

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:

```
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 properly. 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.

```
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 (excl. 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
##
```

interp

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

```
args(interp)
```

```
## 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	0.31833673	-0.26774095	-0.46288318
## 2	2	-1.42379885	1.58585916	-0.88455498
## 3	3	-0.40509086	0.04690059	-1.63092757
## 4	4	0.99538657	0.35649678	0.56223059
## 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	-1.10363778	-0.46205239	-0.09514776
## 12	12	0.44418506	-0.22509928	1.13878050
## 13	13	-0.20495061	-0.84644780	0.50231463
## 14	14	1.67563243	0.07304632	-0.51541405
## 15	15	-0.13132225	-0.27503642	-2.46839047
## 16	16	-0.19988298	-0.38642636	-0.87255274
## 17	17	0.05491242	-0.04620314	0.96408808
## 18	18	-0.68216549	-0.82589372	0.91079625
## 19	19	-0.72770415	-0.85403424	1.92580884
## 20	20	NA	0.11873681	-0.30290695
## 21	21	NA	0.28359691	-1.05470705
## 22	22	NA	1.93008647	0.41811609
## 23	23	0.17716660	-1.14052762	0.70127282
## 24	24	-0.01250080	-1.32211824	0.24675828
## 25	25	-0.39431713	1.22883161	0.46429516
## 26	26	0.35156293	-0.54845603	-0.39546819
## 27	27	0.87876756	-0.12600749	0.71307031
## 28	28	0.20465408	0.68771872	1.18501256
## 29	29	-0.88738071	0.70520038	-1.91114929
## 30	30	-0.47721606	0.80147843	1.11493056

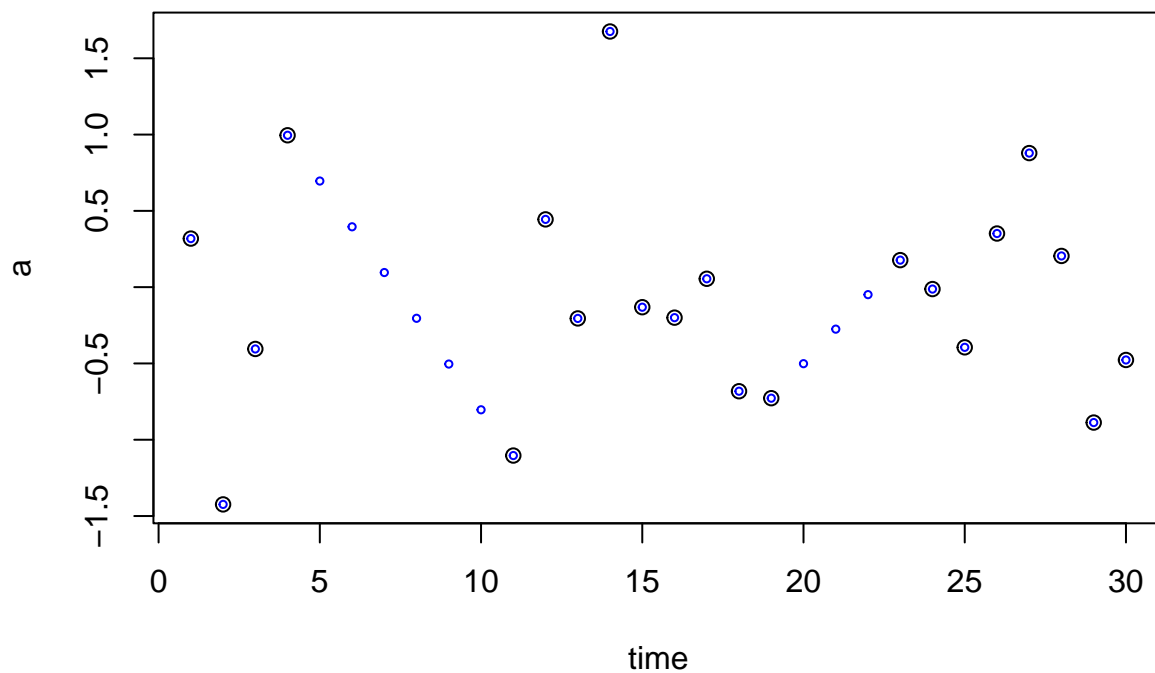
```
dat2 <- interp(dat, 'time', c('a', 'b', 'c'))
```

```
dat2
```

##	time	a	b	c
## 1	1	0.31833673	-0.267740953	-0.46288318
## 2	2	-1.42379885	1.585859163	-0.88455498
## 3	3	-0.40509086	0.046900595	-1.63092757
## 4	4	0.99538657	0.356496777	0.56223059
## 5	5	0.69552594	0.239561182	0.46831939
## 6	6	0.39566532	0.122625586	0.37440820
## 7	7	0.09580470	0.005689991	0.28049701
## 8	8	-0.20405592	-0.111245605	0.18658582
## 9	9	-0.50391654	-0.228181200	0.09267462
## 10	10	-0.80377716	-0.345116796	-0.00123657
## 11	11	-1.10363778	-0.462052391	-0.09514776
## 12	12	0.44418506	-0.225099283	1.13878050
## 13	13	-0.20495061	-0.846447800	0.50231463
## 14	14	1.67563243	0.073046319	-0.51541405
## 15	15	-0.13132225	-0.275036420	-2.46839047
## 16	16	-0.19988298	-0.386426357	-0.87255274
## 17	17	0.05491242	-0.046203141	0.96408808
## 18	18	-0.68216549	-0.825893722	0.91079625

```
## 19 19 -0.72770415 -0.854034238 1.92580884
## 20 20 -0.50148646 0.118736809 -0.30290695
## 21 21 -0.27526878 0.283596906 -1.05470705
## 22 22 -0.04905109 1.930086466 0.41811609
## 23 23 0.17716660 -1.140527618 0.70127282
## 24 24 -0.01250080 -1.322118243 0.24675828
## 25 25 -0.39431713 1.228831605 0.46429516
## 26 26 0.35156293 -0.548456032 -0.39546819
## 27 27 0.87876756 -0.126007492 0.71307031
## 28 28 0.20465408 0.687718722 1.18501256
## 29 29 -0.88738071 0.705200376 -1.91114929
## 30 30 -0.47721606 0.801478429 1.11493056
```

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



Now works 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 = rnorm(30), c = rnorm(30))
dat[5:9, -1:-2] <- NA
dat[c(20, 22), 'a'] <- NA
```

```
dat
```

##	time	group	a	b	c
## 1	1	a	-0.46930021	1.3491063	-0.49751241
## 2	2	a	-1.14800275	-1.6350828	0.76558578
## 3	3	a	1.10399796	-0.7155327	0.50139307
## 4	4	a	-0.28924992	-0.9164943	0.33191117
## 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 -0.12892591  0.4598761  0.62192056
## 11      1     b  0.29911231  0.1677058 -1.47180517
## 12      2     b  0.01829921 -0.2750030  0.79819546
## 13      3     b -1.15006133  0.4923226  0.27946251
## 14      4     b -0.42939635  1.3397546  0.65787426
## 15      5     b  0.85000298 -0.4767442  0.59372632
## 16      6     b -0.21482949 -1.4967955 -0.57792651
## 17      7     b -0.61741402 -1.2014240  0.47606273
## 18      8     b  0.10942738 -0.4074610 -1.18763362
## 19      9     b -0.70651106 -0.9477481  0.71768805
## 20     10     b      NA  0.8516664  0.13469724
## 21      1     c  0.35080172 -0.5884310  1.42208034
## 22      2     c      NA -1.3624731  0.04073091
## 23      3     c -0.03337353 -0.4496379 -0.33840975
## 24      4     c  0.36144400 -0.6433250 -0.55811539
## 25      5     c  1.04557564 -0.5107456  1.02781348
## 26      6     c  0.10321239  0.2572020 -0.22547048
## 27      7     c  0.33179556 -0.4195554 -1.23163005
## 28      8     c -1.56855881 -0.9522277 -0.75824474
## 29      9     c  0.44026994  0.5123934  1.08551815
## 30     10     c  1.88187132 -1.7442677 -1.04909888
```

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

```
##      time group      a      b      c
## 1      1      a -0.46930021  1.34910627 -0.49751241
## 2      2      a -1.14800275 -1.63508277  0.76558578
## 3      3      a  1.10399796 -0.71553273  0.50139307
## 4      4      a -0.28924992 -0.91649432  0.33191117
## 5      5      a -0.26252925 -0.68709924  0.38024607
## 6      6      a -0.23580858 -0.45770417  0.42858097
## 7      7      a -0.20908791 -0.22830909  0.47691586
## 8      8      a -0.18236724  0.00108599  0.52525076
## 9      9      a -0.15564657  0.23048107  0.57358566
## 10     10     a -0.12892591  0.45987615  0.62192056
## 11      1     b  0.29911231  0.16770583 -1.47180517
## 12      2     b  0.01829921 -0.27500302  0.79819546
## 13      3     b -1.15006133  0.49232262  0.27946251
## 14      4     b -0.42939635  1.33975457  0.65787426
## 15      5     b  0.85000298 -0.47674422  0.59372632
## 16      6     b -0.21482949 -1.49679554 -0.57792651
## 17      7     b -0.61741402 -1.20142397  0.47606273
## 18      8     b  0.10942738 -0.40746103 -1.18763362
## 19      9     b -0.70651106 -0.94774810  0.71768805
## 20     10     b      NA  0.85166640  0.13469724
## 21      1     c  0.35080172 -0.58843101  1.42208034
## 22      2     c  0.15871409 -1.36247310  0.04073091
## 23      3     c -0.03337353 -0.44963794 -0.33840975
## 24      4     c  0.36144400 -0.64332503 -0.55811539
## 25      5     c  1.04557564 -0.51074555  1.02781348
## 26      6     c  0.10321239  0.25720201 -0.22547048
## 27      7     c  0.33179556 -0.41955545 -1.23163005
```

```
## 28      8      c -1.56855881 -0.95222766 -0.75824474
## 29      9      c  0.44026994  0.51239341  1.08551815
## 30     10      c  1.88187132 -1.74426769 -1.04909888
```

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

```
##      time group      a      b      c
## 1      1      a -0.46930021  1.34910627 -0.49751241
## 2      2      a -1.14800275 -1.63508277  0.76558578
## 3      3      a  1.10399796 -0.71553273  0.50139307
## 4      4      a -0.28924992 -0.91649432  0.33191117
## 5      5      a -0.26252925 -0.68709924  0.38024607
## 6      6      a -0.23580858 -0.45770417  0.42858097
## 7      7      a -0.20908791 -0.22830909  0.47691586
## 8      8      a -0.18236724  0.00108599  0.52525076
## 9      9      a -0.15564657  0.23048107  0.57358566
## 10     10     a -0.12892591  0.45987615  0.62192056
## 11     1      b  0.29911231  0.16770583 -1.47180517
## 12     2      b  0.01829921 -0.27500302  0.79819546
## 13     3      b -1.15006133  0.49232262  0.27946251
## 14     4      b -0.42939635  1.33975457  0.65787426
## 15     5      b  0.85000298 -0.47674422  0.59372632
## 16     6      b -0.21482949 -1.49679554 -0.57792651
## 17     7      b -0.61741402 -1.20142397  0.47606273
## 18     8      b  0.10942738 -0.40746103 -1.18763362
## 19     9      b -0.70651106 -0.94774810  0.71768805
## 20     10     b -0.70651106  0.85166640  0.13469724
## 21     1      c  0.35080172 -0.58843101  1.42208034
## 22     2      c  0.15871409 -1.36247310  0.04073091
## 23     3      c -0.03337353 -0.44963794 -0.33840975
## 24     4      c  0.36144400 -0.64332503 -0.55811539
## 25     5      c  1.04557564 -0.51074555  1.02781348
## 26     6      c  0.10321239  0.25720201 -0.22547048
## 27     7      c  0.33179556 -0.41955545 -1.23163005
## 28     8      c -1.56855881 -0.95222766 -0.75824474
## 29     9      c  0.44026994  0.51239341  1.08551815
## 30     10     c  1.88187132 -1.74426769 -1.04909888
```

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

```
##      time group      a      b      c
## 1:      1      a -0.46930021  1.3491063 -0.49751241
## 2:      2      a -1.14800275 -1.6350828  0.76558578
## 3:      3      a  1.10399796 -0.7155327  0.50139307
## 4:      4      a -0.28924992 -0.9164943  0.33191117
## 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 -0.12892591  0.4598761  0.62192056
## 11:      1      b  0.29911231  0.1677058 -1.47180517
## 12:      2      b  0.01829921 -0.2750030  0.79819546
## 13:      3      b -1.15006133  0.4923226  0.27946251
## 14:      4      b -0.42939635  1.3397546  0.65787426
```



```
## 15: 5 b 0.85000298 -0.4767442 0.59372632
## 16: 6 b -0.21482949 -1.4967955 -0.57792651
## 17: 7 b -0.61741402 -1.2014240 0.47606273
## 18: 8 b 0.10942738 -0.4074610 -1.18763362
## 19: 9 b -0.70651106 -0.9477481 0.71768805
## 20: 10 b NA 0.8516664 0.13469724
## 21: 1 c 0.35080172 -0.5884310 1.42208034
## 22: 2 c NA -1.3624731 0.04073091
## 23: 3 c -0.03337353 -0.4496379 -0.33840975
## 24: 4 c 0.36144400 -0.6433250 -0.55811539
## 25: 5 c 1.04557564 -0.5107456 1.02781348
## 26: 6 c 0.10321239 0.2572020 -0.22547048
## 27: 7 c 0.33179556 -0.4195554 -1.23163005
## 28: 8 c -1.56855881 -0.9522277 -0.75824474
## 29: 9 c 0.44026994 0.5123934 1.08551815
## 30: 10 c 1.88187132 -1.7442677 -1.04909888
## time group a b c
```

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

```
## time group a b c
## 1: 1 a -0.46930021 1.34910627 -0.49751241
## 2: 2 a -1.14800275 -1.63508277 0.76558578
## 3: 3 a 1.10399796 -0.71553273 0.50139307
## 4: 4 a -0.28924992 -0.91649432 0.33191117
## 5: 5 a -0.26252925 -0.68709924 0.38024607
## 6: 6 a -0.23580858 -0.45770417 0.42858097
## 7: 7 a -0.20908791 -0.22830909 0.47691586
## 8: 8 a -0.18236724 0.00108599 0.52525076
## 9: 9 a -0.15564657 0.23048107 0.57358566
## 10: 10 a -0.12892591 0.45987615 0.62192056
## 11: 1 b 0.29911231 0.16770583 -1.47180517
## 12: 2 b 0.01829921 -0.27500302 0.79819546
## 13: 3 b -1.15006133 0.49232262 0.27946251
## 14: 4 b -0.42939635 1.33975457 0.65787426
## 15: 5 b 0.85000298 -0.47674422 0.59372632
## 16: 6 b -0.21482949 -1.49679554 -0.57792651
## 17: 7 b -0.61741402 -1.20142397 0.47606273
## 18: 8 b 0.10942738 -0.40746103 -1.18763362
## 19: 9 b -0.70651106 -0.94774810 0.71768805
## 20: 10 b NA 0.85166640 0.13469724
## 21: 1 c 0.35080172 -0.58843101 1.42208034
## 22: 2 c 0.15871409 -1.36247310 0.04073091
## 23: 3 c -0.03337353 -0.44963794 -0.33840975
## 24: 4 c 0.36144400 -0.64332503 -0.55811539
## 25: 5 c 1.04557564 -0.51074555 1.02781348
## 26: 6 c 0.10321239 0.25720201 -0.22547048
## 27: 7 c 0.33179556 -0.41955545 -1.23163005
## 28: 8 c -1.56855881 -0.95222766 -0.75824474
## 29: 9 c 0.44026994 0.51239341 1.08551815
## 30: 10 c 1.88187132 -1.74426769 -1.04909888
## time group a b c
```

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

```
##      time group      a      b      c
##  1:     1     a -0.46930021  1.34910627 -0.49751241
##  2:     2     a -1.14800275 -1.63508277  0.76558578
##  3:     3     a  1.10399796 -0.71553273  0.50139307
##  4:     4     a -0.28924992 -0.91649432  0.33191117
##  5:     5     a -0.26252925 -0.68709924  0.38024607
##  6:     6     a -0.23580858 -0.45770417  0.42858097
##  7:     7     a -0.20908791 -0.22830909  0.47691586
##  8:     8     a -0.18236724  0.00108599  0.52525076
##  9:     9     a -0.15564657  0.23048107  0.57358566
## 10:    10     a -0.12892591  0.45987615  0.62192056
## 11:     1     b  0.29911231  0.16770583 -1.47180517
## 12:     2     b  0.01829921 -0.27500302  0.79819546
## 13:     3     b -1.15006133  0.49232262  0.27946251
## 14:     4     b -0.42939635  1.33975457  0.65787426
## 15:     5     b  0.85000298 -0.47674422  0.59372632
## 16:     6     b -0.21482949 -1.49679554 -0.57792651
## 17:     7     b -0.61741402 -1.20142397  0.47606273
## 18:     8     b  0.10942738 -0.40746103 -1.18763362
## 19:     9     b -0.70651106 -0.94774810  0.71768805
## 20:    10     b -0.70651106  0.85166640  0.13469724
## 21:     1     c  0.35080172 -0.58843101  1.42208034
## 22:     2     c  0.15871409 -1.36247310  0.04073091
## 23:     3     c -0.03337353 -0.44963794 -0.33840975
## 24:     4     c  0.36144400 -0.64332503 -0.55811539
## 25:     5     c  1.04557564 -0.51074555  1.02781348
## 26:     6     c  0.10321239  0.25720201 -0.22547048
## 27:     7     c  0.33179556 -0.41955545 -1.23163005
## 28:     8     c -1.56855881 -0.95222766 -0.75824474
## 29:     9     c  0.44026994  0.51239341  1.08551815
## 30:    10     c  1.88187132 -1.74426769 -1.04909888
##      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  0.29039955 a
## 2    2 -0.44757343 b
## 3    3 -0.21886077 c
## 4    4 -1.60544351 d
## 5    5  0.07483969 e
## 6    6 -0.52191870 f
## 7    7 -0.06490237 g
## 8    8  1.05136997 h
## 9    9 -1.74215826 i
## 10 10  1.73254803 j
```

```
rounddf(dat)
```

```
##      a      b c
## 1    1  0.29 a
## 2    2 -0.45 b
## 3    3 -0.22 c
## 4    4 -1.61 d
## 5    5  0.07 e
## 6    6 -0.52 f
## 7    7 -0.06 g
## 8    8  1.05 h
## 9    9 -1.74 i
## 10 10  1.73 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  0.2904 a
## 2    2 -0.4476 b
## 3    3 -0.2189 c
## 4    4 -1.6054 d
## 5    5  0.0748 e
## 6    6 -0.5219 f
## 7    7 -0.0649 g
## 8    8  1.0514 h
## 9    9 -1.7422 i
## 10 10  1.7325 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  0.29040 a
## 2    2 -0.44760 b
## 3    3 -0.21890 c
## 4    4 -1.60500 d
## 5    5  0.07484 e
```

```
## 6 6 -0.52190 f
## 7 7 -0.06490 g
## 8 8 1.05100 h
## 9 9 -1.74200 i
## 10 10 1.73300 j
```

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

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

```
##      a      b c
## 1 1 0.290 a
## 2 2 -0.450 b
## 3 3 -0.220 c
## 4 4 -1.600 d
## 5 5 0.075 e
## 6 6 -0.520 f
## 7 7 -0.065 g
## 8 8 1.100 h
## 9 9 -1.700 i
## 10 10 1.700 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
```

```
roundddf(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
```

```
roundddf(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
```

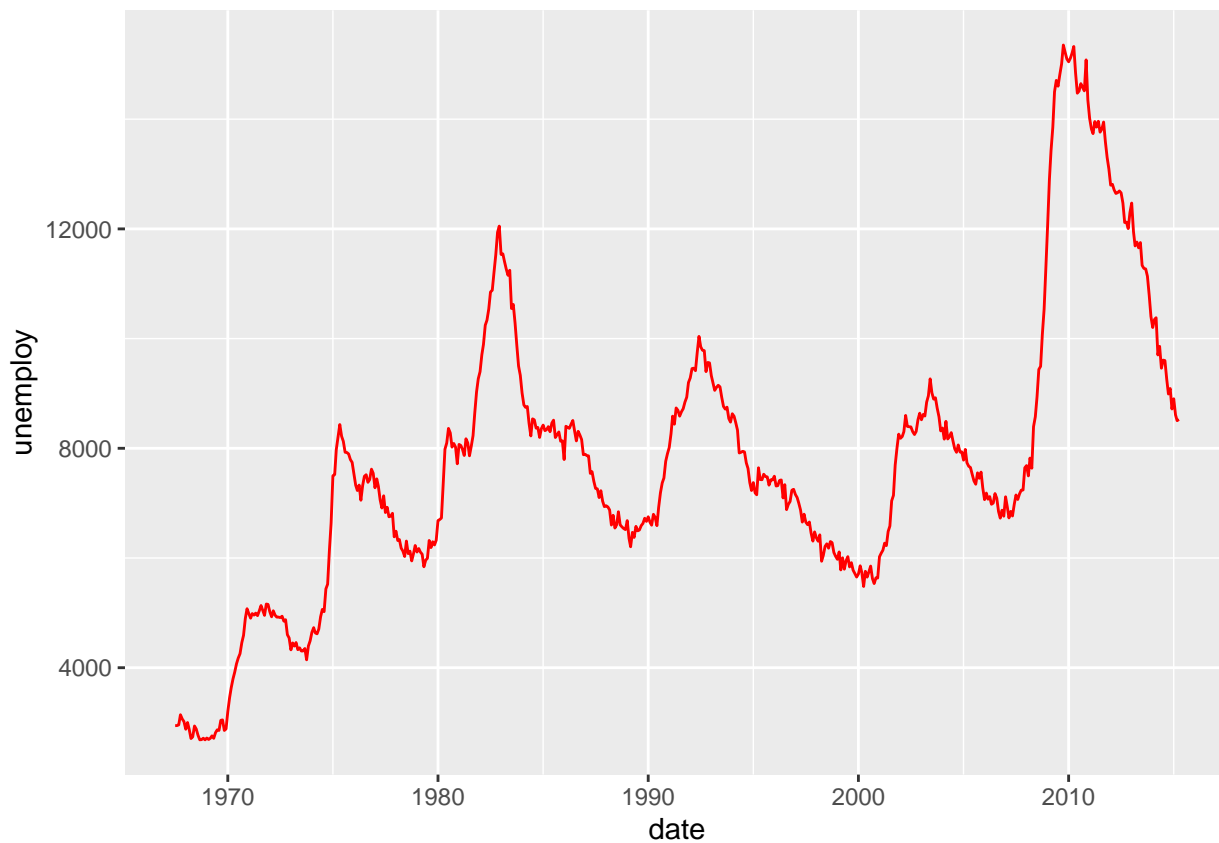
```
dat <- roundddf(dat, pad = TRUE)
summary(dat)
```

```
##      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.

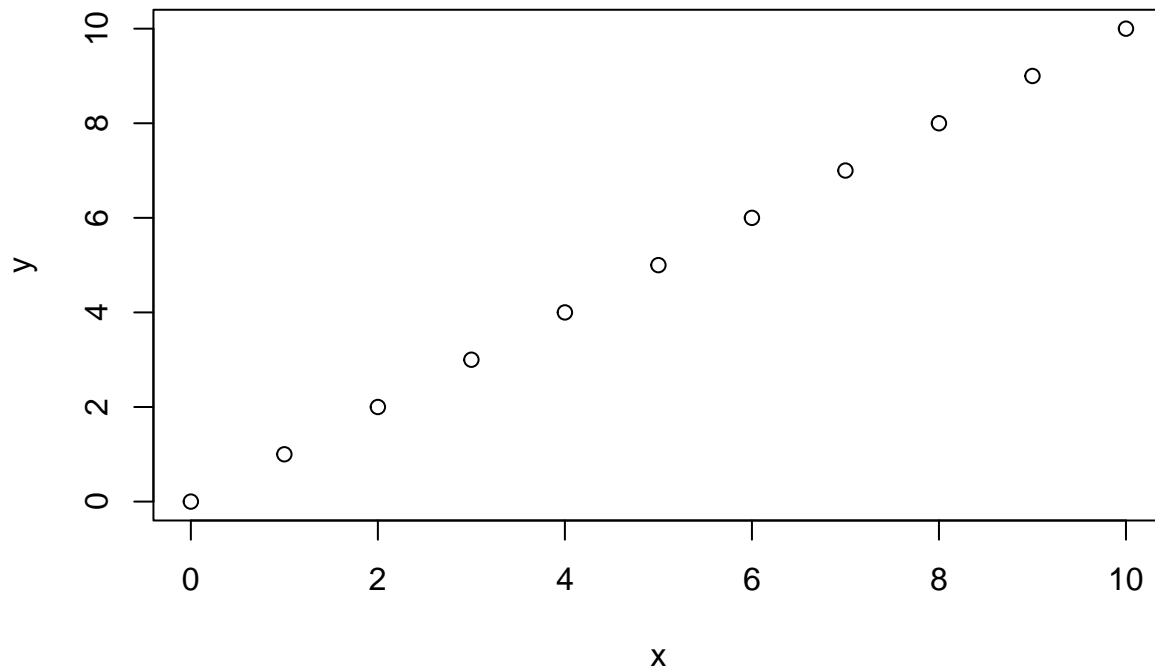
fintegrate

Integrate flux 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')
```

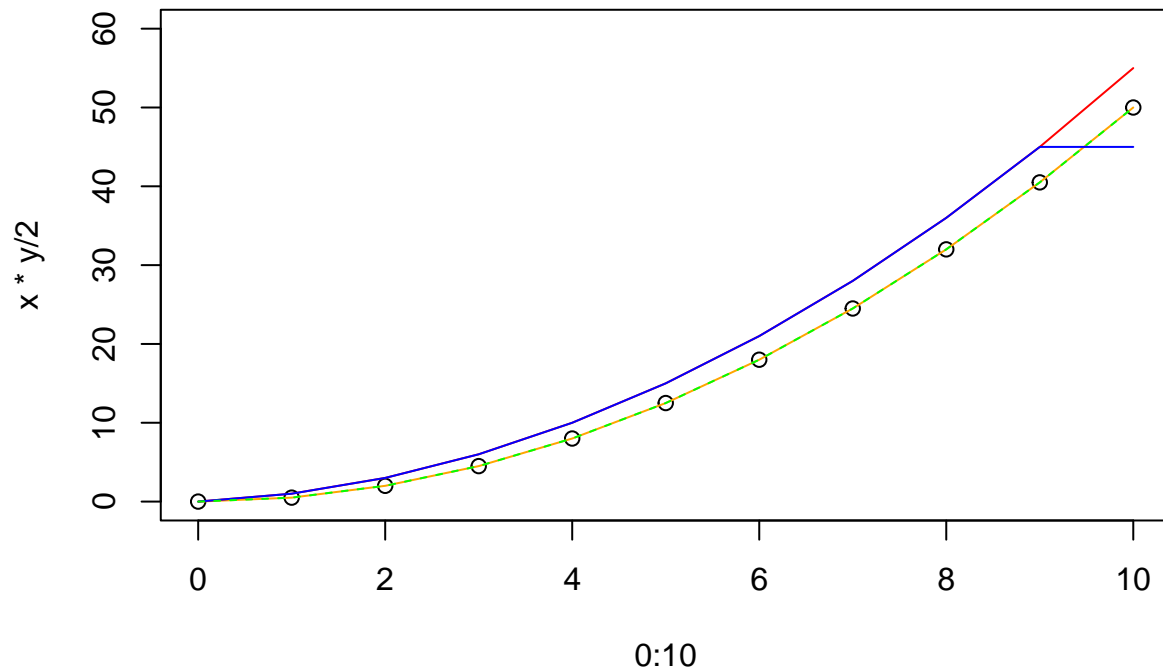
```
## [1] 0 1 3 6 10 15 21 28 36 45 45
```

```
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
```

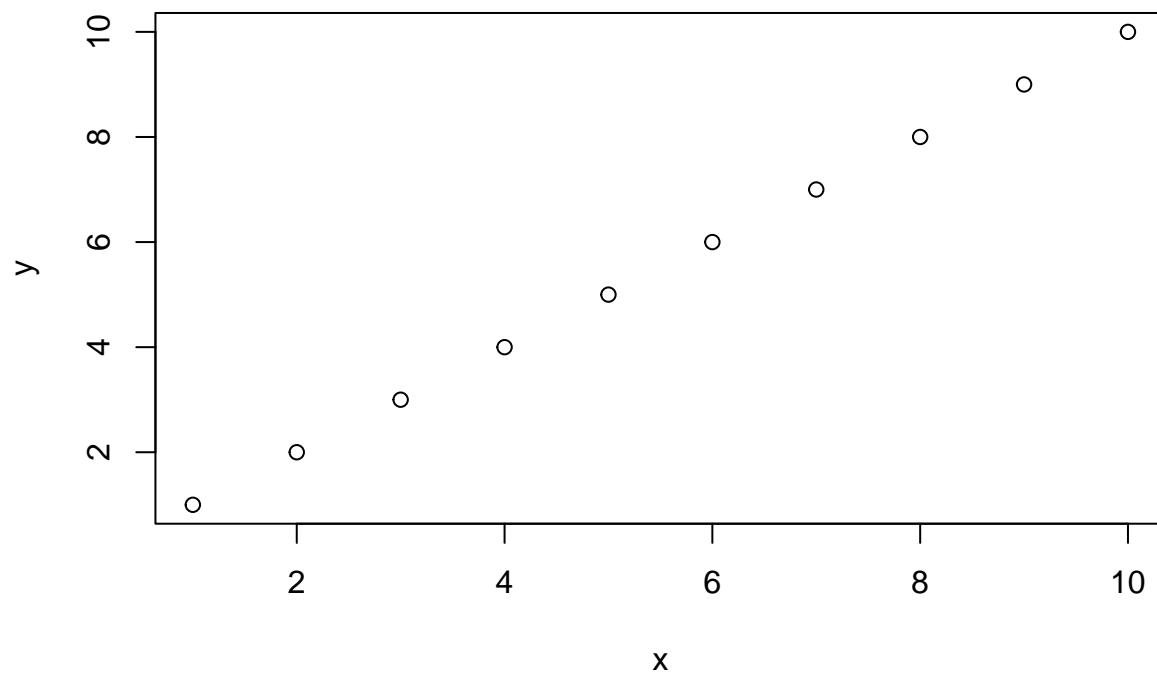
Note differences on the way up.

```
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)
```

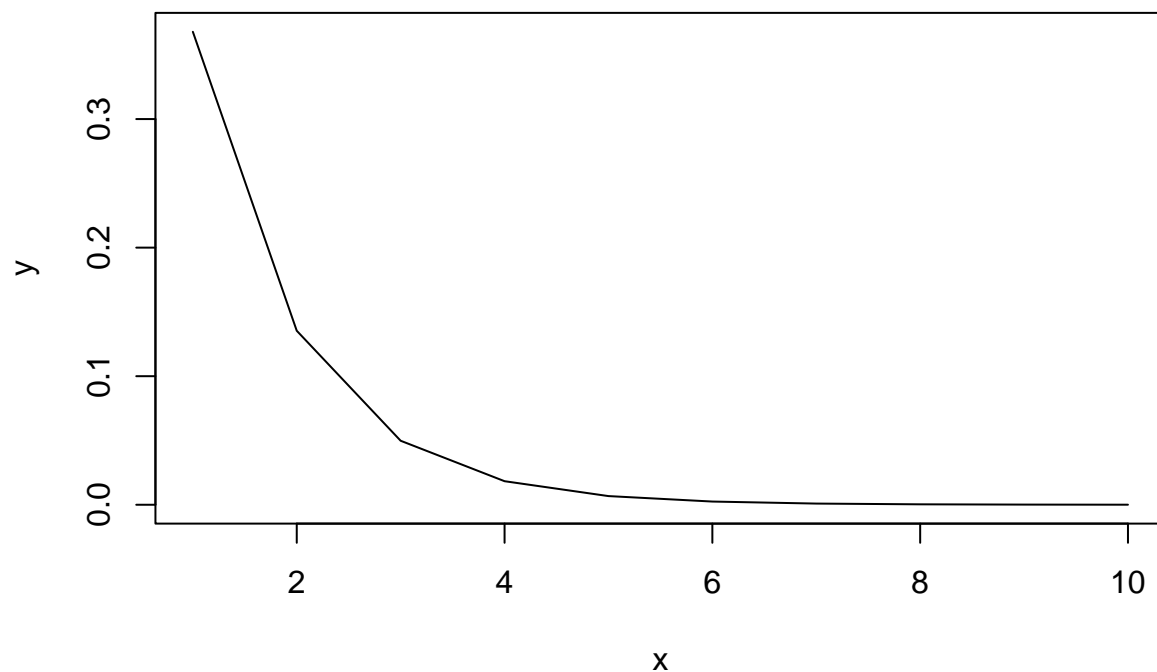
```
## [1] 1 3 6 10 15 21 28 36 45 45
```

```
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

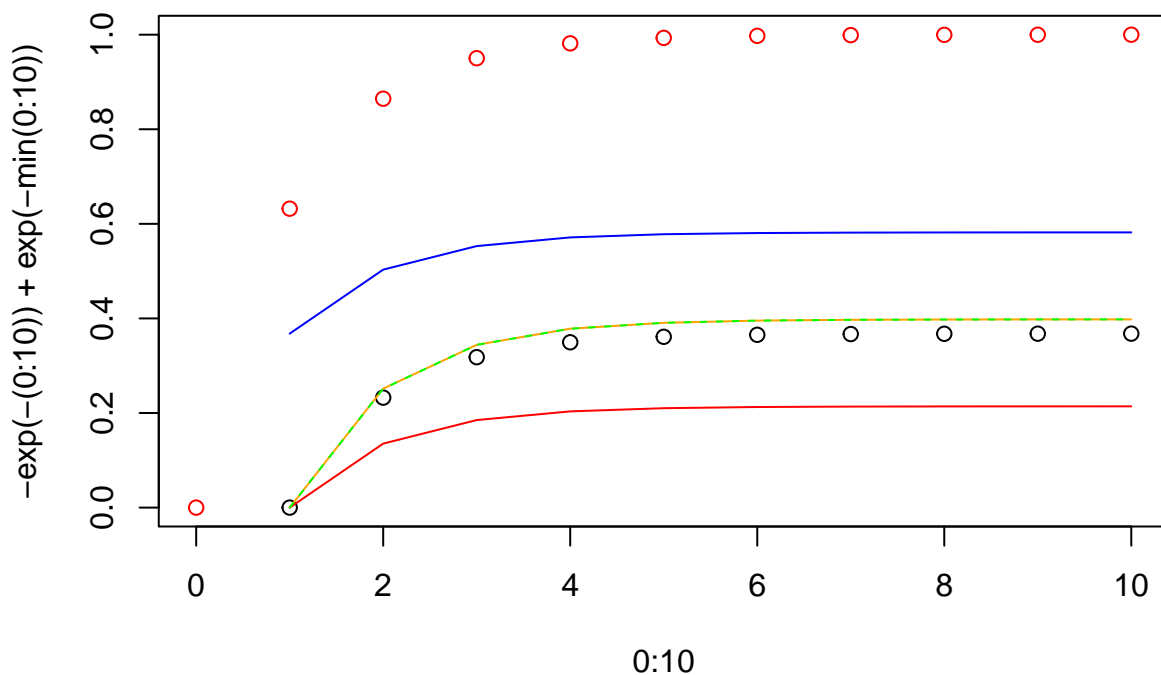
fintegrate(x, y, 'right', value = 'total')

## [1] 0.5819049

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

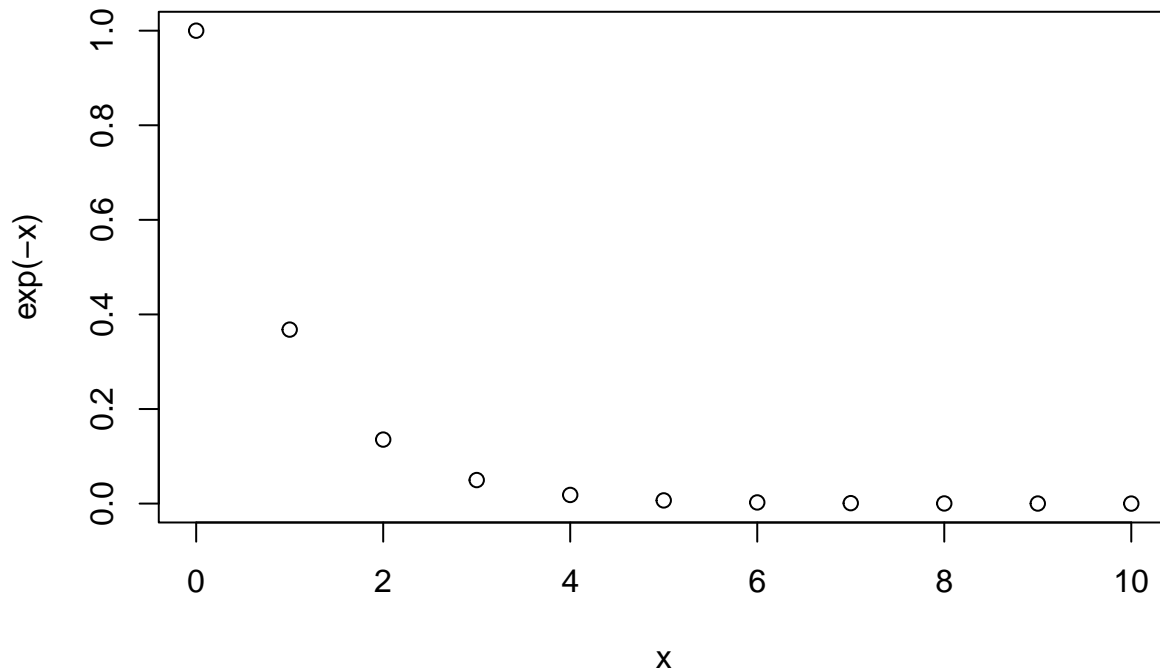
## [1] 0.3979879

plot(0:10, -exp(-(0:10)) + exp(-min(0:10)), col = 'red')
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)
```



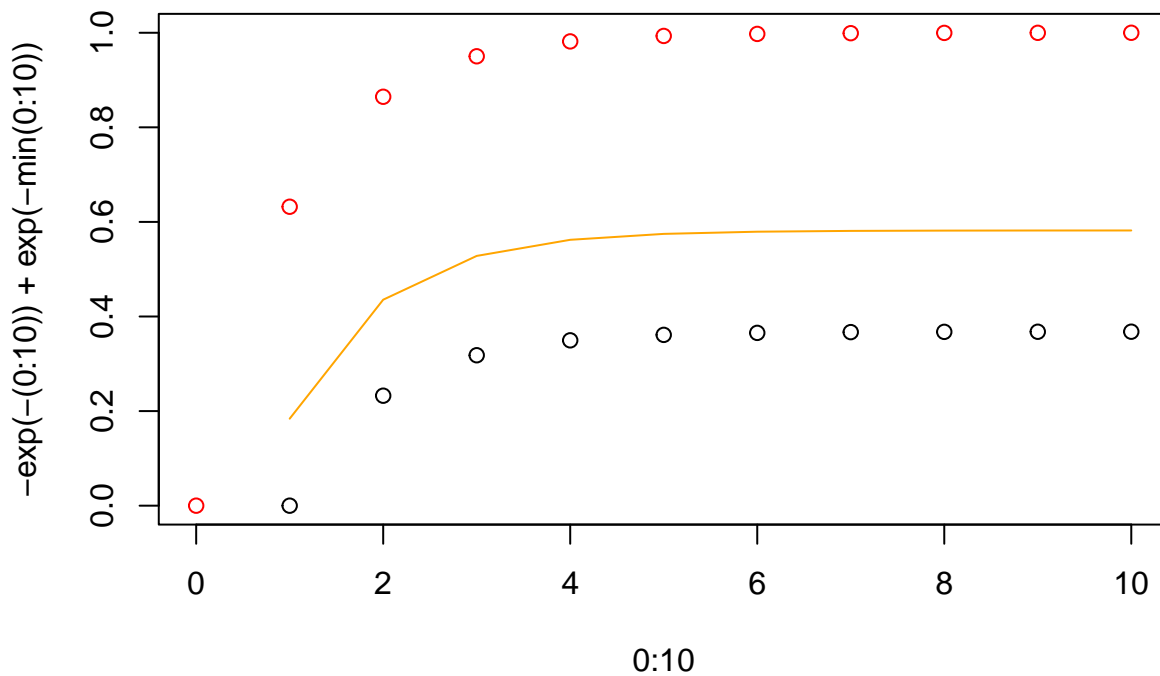
None is perfect, but midpoint and trapezoid (identical in this implementation) are the best, only slightly overestimating. Note that they all do poorly compared to a true integral that starts at 0 (red points). This cannot really be helped—how could we infer the true high values of y close to 0 from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

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



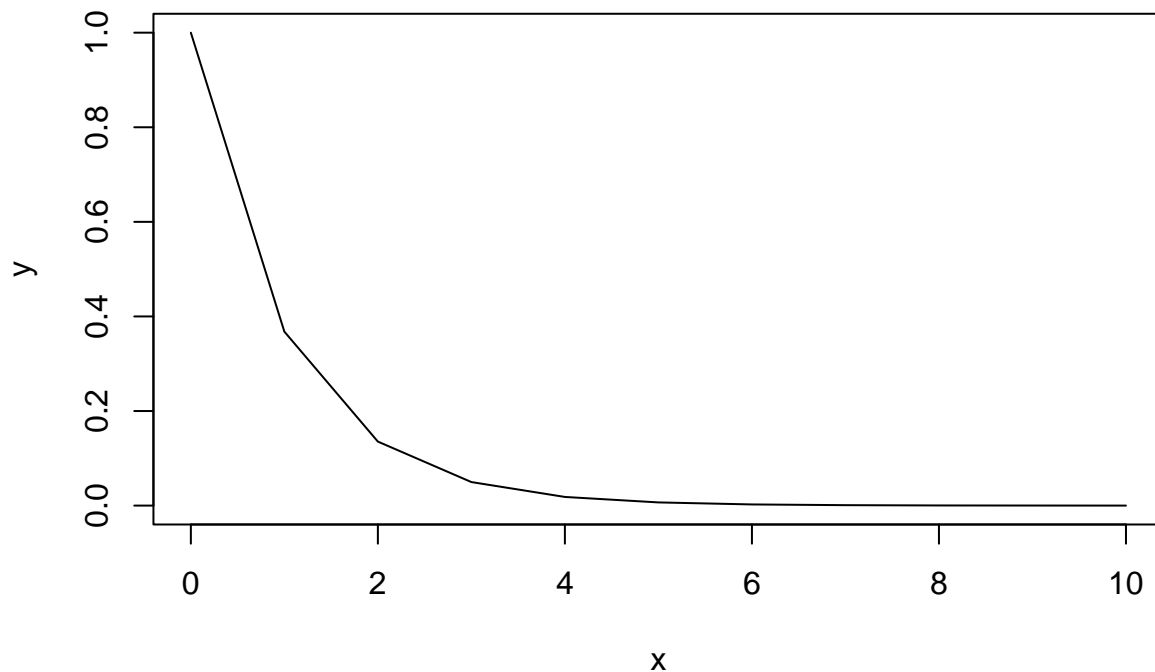
The `start` argument can extend the first rate back to 0 or any arbitrary starting point, which helps a bit.

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



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

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



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

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

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

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

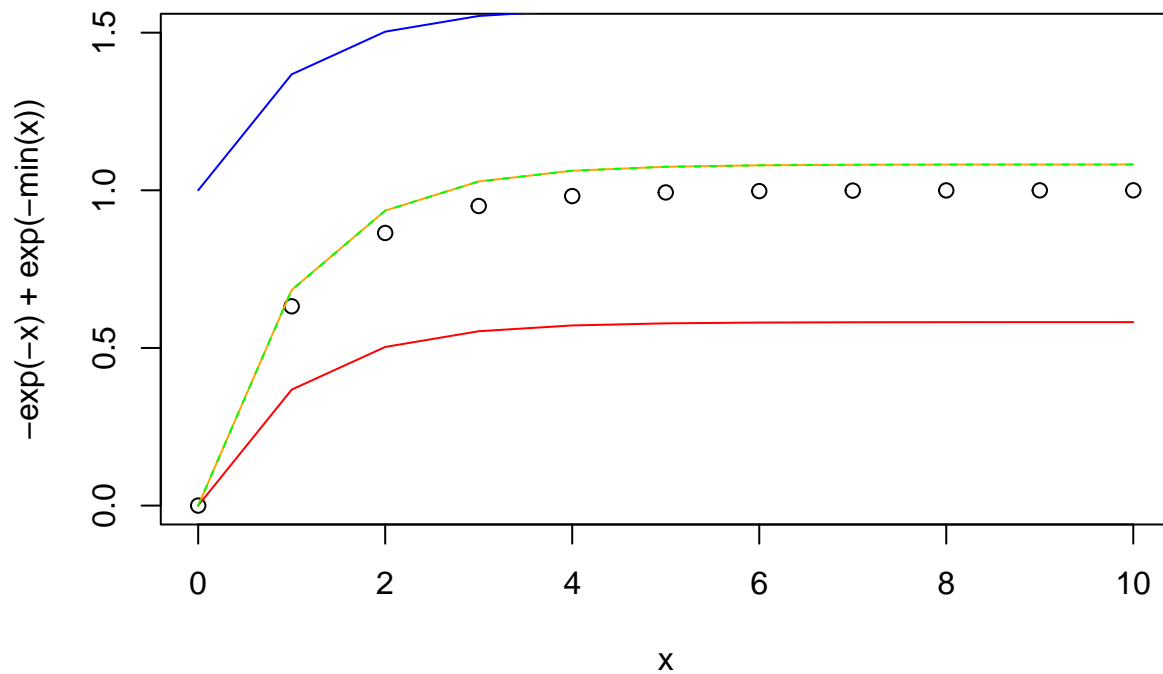
```
fintegrate(x, y, 'right', value = 'total')
```

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

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

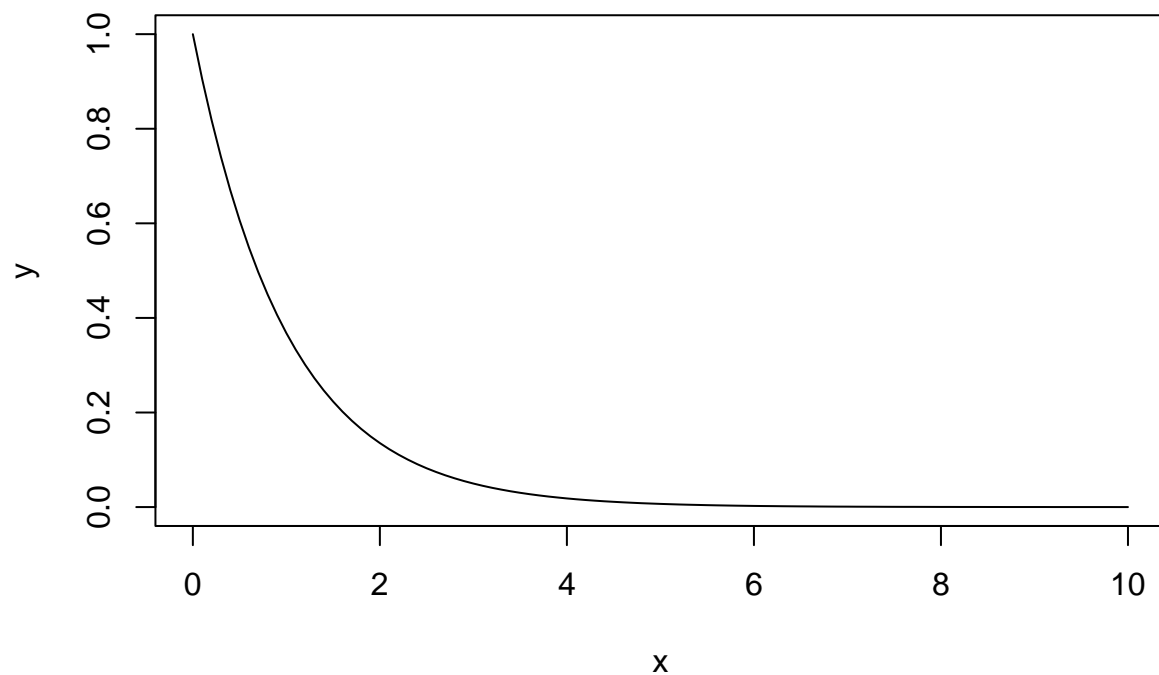
```
## [1] 1.081928
```

```
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)
```



We can 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
```

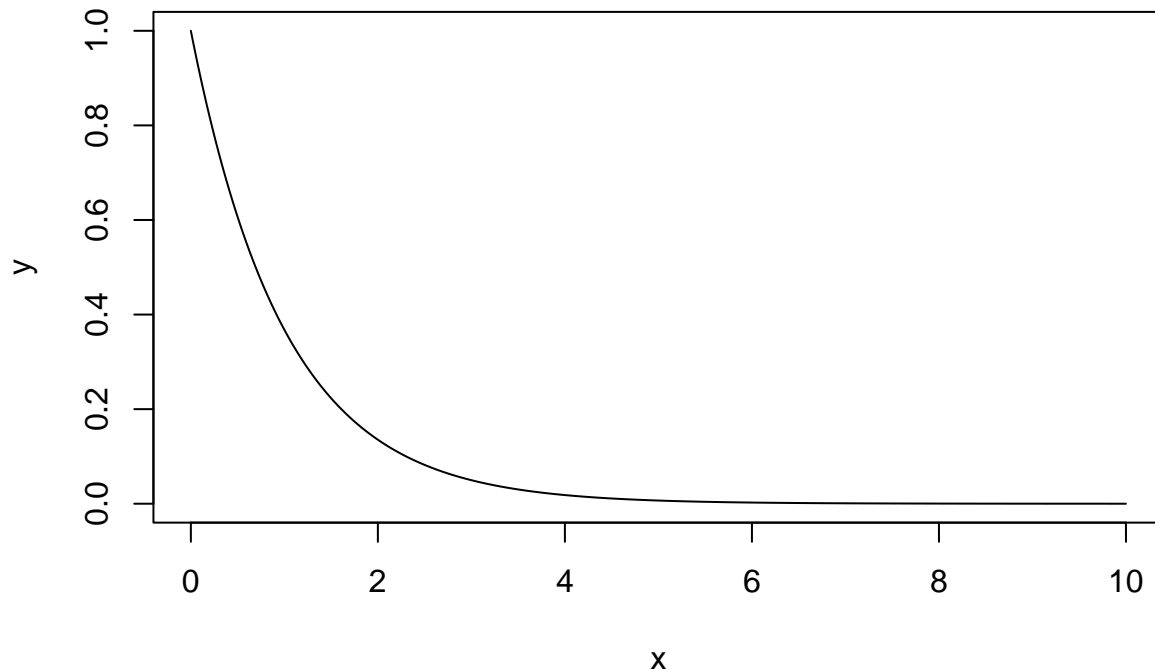
```
fintegrate(x, y, 'trap', value = 'total')
```

```
## [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
```

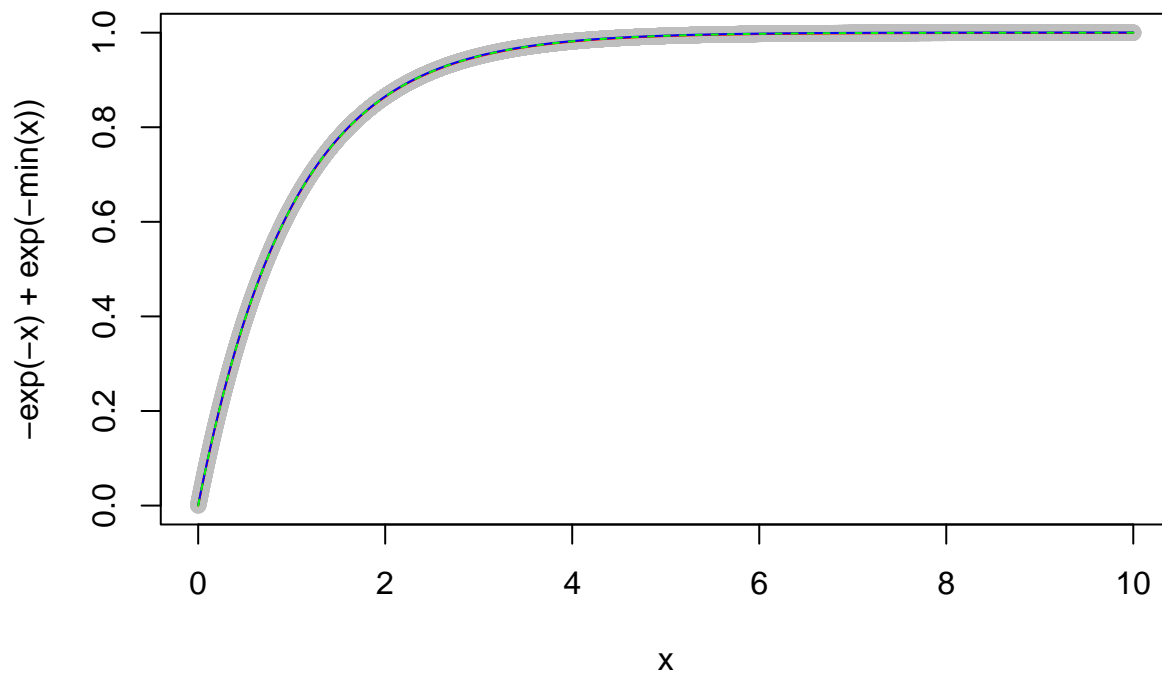
```
plot(x, -exp(-x) + exp(-min(x)), col = 'gray')
```

```
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
```