1. Loop Functions

Contents

1 1. Loop Functions

Looping on the Command Line

Writing for, while loops is useful when programming but not particularly easy when working interactively on the command line. There are some functions which implement looping to make life easier.

· lapply: Loop over a list and evaluate a function on each element (list) - return list · sapply: Same as lapply but try to simplify the result (simple) · apply: Apply a function over the margins of an array (apply - on row or col of a matrix) · tapply: Apply a function over subsets of a vector (table - on a sub set) · mapply: Multivariate version of lapply (multivariate) An auxiliary function split is also useful, particularly in conjunction with lapply.

##lapply

lapply takes three arguments: (1) a list X; (2) a function (or the name of a function) FUN; (3) other arguments via its … argument. If X is not a list, it will be coerced to a list using as.list.

```
lapply
```

```
## function (X, FUN, ...)
## {
##     FUN <- match.fun(FUN)
##     if (!is.vector(X) || is.object(X))
##         X <- as.list(X)
##     .Internal(lapply(X, FUN))
## }
## <bytecode: 0x129134598>
## <environment: namespace:base>
```

The actual looping is done internally in C code.

lapply always returns a list, regardless of the class of the input.

```r
x <- list(a = 1:5, b = rnorm(10))
x
```

```
## $a
## [1] 1 2 3 4 5
##
## $b
##  [1] -0.40597213 -1.08081077 -0.03532611 -0.80269565 -0.14581009  0.46971361
##  [7]  1.76581114 -0.37855759 -1.38144380 -0.01310611
```

```r
lapply(x, mean)
```

```
## $a
## [1] 3
##
## $b
## [1] -0.2008198
```

```r
x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
x
```

```
## $a
## [1] 1 2 3 4
##
## $b
##  [1] -0.811294506  0.271824867 -0.091850172  0.517214060  2.354491729
##  [6] -1.635872433 -0.004686738 -0.415469898  0.683846914  0.394077311
##
## $c
##  [1]  1.51750308  1.97767733  1.23058692 -0.49357058  1.53983601  1.02420518
##  [7]  0.15512524  2.26711168  0.36016289  0.03707857  0.84832691  1.32100806
## [13]  1.11683621 -0.43701680  0.18724741  0.23833730  0.68498623  0.09029907
## [19]  2.06352434  0.22500926
##
## $d
##   [1] 5.350292 5.363950 4.714007 5.063120 5.206988 4.120681 7.185829 4.471640
##   [9] 5.314663 4.213670 6.146022 3.756900 5.353812 4.718872 3.842965 4.519930
##  [17] 5.220888 5.055310 6.325086 5.032320 3.392397 5.972497 4.830800 4.514963
##  [25] 4.470215 4.632025 4.121490 5.194538 4.556984 5.546430 4.996492 5.356584
##  [33] 6.271134 2.131709 4.329464 5.807723 6.055467 5.153145 4.982345 5.510222
##  [41] 4.912302 4.199912 4.148834 2.958660 5.256693 6.927872 4.381719 5.063359
##  [49] 4.507268 5.885052 6.248393 5.904976 5.171119 3.527428 3.557895 5.422567
##  [57] 6.834431 4.251573 6.999054 5.808154 4.274716 2.872507 6.445044 4.014029
##  [65] 4.106864 4.958942 6.123648 5.995729 4.595391 5.476944 4.758604 5.609710
##  [73] 6.443839 6.090177 3.365248 6.108965 4.486481 5.644918 4.709955 4.142245
##  [81] 5.513538 2.788098 5.973399 4.630968 3.026506 4.383709 5.091307 5.934585
##  [89] 5.440869 5.258992 5.671972 6.310989 4.941151 5.389017 5.554198 6.225429
##  [97] 5.969112 4.943486 4.014466 4.459933
```

```r
lapply(x, mean)
```

```
## $a
## [1] 2.5
##
## $b
## [1] 0.1262281
```

```
##
## $c
## [1] 0.7977137
##
## $d
## [1] 5.025165
```

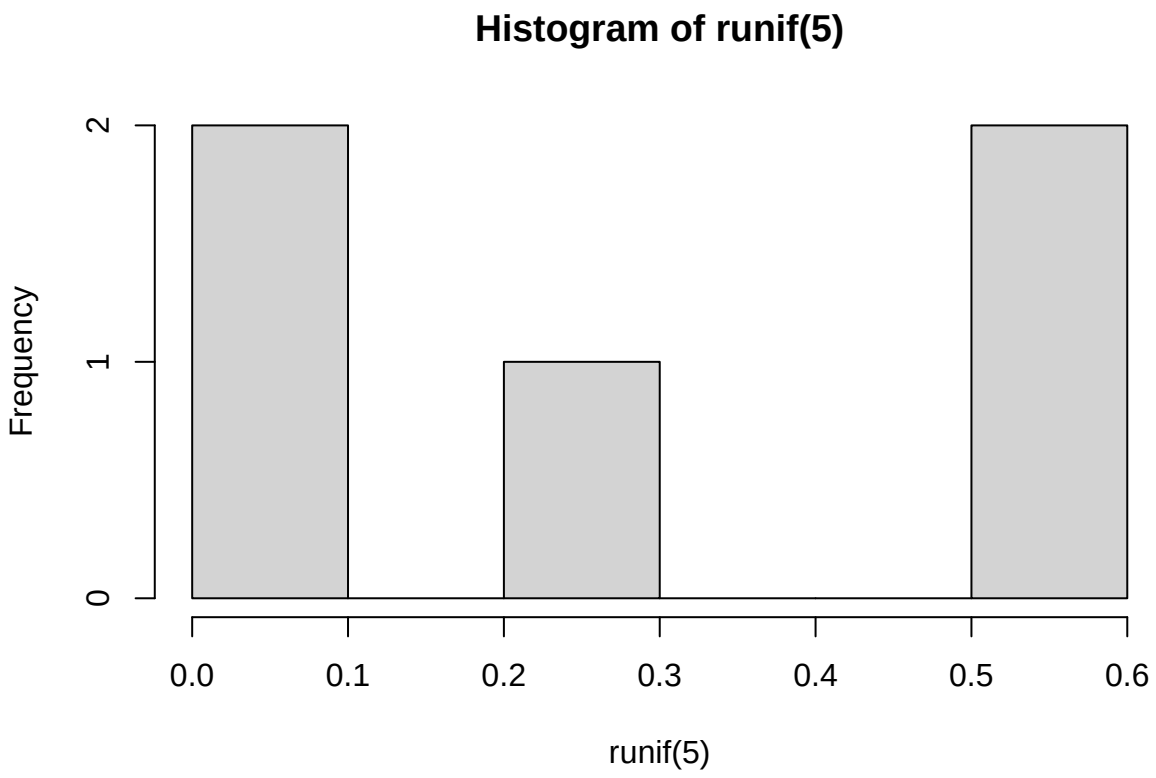
```
x<-1:4
x
```

```
## [1] 1 2 3 4
```

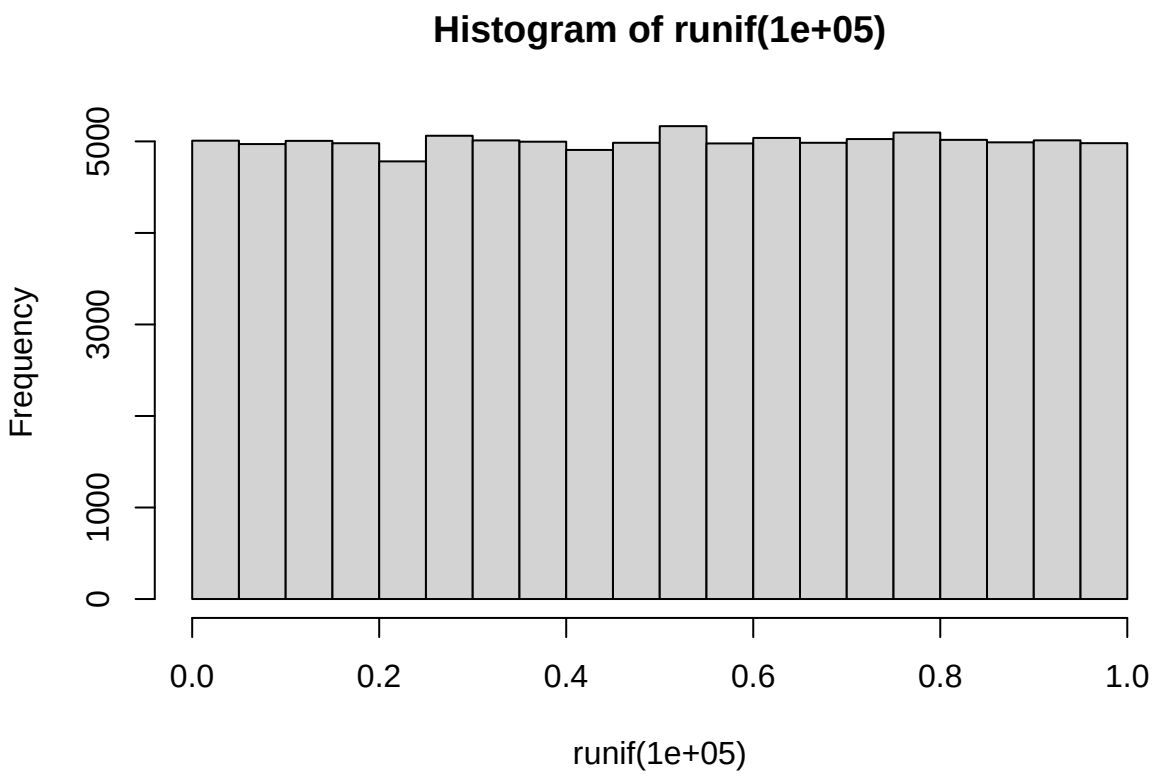
```
# The runif function in R generates random numbers from a uniform distribution.
lapply(x, runif) # r: random , unif : uniform
```

```
## [[1]]
## [1] 0.6629462
##
## [[2]]
## [1] 0.3520533 0.7762519
##
## [[3]]
## [1] 0.3724175 0.2598497 0.5822212
##
## [[4]]
## [1] 0.1262403 0.7783322 0.9908016 0.9499929
```

```
hist(runif(5))
```



```r
hist(runif(100000))
```




```r
x<-1:4
x
```

```
## [1] 1 2 3 4
```

```r
lapply(x, runif, min = 0, max = 10)
```

```
## [[1]]
## [1] 8.835805
##
## [[2]]
## [1] 0.8960222 6.4760318
##
## [[3]]
## [1] 7.264832 6.991576 3.133491
##
## [[4]]
## [1] 9.170640 1.946949 3.632586 9.991785
```

lapply and friends make heavy use of anonymous functions.

```r
x <- list(a = matrix(1:4, 2, 2), b = matrix(1:6, 3, 2))
x
```

```
## $a
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## $b
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

An anonymous function for extracting the first column of each matrix.

```
lapply(x, function(elt) elt[,1])
```

```
## $a
## [1] 1 2
##
## $b
## [1] 1 2 3
```

1.1 sapply

sapply will try to simplify the result of lapply if possible. · If the result is a list where every element is length 1, then a vector is returned · If the result is a list where every element is a vector of the same length (> 1), a matrix is returned. · If it can't figure things out, a list is returned

```
x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
x
```

```
## $a
## [1] 1 2 3 4
##
## $b
##  [1] -0.7527978 -0.4109084 -0.3614250  0.4468058 -1.0588733  0.7286578
##  [7]  0.1118698 -1.7287156  1.0982578  0.5730390
##
## $c
##  [1]  1.4394268  1.2043710  1.5957636 -0.3475811  1.5091591  1.8811671
##  [7]  1.1975935  2.0425442  2.0250294 -0.4876958 -0.4795813  2.6984124
## [13]  1.3152551  0.1730725  0.2638660  1.1581306  0.7584103  1.4799300
## [19]  2.0086327  0.8025828
##
## $d
##   [1] 3.346437 5.298279 5.078638 4.342333 5.486375 6.419879 6.466589 2.901612
##   [9] 6.762977 6.100448 5.532624 5.739986 4.836997 6.186923 4.009555 4.630101
##  [17] 4.784589 6.079246 4.995203 4.572738 5.006622 5.426502 4.293022 4.749290
##  [25] 4.096002 4.350454 4.552311 5.274547 5.750382 5.191912 4.445086 3.661717
##  [33] 4.832482 3.346163 6.152534 4.956238 6.278906 3.853878 3.538979 2.478725
##  [41] 5.798005 2.795016 4.654517 5.916448 5.355085 5.609216 4.190922 4.422246
##  [49] 5.105742 4.623147 5.592393 6.392032 1.240860 7.217929 3.569799 4.975379
##  [57] 6.199542 3.496772 5.765657 3.470058 5.774220 6.373582 5.916641 6.233327
##  [65] 6.085223 4.821776 5.136646 5.163092 4.502382 5.231246 3.605523 6.986898
##  [73] 6.459867 5.349342 6.166439 5.538921 5.318379 4.522792 4.201306 6.147778
##  [81] 5.990069 3.534440 4.901003 4.870423 5.142621 4.887559 4.043663 6.385410
##  [89] 5.397528 5.826310 3.915970 6.653829 4.262762 4.595193 5.676256 4.143646
##  [97] 5.232214 5.833385 4.704439 3.852878
```

```
lapply(x, mean)
```

```
## $a
## [1] 2.5
##
```

```
## $b
## [1] -0.135409
##
## $c
## [1] 1.111924
##
## $d
## [1] 5.01583
```

```
sapply(x, mean)
```

```
##         a         b         c         d
##  2.500000 -0.135409  1.111924  5.015830
```

```
class(lapply(x, mean))
```

```
## [1] "list"
```

```
class(sapply(x, mean))
```

```
## [1] "numeric"
```

```
mean(x)
```

```
## Warning in mean.default(x): argument is not numeric or logical: returning NA
```

```
## [1] NA
```

1.2 apply

apply is used to a evaluate a function (often an anonymous one) over the margins of an array. · It is most often used to apply a function to the rows or columns of a matrix · It can be used with general arrays, e.g. taking the average of an array of matrices · It is not really faster than writing a loop, but it works in one line!

```
str(apply)
```

```
## function (X, MARGIN, FUN, ..., simplify = TRUE)
```

X is an array · MARGIN is an integer vector indicating which margins should be "retained". · FUN is a function to be applied · … is for other arguments to be passed to FUN

The MARGIN argument in the apply function in R indicates the dimension over which the function should be applied.

MARGIN can take the following values:

MARGIN = *1*: The function is applied over *rows*. MARGIN = *2*: The function is applied over *columns*. MARGIN = c(1, 2): The function is applied to each element of the array/matrix.

```
x <- matrix(rnorm(200), 20, 10)
x
```

```
##              [,1]        [,2]       [,3]       [,4]        [,5]        [,6]
##  [1,] -0.22068768  0.94120534 -0.3017430 -0.9365954 -0.08811322 -0.22460361
##  [2,]  0.61722630 -1.01429682  1.2047353  0.9166133 -1.00277605 -1.39606802
##  [3,] -0.40972519  0.17334830  0.7753451 -0.3168669 -0.53404043 -0.93703402
##  [4,]  0.31135919  0.76901738 -1.0085899 -0.4537235  0.19888330 -1.37069056
##  [5,]  0.15169878  0.49609157 -0.9471089  2.8931672 -0.48344719 -1.21870576
##  [6,]  0.11226258 -2.54184564  0.9852258 -1.9368927  0.94169247 -0.44896156
##  [7,]  1.05942985 -0.42150893 -3.0379951  0.8347619  0.83170104  1.29019108
```

```
##  [8,]  0.28228747 -0.83989734  0.8570389  0.5669083 -0.22858759 -1.07509515
##  [9,] -0.21765344  0.92083777  0.3604083  0.6118067  0.76107106 -0.09904831
## [10,]  0.21225034  0.43446420 -0.9663597  0.7408545  0.06157827  1.44768244
## [11,] -0.56218458  0.14153451  1.5728229  1.2951507 -2.02628591  0.35829183
## [12,]  0.66611351 -1.01689886 -0.4693431 -0.7594419 -0.64876855  0.01388809
## [13,] -1.07892808 -1.17000517 -0.6575501  0.4117644 -0.55078406  0.12096139
## [14,] -1.74857291 -0.05085377  0.2921307  0.6298072 -0.93960706  0.87818284
## [15,]  0.07042026 -0.45835414  0.7097704  0.8373039 -0.66837227  1.49225286
## [16,]  0.76220620 -0.71719202 -0.4442763 -0.2688799 -1.98844040 -0.31162721
## [17,] -0.27673546 -0.70395316  0.4973278 -1.1678597 -0.34070976  0.59575477
## [18,] -1.24321449  0.95794673  1.6553763 -1.4098958  1.31532951 -0.65577353
## [19,] -0.01870261 -0.52716197 -1.2866519 -0.1938447  1.18440300 -0.59456996
## [20,]  0.26092205  0.16189801 -0.1376149  0.3052933 -0.66906560 -0.46513147
##              [,7]        [,8]         [,9]        [,10]
##  [1,]  2.04822684  0.58343750  0.277435757  0.890527829
##  [2,] -0.10420937  2.10106434 -1.556714641  0.355043601
##  [3,]  0.43845549 -0.85951719  0.189534680  0.385316859
##  [4,]  0.07900070  0.73160188 -0.985490946  1.584982010
##  [5,] -1.01765230 -0.04551298  0.008209843  0.008093165
##  [6,] -1.55177482 -0.65277433  0.601612212 -0.108318571
##  [7,]  0.68730288 -1.31445854 -0.331148221  0.346052671
##  [8,]  0.55551402 -0.27336462  0.412121604  0.747197688
##  [9,]  0.49777139 -1.72535994 -0.007497773  0.652027566
## [10,] -2.31415385 -0.04285965  0.722696731 -0.109896484
## [11,]  0.01999203 -0.46929971 -1.060064712  0.863730728
## [12,]  0.07450287  0.32392349  1.941838409  1.006425148
## [13,]  0.65900685 -0.73902864 -0.455897266 -0.265994545
## [14,] -1.19086588 -1.42576434 -0.396454732 -0.526285698
## [15,]  1.28324011 -0.01483858 -0.532794717 -0.750156365
## [16,] -0.80222152  0.85171974  0.686500840  0.101926705
## [17,]  0.12873690 -1.39456790  0.737846250  0.152713029
## [18,]  3.08270404  0.83477295 -0.565475057  0.457803804
## [19,] -0.36092257 -1.11883964 -0.956392186 -0.051863596
## [20,] -0.90749675  1.03971638 -2.720964145 -0.915509761
```

```
apply(x, 2, mean) # apply over columns, we have 10 columns
```

```
##  [1] -0.06351140 -0.22328120 -0.01735257  0.12997155 -0.24371697 -0.13000519
##  [7]  0.06525785 -0.18049749 -0.19955490  0.24119079
```

```
apply(x, 1, sum) # apply over 20 rows
```

```
##  [1]  2.96909038  0.12061795 -1.09518328 -0.14365043 -0.15516658 -4.59977456
##  [7] -0.05567142  1.00412333  1.75436329  0.18625680  0.13368776  1.13223907
## [13] -3.72645518 -4.47828362  1.96847152 -2.13028394 -1.77144722  4.42957454
## [19] -3.92454610 -4.04795286
```

1.2.1 col/row sums and means

For sums and means of matrix dimensions, we have some shortcuts. · rowSums = apply(x, 1, sum) · rowMeans = apply(x, 1, mean) · colSums = apply(x, 2, sum) · colMeans = apply(x, 2, mean) The shortcut functions are much faster, but you won't notice unless you're using a large matrix.

1.2.2 Other Ways to Apply

Quantiles of the rows of a matrix.

```r
x <- matrix(rnorm(200), 20, 10)
x
```

```
##               [,1]        [,2]          [,3]       [,4]         [,5]
##  [1,]  0.101718139  0.12750234  0.4430041471 -0.2392441 -0.263761959
##  [2,]  1.664885509 -1.61017568  0.1109072674 -0.3629474 -0.731277495
##  [3,] -2.017437779 -0.64251215  1.0343130365 -0.6079262 -0.615398316
##  [4,]  0.104257454  0.22240639  0.3959583932  0.6876956  2.149563940
##  [5,]  0.923492317 -0.07385803 -0.7097568541 -0.4135593  0.801346083
##  [6,] -0.396552311 -0.40748056 -0.4512120963 -0.1082280 -0.728740526
##  [7,] -1.498633248 -0.59759737 -0.4949419958  0.7397065  1.024406023
##  [8,] -1.989787721  1.53159507 -1.0280193743 -1.1868921 -0.209998841
##  [9,] -0.104207061  1.82711196 -0.2844113260 -1.6861740  0.025132872
## [10,] -0.671024844  1.32750221  0.3957152191  0.9466139 -0.009087485
## [11,] -0.142819234  1.08232633 -0.3865494366  0.7364978 -1.415047413
## [12,] -0.030203031 -1.21391331  2.1204139608 -1.9082870 -0.888012846
## [13,] -0.755898703 -0.76646701  0.0003694094  1.3871221  1.515224386
## [14,] -0.014366437 -0.44985651 -0.2996091434 -0.7608138  0.195181834
## [15,] -1.202985932  0.48205033 -0.7538775375  0.5176744 -0.083037733
## [16,]  0.337119518  0.57266870 -2.2037638455 -0.5301931 -0.635484044
## [17,] -0.336353091  0.05360735  0.3344750533 -1.1114481 -2.055270394
## [18,]  1.262446166  0.67453237 -1.7947471148 -0.6763729  0.465849939
## [19,]  0.005694951 -2.68756435  0.1427888908 -2.3554121  1.037200982
## [20,]  1.860030406 -0.07690457 -0.4837824180 -0.3292394  0.408517588
##                [,6]         [,7]          [,8]       [,9]         [,10]
##  [1,]  2.8565114824  0.72638302  0.137930578 -1.5813637  0.502554444
##  [2,] -0.1578096534  0.13555065  1.028660193  0.1108585  0.169311519
##  [3,] -0.6038577044  0.36831308  0.935244831 -0.5325809  1.157709746
##  [4,] -1.4411311521  0.35524029  1.520297698  1.9374423 -0.211851204
##  [5,] -0.5744392158 -0.11930075  0.464933221  1.1208981 -0.163232035
##  [6,]  0.6315302285  0.76721620 -0.480202853 -0.6740708  0.422513871
##  [7,]  2.0318341729  1.61702063  0.643123509  1.1908467  2.498919280
##  [8,]  0.9512746928  1.69317216 -1.672979060 -0.3485804 -0.002719868
##  [9,] -0.5670187793 -0.92970926 -0.533480130  0.5844684 -0.743782014
## [10,]  1.2754971052 -1.43259006 -0.794010060  0.6074827 -0.934824905
## [11,]  0.4883255667  0.46455461 -0.414974600  1.7310167 -1.690818890
## [12,] -0.4684922299  2.27452807  0.007856386 -0.4475992 -0.424582969
## [13,] -0.5686772335 -0.57922419 -0.196061207  0.2106154 -2.312137442
## [14,] -0.2590553983  1.23507759 -0.950524333 -1.5457363 -0.453326907
## [15,]  1.7897320383 -2.02047996  1.881744052  1.2417745 -0.208528884
## [16,]  0.7400122920  0.72198725  0.102407450 -0.5083530 -0.590168887
## [17,]  1.9187673446  0.84528383  1.570237358  0.5375610 -0.904608465
## [18,]  1.2065823805  1.20184803 -0.725718145  0.4575724  0.306872661
## [19,]  0.3682001461  0.02064885  0.392269508 -1.7640265 -1.263441281
## [20,] -0.0008222655 -1.58455477 -0.059228624 -0.1213307 -0.312077208
```

```r
apply(x, 1, quantile, probs = c(0.25, 0.75))
```

```
##           [,1]       [,2]       [,3]      [,4]       [,5]       [,6]       [,7]
## 25% -0.1540036 -0.3116630 -0.6135303 0.1337947 -0.3509775 -0.4729552 -0.2104256
## 75%  0.4876669  0.1608713  0.7935119 1.3121472  0.7172429  0.2898284  1.5104771
##           [,8]         [,9]      [,10]      [,11]        [,12]      [,13]
## 25% -1.1471739 -0.699591205 -0.7632638 -0.4078683 -0.783132692 -0.7117301
## 75%  0.7127761 -0.007202111  0.8618311  0.6744548 -0.001658468  0.1580539
##           [,14]      [,15]      [,16]      [,17]      [,18]      [,19]
```

```
## 25% -0.68394207 -0.6175404 -0.5751750 -0.7625446 -0.4305615 -1.6388802
## 75% -0.07553868  1.0607494  0.5137814  0.7683531  1.0700191  0.3118473
##           [,20]
## 25% -0.32494884
## 75% -0.01542386
```

The probs parameter in the quantile function specifies the probabilities for which you want to find the quantiles.

1.2.3 Average matrix in an array

```
a <- array(rnorm(2 * 2 * 10), c(2, 2, 10))
a
```

```
## , , 1
##
##            [,1]       [,2]
## [1,] -0.2364384 -1.0090086
## [2,]  0.2102662 -0.1767762
##
## , , 2
##
##            [,1]       [,2]
## [1,]  0.8629358 -0.7310878
## [2,] -0.3301001 -0.2400498
##
## , , 3
##
##            [,1]       [,2]
## [1,] -0.5661198 -1.4106237
## [2,] -0.2112341 -0.9058156
##
## , , 4
##
##          [,1]       [,2]
## [1,] 1.682350  0.9358234
## [2,] 1.405745 -0.2160457
##
## , , 5
##
##            [,1]       [,2]
## [1,]  0.107886 -0.7459053
## [2,] -1.177250 -0.9066917
##
## , , 6
##
##            [,1]      [,2]
## [1,] -0.1822736 0.1923667
## [2,]  1.0430767 1.2131071
##
## , , 7
##
##           [,1]       [,2]
## [1,] 1.7141178 0.06156547
## [2,] 0.9892649 1.66438526
```

```
##
## , , 8
##
##            [,1]        [,2]
## [1,] -0.6307767 -0.04188541
## [2,]  0.4884695 -0.48907013
##
## , , 9
##
##           [,1]      [,2]
## [1,] 0.8519977 0.3356860
## [2,] 0.1467760 0.3463038
##
## , , 10
##
##           [,1]       [,2]
## [1,] -0.116708 -1.1839860
## [2,]  1.538831  0.8114847
```

```
apply(a, c(1, 2), mean)
```

```
##           [,1]       [,2]
## [1,] 0.3486970 -0.3597055
## [2,] 0.4103845  0.1100832
```

```
rowMeans(a, dims = 2)
```

```
##           [,1]       [,2]
## [1,] 0.3486970 -0.3597055
## [2,] 0.4103845  0.1100832
```

1.3 mapply

mapply is a multivariate apply of sorts which applies a function in parallel over a set of arguments.

```
str(mapply)
```

```
## function (FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)
```

FUN is a function to apply · ... contains arguments to apply over · MoreArgs is a list of other arguments to FUN. · SIMPLIFY indicates whether the result should be simplified

The following is tedious to type

```
list(rep(1, 4), rep(2, 3), rep(3, 2), rep(4, 1))
```

```
## [[1]]
## [1] 1 1 1 1
##
## [[2]]
## [1] 2 2 2
##
## [[3]]
## [1] 3 3
##
## [[4]]
## [1] 4
```

Instead we can do

```r
mapply(rep, 1:4, 4:1)
```

```
## [[1]]
## [1] 1 1 1 1
##
## [[2]]
## [1] 2 2 2
##
## [[3]]
## [1] 3 3
##
## [[4]]
## [1] 4
```

1.3.1 Vectorizing a Function

```r
noise <- function(n, mean, sd) {
  rnorm(n, mean, sd)
}
```

```r
noise(5, 1, 2)
```

```
## [1]  0.7572779 -0.1320194 -0.9467855  2.6007378  0.4782336
```

```r
noise(1:5, 1:5, 2) # this is not the correct way, look for the next one
```

```
## [1] 1.8311716 3.9972630 0.8636572 8.8890880 4.2100457
```

1.3.2 Instant Vectorization

```r
mapply(noise, 1:5, 1:5, 2)
```

```
## [[1]]
## [1] -0.03314353
##
## [[2]]
## [1]  1.525988 -1.143901
##
## [[3]]
## [1] 1.804137 1.182077 5.218762
##
## [[4]]
## [1] 8.652957 1.463361 3.232544 7.489554
##
## [[5]]
## [1] 5.205798 4.650116 7.157108 8.827230 5.668318
```

Which is the same as

```r
list(noise(1, 1, 2),
     noise(2, 2, 2),
     noise(3, 3, 2),
     noise(4, 4, 2),
     noise(5, 5, 2))
```

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

```
## [1] -1.208416
##
## [[2]]
## [1] -1.751806  1.900384
##
## [[3]]
## [1] 6.170934 2.061631 7.116297
##
## [[4]]
## [1] 4.016984 3.254409 4.929264 1.660983
##
## [[5]]
## [1] 3.659988 6.861264 4.179012 4.607696 4.518465
```

1.4 tapply

tapply is used to apply a function over subsets of a vector. I don't know why it's called tapply.

```
str(tapply)
```

```
## function (X, INDEX, FUN = NULL, ..., default = NA, simplify = TRUE)
```

· X is a vector · INDEX is a factor or a list of factors (or else they are coerced to factors) · FUN is a function to be applied · … contains other arguments to be passed FUN · simplify, should we simplify the result?

Take group means.

```
x <- c(rnorm(10), runif(10), rnorm(10, 1))
x
```

```
##  [1]  0.74637267  1.01106708 -0.41198136 -0.34405766 -0.69093771  0.95529043
##  [7] -1.20777508  0.04614183 -1.78283801 -0.29948072  0.06710856  0.78413039
## [13]  0.27710522  0.57269041  0.86578634  0.24723381  0.51455863  0.12126549
## [19]  0.64070257  0.69822968  2.57680784  1.48146027  0.85836541  1.08629543
## [25]  1.67392506 -0.03278327  0.66983423 -0.08526913 -0.40696337 -0.38116261
```

```
f<-gl(3,10)
f
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
## Levels: 1 2 3
```

The gl() function in R generates factors by specifying the pattern of their levels. The function stands for "generate levels".

In your case, f <- gl(3,10) generates a factor f of length 30 (3*10), with three levels, each repeating 10 times. Here, 3 is the number of levels, and 10 is the number of replications of each level.

```
tapply(x, f, mean)
```

```
##          1          2          3
## -0.1978199  0.4788811  0.7440510
```

Take group means without simplification.

```
tapply(x, f, mean, simplify = FALSE)
```

```
## $`1`
## [1] -0.1978199
##
```

```
## $`2`
## [1] 0.4788811
##
## $`3`
## [1] 0.744051
```

Find group ranges.

```
tapply(x, f, range)
```

```
## $`1`
## [1] -1.782838  1.011067
##
## $`2`
## [1] 0.06710856 0.86578634
##
## $`3`
## [1] -0.4069634  2.5768078
```

1.5 split

split takes a vector or other objects and splits it into groups determined by a factor or list of factors.

```
str(split)
```

```
## function (x, f, drop = FALSE, ...)
```

· x is a vector (or list) or data frame · f is a factor (or coerced to one) or a list of factors · drop indicates whether empty factors levels should be dropped

```
x <- c(rnorm(10), runif(10), rnorm(10, 1))
f<-gl(3,10)
split(x, f)
```

```
## $`1`
##  [1]  1.39718478  0.34199695 -0.02260777 -1.38399136  0.11029089  0.26311995
##  [7]  0.29237853  1.46748721 -0.94098389  0.44667753
##
## $`2`
##  [1] 0.972819355 0.002337378 0.589184547 0.758757015 0.674663394 0.090090729
##  [7] 0.571457118 0.834366249 0.232676006 0.173581587
##
## $`3`
##  [1] -0.7907686  0.8088647  0.6580380  2.6455654 -0.7210670  2.6854660
##  [7]  1.3434950  1.8365425  0.8908135  1.7340238
```

A common idiom is split followed by an lapply.

```
lapply(split(x, f), mean)
```

```
## $`1`
## [1] 0.1971553
##
## $`2`
## [1] 0.4899933
##
## $`3`
## [1] 1.109097
```

1.5.1 Splitting a Data Frame

```
library(datasets)
head(airquality)
```

Ozone	Solar.R	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

```
s <- split(airquality, airquality$Month)
head(s)
```

```
## $`5`
##    Ozone Solar.R Wind Temp Month Day
## 1     41     190  7.4   67     5   1
## 2     36     118  8.0   72     5   2
## 3     12     149 12.6   74     5   3
## 4     18     313 11.5   62     5   4
## 5     NA      NA 14.3   56     5   5
## 6     28      NA 14.9   66     5   6
## 7     23     299  8.6   65     5   7
## 8     19      99 13.8   59     5   8
## 9      8      19 20.1   61     5   9
## 10    NA     194  8.6   69     5  10
## 11     7      NA  6.9   74     5  11
## 12    16     256  9.7   69     5  12
## 13    11     290  9.2   66     5  13
## 14    14     274 10.9   68     5  14
## 15    18      65 13.2   58     5  15
## 16    14     334 11.5   64     5  16
## 17    34     307 12.0   66     5  17
## 18     6      78 18.4   57     5  18
## 19    30     322 11.5   68     5  19
## 20    11      44  9.7   62     5  20
## 21     1       8  9.7   59     5  21
## 22    11     320 16.6   73     5  22
## 23     4      25  9.7   61     5  23
## 24    32      92 12.0   61     5  24
## 25    NA      66 16.6   57     5  25
## 26    NA     266 14.9   58     5  26
## 27    NA      NA  8.0   57     5  27
## 28    23      13 12.0   67     5  28
## 29    45     252 14.9   81     5  29
## 30   115     223  5.7   79     5  30
## 31    37     279  7.4   76     5  31
##
## $`6`
##    Ozone Solar.R Wind Temp Month Day
## 32    NA     286  8.6   78     6   1
```

```
## 33    NA     287  9.7   74     6   2
## 34    NA     242 16.1   67     6   3
## 35    NA     186  9.2   84     6   4
## 36    NA     220  8.6   85     6   5
## 37    NA     264 14.3   79     6   6
## 38    29     127  9.7   82     6   7
## 39    NA     273  6.9   87     6   8
## 40    71     291 13.8   90     6   9
## 41    39     323 11.5   87     6  10
## 42    NA     259 10.9   93     6  11
## 43    NA     250  9.2   92     6  12
## 44    23     148  8.0   82     6  13
## 45    NA     332 13.8   80     6  14
## 46    NA     322 11.5   79     6  15
## 47    21     191 14.9   77     6  16
## 48    37     284 20.7   72     6  17
## 49    20      37  9.2   65     6  18
## 50    12     120 11.5   73     6  19
## 51    13     137 10.3   76     6  20
## 52    NA     150  6.3   77     6  21
## 53    NA      59  1.7   76     6  22
## 54    NA      91  4.6   76     6  23
## 55    NA     250  6.3   76     6  24
## 56    NA     135  8.0   75     6  25
## 57    NA     127  8.0   78     6  26
## 58    NA      47 10.3   73     6  27
## 59    NA      98 11.5   80     6  28
## 60    NA      31 14.9   77     6  29
## 61    NA     138  8.0   83     6  30
##
## $`7`
##    Ozone Solar.R Wind Temp Month Day
## 62   135     269  4.1   84     7   1
## 63    49     248  9.2   85     7   2
## 64    32     236  9.2   81     7   3
## 65    NA     101 10.9   84     7   4
## 66    64     175  4.6   83     7   5
## 67    40     314 10.9   83     7   6
## 68    77     276  5.1   88     7   7
## 69    97     267  6.3   92     7   8
## 70    97     272  5.7   92     7   9
## 71    85     175  7.4   89     7  10
## 72    NA     139  8.6   82     7  11
## 73    10     264 14.3   73     7  12
## 74    27     175 14.9   81     7  13
## 75    NA     291 14.9   91     7  14
## 76     7      48 14.3   80     7  15
## 77    48     260  6.9   81     7  16
## 78    35     274 10.3   82     7  17
## 79    61     285  6.3   84     7  18
## 80    79     187  5.1   87     7  19
## 81    63     220 11.5   85     7  20
## 82    16       7  6.9   74     7  21
## 83    NA     258  9.7   81     7  22
```

```
## 84    NA     295 11.5   82     7  23
## 85    80     294  8.6   86     7  24
## 86   108     223  8.0   85     7  25
## 87    20      81  8.6   82     7  26
## 88    52      82 12.0   86     7  27
## 89    82     213  7.4   88     7  28
## 90    50     275  7.4   86     7  29
## 91    64     253  7.4   83     7  30
## 92    59     254  9.2   81     7  31
##
## $`8`
##     Ozone Solar.R Wind Temp Month Day
## 93     39      83  6.9   81     8   1
## 94      9      24 13.8   81     8   2
## 95     16      77  7.4   82     8   3
## 96     78      NA  6.9   86     8   4
## 97     35      NA  7.4   85     8   5
## 98     66      NA  4.6   87     8   6
## 99    122     255  4.0   89     8   7
## 100    89     229 10.3   90     8   8
## 101   110     207  8.0   90     8   9
## 102    NA     222  8.6   92     8  10
## 103    NA     137 11.5   86     8  11
## 104    44     192 11.5   86     8  12
## 105    28     273 11.5   82     8  13
## 106    65     157  9.7   80     8  14
## 107    NA      64 11.5   79     8  15
## 108    22      71 10.3   77     8  16
## 109    59      51  6.3   79     8  17
## 110    23     115  7.4   76     8  18
## 111    31     244 10.9   78     8  19
## 112    44     190 10.3   78     8  20
## 113    21     259 15.5   77     8  21
## 114     9      36 14.3   72     8  22
## 115    NA     255 12.6   75     8  23
## 116    45     212  9.7   79     8  24
## 117   168     238  3.4   81     8  25
## 118    73     215  8.0   86     8  26
## 119    NA     153  5.7   88     8  27
## 120    76     203  9.7   97     8  28
## 121   118     225  2.3   94     8  29
## 122    84     237  6.3   96     8  30
## 123    85     188  6.3   94     8  31
##
## $`9`
##     Ozone Solar.R Wind Temp Month Day
## 124    96     167  6.9   91     9   1
## 125    78     197  5.1   92     9   2
## 126    73     183  2.8   93     9   3
## 127    91     189  4.6   93     9   4
## 128    47      95  7.4   87     9   5
## 129    32      92 15.5   84     9   6
## 130    20     252 10.9   80     9   7
## 131    23     220 10.3   78     9   8
```

```
## 132    21     230 10.9   75     9   9
## 133    24     259  9.7   73     9  10
## 134    44     236 14.9   81     9  11
## 135    21     259 15.5   76     9  12
## 136    28     238  6.3   77     9  13
## 137     9      24 10.9   71     9  14
## 138    13     112 11.5   71     9  15
## 139    46     237  6.9   78     9  16
## 140    18     224 13.8   67     9  17
## 141    13      27 10.3   76     9  18
## 142    24     238 10.3   68     9  19
## 143    16     201  8.0   82     9  20
## 144    13     238 12.6   64     9  21
## 145    23      14  9.2   71     9  22
## 146    36     139 10.3   81     9  23
## 147     7      49 10.3   69     9  24
## 148    14      20 16.6   63     9  25
## 149    30     193  6.9   70     9  26
## 150    NA     145 13.2   77     9  27
## 151    14     191 14.3   75     9  28
## 152    18     131  8.0   76     9  29
## 153    20     223 11.5   68     9  30
```

```
lapply(s, function(x) colMeans(x[, c("Ozone", "Solar.R", "Wind")]))
```

```
## $`5`
##    Ozone  Solar.R     Wind
##       NA       NA 11.62258
##
## $`6`
##     Ozone   Solar.R      Wind
##        NA 190.16667  10.26667
##
## $`7`
##      Ozone    Solar.R       Wind
##         NA 216.483871   8.941935
##
## $`8`
##    Ozone  Solar.R     Wind
##       NA       NA 8.793548
##
## $`9`
##    Ozone  Solar.R     Wind
##       NA 167.4333  10.1800
```

```
sapply(s, function(x) colMeans(x[, c("Ozone", "Solar.R", "Wind")]))
```

```
##                5         6          7        8        9
## Ozone         NA        NA         NA       NA       NA
## Solar.R       NA 190.16667 216.483871       NA 167.4333
## Wind    11.62258  10.26667   8.941935 8.793548  10.1800
```

```
sapply(s, function(x) colMeans(x[, c("Ozone", "Solar.R", "Wind")], na.rm = TRUE))
```

```
##                 5         6          7          8         9
## Ozone    23.61538  29.44444  59.115385  59.961538  31.44828
```

```
## Solar.R 181.29630 190.16667 216.483871 171.857143 167.43333
## Wind     11.62258  10.26667   8.941935   8.793548  10.18000
```

1.5.2 Splitting on More than One Level

```
x <- rnorm(10)
x
```

```
##  [1]  0.06413982 -0.36320443  0.84034366 -0.97129492 -1.16068637 -0.91098865
##  [7]  0.73235292 -0.47821776 -0.76956633 -0.59640251
```

```
f1<-gl(2,5)
f1
```

```
##  [1] 1 1 1 1 1 2 2 2 2 2
## Levels: 1 2
```

```
f2<-gl(5,2)
f2
```

```
##  [1] 1 1 2 2 3 3 4 4 5 5
## Levels: 1 2 3 4 5
```

```
interaction(f1, f2)
```

```
##  [1] 1.1 1.1 1.2 1.2 1.3 2.3 2.4 2.4 2.5 2.5
## Levels: 1.1 2.1 1.2 2.2 1.3 2.3 1.4 2.4 1.5 2.5
```

The interaction function in R generates a factor which represents the interaction of the given factors.

The interaction(f1, f2) function will return a new factor representing all combinations of f1 and f2.

The resulting factor has 10 levels, corresponding to all combinations of the levels in f1 and f2. Note that the levels are ordered first by the levels of f1, then by the levels of f2. This can be useful in statistical analyses where the interaction between factors is of interest.

Interactions can create empty levels.

```
list(f1, f2)
```

```
## [[1]]
##  [1] 1 1 1 1 1 2 2 2 2 2
## Levels: 1 2
##
## [[2]]
##  [1] 1 1 2 2 3 3 4 4 5 5
## Levels: 1 2 3 4 5
```

```
str(split(x, list(f1, f2)))
```

```
## List of 10
##  $ 1.1: num [1:2] 0.0641 -0.3632
##  $ 2.1: num(0)
##  $ 1.2: num [1:2] 0.84 -0.971
##  $ 2.2: num(0)
##  $ 1.3: num -1.16
##  $ 2.3: num -0.911
##  $ 1.4: num(0)
##  $ 2.4: num [1:2] 0.732 -0.478
##  $ 1.5: num(0)
```

```
##  $ 2.5: num [1:2] -0.77 -0.596
```

Empty levels can be dropped.

```
str(split(x, list(f1, f2), drop = TRUE))
```

```
## List of 6
##  $ 1.1: num [1:2] 0.0641 -0.3632
##  $ 1.2: num [1:2] 0.84 -0.971
##  $ 1.3: num -1.16
##  $ 2.3: num -0.911
##  $ 2.4: num [1:2] 0.732 -0.478
##  $ 2.5: num [1:2] -0.77 -0.596
```