Norwegian University
of Science and Technology
Department of Mathematical Sciences

Statistical modelling for biologists and biotechnologists, ST2304

Trial exam

Permitted aids: One handwritten yellow A4 paper, pocket calculator, "Tabeller og formler i statistikk" (Tapir forlag), K. Rottmann: Matematisk formelsamling.

Help pages for some R functions you may need to use follows on page 6.

Problem 1

Suppose that $X \sim \text{bin}(n, p)$ and that we want to test the null hypothesis $H_0 : p \leq p_0$ against the alternative hypothesis $H_1 : p > p_0$. The power of a test based on a normal approximation is then given by

$$\gamma(p) = \phi\left(\frac{p - p_0}{\sqrt{p(1-p)/n}} - z_\alpha \sqrt{\frac{p_0(1-p_0)}{p(1-p)}}\right), \tag{1}$$

where ϕ is the cumulative density function of the standard normal distribution.

a) Write an R function `power` which takes α, p_0, p and n as input arguments an which returns the power γ.

b) Explain briefly what would happen if you make the following call to your function `power`,

```
> power(p0=0.5, p = seq(0,1,length=11), alpha=0.05, n=50)
```

in particular, what is the effect of the second argument being a vector?

c) Draw a qualitatively correct graph (by hand) showing the relationship between γ and p given the choice of α and p_0 in the previous point.

Problem 2 A biologist researching the effects of climate change has collected the follow data set containing the number of birch trees inside different rectangular sampling sites.

```
> birchcounts
   trees area altitude time
1     35  100      740 1980
2     28  100      820 1980
3     41  100      720 1980
4     27  100      960 1980
5     12   50      830 1980
6     15   50      720 1980
7     12   50      850 1980
8     14   50      870 1980
9     60  100      740 2010
10    55  100      820 2010
11    59  100      720 2010
12    48  100      960 2010
13    28   50      830 2010
14    27   50      720 2010
15    14   50      850 2010
16    26   50      870 2010
```

The counts have been taken at two points in time in 1980 and in 2010 (the variable `time`) and at different altitudes (the variable `altitude` in meters above see level). The variable `area` is the area of each sampling rectangle (in square meters). The variable `trees` contains the number of individual birch trees inside each sampling rectangle. Note that `time` is encoded as a factor with two levels and 1980 as the reference level. All other variables are numerical.

Suppose that we fit a generalized linear model to these data in R as follows.

```
> attach(birchcounts)
> model <- glm(trees~altitude+time+log(area),family=poisson)
> summary(model)

Call:
glm(formula = trees ~ altitude + time + log(area), family = poisson)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.92027  -0.30845   0.00561   0.26495   1.00634

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
```

```
(Intercept) -1.2496809  0.7900113  -1.582   0.1137
altitude    -0.0011961  0.0005415  -2.209   0.0272 *
time2010     0.5439660  0.0926788   5.869 4.37e-09 ***
log(area)    1.2357436  0.1416743   8.722  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 134.277  on 15  degrees of freedom
Residual deviance:   7.155  on 12  degrees of freedom
AIC: 97.465

Number of Fisher Scoring iterations: 4
```

a) Explain why it is a reasonable starting assumption that the number of birch trees inside each sampling rectangle is Poisson distributed and why a log link-function is reasonable.

b) Write down a formula for the fitted model in mathematical notation. Based on the this formula and the above parameter estimates, what is the expected number of birch trees inside a 200 square meter rectangle at an altitude of 1000 meters above see level in 2010?

c) According to the fitted model, by which factor does the expected number of birch trees increase if increasing the area of a sampling rectangle from 50 to 100 square meters? Does this agree with your prior beliefs? Discuss how you might change the model such that the magnitude of the effect of area is assumed a priori rather than estimated?

d) Aret here any signs of over- or under-dispersion in the data? Discuss briefly some mechanisms which may lead to over- and under-dispersion in this case.

Problem 3

Suppose that we observe the lifespans $x_1, x_2, \ldots, x_3$ of $n = 100$ individuals of the butterfly species *Bia acterion*. We believe that these lifespans follow a gamma distribution with density function

$$f(x) = \frac{1}{b^a \Gamma(a)} x^{a-1} e^{-x/b} \tag{2}$$

for $x > 0$. Suppose that the vector `x` in R contains the observations.

a) Write a function `lnL` that takes a vector `p` containing the parameters a and b and a vector `x` containing the observations and that returns the negative log likelihood as it's value.

Suppose that we then run the following commands in R:

```
> fit <- optim(c(1,1),lnL,x=x,hessian=TRUE)
> fit
$par
[1] 2.448359 8.256129

$value
[1] 382.7134

$counts
function gradient
      59       NA

$convergence
[1] 0

$message
NULL

$hessian
         [,1]      [,2]
[1,] 50.28615 12.112213
[2,] 12.11221  3.592935

> solve(fit$hessian)
           [,1]       [,2]
[1,]  0.1057706 -0.3565652
[2,] -0.3565652  1.4803481
```

b) What are the maximum likelihood estimates $\hat{a}$ and $\hat{b}$ of the parameters a and b?

c) What are the standard errors of the estimates?

d) Compute an estimate of the expected lifespan EX. Find the standard error of this estimate using the delta method. Note that an expression for EX is available in the help page.

e) Suppose that we also fitted an exponential model to the data and obtained a (positive) maximum log likelihood of -400.6. If we rely on a certain aymptotic approximation, can we on the basis of this reject the simpler exponential model in favour of the gamma model?

```
Normal                  package:stats                  R Documentation

The Normal Distribution

Description:

     Density, distribution function, quantile function and random
     generation for the normal distribution with mean equal to 'mean'
     and standard deviation equal to 'sd'.

Usage:

     dnorm(x, mean = 0, sd = 1, log = FALSE)
     pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
     qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
     rnorm(n, mean = 0, sd = 1)

Arguments:

     x,q: vector of quantiles.

       p: vector of probabilities.

       n: number of observations. If 'length(n) > 1', the length is
          taken to be the number required.

    mean: vector of means.

      sd: vector of standard deviations.

log, log.p: logical; if TRUE, probabilities p are given as log(p).

lower.tail: logical; if TRUE (default), probabilities are P[X <= x]
          otherwise, P[X > x].

Details:

     If 'mean' or 'sd' are not specified they assume the default values
     of '0' and '1', respectively.

     The normal distribution has density

           f(x) = 1/(sqrt(2 pi) sigma) e^-((x - mu)^2/(2 sigma^2))

     where mu is the mean of the distribution and sigma the standard
     deviation.

     'qnorm' is based on Wichura's algorithm AS 241 which provides
     precise results up to about 16 digits.

Value:

     'dnorm' gives the density, 'pnorm' gives the distribution
     function, 'qnorm' gives the quantile function, and 'rnorm'
     generates random deviates.

Source:

     For 'pnorm', based on

     Cody, W. D. (1993) Algorithm 715: SPECFUN - A portable FORTRAN
     package of special function routines and test drivers.  _ACM
     Transactions on Mathematical Software_ *19*, 22-32.

     For 'qnorm', the code is a C translation of

     Wichura, M. J. (1988) Algorithm AS 241: The Percentage Points of
     the Normal Distribution.  _Applied Statistics_, *37*, 477-484.

     For 'rnorm', see RNG for how to select the algorithm and for
     references to the supplied methods.

References:

     Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S
     Language_.  Wadsworth & Brooks/Cole.

     Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) _Continuous
     Univariate Distributions_, volume 1, chapter 13.  Wiley, New York.

See Also:

     Distributions for other standard distributions, including 'dlnorm'
     for the _Log_normal distribution.

Examples:

     require(graphics)

     dnorm(0) == 1/ sqrt(2*pi)
     dnorm(1) == exp(-1/2)/ sqrt(2*pi)
     dnorm(1) == 1/ sqrt(2*pi*exp(1))

     ## Using "log = TRUE" for an extended range :
     par(mfrow=c(2,1))
     plot(function(x) dnorm(x, log=TRUE), -60, 50,
          main = "log { Normal density }")
     curve(log(dnorm(x)), add=TRUE, col="red",lwd=2)
     mtext("dnorm(x, log=TRUE)", adj=0)
     mtext("log(dnorm(x))", col="red", adj=1)

     plot(function(x) pnorm(x, log.p=TRUE), -50, 10,
          main = "log { Normal Cumulative }")
     curve(log(pnorm(x)), add=TRUE, col="red",lwd=2)
     mtext("pnorm(x, log=TRUE)", adj=0)
     mtext("log(pnorm(x))", col="red", adj=1)

     ## if you want the so-called 'error function'
     erf <- function(x) 2 * pnorm(x * sqrt(2)) - 1
     ## (see Abramowitz and Stegun 29.2.29)
     ## and the so-called 'complementary error function'
     erfc <- function(x) 2 * pnorm(x * sqrt(2), lower = FALSE)
     ## and the inverses
     erfinv <- function (x) qnorm((1 + x)/2)/sqrt(2)
     erfcinv <- function (x) qnorm(x/2, lower = FALSE)/sqrt(2)

--------------------------------------------------------------------------
seq                     package:base                     R Documentation

Sequence Generation

Description:

     Generate regular sequences.  'seq' is a standard generic with a
     default method.  'seq.int' is a primitive which can be much faster
     but has a few restrictions.  'seq_along' and 'seq_len' are very
     fast primitives for two common cases.

Usage:

     seq(...)

     ## Default S3 method:
     seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
         length.out = NULL, along.with = NULL, ...)

     seq.int(from, to, by, length.out, along.with, ...)

     seq_along(along.with)
     seq_len(length.out)

Arguments:

     ...: arguments passed to or from methods.

from, to: the starting and (maximal) end value of the sequence.

      by: number: increment of the sequence.

length.out: desired length of the sequence.  A non-negative number,
          which for 'seq' and 'seq.int' will be rounded up if
          fractional.

along.with: take the length from the length of this argument.

Details:

     The interpretation of the unnamed arguments of 'seq' and 'seq.int'
     is _not_ standard, and it is recommended always to name the
     arguments when programming.

     'seq' is generic, and only the default method is described here.
     Note that it dispatches on the class of the *first* argument
     irrespective of argument names.  This can have unintended
     consequences if it is called with just one argument intending this
     to be taken as 'along.with': it is much better to use 'seg_along'
```

in that case.

'seq.int' is an internal generic which dispatches on methods for '"seq"' based on the class of the first supplied argument (before argument matching).

Typical usages are

```
seq(from, to)
seq(from, to, by= )
seq(from, to, length.out= )
seq(along.with= )
seq(from)
seq(length.out= )
```

The first form generates the sequence 'from, from+/-1, ..., to' (identical to 'from:to').

The second form generates 'from, from+by', ..., up to the sequence value less than or equal to 'to'. Specifying 'to - from' and 'by' of opposite signs is an error. Note that the computed final value can go just beyond 'to' to allow for rounding error, but (as from R 2.9.0) is truncated to 'to'. ('Just beyond' is by up to 1e-10 times 'abs(from - to)' as from R 2.11.0: previously it was 1e-7 times.)

The third generates a sequence of 'length.out' equally spaced values from 'from' to 'to'. ('length.out' is usually abbreviated to 'length' or 'len', and 'seq_len' is much faster.)

The fourth form generates the integer sequence '1, 2, ..., length(along.with)'. ('along.with' is usually abbreviated to 'along', and 'seq_along' is much faster.)

The fifth form generates the sequence '1, 2, ..., length(from)' (as if argument 'along.with' had been specified), _unless_ the argument is numeric of length 1 when it is interpreted as '1:from' (even for 'seq(0)' for compatibility with S). Using either 'seq_along' or 'seq_len' is much preferred (unless strict S compatibility is essential).

The final form generates the integer sequence '1, 2, ..., length.out' unless 'length.out = 0', when it generates 'integer(0)'.

Very small sequences (with 'from - to' of the order of 10^{-14} times the larger of the ends) will return 'from'.

For 'seq' (only), up to two of 'from', 'to' and 'by' can be supplied as complex values provided 'length.out' or 'along.with' is specified. More generally, the default method of 'seq' will handle classed objects with methods for the 'Math', 'Ops' and 'Summary' group generics.

'seq.int', 'seq_along' and 'seq_len' are primitive.

Value:

'seq.int' and the default method of 'seq' for numeric arguments return a vector of type '"integer"' or '"double"': programmers should not rely on which.

'seq_along' and 'seq_len' always return an integer vector.

References:

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S Language_. Wadsworth & Brooks/Cole.

See Also:

The methods 'seq.Date' and 'seq.POSIXt'.

':', 'rep', 'sequence', 'row', 'col'.

Examples:

```
seq(0, 1, length.out=11)
seq(stats::rnorm(20)) # effectively 'along'
seq(1, 9, by = 2)     # matches 'end'
seq(1, 9, by = pi)    # stays below 'end'
seq(1, 6, by = 3)
seq(1.575, 5.125, by=0.05)
seq(17) # same as 1:17, or even better seq_len(17)
```

---

plot package:graphics R Documentation

Generic X-Y Plotting

Description:

Generic function for plotting of R objects. For more details about the graphical parameter arguments, see 'par'.

Usage:

```
plot(x, y, ...)
```

Arguments:

x: the coordinates of points in the plot. Alternatively, a single plotting structure, function or _any R object with a 'plot' method_ can be provided.

y: the y coordinates of points in the plot, _optional_ if 'x' is an appropriate structure.

...: Arguments to be passed to methods, such as graphical parameters (see 'par'). Many methods will accept the following arguments:

'type' what type of plot should be drawn. Possible types are

- '"p"' for *p*oints,
- '"l"' for *l*ines,
- '"b"' for *b*oth,
- '"c"' for the lines part alone of '"b"',
- '"o"' for both '*o*verplotted',
- '"h"' for '*h*istogram' like (or 'high-density') vertical lines,
- '"s"' for stair *s*teps,
- '"S"' for other *s*teps, see 'Details' below,
- '"n"' for no plotting.

All other 'type's give a warning or an error; using, e.g., 'type = "punkte"' being equivalent to 'type = "p"' for S compatibility. Note that some methods, e.g. 'plot.factor', do not accept this.

'main' an overall title for the plot: see 'title'.

'sub' a sub title for the plot: see 'title'.

'xlab' a title for the x axis: see 'title'.

'ylab' a title for the y axis: see 'title'.

'asp' the y/x aspect ratio, see 'plot.window'.

Details:

For simple scatter plots, 'plot.default' will be used. However, there are 'plot' methods for many R objects, including 'function's, 'data.frame's, 'density' objects, etc. Use 'methods(plot)' and the documentation for these.

The two step types differ in their x-y preference: Going from (x1,y1) to (x2,y2) with x1 < x2, 'type = "s"' moves first horizontal, then vertical, whereas 'type = "S"' moves the other way around.

See Also:

'plot.default', 'plot.formula' and other methods; 'points', 'lines', 'par'.

```
     For X-Y-Z plotting see 'contour', 'persp' and 'image'.

Examples:

     require(stats)
     plot(cars)
     lines(lowess(cars))

     plot(sin, -pi, 2*pi)

     ## Discrete Distribution Plot:
     plot(table(rpois(100,5)), type = "h", col = "red", lwd=10,
          main="rpois(100,lambda=5)")

     ## Simple quantiles/ECDF, see ecdf() {library(stats)} for a better one:
     plot(x <- sort(rnorm(47)), type = "s", main = "plot(x, type = \"s\")")
     points(x, cex = .5, col = "dark red")
--------------------------------------------------------------------------
curve                 package:graphics                 R Documentation

Draw Function Plots

Description:

     Draws a curve corresponding to the given function or, for
     'curve()' also an expression (in 'x') over the interval
     '[from,to]'.

Usage:

     curve(expr, from = NULL, to = NULL, n = 101, add = FALSE,
           type = "l", ylab = NULL, log = NULL, xlim = NULL, ...)

     ## S3 method for class 'function'
     plot(x, y = 0, to = 1, from = y, xlim = NULL, ...)

Arguments:

    expr: a call or an expression written as a function of 'x', or
          alternatively the name of a function which will be plotted.

       x: a 'vectorizing' numeric R function.

 from,to: the range over which the function will be plotted.

       n: integer; the number of x values at which to evaluate.

     add: logical; if 'TRUE' add to already existing plot.

    xlim: numeric of length 2; if specified, it serves as default for
          'c(from, to)'.

    type: plot type: see 'plot.default'.

       y: alias for 'from' for compatibility with 'plot()'

ylab, log, ...: labels and graphical parameters can also be specified
          as arguments.  'plot.function' passes all these to 'curve'.

Details:

     The evaluation of 'expr' is at 'n' points equally spaced over the
     range '[from, to]', possibly adapted to log scale.  The points
     determined in this way are then joined with straight lines.
     'x(t)' or 'expr' (with 'x' inside) must return a numeric of the
     same length as the argument 't' or 'x'.

     For 'curve()', if either of 'from' or 'to' is 'NULL', it defaults
     to the corresponding element of 'xlim', and 'xlim' defaults to the
     x-limits of the current plot.  For 'plot(<function>, ..)', the
     defaults for (from, to) are (0, 1).

     'log' is taken from the current plot only when 'add' is true, and
     otherwise defaults to '""' indicating linear scales on both axes.

     This used to be a quick hack which now seems to serve a useful
     purpose, but can give bad results for functions which are not
     smooth.

     For expensive-to-compute 'expr'essions, you should use smarter
     tools.

Value:

     A list with components 'x' and 'y' of the points that were drawn
     is returned invisibly.

See Also:

     'splinefun' for spline interpolation, 'lines'.

Examples:

     plot(qnorm)
     plot(qlogis, main = "The Inverse Logit : qlogis()")
     abline(h=0, v=0:2/2, lty=3, col="gray")

     curve(sin, -2*pi, 2*pi)
     curve(tan, main = "curve(tan)  --> same x-scale as previous plot")

     op <- par(mfrow=c(2,2))
     curve(x^3-3*x, -2, 2)
     curve(x^2-2, add = TRUE, col = "violet")

     ## simple and sophisticated, quite similar:
     plot(cos, -pi,  3*pi)
     plot(cos, xlim = c(-pi,3*pi), n = 1001, col = "blue", add=TRUE)

     chippy <- function(x) sin(cos(x)*exp(-x/2))
     curve(chippy, -8, 7, n=2001)
     plot (chippy, -8, -5)

     for(ll in c("","x","y","xy"))
        curve(log(1+x), 1,100, log=ll, sub=paste("log= '",ll,"'",sep=""))
     par(op)
--------------------------------------------------------------------------
GammaDist               package:stats               R Documentation

The Gamma Distribution

Description:

     Density, distribution function, quantile function and random
     generation for the Gamma distribution with parameters 'shape' and
     'scale'.

Usage:

     dgamma(x, shape, rate = 1, scale = 1/rate, log = FALSE)
     pgamma(q, shape, rate = 1, scale = 1/rate, lower.tail = TRUE,
            log.p = FALSE)
     qgamma(p, shape, rate = 1, scale = 1/rate, lower.tail = TRUE,
            log.p = FALSE)
     rgamma(n, shape, rate = 1, scale = 1/rate)

Arguments:

    x, q: vector of quantiles.

       p: vector of probabilities.

       n: number of observations. If 'length(n) > 1', the length is
          taken to be the number required.

    rate: an alternative way to specify the scale.

shape, scale: shape and scale parameters.  Must be positive, 'scale'
          strictly.

log, log.p: logical; if 'TRUE', probabilities/densities p are returned
          as log(p).

lower.tail: logical; if TRUE (default), probabilities are P[X <= x],
          otherwise, P[X > x].

Details:

     If 'scale' is omitted, it assumes the default value of '1'.

     The Gamma distribution with parameters 'shape' = a and 'scale' = s
     has density

                  f(x)= 1/(s^a Gamma(a)) x^(a-1) e^-(x/s)

     for x >= 0, a > 0 and s > 0.  (Here Gamma(a) is the function
     implemented by R's 'gamma()' and defined in its help.  Note that
```

```
     a=0 corresponds to the trivial distribution with all mass at point
     0.)

     The mean and variance are E(X) = a*s and Var(X) = a*s^2.

     The cumulative hazard H(t) = - log(1 - F(t)) is '-pgamma(t, ...,
     lower = FALSE, log = TRUE)'.

     Note that for smallish values of 'shape' (and moderate 'scale') a
     large parts of the mass of the Gamma distribution is on values of
     x so near zero that they will be represented as zero in computer
     arithmetic.  So 'rgamma' can well return values which will be
     represented as zero.  (This will also happen for very large values
     of 'scale' since the actual generation is done for 'scale=1'.)

Value:

     'dgamma' gives the density, 'pgamma' gives the distribution
     function, 'qgamma' gives the quantile function, and 'rgamma'
     generates random deviates.

     Invalid arguments will result in return value 'NaN', with a
     warning.

Note:

     The S parametrization is via 'shape' and 'rate': S has no 'scale'
     parameter.

     'pgamma' is closely related to the incomplete gamma function.  As
     defined by Abramowitz and Stegun 6.5.1 (and by 'Numerical
     Recipes') this is

          P(a,x) =      1/Gamma(a) integral_0^x t^(a-1) exp(-t) dt

     P(a, x) is 'pgamma(x, a)'.  Other authors (for example Karl
     Pearson in his 1922 tables) omit the normalizing factor, defining
     the incomplete gamma function as 'pgamma(x, a) * gamma(a)'.  A few
     use the 'upper' incomplete gamma function, the integral from x to
     infinity which can be computed by 'pgamma(x, a, lower=FALSE) *
     gamma(a)', or its normalized version. See also <URL:
     http://en.wikipedia.org/wiki/Incomplete_gamma_function>.

Source:

     'dgamma' is computed via the Poisson density, using code
     contributed by Catherine Loader (see 'dbinom').

     'pgamma' uses an unpublished (and not otherwise documented)
     algorithm 'mainly by Morten Welinder'.

     'qgamma' is based on a C translation of

     Best, D. J. and D. E. Roberts (1975).  Algorithm AS91. Percentage
     points of the chi-squared distribution.  _Applied Statistics_,
     *24*, 385-388.

     plus a final Newton step to improve the approximation.

     'rgamma' for 'shape >= 1' uses

     Ahrens, J. H. and Dieter, U. (1982).  Generating gamma variates by
     a modified rejection technique.  _Communications of the ACM_,
     *25*, 47-54,

     and for '0 < shape < 1' uses

     Ahrens, J. H. and Dieter, U. (1974).  Computer methods for
     sampling from gamma, beta, Poisson and binomial distributions.
     _Computing_, *12*, 223-246.

References:

     Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S
     Language_.  Wadsworth & Brooks/Cole.

     Shea, B. L. (1988) Algorithm AS 239, Chi-squared and incomplete
     Gamma integral, _Applied Statistics (JRSS C)_ *37*, 466-473.

     Abramowitz, M. and Stegun, I. A. (1972) _Handbook of Mathematical
     Functions._ New York: Dover.  Chapter 6: Gamma and Related
     Functions.

See Also:

     'gamma' for the gamma function.

     Distributions for other standard distributions, including 'dbeta'
     for the Beta distribution and 'dchisq' for the chi-squared
     distribution which is a special case of the Gamma distribution.

Examples:

     -log(dgamma(1:4, shape=1))
     p <- (1:9)/10
     pgamma(qgamma(p,shape=2), shape=2)
     1 - 1/exp(qgamma(p, shape=1))

     # even for shape = 0.001 about half the mass is on numbers
     # that cannot be represented accurately (and most of those as zero)
     pgamma(.Machine$double.xmin, 0.001)
     pgamma(5e-324, 0.001)  # on most machines 5e-324 is the smallest
                            # representable non-zero number
     table(rgamma(1e4, 0.001) == 0)/1e4
```