jumbled demonstrations

Sasha D. Hafner

18 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:

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

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

To include sd and n, use `aggregate3`:

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

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

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

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

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

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

dfcombos

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

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

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

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

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

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

dfsumm

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

```
dfsumm(attenu)
```

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

Compare to `summary`.

```
summary(attenu)
```

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

interpm

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

```
args(interpm)
```

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

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

dat
```

```
##    time           a           b           c
## 1     1 -1.25711093 -1.34752527 -0.78104469
## 2     2  0.26033198  1.22572846 -0.56483413
## 3     3 -0.03851662 -2.13396273  0.92195052
## 4     4  0.90802529 -0.89238110 -0.83751348
## 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 -0.26741969 -1.08083952  0.03766624
## 12   12  0.11207296 -0.33852791  0.22233735
## 13   13 -0.38218873  0.38392425 -0.43612520
## 14   14  1.30175471  1.02582301 -1.29361059
## 15   15  0.34288565 -0.37910652  0.69191710
## 16   16 -0.86474372  2.37943961  0.78975253
## 17   17  0.84663077  1.03572380 -1.27168223
## 18   18 -0.37225376 -1.45899664  0.85448350
## 19   19 -1.27620998  0.98871827 -0.42596555
## 20   20          NA -1.53710074 -0.33221366
## 21   21          NA -0.06256062 -0.53817742
## 22   22          NA  1.55889852  0.23122631
## 23   23  0.89101154  0.01260965 -0.53130269
## 24   24 -0.04019999  0.92404047 -0.34017356
## 25   25 -0.81607862 -0.94117159  0.91018851
## 26   26  0.75423209  1.90295897 -1.11864883
## 27   27 -0.14147344 -0.31502092  0.81692305
## 28   28  0.23835815 -1.85827161  0.26968847
## 29   29  0.87960727  0.83424978 -0.66968974
## 30   30  0.01296577 -0.12071613 -0.80968421
```

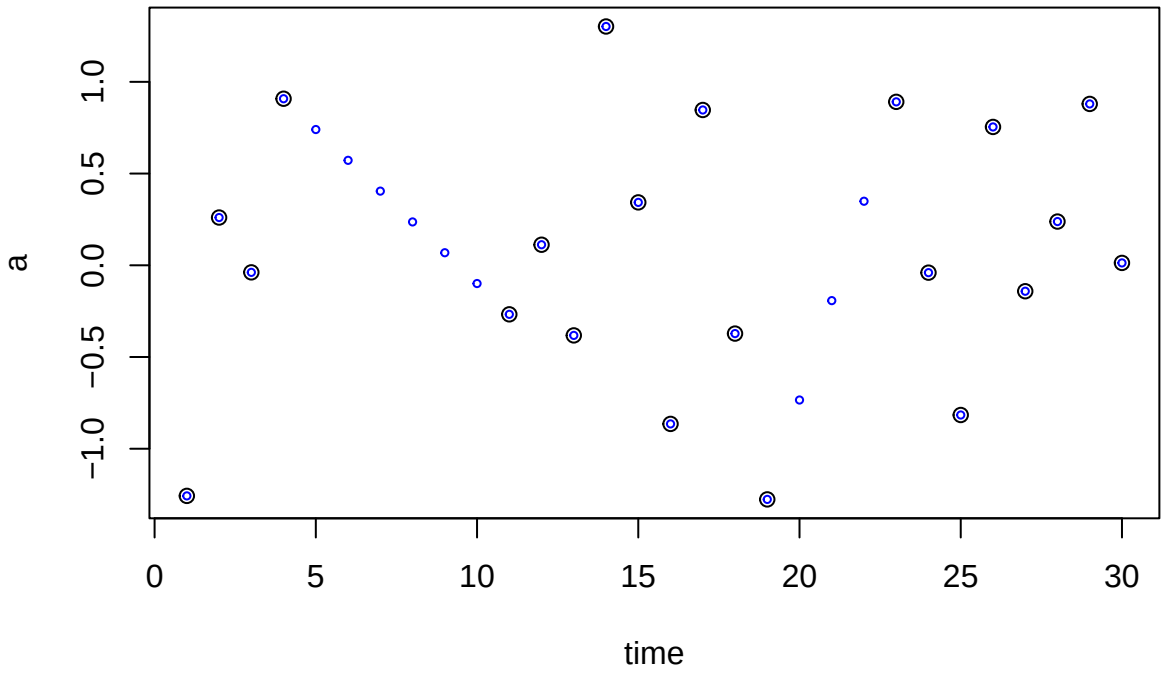
```
dat2 <- interpm(dat, 'time', c('a', 'b', 'c'))

dat2
```

```
##    time           a           b           c
## 1     1 -1.25711093 -1.34752527 -0.78104469
## 2     2  0.26033198  1.22572846 -0.56483413
## 3     3 -0.03851662 -2.13396273  0.92195052
## 4     4  0.90802529 -0.89238110 -0.83751348
## 5     5  0.74010458 -0.91930373 -0.71248780
## 6     6  0.57218386 -0.94622636 -0.58746213
## 7     7  0.40426315 -0.97314899 -0.46243646
## 8     8  0.23634244 -1.00007163 -0.33741078
## 9     9  0.06842173 -1.02699426 -0.21238511
## 10   10 -0.09949898 -1.05391689 -0.08735944
## 11   11 -0.26741969 -1.08083952  0.03766624
## 12   12  0.11207296 -0.33852791  0.22233735
## 13   13 -0.38218873  0.38392425 -0.43612520
## 14   14  1.30175471  1.02582301 -1.29361059
## 15   15  0.34288565 -0.37910652  0.69191710
## 16   16 -0.86474372  2.37943961  0.78975253
## 17   17  0.84663077  1.03572380 -1.27168223
## 18   18 -0.37225376 -1.45899664  0.85448350
```

```
## 19   19 -1.27620998  0.98871827 -0.42596555
## 20   20 -0.73440460 -1.53710074 -0.33221366
## 21   21 -0.19259922 -0.06256062 -0.53817742
## 22   22  0.34920616  1.55889852  0.23122631
## 23   23  0.89101154  0.01260965 -0.53130269
## 24   24 -0.04019999  0.92404047 -0.34017356
## 25   25 -0.81607862 -0.94117159  0.91018851
## 26   26  0.75423209  1.90295897 -1.11864883
## 27   27 -0.14147344 -0.31502092  0.81692305
## 28   28  0.23835815 -1.85827161  0.26968847
## 29   29  0.87960727  0.83424978 -0.66968974
## 30   30  0.01296577 -0.12071613 -0.80968421
```

```
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 -1.7673566 -1.13155748 -0.9616009
## 2     2     a -0.5928466  0.08939247  1.1955004
## 3     3     a  0.2340263 -0.06156184  0.5146983
## 4     4     a -0.3674743  0.69780691 -1.1727698
## 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  0.5405469  1.65035192 -1.0936131
## 11    1     b  0.5917373 -0.21568442  0.6152655
## 12    2     b  0.6981890 -0.77017927 -0.6734280
## 13    3     b  0.3009973 -0.49973001  0.9553970
## 14    4     b -0.1859507  0.76995754  0.5807187
## 15    5     b  0.2110899 -0.62237216  0.4597081
## 16    6     b -1.3400247  0.60466794  0.9165368
## 17    7     b -1.3000646 -0.64265007 -1.2312324
## 18    8     b -0.7386242 -0.52487221 -0.1322889
## 19    9     b -0.7926353 -0.77813125  1.5027159
## 20   10     b         NA  1.23366045  0.9809911
## 21    1     c -1.2667770 -1.22095074 -1.1567717
## 22    2     c         NA -1.03049944  0.3252181
## 23    3     c  0.8745463  0.63574102 -0.1189843
## 24    4     c -0.9715340 -0.01966468  0.1031091
## 25    5     c  1.0977061  1.00575901  0.2227572
## 26    6     c -2.1088220 -0.58850202  0.6172786
## 27    7     c  1.5967571 -0.24410175 -0.3077223
## 28    8     c  0.5693480 -0.85896061  1.3095515
## 29    9     c -1.2833122 -1.35098881  0.2331185
## 30   10     c -0.1382834  0.60622423 -0.2979842
```

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

```
##    time group          a          b          c
## 1     1     a -1.76735663 -1.13155748 -0.9616009
## 2     2     a -0.59284658  0.08939247  1.1955004
## 3     3     a  0.23402633 -0.06156184  0.5146983
## 4     4     a -0.36747431  0.69780691 -1.1727698
## 5     5     a -0.21613744  0.85656441 -1.1595770
## 6     6     a -0.06480057  1.01532191 -1.1463843
## 7     7     a  0.08653631  1.17407942 -1.1331915
## 8     8     a  0.23787318  1.33283692 -1.1199987
## 9     9     a  0.38921005  1.49159442 -1.1068059
## 10   10     a  0.54054692  1.65035192 -1.0936131
## 11    1     b  0.59173732 -0.21568442  0.6152655
## 12    2     b  0.69818903 -0.77017927 -0.6734280
## 13    3     b  0.30099725 -0.49973001  0.9553970
## 14    4     b -0.18595069  0.76995754  0.5807187
## 15    5     b  0.21108985 -0.62237216  0.4597081
## 16    6     b -1.34002472  0.60466794  0.9165368
## 17    7     b -1.30006460 -0.64265007 -1.2312324
## 18    8     b -0.73862417 -0.52487221 -0.1322889
## 19    9     b -0.79263526 -0.77813125  1.5027159
## 20   10     b          NA  1.23366045  0.9809911
## 21    1     c -1.26677695 -1.22095074 -1.1567717
## 22    2     c -0.19611531 -1.03049944  0.3252181
## 23    3     c  0.87454633  0.63574102 -0.1189843
## 24    4     c -0.97153398 -0.01966468  0.1031091
## 25    5     c  1.09770606  1.00575901  0.2227572
## 26    6     c -2.10882201 -0.58850202  0.6172786
## 27    7     c  1.59675706 -0.24410175 -0.3077223
```

```
## 28    8     c  0.56934798 -0.85896061  1.3095515
## 29    9     c -1.28331224 -1.35098881  0.2331185
## 30   10     c -0.13828335  0.60622423 -0.2979842
```

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

```
##    time group          a          b          c
## 1     1     a -1.76735663 -1.13155748 -0.9616009
## 2     2     a -0.59284658  0.08939247  1.1955004
## 3     3     a  0.23402633 -0.06156184  0.5146983
## 4     4     a -0.36747431  0.69780691 -1.1727698
## 5     5     a -0.21613744  0.85656441 -1.1595770
## 6     6     a -0.06480057  1.01532191 -1.1463843
## 7     7     a  0.08653631  1.17407942 -1.1331915
## 8     8     a  0.23787318  1.33283692 -1.1199987
## 9     9     a  0.38921005  1.49159442 -1.1068059
## 10   10     a  0.54054692  1.65035192 -1.0936131
## 11    1     b  0.59173732 -0.21568442  0.6152655
## 12    2     b  0.69818903 -0.77017927 -0.6734280
## 13    3     b  0.30099725 -0.49973001  0.9553970
## 14    4     b -0.18595069  0.76995754  0.5807187
## 15    5     b  0.21108985 -0.62237216  0.4597081
## 16    6     b -1.34002472  0.60466794  0.9165368
## 17    7     b -1.30006460 -0.64265007 -1.2312324
## 18    8     b -0.73862417 -0.52487221 -0.1322889
## 19    9     b -0.79263526 -0.77813125  1.5027159
## 20   10     b -0.79263526  1.23366045  0.9809911
## 21    1     c -1.26677695 -1.22095074 -1.1567717
## 22    2     c -0.19611531 -1.03049944  0.3252181
## 23    3     c  0.87454633  0.63574102 -0.1189843
## 24    4     c -0.97153398 -0.01966468  0.1031091
## 25    5     c  1.09770606  1.00575901  0.2227572
## 26    6     c -2.10882201 -0.58850202  0.6172786
## 27    7     c  1.59675706 -0.24410175 -0.3077223
## 28    8     c  0.56934798 -0.85896061  1.3095515
## 29    9     c -1.28331224 -1.35098881  0.2331185
## 30   10     c -0.13828335  0.60622423 -0.2979842
```

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

```
##     time group          a          b          c
##  1:    1     a -1.7673566 -1.13155748 -0.9616009
##  2:    2     a -0.5928466  0.08939247  1.1955004
##  3:    3     a  0.2340263 -0.06156184  0.5146983
##  4:    4     a -0.3674743  0.69780691 -1.1727698
##  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  0.5405469  1.65035192 -1.0936131
## 11:    1     b  0.5917373 -0.21568442  0.6152655
## 12:    2     b  0.6981890 -0.77017927 -0.6734280
## 13:    3     b  0.3009973 -0.49973001  0.9553970
## 14:    4     b -0.1859507  0.76995754  0.5807187
```

```
## 15:    5     b  0.2110899 -0.62237216  0.4597081
## 16:    6     b -1.3400247  0.60466794  0.9165368
## 17:    7     b -1.3000646 -0.64265007 -1.2312324
## 18:    8     b -0.7386242 -0.52487221 -0.1322889
## 19:    9     b -0.7926353 -0.77813125  1.5027159
## 20:   10     b         NA  1.23366045  0.9809911
## 21:    1     c -1.2667770 -1.22095074 -1.1567717
## 22:    2     c         NA -1.03049944  0.3252181
## 23:    3     c  0.8745463  0.63574102 -0.1189843
## 24:    4     c -0.9715340 -0.01966468  0.1031091
## 25:    5     c  1.0977061  1.00575901  0.2227572
## 26:    6     c -2.1088220 -0.58850202  0.6172786
## 27:    7     c  1.5967571 -0.24410175 -0.3077223
## 28:    8     c  0.5693480 -0.85896061  1.3095515
## 29:    9     c -1.2833122 -1.35098881  0.2331185
## 30:   10     c -0.1382834  0.60622423 -0.2979842
##     time group          a           b          c
```

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

```
##     time group          a           b          c
##  1:    1     a -1.76735663 -1.13155748 -0.9616009
##  2:    2     a -0.59284658  0.08939247  1.1955004
##  3:    3     a  0.23402633 -0.06156184  0.5146983
##  4:    4     a -0.36747431  0.69780691 -1.1727698
##  5:    5     a -0.21613744  0.85656441 -1.1595770
##  6:    6     a -0.06480057  1.01532191 -1.1463843
##  7:    7     a  0.08653631  1.17407942 -1.1331915
##  8:    8     a  0.23787318  1.33283692 -1.1199987
##  9:    9     a  0.38921005  1.49159442 -1.1068059
## 10:   10     a  0.54054692  1.65035192 -1.0936131
## 11:    1     b  0.59173732 -0.21568442  0.6152655
## 12:    2     b  0.69818903 -0.77017927 -0.6734280
## 13:    3     b  0.30099725 -0.49973001  0.9553970
## 14:    4     b -0.18595069  0.76995754  0.5807187
## 15:    5     b  0.21108985 -0.62237216  0.4597081
## 16:    6     b -1.34002472  0.60466794  0.9165368
## 17:    7     b -1.30006460 -0.64265007 -1.2312324
## 18:    8     b -0.73862417 -0.52487221 -0.1322889
## 19:    9     b -0.79263526 -0.77813125  1.5027159
## 20:   10     b          NA  1.23366045  0.9809911
## 21:    1     c -1.26677695 -1.22095074 -1.1567717
## 22:    2     c -0.19611531 -1.03049944  0.3252181
## 23:    3     c  0.87454633  0.63574102 -0.1189843
## 24:    4     c -0.97153398 -0.01966468  0.1031091
## 25:    5     c  1.09770606  1.00575901  0.2227572
## 26:    6     c -2.10882201 -0.58850202  0.6172786
## 27:    7     c  1.59675706 -0.24410175 -0.3077223
## 28:    8     c  0.56934798 -0.85896061  1.3095515
## 29:    9     c -1.28331224 -1.35098881  0.2331185
## 30:   10     c -0.13828335  0.60622423 -0.2979842
##     time group          a           b          c
```

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

```
##     time group           a           b          c
##  1:    1     a -1.76735663 -1.13155748 -0.9616009
##  2:    2     a -0.59284658  0.08939247  1.1955004
##  3:    3     a  0.23402633 -0.06156184  0.5146983
##  4:    4     a -0.36747431  0.69780691 -1.1727698
##  5:    5     a -0.21613744  0.85656441 -1.1595770
##  6:    6     a -0.06480057  1.01532191 -1.1463843
##  7:    7     a  0.08653631  1.17407942 -1.1331915
##  8:    8     a  0.23787318  1.33283692 -1.1199987
##  9:    9     a  0.38921005  1.49159442 -1.1068059
## 10:   10     a  0.54054692  1.65035192 -1.0936131
## 11:    1     b  0.59173732 -0.21568442  0.6152655
## 12:    2     b  0.69818903 -0.77017927 -0.6734280
## 13:    3     b  0.30099725 -0.49973001  0.9553970
## 14:    4     b -0.18595069  0.76995754  0.5807187
## 15:    5     b  0.21108985 -0.62237216  0.4597081
## 16:    6     b -1.34002472  0.60466794  0.9165368
## 17:    7     b -1.30006460 -0.64265007 -1.2312324
## 18:    8     b -0.73862417 -0.52487221 -0.1322889
## 19:    9     b -0.79263526 -0.77813125  1.5027159
## 20:   10     b -0.79263526  1.23366045  0.9809911
## 21:    1     c -1.26677695 -1.22095074 -1.1567717
## 22:    2     c -0.19611531 -1.03049944  0.3252181
## 23:    3     c  0.87454633  0.63574102 -0.1189843
## 24:    4     c -0.97153398 -0.01966468  0.1031091
## 25:    5     c  1.09770606  1.00575901  0.2227572
## 26:    6     c -2.10882201 -0.58850202  0.6172786
## 27:    7     c  1.59675706 -0.24410175 -0.3077223
## 28:    8     c  0.56934798 -0.85896061  1.3095515
## 29:    9     c -1.28331224 -1.35098881  0.2331185
## 30:   10     c -0.13828335  0.60622423 -0.2979842
##     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  0.68133288 a
## 2   2  0.06902602 b
## 3   3  1.84371076 c
## 4   4 -0.97961204 d
## 5   5 -0.03407835 e
## 6   6 -0.43719954 f
## 7   7  1.10793722 g
## 8   8  0.23775640 h
## 9   9 -1.03145624 i
## 10 10 -1.97722700 j
```

```
rounddf(dat)
```

```
##     a     b c
## 1   1  0.68 a
## 2   2  0.07 b
## 3   3  1.84 c
## 4   4 -0.98 d
## 5   5 -0.03 e
## 6   6 -0.44 f
## 7   7  1.11 g
## 8   8  0.24 h
## 9   9 -1.03 i
## 10 10 -1.98 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  0.6813 a
## 2   2  0.0690 b
## 3   3  1.8437 c
## 4   4 -0.9796 d
## 5   5 -0.0341 e
## 6   6 -0.4372 f
## 7   7  1.1079 g
## 8   8  0.2378 h
## 9   9 -1.0315 i
## 10 10 -1.9772 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  0.68130 a
## 2   2  0.06903 b
## 3   3  1.84400 c
## 4   4 -0.97960 d
## 5   5 -0.03408 e
```

```
## 6   6 -0.43720 f
## 7   7  1.10800 g
## 8   8  0.23780 h
## 9   9 -1.03100 i
## 10 10 -1.97700 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  0.680 a
## 2   2  0.069 b
## 3   3  1.800 c
## 4   4 -0.980 d
## 5   5 -0.034 e
## 6   6 -0.440 f
## 7   7  1.100 g
## 8   8  0.240 h
## 9   9 -1.000 i
## 10 10 -2.000 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
```

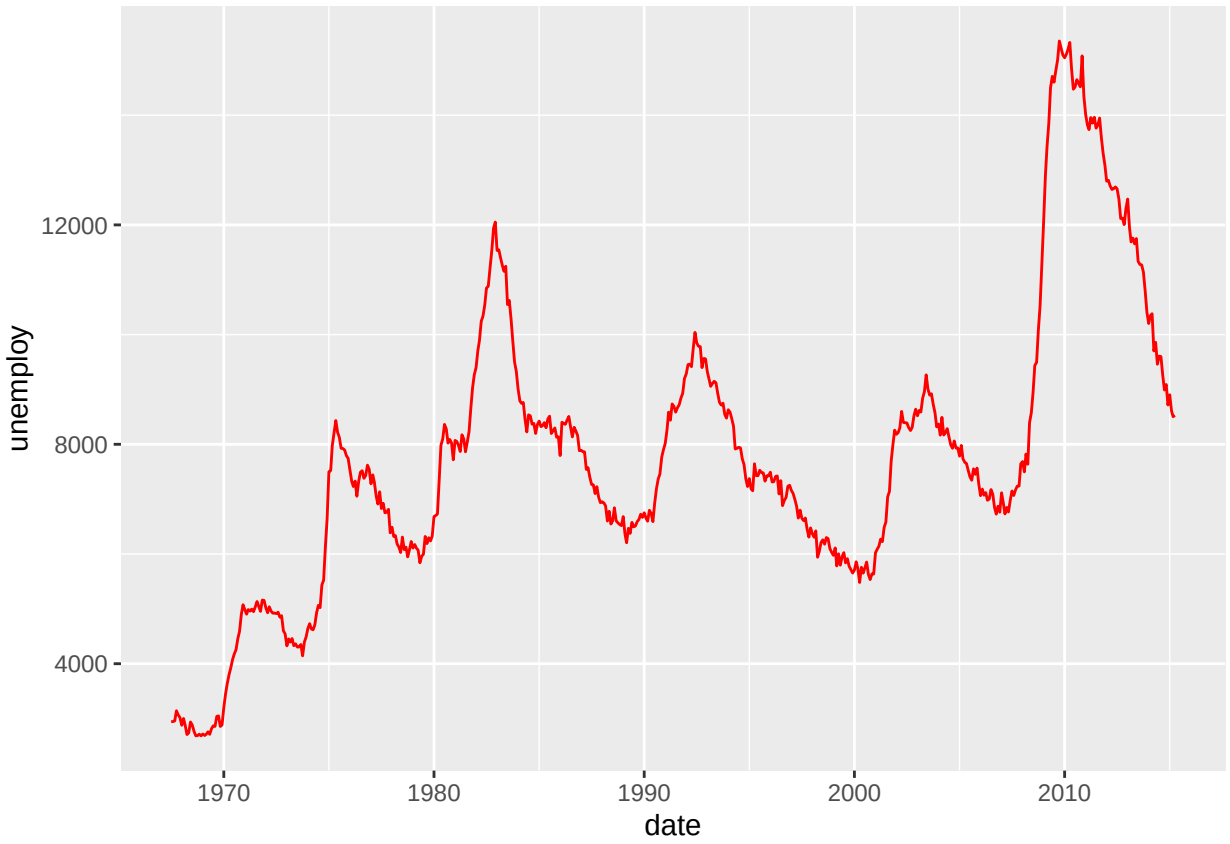
```
dat <- rounddf(dat, pad = TRUE)
summary(dat)
```

```
##        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.

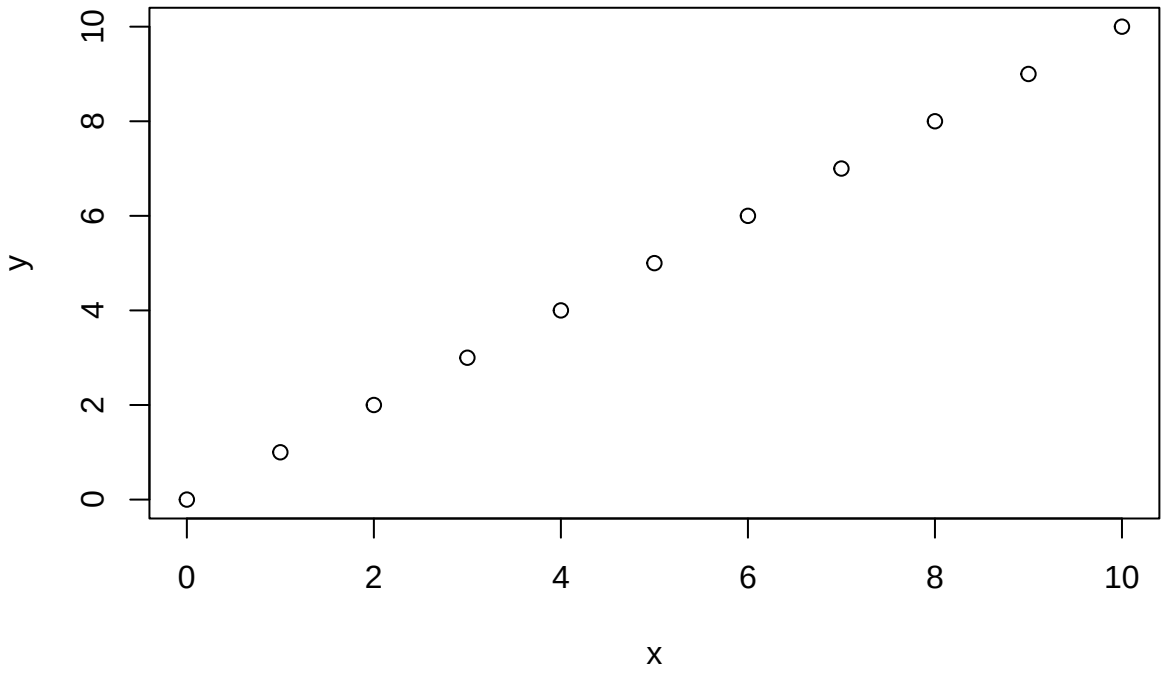
`fintegrate`

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  1  3  6 10 15 21 28 36 45 50
```

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

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

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

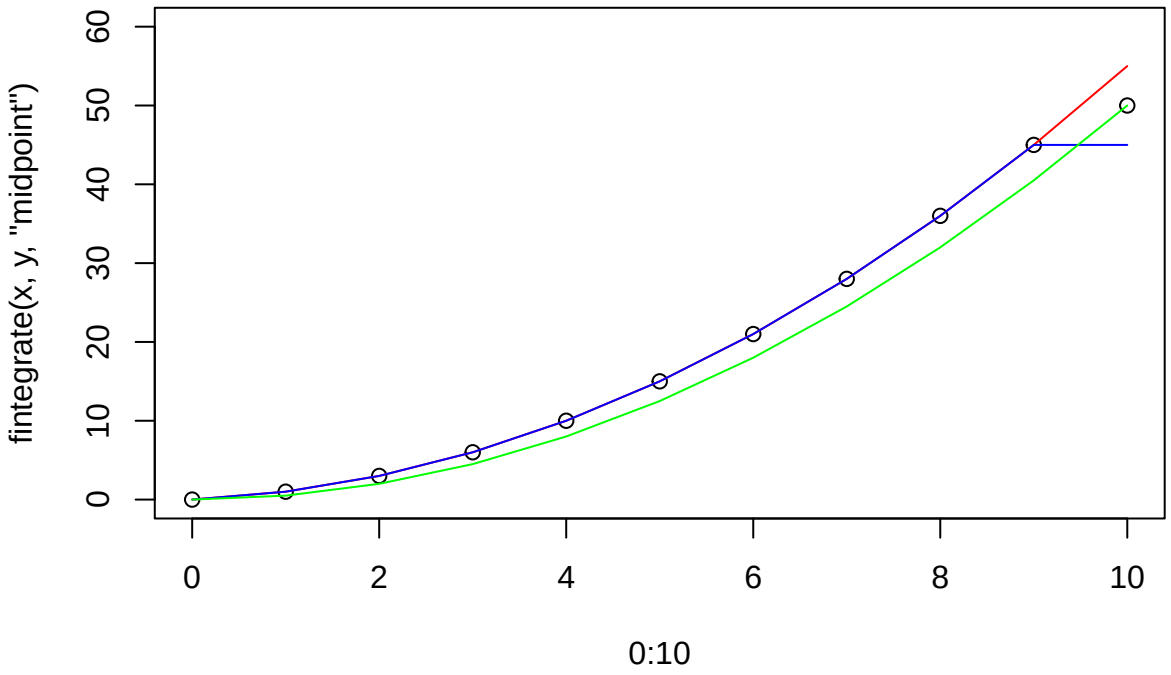
```
##  [1]  0  1  3  6 10 15 21 28 36 45 45
```

```
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
```

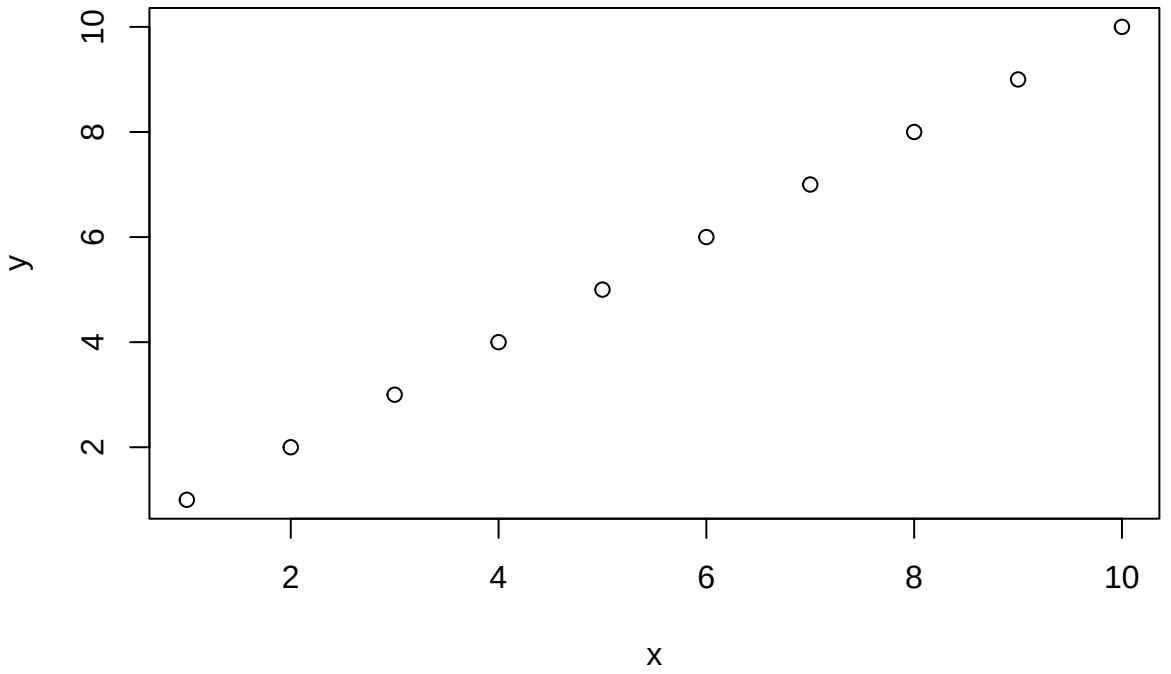
Note differences on the way up.

```
plot(0:10, fintegrate(x, y, 'midpoint'), ylim = c(0, 60))
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')
```



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]  0.5  2.5  5.5  9.5 14.5 20.5 27.5 35.5 44.5 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]  1  3  6 10 15 21 28 36 45 50
```

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

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

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

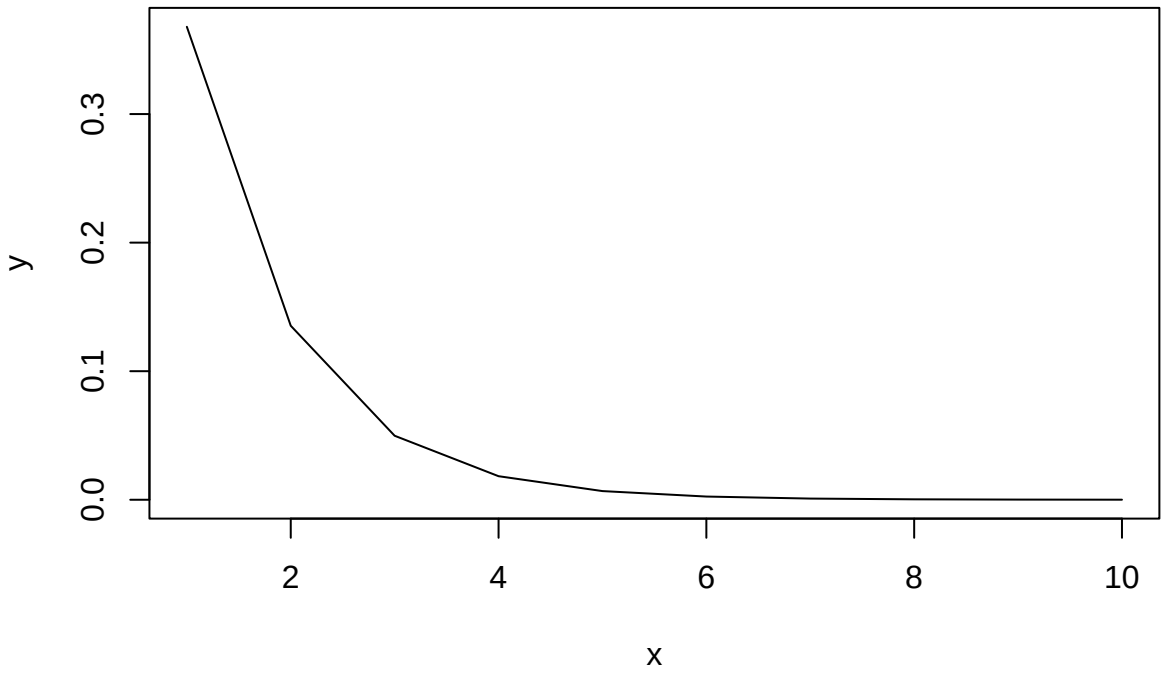
```
##  [1]  1  3  6 10 15 21 28 36 45 45
```

```
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
```

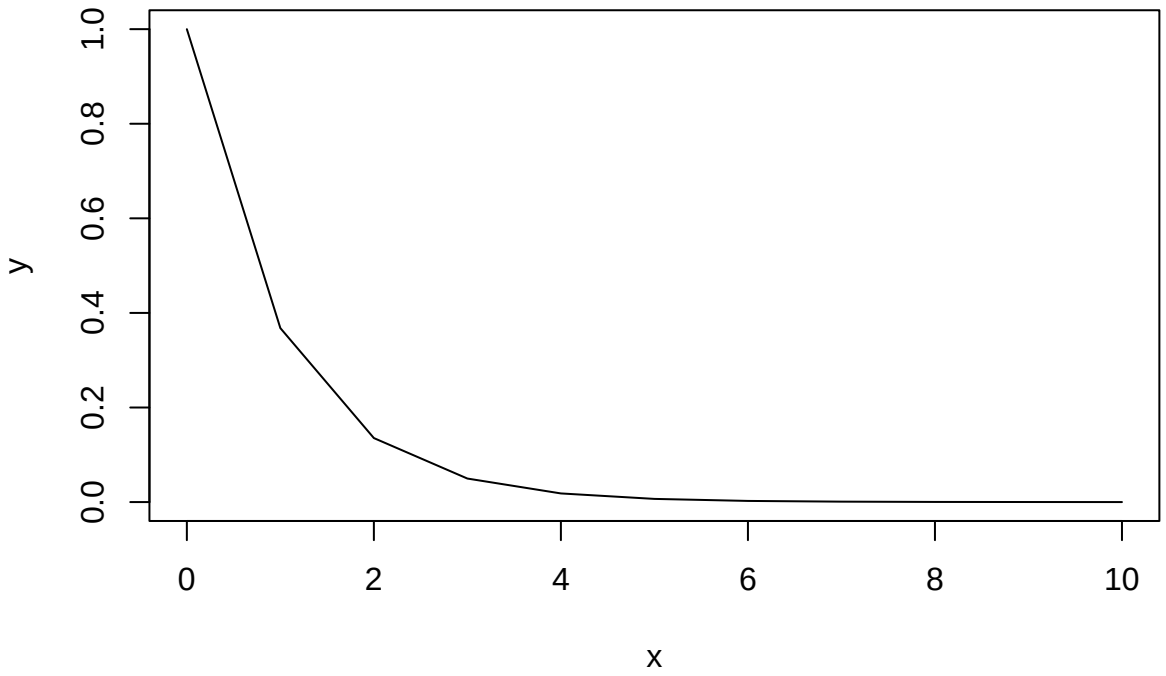
```
fintegrate(x, y, 'trap', value = 'total')
```

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

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')
```

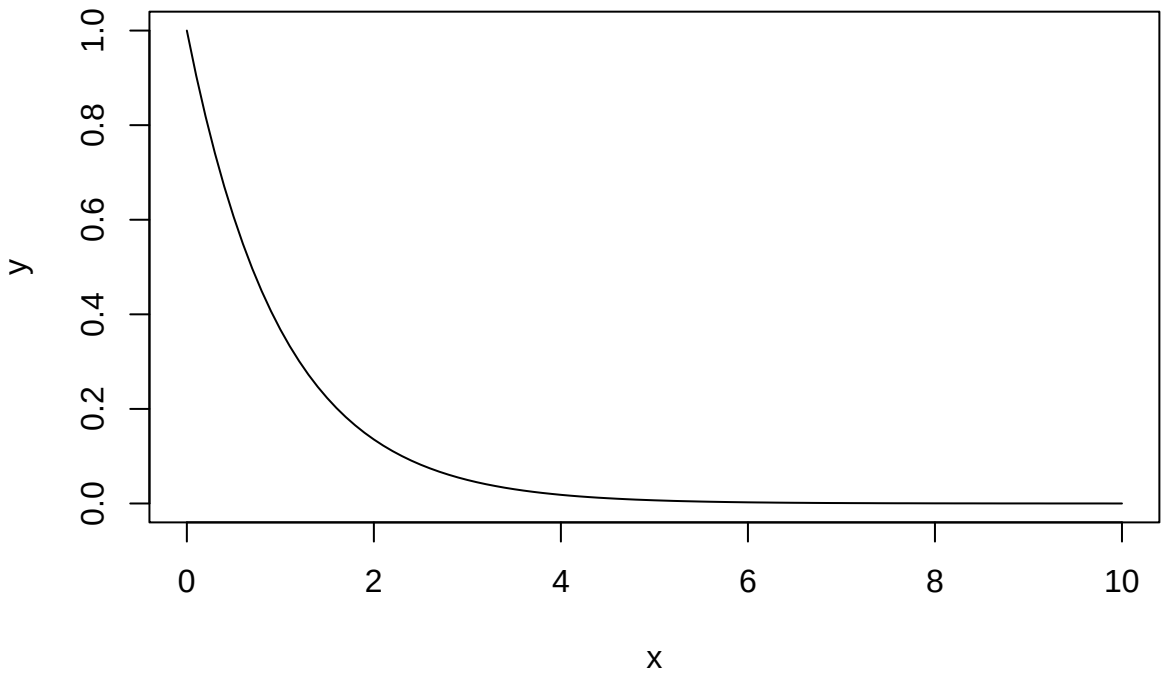
```
## [1] 1.581905
```

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

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

Prove that all methods become accurate with very high resolution.

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




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

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

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

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

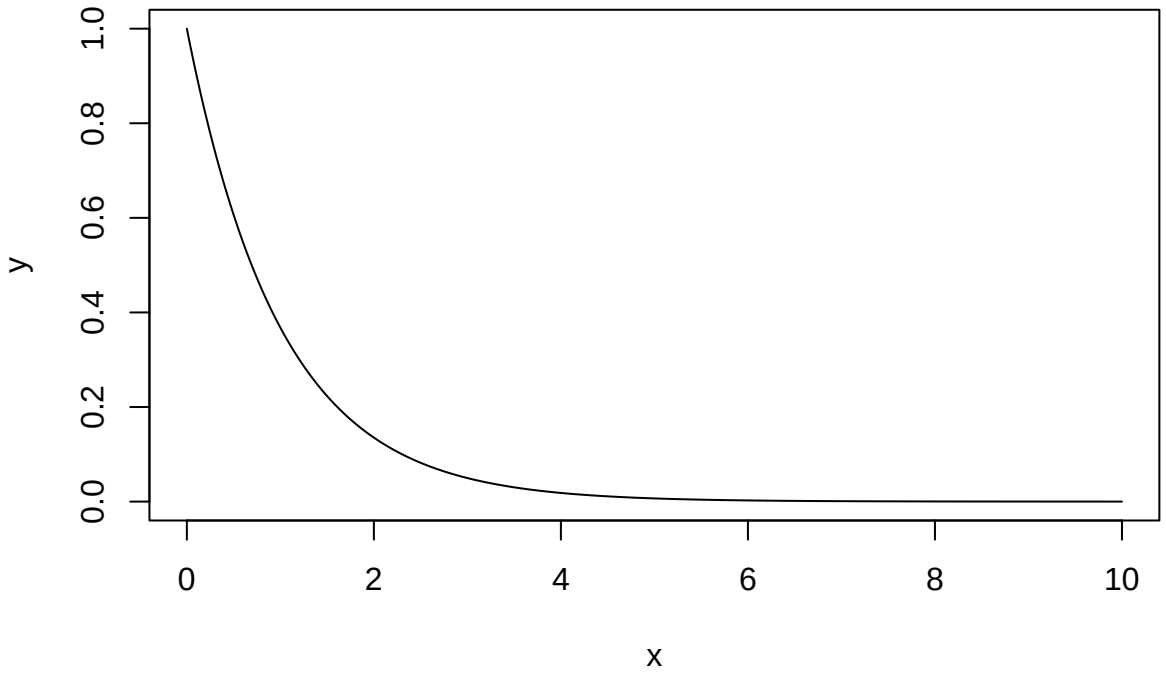
```
fintegrate(x, y, 'right', value = 'total')
```

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

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

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

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




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

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

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

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

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

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

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

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

Note that data need not be sorted by x.

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

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

```
##  [1] 0.5000000 0.8678794 1.0032147 1.0530018 1.0713174 1.0780554 1.0805341
##  [8] 1.0814460 1.0817815 1.0819049 1.0819276
```

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

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

```
##  [1] 1.0713174 0.8678794 1.0032147 1.0530018 0.5000000 1.0780554 1.0805341
##  [8] 1.0814460 1.0817815 1.0819049 1.0819276
```