Norges teknisk-naturvitenskapelige universitet
Institutt for matematiske fag

Bokmål

Faglig kontakt under eksamen: Professor Jarle Tufto
Telefon: 99 70 55 19

Statistisk modellering for biologer og bioteknologer, ST2304

12. august, 2013
Kl. 9–13
Sensur: 3. september

Hjelpemidler: Ett håndskrevet gult A4-ark, bestemt enkel kalkulator, "Tabeller og formler i statistikk" (Tapir forlag), K. Rottmann: Matematisk formelsamling.

Alle svar skal begrunnes.

Oppgave 1 Anta at vi skal utføre en to-utvalgs t-test basert på normalfordelte data. Størrelsene av hvert av de to utvalgene er henholdsvis $n_1 = 10$ og $n_2 = 13$. Vi ønsker å teste nullhypotesen $H_0 : \mu_1 \leq \mu_2$ mot alternativ hypotese $H_1 : \mu_1 > \mu_2$.

a) Skriv et R-uttrykk som beregner testens kritiske verdi.

b) Skriv et R-uttrykk som beregner testens p-verdi gitt at observert verdi at testobservatoren er lik 2.31.

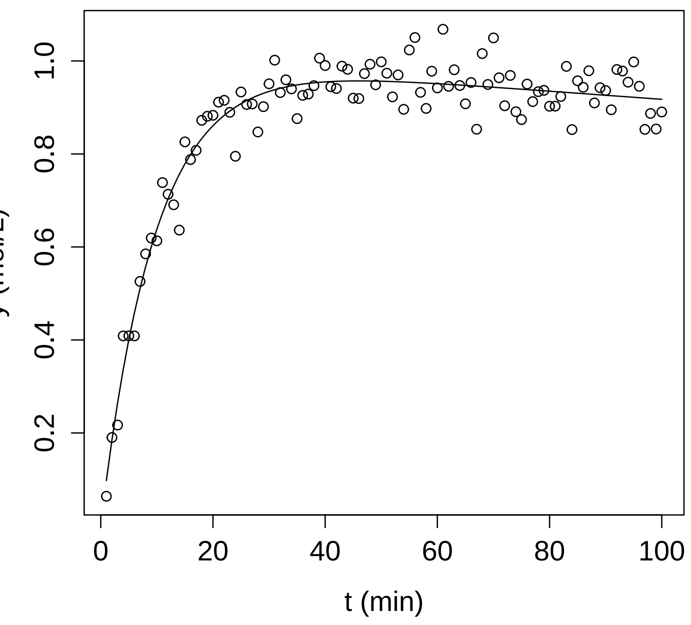
c) Er t-observatoren kontinuerlig eller diskret fordelt? Skriv et R-uttrykk som beregner sannsynlighetstettheten i $t = 0$.

Oppgave 2 En bioteknolog studerer en kjemisk reaksjon hvor det kjemiske stoffet A i en primær reaksjon omdannes til B med en rate k_1 per tidsenhet. Videre mistenker bioteknologen at B, i en sekundær reaksjon, langsomt omdannes til et tredje stoff C med en rate k_2. Ved tidspunkt $t = 0$ er kun stoffet A tilstede i løsningen med kjent konsentrasjon $x_0 = 1$ målt i mol/L. I følge teori fra kjemisk kinetikk vil konsentrasjonen av stoffet B ved et vilkårlig tidpunkt t da være være gitt ved

$$y(t) = \frac{k_1 x_0}{k_2 - k_1}(e^{-k_1 t} - e^{-k_2 t}), \tag{1}$$

se figur.

Bioteknologen ønsker å estimere raten k_1 i den primære reaksjonen og utfører derfor et eksperiment hvor han observerer konsentrasjonen $y_1, y_2, \ldots, y_{100}$ av stoffet B på gitte tidspunkt $t_1, t_2, \ldots, t_{100}$ slik at han får dataene observert i følgende figur.



Vi antar at hver måling $y_1, y_2, \ldots, y_{100}$ er normalfordelt med forventning gitt ved uttrykket over og varians σ^2 og tilpasser modellen på følgende måte i R.

```
> lnL1 <- function(p,y,t) {
+   k1 <- p[1]
+   k2 <- p[2]
+   sigma <- p[3]
+   -sum(dnorm(y,mean=k1/(k2-k1)*(exp(-k1*t)-exp(-k2*t)),sd=sigma,log=T))
+ }
```

```
> fit1 <- optim(c(0.2,0,.1),lnL1,t=t,y=y,lower=c(-Inf,-Inf,0),hessian=TRUE)
Warning message:
In optim(c(0.2, 0, 0.1), lnL1, t = t, y = y, lower = c(-Inf, -Inf,  :
  bounds can only be used with method L-BFGS-B (or Brent)
> fit1
$par
[1] 0.1026603457 0.0009543398 0.0446979276

$value
[1] -168.9174

$counts
function gradient
      81       81

$convergence
[1] 52

$message
[1] "ERROR: ABNORMAL_TERMINATION_IN_LNSRCH"

$hessian
              [,1]          [,2]         [,3]
[1,]  111308.80058 -4.068961e+05    -14.14546
[2,] -406896.13278  1.107652e+08    222.60217
[3,]     -14.14546  2.226022e+02 100471.55345

> solve(fit$hessian)
             [,1]          [,2]          [,3]
[1,] 7.988289e-06  2.872830e-08  2.099499e-08
[2,] 2.872830e-08  8.454731e-09 -3.118829e-10
[3,] 2.099499e-08 -3.118829e-10  9.175693e-06
```

Kurven i plottet over representerer denne tilpassede modellen.

a) Hva er sannsynlighetsmaksimeringsestimatene og tilhørende standardfeil til parameterne i modellen?

b) Skriv opp uttrykket for y som funksjon av t dersom stoffet B ikke omdannes til C, altså at raten i den sekundære reaksjonen $k_2 = 0$. Lag en skisse av y som funksjon av

t når modellen har denne formen. Mot hvilken grense går konsentrasjonen av B i dette tilfelle når tiden t går mot uendelig? Forklar hvorfor denne modellen er nøstet i modellen beskrevet ved ligning (1).

c) Skriv en ny funksjon `lnL0` samt annen nødvendig programkode du ville brukt i R for å tilpasse den forenklede modellen i forrige punkt.

Vi lagrer resultatet av tilpasningen av den forenklede modellen i et objekt `fit0` som får følgende verdi.

```
> fit0
$par
[1] 0.09772798 0.06266032

$value
[1] -135.109

$counts
function gradient
      24       24

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"

$hessian
            [,1]        [,2]
[1,] 61900.11294    14.73557
[2,]    14.73557 51055.01290
```

d) Har vi noe grunnlag for å si hvilke av de to alternative modellene som er å foretrekke? Utfør en formell hypotesetest av dette. Virker dette rimelig ut i fra de observerte dataene? Hvilket estimat av raten k_1 i den primære reaksjonen bør bioteknologen benytte?

Oppgave 3 Anta at vi studerer variasjon i antall individ av stivstarr innenfor ulike prøveruter av forskjellig areal plassert ut i terrenget i fem ulike år. Noen av rutene ligger i nordvendt terreng og andre i sørvendt. Rutene er også plassert i forskjellig høyde over havet og ved forskjellig breddegrad.

a) Foreslå en mulig statistisk modell for hvordan responsvariabelen (antall individ i hver rute) avhenger av de ulike andre variablene.

Hva slags fordeling er det rimelig å bruke for responsvariabelen og hvorfor?

Hvilken link-funksjon vil du bruke og hvorfor?

Hvilke forklaringsvariable er numeriske og hvilke forklaringsvariable vil du velge å modellere som kategoriske?

Kan det være fornuftig å log-transformere noen av variablene?

Forklar valg av eventuelle offset-variable.

b) Skriv opp modellen i forrige punkt i matematisk notasjon.

c) Hvilke biologiske mekanismer kan tenkes å generere henholdsvis under- og over-dispersjon i et slikt datasett?

```
TDist                  package:stats                   R Documentation

The Student t Distribution

Description:

     Density, distribution function, quantile function and random
     generation for the t distribution with 'df' degrees of freedom
     (and optional non-centrality parameter 'ncp').

Usage:

     dt(x, df, ncp, log = FALSE)
     pt(q, df, ncp, lower.tail = TRUE, log.p = FALSE)
     qt(p, df, ncp, lower.tail = TRUE, log.p = FALSE)
     rt(n, df, ncp)

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.

      df: degrees of freedom (> 0, maybe non-integer).  'df = Inf' is
          allowed.

     ncp: non-centrality parameter delta; currently except for 'rt()',
          only for 'abs(ncp) <= 37.62'.  If omitted, use the central t
          distribution.

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:

     The t distribution with 'df' = n degrees of freedom has density

     f(x) = Gamma((n+1)/2) / (sqrt(n pi) Gamma(n/2)) (1 + x^2/n)^-((n+1)/2)

     for all real x.  It has mean 0 (for n > 1) and variance n/(n-2)
     (for n > 2).

     The general _non-central_ t with parameters (df, Del) '= (df,
     ncp)' is defined as the distribution of T(df, Del) := (U + Del) /
     sqrt(V/df) where U and V are independent random variables, U ~
     N(0,1) and V ~ chi^2(df) (see Chisquare).

     The most used applications are power calculations for t-tests:
     Let T= (mX - m0) / (S/sqrt(n)) where mX is the 'mean' and S the
     sample standard deviation ('sd') of X_1, X_2, ..., X_n which are
     i.i.d. N(mu, sigma^2) Then T is distributed as non-central t with
     'df'= n - 1 degrees of freedom and *n*on-*c*entrality *p*arameter
     'ncp' = (mu - m0) * sqrt(n)/sigma.

Value:

     'dt' gives the density, 'pt' gives the distribution function, 'qt'
     gives the quantile function, and 'rt' generates random deviates.

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

Note:

     Supplying 'ncp = 0' uses the algorithm for the non-central
     distribution, which is not the same algorithm used if 'ncp' is
     omitted.  This is to give consistent behaviour in extreme cases
     with values of 'ncp' very near zero.

     The code for non-zero 'ncp' is principally intended to be used for
     moderate values of 'ncp': it will not be highly accurate,
     especially in the tails, for large values.

Source:

     The central 'dt' is computed via an accurate formula provided by
     Catherine Loader (see the reference in 'dbinom').

     For the non-central case of 'dt', C code contributed by Claus
     Ekstroem based on the relationship (for x != 0) to the cumulative
     distribution.

     For the central case of 'pt', a normal approximation in the tails,
     otherwise via 'pbeta'.

     For the non-central case of 'pt' based on a C translation of

     Lenth, R. V. (1989). _Algorithm AS 243_ - Cumulative distribution
     function of the non-central t distribution, _Applied Statistics_
     *38*, 185-189.

     This computes the lower tail only, so the upper tail suffers from
     cancellation and a warning will be given when this is likely to be
     significant.

     For central 'qt', a C translation of

     Hill, G. W. (1970) Algorithm 396: Student's t-quantiles.
     _Communications of the ACM_, *13(10)*, 619-620.

     altered to take account of

     Hill, G. W. (1981) Remark on Algorithm 396, _ACM Transactions on
     Mathematical Software_, *7*, 250-1.

     The non-central case is done by inversion.

References:

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

     Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) _Continuous
     Univariate Distributions_, volume 2, chapters 28 and 31.  Wiley,
     New York.

See Also:

     Distributions for other standard distributions, including 'df' for
     the F distribution.

Examples:

     require(graphics)

     1 - pt(1:5, df = 1)
     qt(.975, df = c(1:10,20,50,100,1000))

     tt <- seq(0,10, len=21)
     ncp <- seq(0,6, len=31)
     ptn <- outer(tt,ncp, function(t,d) pt(t, df = 3, ncp=d))
     t.tit <- "Non-central t - Probabilities"
     image(tt,ncp,ptn, zlim=c(0,1), main = t.tit)
     persp(tt,ncp,ptn, zlim=0:1, r=2, phi=20, theta=200, main=t.tit,
           xlab = "t", ylab = "non-centrality parameter",
           zlab = "Pr(T <= t)")

     plot(function(x) dt(x, df = 3, ncp = 2), -3, 11, ylim = c(0, 0.32),
          main="Non-central t - Density", yaxs="i")

------------------------------------------------------------------------
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)
----------------------------------------------------------------------
optim                  package:stats                   R Documentation

General-purpose Optimization

Description:

     General-purpose optimization based on Nelder-Mead, quasi-Newton
     and conjugate-gradient algorithms. It includes an option for
     box-constrained optimization and simulated annealing.

Usage:

     optim(par, fn, gr = NULL, ...,
           method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
           lower = -Inf, upper = Inf,
           control = list(), hessian = FALSE)

     optimHess(par, fn, gr = NULL, ..., control = list())

Arguments:

     par: Initial values for the parameters to be optimized over.

      fn: A function to be minimized (or maximized), with first
          argument the vector of parameters over which minimization is
          to take place.  It should return a scalar result.

      gr: A function to return the gradient for the '"BFGS"', '"CG"'
          and '"L-BFGS-B"' methods.  If it is 'NULL', a
          finite-difference approximation will be used.

          For the '"SANN"' method it specifies a function to generate a
          new candidate point.  If it is 'NULL' a default Gaussian
          Markov kernel is used.

     ...: Further arguments to be passed to 'fn' and 'gr'.

  method: The method to be used. See 'Details'.

lower, upper: Bounds on the variables for the '"L-BFGS-B"' method, or
          bounds in which to _search_ for method '"Brent"'.

 control: A list of control parameters. See 'Details'.

 hessian: Logical. Should a numerically differentiated Hessian matrix
          be returned?

Details:

     Note that arguments after '...' must be matched exactly.

     By default 'optim' performs minimization, but it will maximize if
     'control$fnscale' is negative.  'optimHess' is an auxiliary
     function to compute the Hessian at a later stage if 'hessian =
     TRUE' was forgotten.

     The default method is an implementation of that of Nelder and Mead
     (1965), that uses only function values and is robust but
     relatively slow.  It will work reasonably well for
     non-differentiable functions.

     Method '"BFGS"' is a quasi-Newton method (also known as a variable
     metric algorithm), specifically that published simultaneously in
     1970 by Broyden, Fletcher, Goldfarb and Shanno.  This uses
     function values and gradients to build up a picture of the surface
     to be optimized.

     Method '"CG"' is a conjugate gradients method based on that by
     Fletcher and Reeves (1964) (but with the option of Polak-Ribiere
     or Beale-Sorenson updates).  Conjugate gradient methods will
     generally be more fragile than the BFGS method, but as they do not
     store a matrix they may be successful in much larger optimization
     problems.

     Method ’"L-BFGS-B"’ is that of Byrd _et. al._ (1995) which allows
     _box constraints_, that is each variable can be given a lower
     and/or upper bound. The initial value must satisfy the
     constraints.  This uses a limited-memory modification of the BFGS
     quasi-Newton method. If non-trivial bounds are supplied, this
     method will be selected, with a warning.

     Nocedal and Wright (1999) is a comprehensive reference for the
     previous three methods.

     Method ’"SANN"’ is by default a variant of simulated annealing
     given in Belisle (1992). Simulated-annealing belongs to the class
     of stochastic global optimization methods. It uses only function
     values but is relatively slow. It will also work for
     non-differentiable functions. This implementation uses the
     Metropolis function for the acceptance probability. By default the
     next candidate point is generated from a Gaussian Markov kernel
     with scale proportional to the actual temperature. If a function
     to generate a new candidate point is given, method ’"SANN"’ can
     also be used to solve combinatorial optimization problems.
     Temperatures are decreased according to the logarithmic cooling
     schedule as given in Belisle (1992, p. 890); specifically, the
     temperature is set to ’temp / log(((t-1) %/% tmax)*tmax +
     exp(1))’, where ’t’ is the current iteration step and ’temp’ and
     ’tmax’ are specifiable via ’control’, see below.  Note that the
     ’"SANN"’ method depends critically on the settings of the control
     parameters. It is not a general-purpose method but can be very
     useful in getting to a good value on a very rough surface.

     Method ’"Brent"’ is for one-dimensional problems only, using
     ’optimize()’.  It can be useful in cases where ’optim()’ is used
     inside other functions where only ’method’ can be specified, such
     as in ’mle’ from package ’stats4’.

     Function ’fn’ can return ’NA’ or ’Inf’ if the function cannot be
     evaluated at the supplied value, but the initial value must have a
     computable finite value of ’fn’.  (Except for method ’"L-BFGS-B"’
     where the values should always be finite.)

     ’optim’ can be used recursively, and for a single parameter as
     well as many.  It also accepts a zero-length ’par’, and just
     evaluates the function with that argument.

     The ’control’ argument is a list that can supply any of the
     following components:

     ’trace’ Non-negative integer. If positive, tracing information on
          the progress of the optimization is produced. Higher values
          may produce more tracing information: for method ’"L-BFGS-B"’
          there are six levels of tracing.  (To understand exactly what
          these do see the source code: higher levels give more
          detail.)

     ’fnscale’ An overall scaling to be applied to the value of ’fn’
          and ’gr’ during optimization. If negative, turns the problem
          into a maximization problem. Optimization is performed on
          ’fn(par)/fnscale’.

     ’parscale’ A vector of scaling values for the parameters.
          Optimization is performed on ’par/parscale’ and these should
          be comparable in the sense that a unit change in any element
          produces about a unit change in the scaled value.  Not used
          (nor needed) for ’method = "Brent"’.

     ’ndeps’ A vector of step sizes for the finite-difference
          approximation to the gradient, on ’par/parscale’ scale.
          Defaults to ’1e-3’.

     ’maxit’ The maximum number of iterations. Defaults to ’100’ for
          the derivative-based methods, and ’500’ for ’"Nelder-Mead"’.

          For ’"SANN"’ ’maxit’ gives the total number of function
          evaluations: there is no other stopping criterion. Defaults
          to ’10000’.

     ’abstol’ The absolute convergence tolerance. Only useful for
          non-negative functions, as a tolerance for reaching zero.

     ’reltol’ Relative convergence tolerance.  The algorithm stops if
          it is unable to reduce the value by a factor of ’reltol *
          (abs(val) + reltol)’ at a step.  Defaults to
          ’sqrt(.Machine$double.eps)’, typically about ’1e-8’.

     ’alpha’, ’beta’, ’gamma’ Scaling parameters for the
          ’"Nelder-Mead"’ method. ’alpha’ is the reflection factor
          (default 1.0), ’beta’ the contraction factor (0.5) and
          ’gamma’ the expansion factor (2.0).

     ’REPORT’ The frequency of reports for the ’"BFGS"’, ’"L-BFGS-B"’
          and ’"SANN"’ methods if ’control$trace’ is positive. Defaults
          to every 10 iterations for ’"BFGS"’ and ’"L-BFGS-B"’, or
          every 100 temperatures for ’"SANN"’.

     ’type’ for the conjugate-gradients method. Takes value ’1’ for the
          Fletcher-Reeves update, ’2’ for Polak-Ribiere and ’3’ for
          Beale-Sorenson.

     ’lmm’ is an integer giving the number of BFGS updates retained in
          the ’"L-BFGS-B"’ method, It defaults to ’5’.

     ’factr’ controls the convergence of the ’"L-BFGS-B"’ method.
          Convergence occurs when the reduction in the objective is
          within this factor of the machine tolerance. Default is
          ’1e7’, that is a tolerance of about ’1e-8’.

     ’pgtol’ helps control the convergence of the ’"L-BFGS-B"’ method.
          It is a tolerance on the projected gradient in the current
          search direction. This defaults to zero, when the check is
          suppressed.

     ’temp’ controls the ’"SANN"’ method. It is the starting
          temperature for the cooling schedule. Defaults to ’10’.

     ’tmax’ is the number of function evaluations at each temperature
          for the ’"SANN"’ method. Defaults to ’10’.

     Any names given to ’par’ will be copied to the vectors passed to
     ’fn’ and ’gr’.  Note that no other attributes of ’par’ are copied
     over.

     The parameter vector passed to ’fn’ has special semantics and may
     be shared between calls: the function should not change or copy
     it.

Value:

     For ’optim’, a list with components:

     par: The best set of parameters found.

   value: The value of ’fn’ corresponding to ’par’.

  counts: A two-element integer vector giving the number of calls to
          ’fn’ and ’gr’ respectively. This excludes those calls needed
          to compute the Hessian, if requested, and any calls to ’fn’
          to compute a finite-difference approximation to the gradient.

convergence: An integer code. ’0’ indicates successful completion
          (which is always the case for ’"SANN"’ and ’"Brent"’).
          Possible error codes are

          ’1’ indicates that the iteration limit ’maxit’ had been
               reached.

          ’10’ indicates degeneracy of the Nelder-Mead simplex.

          ’51’ indicates a warning from the ’"L-BFGS-B"’ method; see
               component ’message’ for further details.

          ’52’ indicates an error from the ’"L-BFGS-B"’ method; see
               component ’message’ for further details.

 message: A character string giving any additional information returned
          by the optimizer, or ’NULL’.

 hessian: Only if argument ’hessian’ is true. A symmetric matrix giving
          an estimate of the Hessian at the solution found.  Note that
          this is the Hessian of the unconstrained problem even if the
          box constraints are active.
     For ’optimHess’, the description of the ’hessian’ component
     applies.
```

```
Note:

     'optim' will work with one-dimensional 'par's, but the default
     method does not work well (and will warn).  Method '"Brent"' uses
     'optimize' and needs bounds to be available; '"BFGS"' often works
     well enough if not.

Source:

     The code for methods '"Nelder-Mead"', '"BFGS"' and '"CG"' was
     based originally on Pascal code in Nash (1990) that was translated
     by 'p2c' and then hand-optimized.  Dr Nash has agreed that the
     code can be made freely available.

     The code for method '"L-BFGS-B"' is based on Fortran code by Zhu,
     Byrd, Lu-Chen and Nocedal obtained from Netlib (file
     'opt/lbfgs_bcm.shar': another version is in 'toms/778').

     The code for method '"SANN"' was contributed by A. Trapletti.

References:

     Belisle, C. J. P. (1992) Convergence theorems for a class of
     simulated annealing algorithms on Rd. _J. Applied Probability_,
     *29*, 885-895.

     Byrd, R. H., Lu, P., Nocedal, J. and Zhu, C.  (1995) A limited
     memory algorithm for bound constrained optimization.  _SIAM J.
     Scientific Computing_, *16*, 1190-1208.

     Fletcher, R. and Reeves, C. M. (1964) Function minimization by
     conjugate gradients. _Computer Journal_ *7*, 148-154.

     Nash, J. C. (1990) _Compact Numerical Methods for Computers.
     Linear Algebra and Function Minimisation._ Adam Hilger.

     Nelder, J. A. and Mead, R. (1965) A simplex algorithm for function
     minimization. _Computer Journal_ *7*, 308-313.

     Nocedal, J. and Wright, S. J. (1999) _Numerical Optimization_.
     Springer.

See Also:

     'nlm', 'nlminb'.

     'optimize' for one-dimensional minimization and 'constrOptim' for
     constrained optimization.

Examples:

     require(graphics)

     fr <- function(x) {   ## Rosenbrock Banana function
         x1 <- x[1]
         x2 <- x[2]
         100 * (x2 - x1 * x1)^2 + (1 - x1)^2
     }
     grr <- function(x) { ## Gradient of 'fr'
         x1 <- x[1]
         x2 <- x[2]
         c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
            200 *      (x2 - x1 * x1))
     }
     optim(c(-1.2,1), fr)
     (res <- optim(c(-1.2,1), fr, grr, method = "BFGS"))
     optimHess(res$par, fr, grr)
     optim(c(-1.2,1), fr, NULL, method = "BFGS", hessian = TRUE)
     ## These do not converge in the default number of steps
     optim(c(-1.2,1), fr, grr, method = "CG")
     optim(c(-1.2,1), fr, grr, method = "CG", control=list(type=2))
     optim(c(-1.2,1), fr, grr, method = "L-BFGS-B")

     flb <- function(x)
         { p <- length(x); sum(c(1, rep(4, p-1)) * (x - c(1, x[-p])^2)^2) }
     ## 25-dimensional box constrained
     optim(rep(3, 25), flb, NULL, method = "L-BFGS-B",
           lower=rep(2, 25), upper=rep(4, 25)) # par[24] is *not* at boundary

     ## "wild" function , global minimum at about -15.81515
     fw <- function (x)
         10*sin(0.3*x)*sin(1.3*x^2) + 0.00001*x^4 + 0.2*x+80
plot(fw, -50, 50, n=1000, main = "optim() minimising 'wild function'")

res <- optim(50, fw, method="SANN",
             control=list(maxit=20000, temp=20, parscale=20))
res
## Now improve locally {typically only by a small bit}:
(r2 <- optim(res$par, fw, method="BFGS"))
points(r2$par, r2$value, pch = 8, col = "red", cex = 2)

## Combinatorial optimization: Traveling salesman problem
library(stats) # normally loaded

eurodistmat <- as.matrix(eurodist)

distance <- function(sq) {  # Target function
    sq2 <- embed(sq, 2)
    sum(eurodistmat[cbind(sq2[,2],sq2[,1])])
}

genseq <- function(sq) {  # Generate new candidate sequence
    idx <- seq(2, NROW(eurodistmat)-1)
    changepoints <- sample(idx, size=2, replace=FALSE)
    tmp <- sq[changepoints[1]]
    sq[changepoints[1]] <- sq[changepoints[2]]
    sq[changepoints[2]] <- tmp
    sq
}

sq <- c(1:nrow(eurodistmat), 1)  # Initial sequence: alphabetic
distance(sq)
# rotate for conventional orientation
loc <- -cmdscale(eurodist, add=TRUE)$points
x <- loc[,1]; y <- loc[,2]
s <- seq_len(nrow(eurodistmat))
tspinit <- loc[sq,]

plot(x, y, type="n", asp=1, xlab="", ylab="",
     main="initial solution of traveling salesman problem", axes = FALSE)
arrows(tspinit[s,1], tspinit[s,2], tspinit[s+1,1], tspinit[s+1,2],
       angle=10, col="green")
text(x, y, labels(eurodist), cex=0.8)

set.seed(123) # chosen to get a good soln relatively quickly
res <- optim(sq, distance, genseq, method = "SANN",
             control = list(maxit = 30000, temp = 2000, trace = TRUE,
                            REPORT = 500))
res  # Near optimum distance around 12842

tspres <- loc[res$par,]
plot(x, y, type="n", asp=1, xlab="", ylab="",
     main="optim() 'solving' traveling salesman problem", axes = FALSE)
arrows(tspres[s,1], tspres[s,2], tspres[s+1,1], tspres[s+1,2],
       angle=10, col="red")
text(x, y, labels(eurodist), cex=0.8)
```