```
R version 2.15.1 (2012-06-22) -- "Roasted Marshmallows"
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-apple-darwin9.8.0/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[R.app GUI 1.52 (6188) i386-apple-darwin9.8.0]

> # STAT 100B chapter 7 simulation
> # Purpose: To understand random sampling, central limit theorem and confidence intervals
> hospitals=read.table("http://www.stat.ucla.edu/~hqxu/stat100B/hospitals.txt", h=T)
> summary(hospitals)
   discharges          beds      
 Min.   :  14.0   Min.   : 10.0  
 1st Qu.: 311.0   1st Qu.:102.0  
 Median : 713.0   Median :233.0  
 Mean   : 814.6   Mean   :274.8  
 3rd Qu.:1186.0   3rd Qu.:393.0  
 Max.   :2844.0   Max.   :986.0  
> discharges = hospitals$discharges
> sort(discharges)
  [1]   14   22   23   35   36   41   42   48   49   52   56   57   58   64   64   66   75   76   78   79   81   84   85
 [24]   87   87   89   90   91   92   95   95   98  100  100  103  108  109  115  118  120  120  121  124  125  134  141
 [47]  153  170  173  174  182  186  194  200  209  210  213  215  216  220  222  225  227  228  229  231  231  233  233
 [70]  233  239  240  242  243  243  244  244  247  253  255  258  259  260  265  266  270  273  283  296  297  297  297
 [93]  298  301  308  308  309  310  311  315  318  321  322  323  326  327  337  345  346  355  360  360  362  362  368
[116]  371  373  373  376  377  381  383  384  389  402  410  411  414  414  416  426  427  431  439  439  440  444  446
[139]  453  456  467  467  470  471  475  479  481  486  487  490  498  504  505  518  531  534  535  538  539  543  557
[162]  557  558  573  586  587  590  590  592  592  594  601  609  610  611  621  622  625  629  632  635  637  652  662
[185]  665  669  670  670  684  689  690  695  697  703  705  707  713  715  726  726  732  744  754  764  767  773  778
[208]  778  787  788  795  808  810  811  814  824  828  834  842  852  855  858  861  872  876  876  884  885  887  889
[231]  896  906  912  915  918  925  926  928  931  935  941  944  946  951  955  956  956  956  958  966  974  976  985
[254]  986  989  995  999 1007 1009 1009 1011 1012 1016 1028 1029 1031 1036 1040 1042 1049 1060 1063 1076 1084 1089 1093
[277] 1095 1097 1097 1098 1105 1106 1126 1133 1137 1149 1150 1152 1153 1156 1160 1166 1173 1175 1186 1201 1210 1218 1219
[300] 1219 1225 1226 1231 1232 1232 1239 1239 1258 1272 1283 1287 1301 1315 1322 1326 1346 1347 1350 1355 1369 1370 1373
[323] 1376 1376 1386 1389 1390 1394 1400 1413 1415 1418 1420 1425 1453 1463 1478 1504 1509 1522 1527 1534 1547 1583 1584
[346] 1606 1617 1620 1624 1632 1634 1645 1648 1657 1665 1669 1678 1684 1705 1706 1707 1715 1719 1744 1765 1766 1780 1785
[369] 1785 1789 1805 1828 1835 1893 1894 1948 2031 2034 2051 2058 2089 2116 2135 2150 2154 2171 2190 2240 2268 2700 2766
[392] 2818 2844
> mean(discharges) # mean mu=sum(x_i)/N
[1] 814.6031
> var(discharges)  # sample variance = sum((x_i-mu)^2)/(N-1)
[1] 347766.4
> sd(discharges) # sample standard deviation
[1] 589.7173
> hist(discharges)  # Figure 7.1
> 
> # define a function to compute popu. variance
> pop.var=function(x) mean(x^2) - mean(x)^2
> pop.sd=function(x) sqrt(pop.var(x))
> pop.var(discharges)  # population variance sigma^2 = sum((x_i-mu)^2)/N
[1] 346881.5
> pop.sd(discharges) # population standard deviation
[1] 588.9665
> ## note: the population standard deviation should be 588.9665. The text gives the sample standard deviation=589.7173.
> ## the difference is small.
> 
> ?sd  # var and sd use denominator n - 1 not n.
starting httpd help server ... done
> 392*var(discharges) - 393*pop.var(discharges)  # should be 0
[1] 0
> 
> # take simple random samples with function sample()
> ?sample
> sample(1:10, 5)
[1]  5  9  7  4 10
> sample(1:10, 5)
[1]  2  9  5  4 10
> 
> # simple random sample of size n
> pop = discharges # population
```

```
> (N=length(pop)) # popu size
[1] 393
> n=64  # sample size 8, 16, 32, 64 in Figure 7.2
>
> (x=sample(pop, n))
 [1]   57 2135 1029   90 1894  416  637  228  467 1160  535 1031 2818  345   14  239 2051  194  444  297  270  247 1219
[24]  377  134  887  487  587  592 1390 1634  538  573 1149  225  928  504 1522  629  470  301  431 1076 1415   23 1097
[47]   89  999  381  346 1828  912  308   79 1226  389 1400 1133  744 1632 2154  174  622 1084
> mean(x)
[1] 785.7188
> sd(x)
[1] 624.4496
>
> (x=sample(pop, n))
 [1]  884  355  726  121 1301  103 1715  373 1287  906  594  120  244  587  925  951   89  200 1175  535 1373  926  995
[24]  989 1648 1097   92  411 1645 1415  222 1315  852  227  632   85  467  100  240  273 1547 1504 1186 1785 1042 1272
[47] 2844 1744  186 1418  814  231  896  265  247  592  876 2154  402  360 1007  498  118  778
> mean(x)
[1] 811.8906
> sd(x)
[1] 595.1521
>
> K=1000;
> xbar=rep(0,K)
> for(i in 1:K){
+   (x=sample(pop, n))
+   xbar[i]=mean(x)
+   }
> summary(xbar)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  628.6   768.6   814.5   815.2   860.5  1020.0
> mean(xbar) # mean of xbar from K simulations
[1] 815.231
> sd(xbar)  # empirical standard error of xbar from K simulations
[1] 67.67512
> # examine the distribution of K sample means (xbar).
> hist(xbar, main=paste("Histogram of xbar, n=",n))  # Figure 7.2 approximately normal for large n
>
> # proportion of xbar differed by more than 100 from mu
> sum(abs(xbar-mean(pop))>100)/K
[1] 0.148
>
> mean(pop) # population mean
[1] 814.6031
> pop.sd(pop)/sqrt(n)*sqrt((N-n)/(N-1)) # standard error of xbar
[1] 67.44589
> pop.sd(pop)/sqrt(n) # standard error of xbar, ignoring the finite population correction factor
[1] 73.62081
>
>
> # STAT 100B: confidence interval simulation
> #
> # population: patient discharges
> # simple random sample of size n
> pop = discharges # population
> (N=length(pop)) # popu size
[1] 393
> mu=mean(pop); sigma=pop.sd(pop);  #
> n=25;  # sample size n
> (sigma.xbar = pop.sd(pop)/sqrt(n)*sqrt((N-n)/(N-1))) # standard error of xbar
[1] 114.1304
>
> # 95% CI margin=1.96*sigma.xbar
> (margin=1.96*sigma.xbar)
[1] 223.6956
>
> # draw a simple random sample of n
> (x=sample(pop, n))
 [1] 2051 1287  467  243   42 1583  995  498 1707  872  444  377  446  621  778  999  153  590 1705  381 1893  611   98
[24]  505   90
> mean(x)
[1] 777.44
> # 95% CI for mu, z(.025)=1.96
> c(mean(x) - 1.96*sigma.xbar, mean(x) + 1.96*sigma.xbar)
[1]  553.7444 1001.1356
>
> ## or estiamte sigma.bar by s.xbar
> (s.xbar = sd(x)/sqrt(n)*sqrt(1-n/N)) # estimated standard error of xbar
[1] 116.0533
> c(mean(x) - 1.96*s.xbar, mean(x) + 1.96*s.xbar)
[1]  549.9756 1004.9044
>
> # do it again,
> (x=sample(pop, n))
 [1]   92   98 1029   14  315  453  767  824  855 1665  216  935   36 1744 1126  247 1418  665 1219  610 1386 1322  471
```

```
[24] 1063  918
> mean(x)  # sample mean
[1] 779.52
> # 95% CI for mu is xbar +/- margin;
> c(mean(x) - margin, mean(x) + margin)
[1]  555.8244 1003.2156
>
> # do it again,
> (x=sample(pop, n))
 [1] 1369  999  966 1084 1705 2034 1948  109 1684  311  100  439  889   92 2116  787  690   14  925  590 1355 1418 1606
[24] 1028  255
> mean(x)  # sample mean
[1] 980.52
> # 95% CI for mu is xbar +/- margin;
> c(mean(x) - margin, mean(x) + margin)
[1]  756.8244 1204.2156
>
> ## simulation A: use sigma.bar
> ### repeat the procedure K times, save the K sample means in xbar
> ###
> K=100; xbar  = rep(0, K)
> for(i in 1:K) { x=sample(pop, n); xbar[i] = mean(x) }
>
> (margin=1.96*sigma.xbar) # margin is a number
[1] 223.6956
>
> # K 95% CI's for mu is xbar +/- margin;
> ci=cbind(xbar - margin, xbar + margin)
> ci[1:min(20,K),]  # show first 20 CI's
          [,1]      [,2]
 [1,] 545.1044  992.4956
 [2,] 454.5844  901.9756
 [3,] 494.2644  941.6556
 [4,] 803.1844 1250.5756
 [5,] 829.4644 1276.8556
 [6,] 562.1044 1009.4956
 [7,] 511.8644  959.2556
 [8,] 574.9844 1022.3756
 [9,] 398.8244  846.2156
[10,] 587.8244 1035.2156
[11,] 816.2644 1263.6556
[12,] 788.5444 1235.9356
[13,] 510.1844  957.5756
[14,] 681.1844 1128.5756
[15,] 755.7844 1203.1756
[16,] 569.9844 1017.3756
[17,] 525.5444  972.9356
[18,] 637.7044 1085.0956
[19,] 582.0644 1029.4556
[20,] 592.1444 1039.5356
>
> # plot the K CI's
> matplot(cbind(xbar - margin, xbar + margin), type="n")
> for(i in 1:K) lines(c(i,i),c(xbar[i] - margin, xbar[i] + margin))
> abline(h=mu)   # add a line
>
> # count the number of CI's containing mu, which is not exactly 100(1-alpha)
> contain.mu = xbar-margin < mu & mu < xbar + margin
> sum(contain.mu) # number of CI's containing mu
[1] 91
> (1:K)[!contain.mu]  # list of CI's do not contain mu
[1]  5 11 26 27 39 41 42 74 80
>
>
> ## simulation B: estiamte sigma.bar by (s.xbar = sd(x)/sqrt(n)*sqrt(1-n/N))
> # repeat the procedure K times, save the K sample means in xbar
> K=100; sd.x = xbar = rep(0, K)
> for(i in 1:K) { x=sample(pop, n); xbar[i] = mean(x); sd.x[i]=sd(x) }
>
> (s.xbar = sd.x/sqrt(n)*sqrt(1-n/N)) # estimated standard error of xbar
  [1] 149.04532  95.87504 107.66076 132.87547  95.85226 105.59134 102.57643 107.75369 130.38309 111.13549  84.24756
 [12] 119.32188  85.32578 102.84166 103.02316 116.00645 110.32553 118.40502 123.90642 114.41863 136.74604 126.46866
 [23] 128.76736  99.79612 120.09234 107.33954 118.27789  90.80950 103.07687 107.79002 108.23991  95.70246 111.94577
 [34] 120.34387 127.12453 124.86641 128.18698 115.70020 130.48205 104.34907 100.05659  99.41573 143.34400 116.36737
 [45] 127.34119 103.17494 130.38976 109.77661 129.59046 106.21062 141.49158  87.21732 117.24083 120.87569 115.17148
 [56] 108.41477 110.21362 106.67716  77.92286 124.80054 110.76185 100.88888 125.53959 120.21340 140.25634 112.89065
 [67] 140.20494 130.41759 123.18462  79.93993 112.17361  93.60985 102.81457 139.64338  84.61022  96.46418 112.89077
 [78]  99.50410 130.67686 116.13125  98.56003 105.98414  85.58291 111.43724 118.15411 137.88032 109.97058 133.93434
 [89]  93.95223 107.28861 104.74959 115.95465 107.70985 126.39463 104.95491 124.18902 114.05880  99.05772  86.22106
[100] 140.20415
> (margin = 1.96*s.xbar)  # margin is a vector now
  [1] 292.1288 187.9151 211.0151 260.4359 187.8704 206.9590 201.0498 211.1972 255.5509 217.8256 165.1252 233.8709 167.2385
 [14] 201.5696 201.9254 227.3727 216.2380 232.0738 242.8566 224.2605 268.0222 247.8786 252.3840 195.6004 235.3810 210.3855
 [27] 231.8247 177.9866 202.0307 211.2684 212.1502 187.5768 219.4137 235.8740 249.1641 244.7382 251.2465 226.7724 255.7448
 [40] 204.5242 196.1109 194.8548 280.9542 228.0800 249.5887 202.2229 255.5639 215.1622 253.9973 208.1728 277.3235 170.9460
```

```
 [53] 229.7920 236.9164 225.7361 212.4929 216.0187 209.0872 152.7288 244.6091 217.0932 197.7422 246.0576 235.6183 274.9024
 [66] 221.2657 274.8017 255.6185 241.4419 156.6823 219.8603 183.4753 201.5166 273.7010 165.8360 189.0698 221.2659 195.0280
 [79] 256.1266 227.6173 193.1777 207.7289 167.7425 218.4170 231.5821 270.2454 215.5423 262.5113 184.1464 210.2857 205.3092
 [92] 227.2711 211.1113 247.7335 205.7116 243.4105 223.5553 194.1531 168.9933 274.8001
>
> # K 95% CI's for mu is xbar +/- margin;
> ci=cbind(xbar - margin, xbar + margin)
> ci[1:min(20,K),]  # show first 20 CI's
          [,1]      [,2]
 [1,] 639.8712 1224.1288
 [2,] 654.0449 1029.8751
 [3,] 398.5849  820.6151
 [4,] 556.2041 1077.0759
 [5,] 475.8896  851.6304
 [6,] 522.3210  936.2390
 [7,] 538.1902  940.2898
 [8,] 682.9628 1105.3572
 [9,] 721.2091 1232.3109
[10,] 664.0944 1099.7456
[11,] 483.5548  813.8052
[12,] 620.7691 1088.5109
[13,] 580.3215  914.7985
[14,] 696.1104 1099.2496
[15,] 483.2746  887.1254
[16,] 620.7073 1075.4527
[17,] 491.6020  924.0780
[18,] 514.6862  978.8338
[19,] 638.9434 1124.6566
[20,] 533.2995  981.8205
>
> # plot the K CI's
> matplot(cbind(xbar - margin, xbar + margin), type="n")
> for(i in 1:K) lines(c(i,i),c(xbar[i] - margin[i], xbar[i] + margin[i]))
> abline(h=mu)   # add a line
>
> # count the number of CI's containing mu, which is not exactly 100(1-alpha)
> contain.mu = xbar-margin < mu & mu < xbar + margin
> sum(contain.mu) # number of CI's containing mu
[1] 93
> (1:K)[!contain.mu]  # list of CI's do not contain mu
[1] 11 41 70 79 81 98 99
```