    7     c -1.39738350 -0.10401462 -0.34183738
## 28    8     c  0.81148955 -0.88733501 -0.81816525
## 29    9     c -0.64200980  0.50286180 -0.50142871
## 30   10     c -0.85876505  1.65451395 -0.70650384
```

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

```
##    time group           a           b           c
## 1     1     a  0.29673326 -1.42587814 -0.32878710
## 2     2     a  0.76951554 -1.60323073  0.16911639
## 3     3     a  0.79149868 -0.09754449 -1.48821616
## 4     4     a -1.64973311  0.13508043  1.89529384
## 5     5     a -1.10996128 -0.32481954  1.85345085
## 6     6     a -0.57018945 -0.78471951  1.81160786
## 7     7     a -0.03041761 -1.24461949  1.76976487
## 8     8     a  0.50935422 -1.70451946  1.72792187
## 9     9     a  1.04912605 -2.16441943  1.68607888
## 10   10     a  1.58889788 -2.62431940  1.64423589
## 11    1     b -0.85825773 -1.33304129  1.59765925
## 12    2     b -1.68160945  1.52983965 -1.08174391
## 13    3     b -0.65223401 -2.03491001  0.06655042
## 14    4     b -1.16403176 -0.81412636  1.17031669
## 15    5     b  0.34996642  0.67303911  0.79412086
## 16    6     b  0.51839729  1.33718108 -0.03366862
## 17    7     b -0.46237318  0.76354637  1.02392965
## 18    8     b -1.36003159 -0.89690782  1.80116536
## 19    9     b -0.83526336  0.96410430 -0.84215418
## 20   10     b -0.83526336  0.20760306  0.67035382
## 21    1     c -0.58655411 -0.69715853  1.02604741
## 22    2     c -0.43526418 -0.42522934  0.60000430
## 23    3     c -0.28397425  0.36348994 -0.03185524
## 24    4     c -0.87815231  0.19495920 -0.24331214
## 25    5     c  1.78738423 -1.27292526  1.57949232
## 26    6     c  0.63315546  0.02510162  0.22826516
## 27    7     c -1.39738350 -0.10401462 -0.34183738
## 28    8     c  0.81148955 -0.88733501 -0.81816525
## 29    9     c -0.64200980  0.50286180 -0.50142871
## 30   10     c -0.85876505  1.65451395 -0.70650384
```

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

```
##     time group          a           b           c
##  1:    1     a  0.2967333 -1.42587814 -0.32878710
##  2:    2     a  0.7695155 -1.60323073  0.16911639
##  3:    3     a  0.7914987 -0.09754449 -1.48821616
##  4:    4     a -1.6497331  0.13508043  1.89529384
##  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  1.5888979 -2.62431940  1.64423589
## 11:    1     b -0.8582577 -1.33304129  1.59765925
## 12:    2     b -1.6816094  1.52983965 -1.08174391
## 13:    3     b -0.6522340 -2.03491001  0.06655042
## 14:    4     b -1.1640318 -0.81412636  1.17031669
## 15:    5     b  0.3499664  0.67303911  0.79412086
## 16:    6     b  0.5183973  1.33718108 -0.03366862
## 17:    7     b -0.4623732  0.76354637  1.02392965
## 18:    8     b -1.3600316 -0.89690782  1.80116536
## 19:    9     b -0.8352634  0.96410430 -0.84215418
## 20:   10     b         NA  0.20760306  0.67035382
## 21:    1     c -0.5865541 -0.69715853  1.02604741
## 22:    2     c         NA -0.42522934  0.60000430
## 23:    3     c -0.2839742  0.36348994 -0.03185524
## 24:    4     c -0.8781523  0.19495920 -0.24331214
## 25:    5     c  1.7873842 -1.27292526  1.57949232
## 26:    6     c  0.6331555  0.02510162  0.22826516
## 27:    7     c -1.3973835 -0.10401462 -0.34183738
## 28:    8     c  0.8114896 -0.88733501 -0.81816525
## 29:    9     c -0.6420098  0.50286180 -0.50142871
## 30:   10     c -0.8587650  1.65451395 -0.70650384
##     time group          a           b           c
```

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

```
##     time group           a           b           c
##  1:    1     a  0.29673326 -1.42587814 -0.32878710
##  2:    2     a  0.76951554 -1.60323073  0.16911639
##  3:    3     a  0.79149868 -0.09754449 -1.48821616
##  4:    4     a -1.64973311  0.13508043  1.89529384
##  5:    5     a -1.10996128 -0.32481954  1.85345085
##  6:    6     a -0.57018945 -0.78471951  1.81160786
##  7:    7     a -0.03041761 -1.24461949  1.76976487
##  8:    8     a  0.50935422 -1.70451946  1.72792187
##  9:    9     a  1.04912605 -2.16441943  1.68607888
## 10:   10     a  1.58889788 -2.62431940  1.64423589
## 11:    1     b -0.85825773 -1.33304129  1.59765925
## 12:    2     b -1.68160945  1.52983965 -1.08174391
## 13:    3     b -0.65223401 -2.03491001  0.06655042
## 14:    4     b -1.16403176 -0.81412636  1.17031669
## 15:    5     b  0.34996642  0.67303911  0.79412086
## 16:    6     b  0.51839729  1.33718108 -0.03366862
## 17:    7     b -0.46237318  0.76354637  1.02392965
## 18:    8     b -1.36003159 -0.89690782  1.80116536
## 19:    9     b -0.83526336  0.96410430 -0.84215418
## 20:   10     b          NA  0.20760306  0.67035382
## 21:    1     c -0.58655411 -0.69715853  1.02604741
## 22:    2     c -0.43526418 -0.42522934  0.60000430
## 23:    3     c -0.28397425  0.36348994 -0.03185524
## 24:    4     c -0.87815231  0.19495920 -0.24331214
## 25:    5     c  1.78738423 -1.27292526  1.57949232
## 26:    6     c  0.63315546  0.02510162  0.22826516
## 27:    7     c -1.39738350 -0.10401462 -0.34183738
## 28:    8     c  0.81148955 -0.88733501 -0.81816525
## 29:    9     c -0.64200980  0.50286180 -0.50142871
## 30:   10     c -0.85876505  1.65451395 -0.70650384
##     time group           a           b           c
```

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

```
##     time group           a           b           c
##  1:    1     a  0.29673326 -1.42587814 -0.32878710
##  2:    2     a  0.76951554 -1.60323073  0.16911639
##  3:    3     a  0.79149868 -0.09754449 -1.48821616
##  4:    4     a -1.64973311  0.13508043  1.89529384
##  5:    5     a -1.10996128 -0.32481954  1.85345085
##  6:    6     a -0.57018945 -0.78471951  1.81160786
##  7:    7     a -0.03041761 -1.24461949  1.76976487
##  8:    8     a  0.50935422 -1.70451946  1.72792187
##  9:    9     a  1.04912605 -2.16441943  1.68607888
## 10:   10     a  1.58889788 -2.62431940  1.64423589
## 11:    1     b -0.85825773 -1.33304129  1.59765925
## 12:    2     b -1.68160945  1.52983965 -1.08174391
## 13:    3     b -0.65223401 -2.03491001  0.06655042
## 14:    4     b -1.16403176 -0.81412636  1.17031669
## 15:    5     b  0.34996642  0.67303911  0.79412086
## 16:    6     b  0.51839729  1.33718108 -0.03366862
## 17:    7     b -0.46237318  0.76354637  1.02392965
## 18:    8     b -1.36003159 -0.89690782  1.80116536
## 19:    9     b -0.83526336  0.96410430 -0.84215418
## 20:   10     b -0.83526336  0.20760306  0.67035382
## 21:    1     c -0.58655411 -0.69715853  1.02604741
## 22:    2     c -0.43526418 -0.42522934  0.60000430
## 23:    3     c -0.28397425  0.36348994 -0.03185524
## 24:    4     c -0.87815231  0.19495920 -0.24331214
## 25:    5     c  1.78738423 -1.27292526  1.57949232
## 26:    6     c  0.63315546  0.02510162  0.22826516
## 27:    7     c -1.39738350 -0.10401462 -0.34183738
## 28:    8     c  0.81148955 -0.88733501 -0.81816525
## 29:    9     c -0.64200980  0.50286180 -0.50142871
## 30:   10     c -0.85876505  1.65451395 -0.70650384
##     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.

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

```
##     a           b c
## 1   1 -0.53768654 a
## 2   2  0.86645330 b
## 3   3 -2.03462623 c
## 4   4 -0.29194284 d
## 5   5 -0.67431186 e
## 6   6  0.28148899 f
## 7   7 -0.26587311 g
## 8   8 -1.07065414 h
## 9   9 -1.69594034 i
## 10 10  0.06461751 j
```

```r
rounddf(dat)
```

```
##     a     b c
## 1   1 -0.54 a
## 2   2  0.87 b
## 3   3 -2.03 c
## 4   4 -0.29 d
## 5   5 -0.67 e
## 6   6  0.28 f
## 7   7 -0.27 g
## 8   8 -1.07 h
## 9   9 -1.70 i
## 10 10  0.06 j
```

```r
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 -0.5377 a
## 2   2  0.8665 b
## 3   3 -2.0346 c
## 4   4 -0.2919 d
## 5   5 -0.6743 e
## 6   6  0.2815 f
## 7   7 -0.2659 g
## 8   8 -1.0707 h
## 9   9 -1.6959 i
## 10 10  0.0646 j
```

```r
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 -0.53770 a
## 2   2  0.86650 b
## 3   3 -2.03500 c
## 4   4 -0.29190 d
## 5   5 -0.67430 e
## 6   6  0.28150 f
## 7   7 -0.26590 g
## 8   8 -1.07100 h
## 9   9 -1.69600 i
## 10 10  0.06462 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 -0.540 a
## 2   2  0.870 b
## 3   3 -2.000 c
## 4   4 -0.290 d
## 5   5 -0.670 e
## 6   6  0.280 f
## 7   7 -0.270 g
## 8   8 -1.100 h
## 9   9 -1.700 i
## 10 10  0.065 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
```

```r
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
```

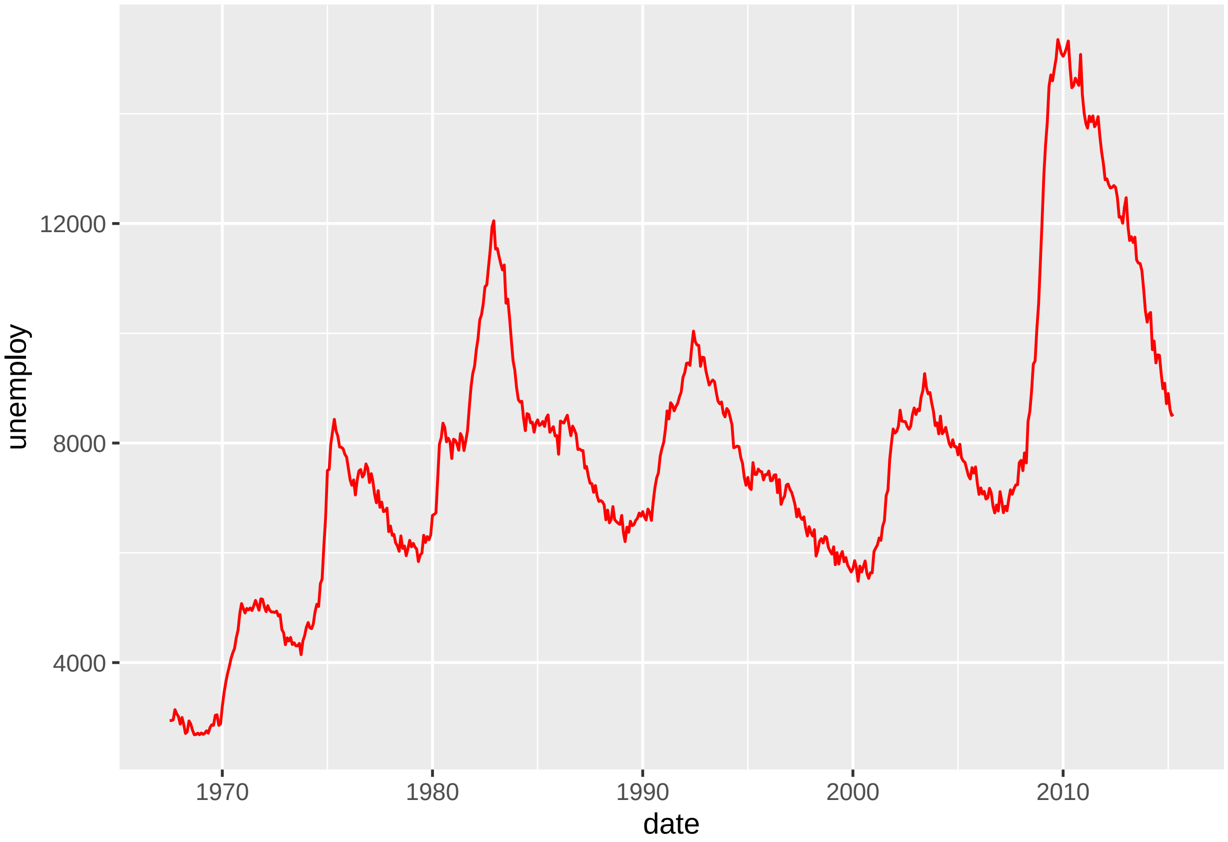
```r
dat <- rounddf(dat, pad = TRUE)
summary(dat)
```

```
##        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.

```r
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.

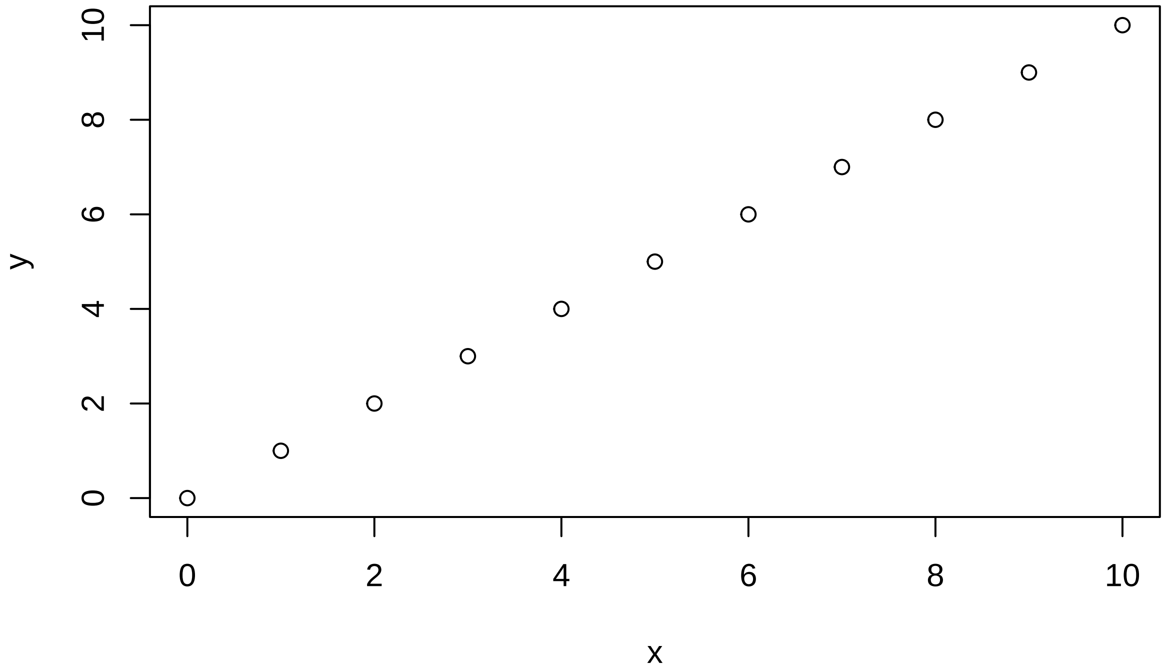
`fintegrate`

Integrate $f$lux measurements for emission.

```
source('fintegrate.R')
```

1. Linear

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



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

```
fintegrate(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
```

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

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

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

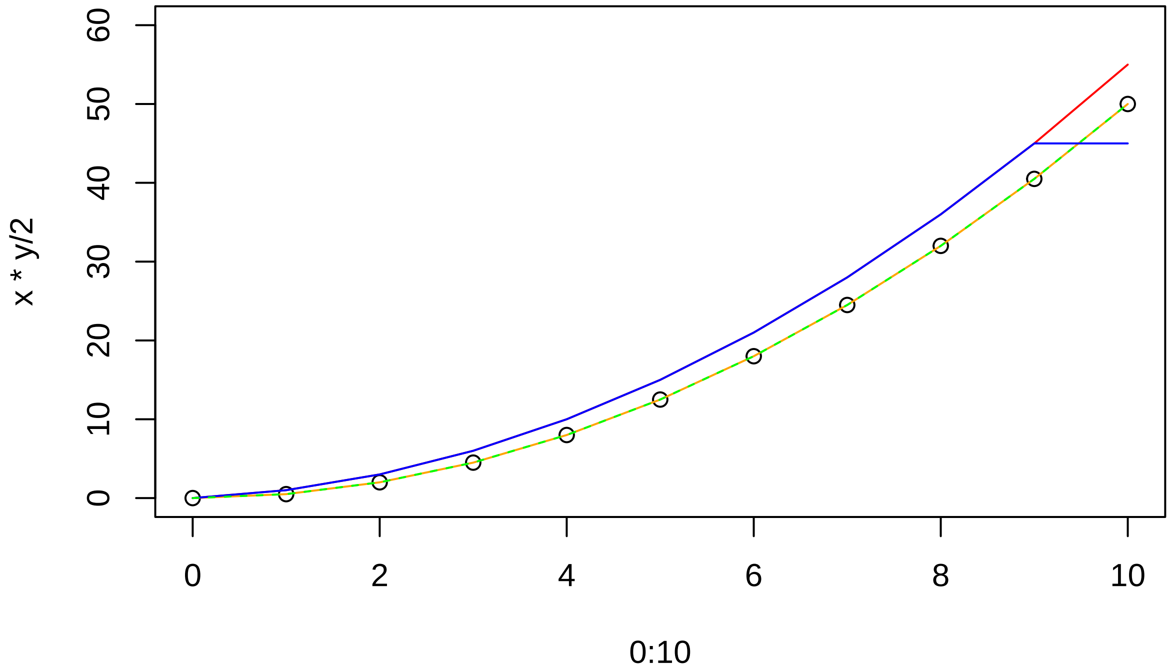
```
##  [1]  0  1  3  6 10 15 21 28 36 45 45
```

```
fintegrate(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
```

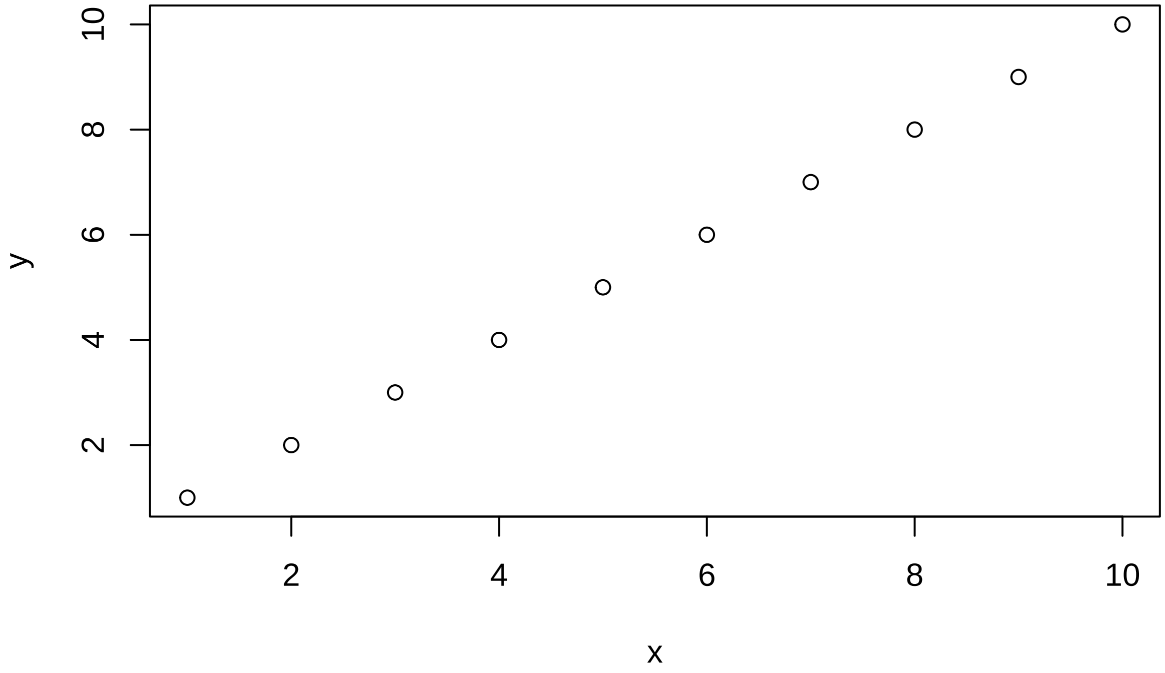
Note differences on the way up.

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

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

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

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

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

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

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

```
fintegrate(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.

```
fintegrate(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
```

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

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

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

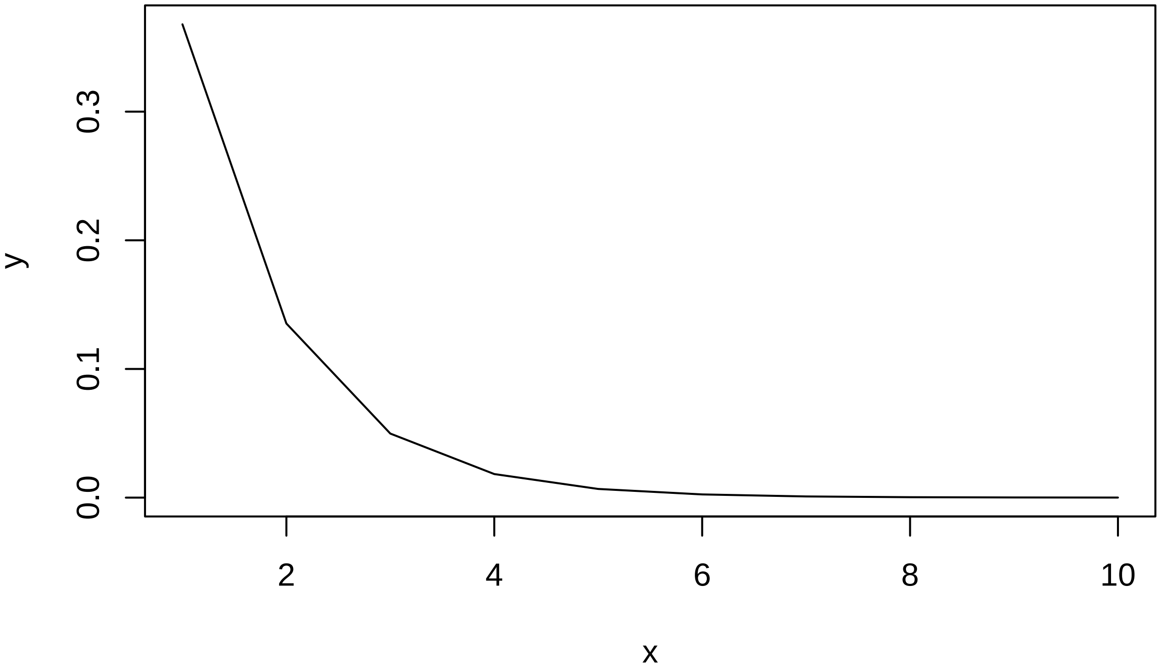
```
## [1]  1  3  6 10 15 21 28 36 45 45
```

```
fintegrate(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.

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

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

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

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

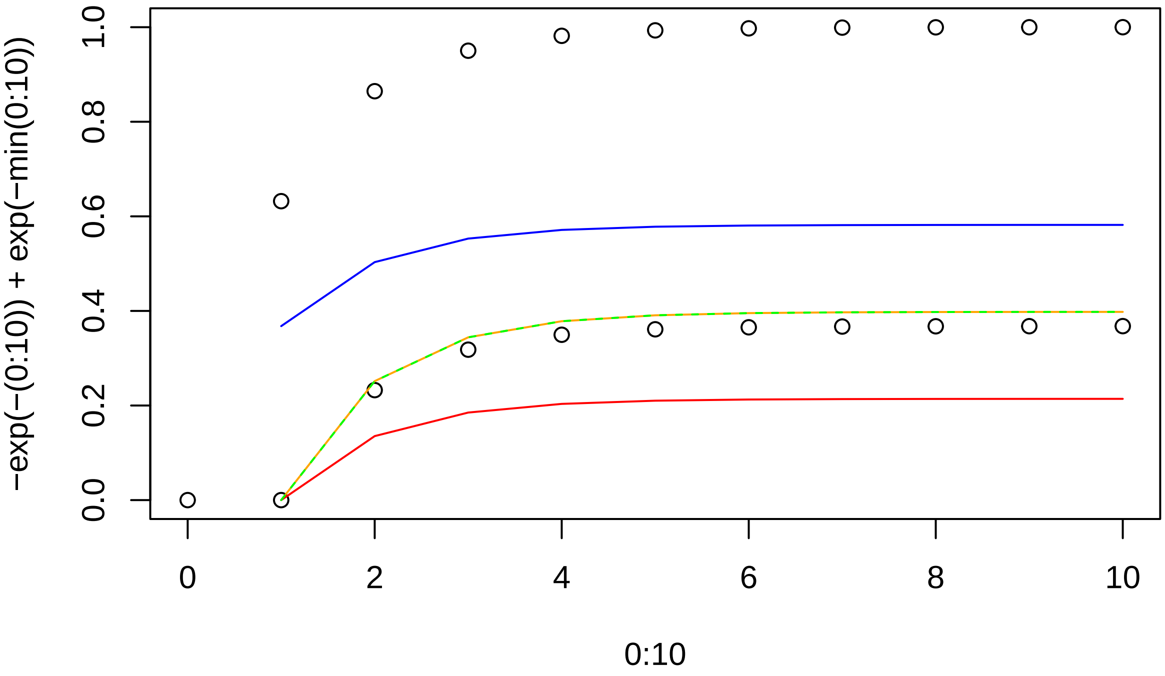
```
fintegrate(x, y, 'right', value = 'total')
```

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

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

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

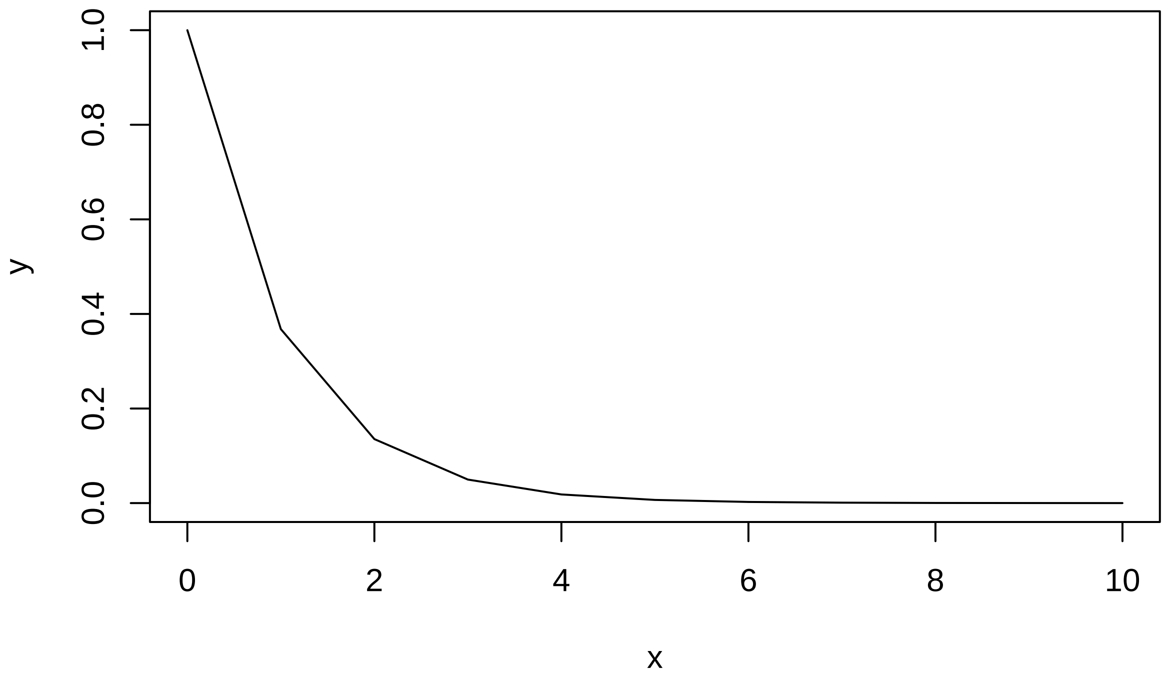
```
plot(0:10, -exp(-(0:10)) + exp(-min(0:10)))
points(x, -exp(-x) + exp(-min(x)), ylim = c(0, 0.7))
lines(x, fintegrate(x, y, 'midpoint'), col = 'orange')
lines(x, fintegrate(x, y, 'left'), col = 'red')
lines(x, fintegrate(x, y, 'right'), col = 'blue')
lines(x, fintegrate(x, y, 'trap'), col = 'green',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



None does very well.

Start at 0.

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



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

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

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

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

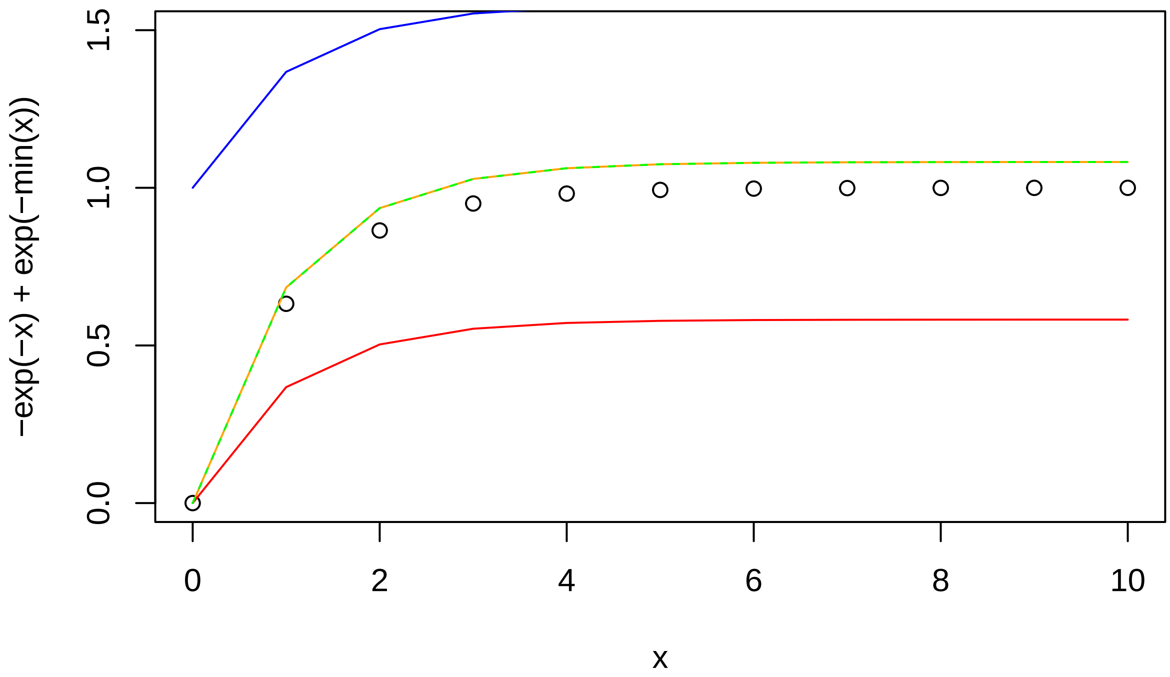
```
fintegrate(x, y, 'right', value = 'total')
```

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

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

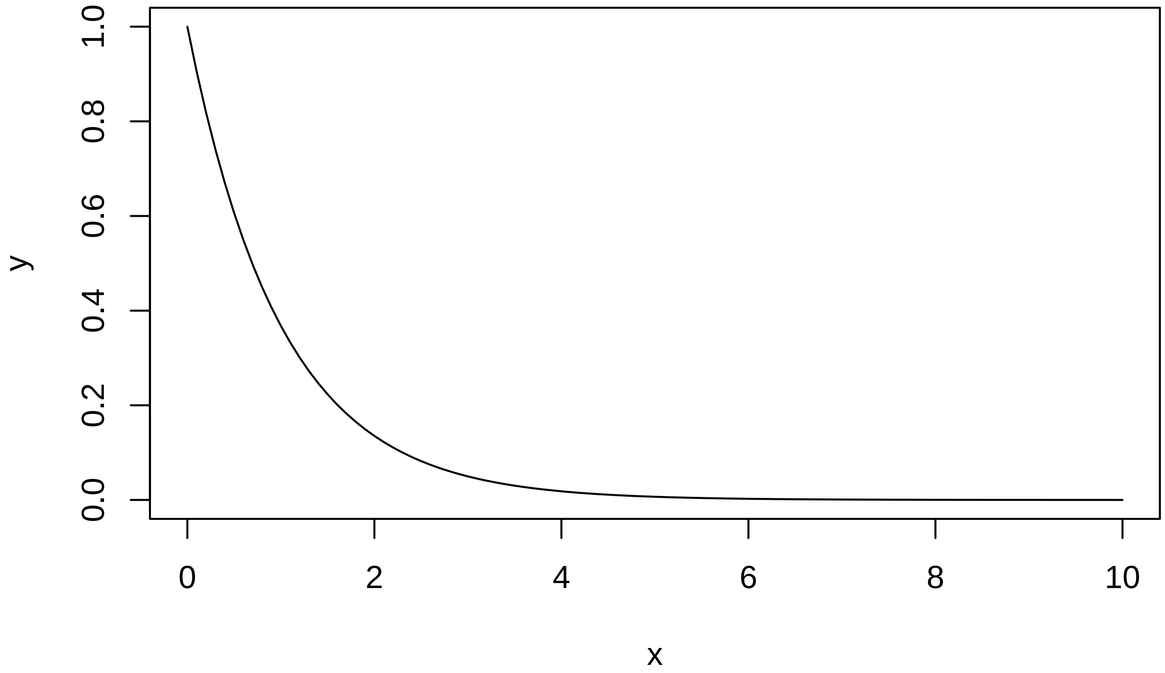
```
## [1] 1.081928
```

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



Prove that all methods become accurate with very high resolution.

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



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

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

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

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

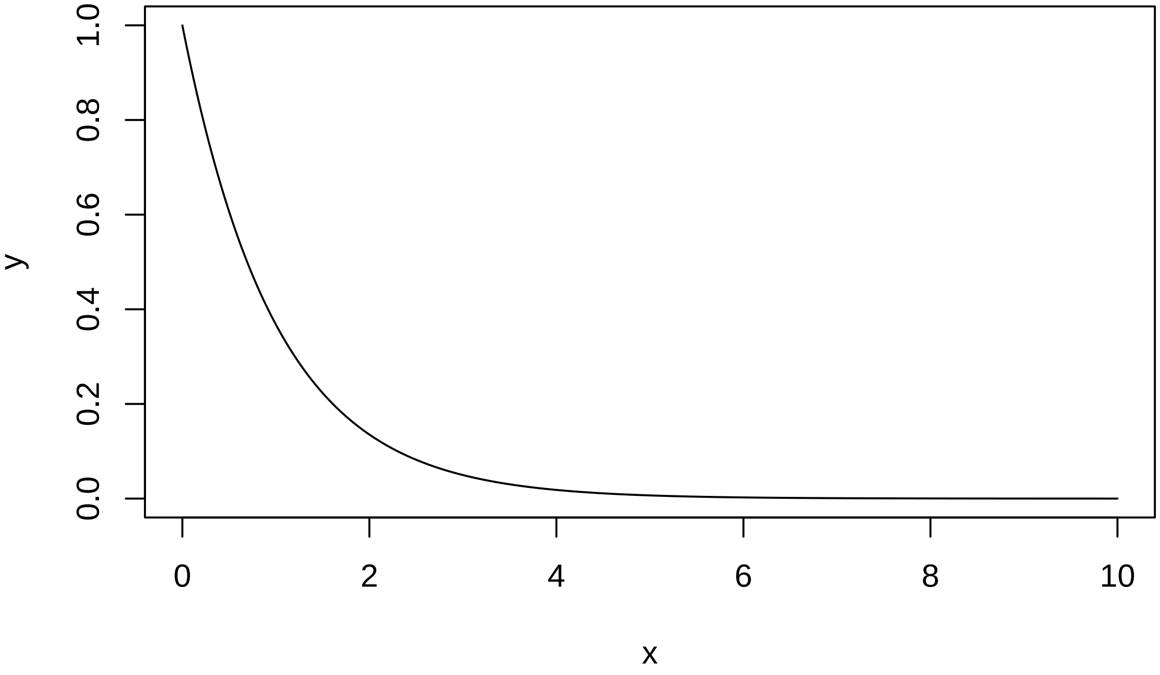
```r
fintegrate(x, y, 'right', value = 'total')
```

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

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

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

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



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

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

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

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

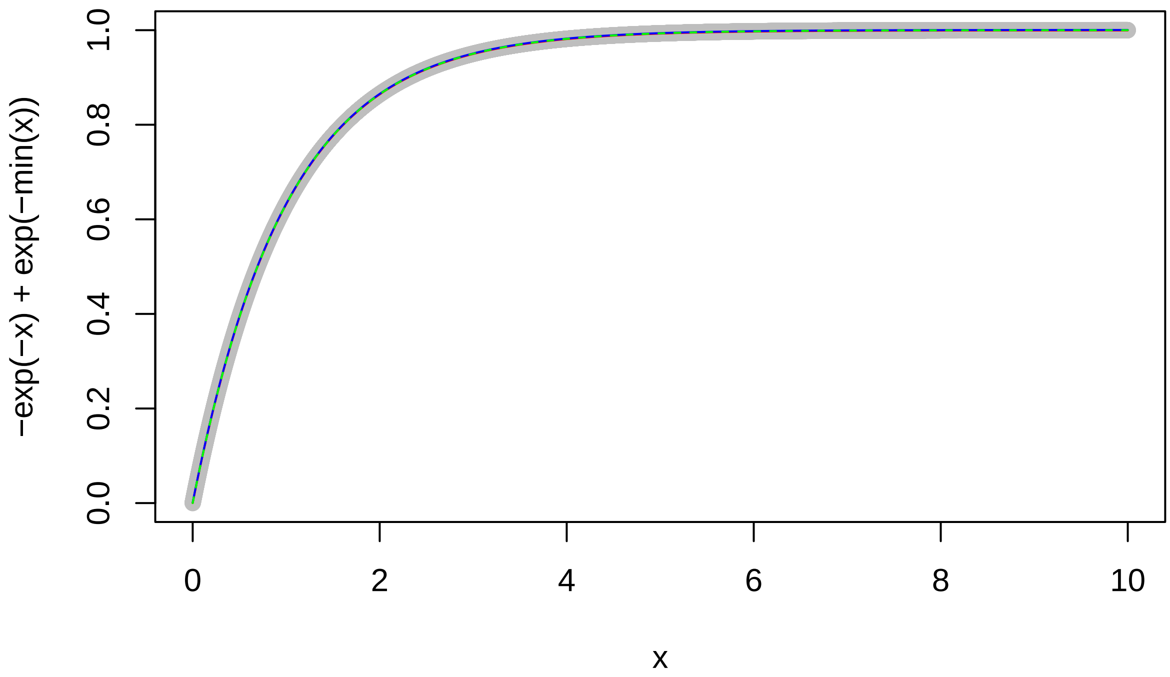
```r
fintegrate(x, y, 'right', value = 'total')
```

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

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

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

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



Note that data need not be sorted by x.

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

```
fintegrate(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)
```

```
fintegrate(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
```