gloptim with *GA* and *DEoptim* Test

This is about an example **gloptim** function, tested with the non-smooth Hald test function. Test functions are represented as lists with components

- `fn` the defining code of the test function
- `dim` the number of variables
- `lb`, `ub`, lower and upper bounds for the variables
- `xmin` the known minimum solution
- `fmin` the function value at this minimum
- `prec` the precision of this minimum value

Here is the 5-parameter Hald function, defined as a minimax funtion of 21 functions. Note that the function is everywhere defined and continuous, with possible singularities.

```
##  Test function HALD, 5 parameters, non-smooth, one local minimum
Hald <- list(
  fn = function(x) {
    stopifnot(is.numeric(x), length(x) == 5)
    if (all(x == 1)) return(exp(1))
    t <- -1 + (c(1:21) - 1)/10
    v <- (x[1]+x[2]*t) / (1 + x[3]*t + x[4]*t^2 + x[5]*t^3) - exp(t)
    max(abs(v))
  },
  dim = 5, lb = rep(-1, 5), ub = rep(1, 5),
  xmin = c(0.99987763, 0.25358844, -0.74660757, 0.24520150, -0.03749029),
  fmin = 0.00012237326, prec = 1.0e-10
)
```

The following is an example of a wrapping function for the stochastic solvers `ga` (from the *GA* package) and `DEoptim` (from the *DEoptim* package).

```
gloptim <- function(fn, lb, ub, x0 = NULL, ...,
        method = c("deoptim", "ga"), type = NULL,
        minimize = TRUE, control = list()) {

    fun = match.fun(fn)
    f <- function(x) fun(x, ...)

    method = match.arg(method)
    cat("Global solver/method:", method, "\n")

    cntrl <- list(info = FALSE,     # shall info/trace be shown
                  popsize = NULL,   # population size
                  itermax = NULL    # max. no. of iterations
                  )
    for (nm in names(control)) {
        if (nm %in% names(cntrl)) {
            cntrl[nm] <- control[nm]
        } else {
            stop("Unknown name in control list: '", nm, "'.", call. = FALSE)
        }
    }

    if (method == "ga") {
```

```
        if (minimize) s <- -1 else s <- 1
        fn <- function(x) s * f(x)
        if (is.null(cntrl$popsize)) popSize <- 100 else popSize <- cntrl$popsize
        if (is.null(cntrl$itermax)) maxiter <- 100 else maxiter <- cntrl$itermax

        sol <- GA::ga(type = "real-valued", fitness = fn,
                      min = lb, max = ub,
                      popSize = popSize,
                      maxiter = maxiter,
                      monitor = cntrl$info)
        return(list(xmin = sol@solution,
                    fmin = s * sol@fitnessValue))

    } else if (method == "deoptim") {
        if (is.null(cntrl$itermax)) maxiter <- 1000 else maxiter <- cntrl$itermax

        sol <- DEoptim::DEoptim(fn, lower = lb, upper = ub,
                                DEoptim::DEoptim.control(
                                trace = cntrl$info, itermax = cntrl$itermax))
        return(list(xmin = sol$optim$bestmem, fmin = sol$optim$bestval))
    } else {
        stop("Argument 'method' has not yet been implemented.")
    }
}
```

Only very few control options are available at the moment:

- `info`: whether a trace or other info from the solver should be show.
- `popsize`: population size for gemetic algorithms, mostly.
- `itermax`: maximum number of iterations.

Some stochastic solvers stop automatically when the solution does not change anymore (below a certain relative tolerance). The genetic algorithm in *GA* only stops when the maximum number of iterations has been reached.

The population size is obviously important for the accuracy to be reached. But big population sizes will make the whole process very slow. By the way, the default values for the `ga` solver are quite low and should be increased by a wrapper.

The application of the solvers to the Hald test function can best be done using the `with`-construct.

```
with(Hald, {
  stime = system.time(
    sol <- gloptim(fn = fn, lb = lb, ub = ub, method = "ga",
                   minimize = TRUE,
                   control = list(popsize = 200, itermax = 1000))
    )
  cat("xmin: ", sol$xmin, '\n')
  cat("fmin: ", sol$fmin, '\n')
  cat("xerr: ", sqrt(sum((sol$xmin-Hald$xmin)^2)), '\n')
  cat("ferr: ", abs( sol$fmin-Hald$fmin), '\n')
  cat("Elapsed time: ", stime["elapsed"], " [s].")
})
```

```
## Global solver/method: ga
## xmin:  0.9907424 0.4301047 -0.5549924 0.06614381 0.008088082
## fmin:  0.02031531
```

```
## xerr:  0.319526
## ferr:  0.02019294
## Elapsed time:  10.387  [s].
```

```
with(Hald, {
  stime = system.time(
    sol <- gloptim(fn = fn, lb = lb, ub = ub, method = "deoptim",
                   minimize = TRUE,
                   control = list(itermax = 1000, info = FALSE))
    )
  cat("xmin: ", sol$xmin, '\n')
  cat("fmin: ", sol$fmin, '\n')
  cat("xerr: ", sqrt(sum((sol$xmin-Hald$xmin)^2)), '\n')
  cat("ferr: ", abs( sol$fmin-Hald$fmin), '\n')
  cat("Elapsed time: ", stime["elapsed"], " [s].")
})
```

```
## Global solver/method: deoptim
## xmin:  0.999875 0.2537378 -0.7464397 0.2450211 -0.03742428
## fmin:  0.0001262106
## xerr:  0.0002956477
## ferr:  3.837325e-06
## Elapsed time:  1.06  [s].
```