Technical University of Košice

Faculty of Electrical Engineering and Informatics



Applied Statistics

Exercise 8 - Worked Solutions in R

Jana Fortes

Reviewed English edition

```
# APPLIED STATISTICS
# Exercise 8 - Worked Solutions in R
# Jana Fortes
# Reviewed English edition, retaining all original exercises.
#
# Each exercise is self-contained. Copy all its continuation pages.
# Textbook problem statements, assumptions and interpretations use #.
# Optional packages: BSDA, fBasics, outliers. Base-R alternatives and
# direct formulas are supplied where appropriate. No auto-installation.
# install.packages(c("BSDA","fBasics","outliers"))
#
# Review changes:
# 5.10: test a median, remove zero differences, use exact sign counts.
# 5.11: account for ties and clarify R's W versus the textbook's U.
# 5.12: recalculate after merging; adjust fitted-parameter df.
# 5.13: do not give an unqualified continuous K-S conclusion with ties.
# 5.14: distinguish exponential AD from the invalid n=6 normality call.
# 5.15: add results/interpretation and normality-test conditions.
# 5.16: define ap/ss; match Grubbs scaling and Dixon tail conventions.
#

# DOCUMENTATION USED FOR CONDITIONS AND FUNCTION BEHAVIOUR
# https://search.r-project.org/CRAN/refmans/BSDA/html/SIGN.test.html
# https://stat.ethz.ch/R-manual/R-devel/library/stats/html/wilcox.test.html
# https://stat.ethz.ch/R-manual/R-devel/library/stats/html/ks.test.html
# https://search.r-project.org/CRAN/refmans/nortest/html/ad.test.html
# https://stat.ethz.ch/R-manual/R-devel/library/stats/html/shapiro.test.html
# https://search.r-project.org/CRAN/refmans/outliers/html/grubbs.test.html
# https://search.r-project.org/CRAN/refmans/outliers/html/dixon.test.html
```

```r
# EXERCISE 5.10 - SIGN TEST (TEXTBOOK p. 90)
# Twelve products are weighed [g]: 21.3, 18.5, 19.7, 26.5, 16.7,
# 23.1, 19.3, 17.5, 19.1, 24.3, 15.9, 20.8.
# At alpha = 0.05, test whether their population median is 19 g.
# The textbook says 'average', but a sign test concerns a MEDIAN.
# H0: median = 19; H1: median != 19.
# Assume independent observations and a continuous distribution.
# Symmetry and normality are not required for this median test.

x=c(21.3,18.5,19.7,26.5,16.7,23.1,19.3,17.5,19.1,24.3,15.9,20.8)
alpha=0.05
x0=19
d=x-x0
zn=sign(d)  # 1 for positive, -1 for negative, 0 for zero.
zn_plus=0
zn_minus=0
n=length(zn)
# Original counting loop, with safe iteration indices.
for (i in seq_along(zn)) {
  if (zn[i] == 1) {
    zn_plus=zn_plus+1
  } else if (zn[i] == -1) {
    zn_minus=zn_minus+1
  }
}
n0=zn_plus+zn_minus  # Exclude zero differences.
T=min(zn_plus,zn_minus)
print(c(positive=zn_plus,negative=zn_minus,nonzero=n0,T=T))

# Simpler equivalent counts (no loop needed).
print(c(positive=sum(d>0),negative=sum(d<0)))
# Under H0, the positive count is Bin(n0,0.5); T itself is not.
# For n0=12, a two-sided 5% exact rejection rule is T <= 2.
# Note the inclusive boundary for this discrete statistic.
KH=2
cat(if (T <= KH) "Reject H0 by the exact count rule.\n" else
    "Do not reject H0 by the exact count rule.\n")

# Simpler exact test included in base R.
fit=binom.test(zn_plus,n0,p=0.5,alternative="two.sided")
print(fit)
p=fit$p.value
if (p > alpha) {
  cat("p > alpha: do not reject H0.\n")
} else {
  cat("p <= alpha: reject H0.\n")
}
# Original package alternative: its statistic is Z+ (positive count).
if (requireNamespace("BSDA",quietly=TRUE)) {
  print(BSDA::SIGN.test(x,md=x0,alternative="two.sided",
                        conf.level=1-alpha))
}
# Interpretation: Z+=8, Z-=4, T=4; p=0.387695.
# At alpha=0.05, there is insufficient evidence that the median
# differs from 19 g. This does not prove that the median is 19 g.
```

```
# EXERCISE 5.11 - WILCOXON RANK-SUM TEST (TEXTBOOK p. 92)
# A pharmaceutical company compares the percentages of patients
# reporting adverse effects for two drugs:
# Drug A: 31,33,29,35; Drug B: 25,34,28,24,33,40.
# Use the Wilcoxon two-sample test at alpha=0.05 to compare them.
# H0: the population distributions are equal; H1: they differ.
# Assume independent observations and independent samples.
# Comparable shapes are needed for a pure location/median reading.

x=c(31,33,29,35)
y=c(25,34,28,24,33,40)
alpha=0.05
n1=length(x)
n2=length(y)
# Direct calculation, as in the textbook; tied ranks are averaged.
ranks=rank(c(x,y),ties.method="average")
T1=sum(ranks[seq_len(n1)])
T2=sum(ranks[n1+seq_len(n2)])
U1=n1*n2+n1*(n1+1)/2-T1
U2=n1*n2+n2*(n2+1)/2-T2
U=min(U1,U2)
print(c(T1=T1,T2=T2,U1=U1,U2=U2,U=U))
# Textbook critical value U_alpha(4,6)=2 is for untied data.
# There is a tie at 33 here; use a tie-aware calculation instead
# of treating the untied table as an exact test for these data.

# Original stats::wilcox.test, with its options made explicit.
fit=wilcox.test(x,y,alternative="two.sided",exact=FALSE,
                correct=TRUE,conf.level=1-alpha)
print(fit)
p=fit$p.value
if (p > alpha) {
  cat("p > alpha: do not reject H0.\n")
} else {
  cat("p <= alpha: reject H0.\n")
}
# exact=FALSE requests the tie-corrected normal approximation;
# correct=TRUE retains the continuity correction.
# conf.level affects an interval only when conf.int=TRUE is requested.
# R reports W=14.5, which is U2 under this textbook's convention.
# The textbook's minimum statistic is U=9.5.
# Expected approximate p=0.668870: do not reject H0 at 0.05.
# There is insufficient evidence of a difference between drugs.
# The test does not establish equal adverse-effect percentages.
```

```r
# EXERCISE 5.12 - PEARSON GOODNESS OF FIT (TEXTBOOK p. 94)
# Observed class intervals: (0,1],(1,2],(2,3],(3,4],(4,5].
# Observed frequencies: 60,20,11,5,4.
# At alpha=0.05, test goodness of fit to an exponential population.
# H0: an exponential model fits; H1: it does not.
# The chapter estimates the exponential MEAN from class midpoints.
# Observations are independent; the chi-square calibration is
# approximate, particularly because the parameter uses grouped data.

t=seq(0,5,by=1)
ni=c(60,20,11,5,4)
alpha=0.05
n=sum(ni)
zi=(t[1:5]+t[2:6])/2
x=rep(zi,ni)  # Replicate each midpoint by its frequency.
m=mean(x)
k=length(ni)
r=1  # One exponential parameter is fitted.
t[k+1]=Inf  # Last theoretical cell covers the remaining upper tail.
gt=1-exp(-t/m)
# Simpler equivalent CDF: pexp(t,rate=1/m).
pi=diff(gt)
expected=n*pi
ch=(ni-expected)^2/expected
chi2=sum(ch)
# Original calculation table, without NaN placeholder rows.
tab=cbind(lower=t[1:k],upper=t[2:(k+1)],probability=pi,
          expected=expected,observed=ni,contribution=ch)
print(tab)
print(c(unmerged_chi2=chi2))
# Original last two expected counts are <5: merge their cells.
# Preserve unmerged data rather than replacing entries by NaN.
ni_merged=c(ni[1:3],sum(ni[4:5]))
pi_merged=c(pi[1:3],sum(pi[4:5]))
k=length(ni_merged)
expected_merged=n*pi_merged
stopifnot(all(expected_merged>=5),abs(sum(pi_merged)-1)<1e-10)
ch_merged=(ni_merged-expected_merged)^2/expected_merged
# Correction: recompute the statistic AFTER merging classes.
chi2=sum(ch_merged)
df=k-r-1
q=qchisq(1-alpha,df)
print(cbind(observed=ni_merged,probability=pi_merged,
            expected=expected_merged,contribution=ch_merged))
print(c(chi2=chi2,df=df,critical=q))
cat("Critical region: (",q,", Inf)\n")
cat(if (chi2>q) "Reject H0.\n" else "Do not reject H0.\n")

# Original stats::chisq.test alternative, with both corrections:
# (1) use the MERGED counts; (2) deduct the fitted parameter from df.
# chisq.test() alone treats probabilities as specified, so it uses
# k-1 degrees of freedom, not the fitted-model value k-r-1.
check=chisq.test(ni_merged,p=pi_merged)
stopifnot(isTRUE(all.equal(unname(check$statistic),chi2)))
check$parameter=c(df=df)
check$p.value=pchisq(chi2,df=df,lower.tail=FALSE)
check$method="Pearson chi-square GOF with fitted-parameter df adjustment"
print(check)
p=check$p.value
if (p > alpha) {
  cat("p > alpha: do not reject H0.\n")
} else {
  cat("p <= alpha: reject H0.\n")
}
# m=1.23; merged chi2=1.237143; df=2; p=0.538713.
# Do not reject the exponential model at 0.05. This is an
# approximate grouped-data test, not proof of exponentiality.
# The last class must cover the full theoretical support.
# Here it is (3,Inf) after merging, as in the textbook.
```

```
# EXERCISE 5.13 - KOLMOGOROV-SMIRNOV (TEXTBOOK p. 98)
# Values xj: 0.5,1.5,2.5,3.5,4.5; frequencies: 60,20,11,5,4.
# At alpha=0.05, compare with an exponential distribution whose
# mean parameter is specified as lambda=1.23.
# H0: F=Ex(1.23); H1: F differs from this specified model.

xi=seq(0.5,4.5,by=1)
ni=c(60,20,11,5,4)
n=sum(ni)
alpha=0.05
lambda=1.23
x=rep(xi,ni)
# Original empirical-CDF calculation, written without hard-coded sizes.
nn=length(xi)
nii=numeric(nn+1)
for (i in seq_len(nn)) nii[i+1]=nii[i]+ni[i]
f=nii/n
g=1-exp(-xi/lambda)  # Equivalent: pexp(xi,rate=1/lambda).
di_minus=numeric(nn)
di_plus=numeric(nn)
for (i in seq_len(nn)) di_minus[i]=abs(g[i]-f[i])
for (i in seq_len(nn)) di_plus[i]=abs(g[i]-f[i+1])
d=max(di_minus,di_plus)
print(cbind(xi,ni,G=g,Fn_left=f[1:nn],Fn_right=f[2:(nn+1)],
            d_minus=di_minus,d_plus=di_plus))
# Simpler equivalent empirical CDF values:
Fn_right=cumsum(ni)/n
Fn_left=c(0,head(Fn_right,-1))
d_simple=max(abs(g-Fn_left),abs(g-Fn_right))
stopifnot(isTRUE(all.equal(d,d_simple)))
# Textbook table: critical value for n=100, alpha=0.05.
KH=0.1358
print(c(D=d,textbook_critical=KH))
# D=0.334034 exceeds 0.1358 in the textbook calculation.
# HOWEVER, the supplied data have many ties (or are class midpoints).
# Standard continuous one-sample K-S critical values/p-values are
# not calibrated for these data. Do not claim a valid 5% rejection
# using that comparison alone. Use a method respecting grouping,
# such as Pearson's grouped-data approach in Exercise 5.12.

# Preserve the original package call for reference only:
# ks.test(x,"pexp",1/lambda,exact=TRUE)
# exact=TRUE does not make a continuous K-S test exact with ties.
# The safe conditional check below runs only for untied raw data.
if (anyDuplicated(x)==0L) {
  fit=ks.test(x,"pexp",rate=1/lambda,exact=TRUE)
  print(fit)
  p=fit$p.value
  cat(if (p>alpha) "Do not reject H0.\n" else "Reject H0.\n")
} else {
  cat("Standard continuous K-S calibration is inapplicable: ties.\n")
}
# Also, lambda must be fixed independently of the tested data.
# If lambda was estimated from x, standard K-S calibration does
# not apply even without ties; use fitted-model calibration.
```

```r
# EXERCISE 5.14 - ANDERSON-DARLING (TEXTBOOK p. 100)
# At alpha=0.05, assess whether 10,20,35,4,30,1 come from an
# exponential population. Its mean lambda is estimated by mean(x).
# H0: a fitted exponential model; H1: lack of fit.
# Assume independent, positive, continuous observations.

x=c(10,20,35,4,30,1)
n=length(x)
alpha=0.05
stopifnot(all(is.finite(x)),all(x>0))
i=seq_len(n)
m=mean(x)
vr=sort(x)
gt=1-exp(-vr/m)  # Equivalent: pexp(vr,rate=1/m).
l1=log(gt)
l2=log1p(-gt)  # log(1-gt), with better numerical accuracy.
l22=rev(l2)  # Simpler than the original hard-coded l2[6:1].
ll=l1+l22
ad=((1/n)*sum((1-2*i)*ll))-n
print(cbind(i,vr,gt,l1,l22,ll))
print(c(mean=m,AD=ad))
# Use the exponential-model critical value given in the chapter.
KH=1.321
cat("Textbook critical region: (",KH,", Inf)\n")
cat(if (ad>KH) "Reject H0.\n" else "Do not reject H0.\n")
# AD=0.258082 <1.321: the chapter's table rule does not reject.
# This does not establish exponentiality; n=6 provides little power.
# Critical values depend on the distribution and parameter fitting.

# Simpler reusable exponential AD-statistic calculation.
AD_exp=function(z) {
  stopifnot(length(z)>1,all(is.finite(z)),all(z>0))
  z=sort(z)
  n=length(z)
  rate=1/mean(z)
  log_F=log(-expm1(-rate*z))
  log_survival=-rate*z
  -n-sum((2*seq_len(n)-1)*(log_F+rev(log_survival)))/n
}
stopifnot(isTRUE(all.equal(ad,AD_exp(x))))

# Optional fitted-model parametric bootstrap for this small sample.
# Refit the mean in EVERY simulated sample. The exponential AD
# statistic is scale-invariant, so simulate from Exp(mean=1).
# Switch this flag to TRUE to run the simulation.
run_bootstrap=FALSE
if (run_bootstrap) {
  set.seed(514)
  B=9999
  simulated_AD=replicate(B,AD_exp(rexp(n,rate=1)))
  p_boot=(1+sum(simulated_AD>=ad))/(B+1)
  print(c(AD=ad,bootstrap_p=p_boot))
  cat(if (p_boot>alpha) "Do not reject H0.\n" else "Reject H0.\n")
}

# Original extra function: fBasics::adTest(x).
# It is an ANDERSON-DARLING NORMALITY test, not an exponential test.
# Its nortest-based normality implementation requires more than 7
# observations. Here n=6: do not run it on these data.
# Preserve the function as an explicitly labelled optional template.
if (n>7 && requireNamespace("fBasics",quietly=TRUE)) {
  normality_result=fBasics::adTest(x)
  print(normality_result)  # Different null: normality, not exponentiality.
} else {
  cat("Original AD normality check not run: n=6; wrong target here.\n")
}
```

```r
# EXERCISE 5.15 - SHAPIRO-WILK NORMALITY (TEXTBOOK p. 102)
# At alpha=0.05, test whether 4.05,4.15,4.20,4.25,4.30,4.35,
# 4.45,4.55 are compatible with a normal population.
# H0: the population is normal; H1: it is not normal.
# Assume independent observations. stats::shapiro.test requires
# 3 to 5000 non-missing observations and nonconstant data.

x=c(4.05,4.15,4.2,4.25,4.3,4.35,4.45,4.55)
alpha=0.05
stopifnot(length(x)>=3,length(x)<=5000,sd(x)>0)
# Original fBasics alternative, if installed.
if (requireNamespace("fBasics",quietly=TRUE)) {
  print(fBasics::shapiroTest(x))
}
# Simpler: stats is included with R; no extra package needed.
fit=shapiro.test(x)
print(fit)
p=fit$p.value
if (p > alpha) {
  cat("p > alpha: do not reject H0.\n")
} else {
  cat("p <= alpha: reject H0.\n")
}
# W approximately 0.990000, p approximately 0.995076.
# Do not reject normality at 0.05. A high p-value does not prove
# normality, especially in this very small sample.
# Optional visual check:
qqnorm(x,main="Exercise 5.15: normal Q-Q plot")
qqline(x)
```

```r
# EXERCISE 5.16 - OUTLIER TESTS (TEXTBOOK p. 106)
# Values: 5.83,5.80,5.85,5.88,5.84,5.83,5.96,5.78,5.82,
# 5.81,5.86,5.82. At alpha=0.05, test whether 5.96 is an outlier
# using a) Grubbs' test and b) Dixon's test.
# H0: no outlier at the specified upper extreme; H1: an upper outlier.
# The population is assumed normal in the exercise. These outlier
# tests are PARAMETRIC, not distribution-free nonparametric tests.
# For new data, check normality plausibility using context and plots;
# a non-significant normality test alone does not prove normality.

xx=c(5.83,5.8,5.85,5.88,5.84,5.83,5.96,5.78,5.82,5.81,5.86,5.82)
x=sort(xx)
alpha=0.05
n=length(x)
# Fix the undefined variables in the original Grubbs calculation.
ap=mean(x)
ss=sd(x)
stopifnot(n>=3,ss>0)
boxplot(x,main="Exercise 5.16: possible upper outlier")

# a) Grubbs' test: direct formula and matching critical region.
G=(x[n]-ap)/ss
T_scaled=G*sqrt(n/(n-1))  # Statistic used in the textbook.
# The textbook lists T_critical=2.387; this does not match the
# reference cutoff below. Retain it as a textbook reference only.
# Never compare a critical value for scaled T directly to package G.
# For a consistent specified-upper-tail Grubbs calibration:
tcrit=qt(1-alpha/n,n-2)
G_critical=(n-1)/sqrt(n)*sqrt(tcrit^2/(n-2+tcrit^2))
T_critical=G_critical*sqrt(n/(n-1))
print(c(mean=ap,SD=ss,G=G,G_critical=G_critical,
        T_scaled=T_scaled,T_critical=T_critical))
cat(if (G>G_critical) "Reject H0: upper outlier.\n" else
    "Do not reject H0.\n")
# G=2.590727; T_scaled=2.705926. The critical-region test rejects.
# The scaling differs, but a correctly transformed critical value
# gives the same decision. Do not mix G and T critical tables.
if (requireNamespace("outliers",quietly=TRUE)) {
  fit=outliers::grubbs.test(x,type=10,two.sided=FALSE)
  print(fit)
  p=fit$p.value
  cat(if (p>alpha) "Do not reject H0.\n" else "Reject H0.\n")
  # Expected specified-tail p approximately 0.007276.
  # If the tail was selected after inspection, also consider both tails:
  print(outliers::grubbs.test(x,type=10,two.sided=TRUE))
} else {
  cat("Package check skipped; direct Grubbs calculation is complete.\n")
}
```

```r
# 5.16 (CONTINUED)

# b) Dixon type 10: upper gap divided by the full sample range.
q=(x[n]-x[n-1])/(x[n]-x[1])
# type=10 must be explicit. With n=12, type=0 chooses another variant.
stopifnot(n>=3,n<=30,x[n]>x[1])
KH=0.376  # Chapter's specified-tail critical value for n=12, alpha=0.05.
if (requireNamespace("outliers",quietly=TRUE)) {
  KH=outliers::qdixon(alpha,n,type=10)
}
print(c(Q=q,critical=KH))
cat(if (q>KH) "Reject H0: upper outlier.\n" else
    "Do not reject H0.\n")
if (requireNamespace("outliers",quietly=TRUE)) {
  # Match the one-sided manual critical value explicitly.
  fit=outliers::dixon.test(x,type=10,two.sided=FALSE)
  print(fit)
  p=fit$p.value
  cat(if (p>alpha) "Do not reject H0.\n" else "Reject H0.\n")
  # Preserve the original default two-sided package comparison too.
  # Its critical value uses alpha/2, not alpha, in qdixon().
  KH_two=outliers::qdixon(alpha/2,n,type=10)
  print(c(Q=q,two_sided_critical=KH_two))
  print(outliers::dixon.test(x,type=10,two.sided=TRUE))
}
# Q=0.444444. It exceeds both 0.376 (specified tail) and 0.426
# (two-sided type-10 critical value). Both conventions reject at 0.05.
# Investigate 5.96 and document any exclusion; do not delete a value
# automatically just because a test or boxplot flags it.
```