**Purpose and Formulation**

This cheatsheet is a quick guide to optimization tools present in the base distribution of the R software for statistical computing.

**General problem:**

Find a vector \( \mathbf{x} \) of \( n \) parameters that minimizes a given function \( f(\mathbf{x}) \) in a certain domain, i.e. subject to inequality constraints \( g(\mathbf{x}) \leq 0 \), and sometimes also equality constraints \( h(\mathbf{x}) = 0 \).

Numerical optimization routines will in general only find local minima that are minimal only in their neighborhood. There may exist several or even many minima in a given domain.

- Most problems are special cases. Base R mainly treats problems that are unconstrained minimizations of a few parameters.
- There are often several ways to formulate a problem for different software tools, and some of these may be much more efficient than others.
- Bounds constraints \( \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \) are a special and easier to handle case of constraints.
- To maximize a function \( f(\mathbf{x}) \), minimize \(-f(\mathbf{x})\)
- Most tools require suitable starting parameters \( \mathbf{x}_0 \). This is particularly important if there are multiple minima. The minimum found is not guaranteed to be the nearest one to \( \mathbf{x}_0 \).
- The lowest of all minima is known as the global minimum. Unfortunately, Base R has no tools for global optimization.
- Iterative algorithms converge, but programs terminate (convergence, computational failure, limit on effort).
- The tools in Base R are intended for smooth functions. Optimizing non-smooth functions usually requires special approaches, though optim() with method Nelder-Mead may sometimes succeed.

For many more optimization tools see the Task View for Optimization.

**REMARK:** In R syntax below, square brackets indicate an optional element of a command. ... indicates additional parameters of the function to be minimized.

---

**One-dimensional optimization (x a scalar)**

\[
\text{result} \leftarrow \text{optimize}(f, \text{interval}=c(\text{lower}, \text{upper}), ..., \\
\text{tol}=\text{.Machine}$\text{double}$^0.25)
\]

- \( f \) is a function of just one parameter, and interval gives the end points of the domain in which a minimum is to be found.
- See the documentation for alternate syntax, including maximum=TRUE. The tol above is the default value (approx. 0.0001). This default tolerance is not small enough for many applications.
- result is a list with elements minimum and objective.
- If there are multiple minima, you only get one of them. If there are no minima – you still get a result! CAUTION.

**Unconstrained minimization**

Function \( f(\mathbf{x}[\ldots]) \) computes the function to be minimized at \( \mathbf{x} \); choose starting parameters \( \mathbf{x}_0 \). Optional, but useful: a vector valued function \( \text{grad}(\cdot) \) that computes the gradient of \( f(\cdot) \) at \( \mathbf{x} \).

optim is the main optimizer function in Base R. There are two other minimizers that may be sometimes useful, nlm and nlminb.

\[
\text{result} \leftarrow \text{optim}(\mathbf{x}_0, f \,[\ldots], \text{method} = \text{"Method"}[\ldots] \\
\text{control} = \text{list}())
\]

where Method is one of 'Nelder-Mead', 'BFGS', 'L-BFGS-B', 'CG', or 'SANN'. The default is 'Nelder-Mead', a gradient-free – but slow – optimization solver.

NOTE: CG (JN is author!) and SANN are NOT recommended.

The most important control option is the tolerance, called reltol for methods 'Nelder-Mead' and 'BFGS', and factr for method 'L-BFGS-B'. Its default value is about 1e-08.

nlm carries out a Newton-type algorithm and returns a list with minimum for the function value and estimate for the solution vector.

\[
\text{result} \leftarrow \text{nlm}(f, \mathbf{x}_0)
\]

- A separate gradient function is not used. Look at the help page to see how the gradient can be defined and used in nlm. Also note the order of the call for nlm: function first, then the starting parameters.
Unconstrained minimization (continued)

`nlminb` uses the PORT routines, a FORTRAN implementation of quasi-Newton BFGS. It has been included with Base R for historical reasons.

```r
result <- nlminb(x0, f [, grad])
```

It returns `par` and `objective` as list (with some more information).

- To minimize functions of many parameters, use `optim()` with method L-BFGS-B or find efficient solvers in the Task View for Optimization.
- When a gradient is needed but not specified, methods use numerical approximations (finite differences). This generally gives slightly less satisfactory solutions and may require more computational effort.

Bounds constrained optimization

From `optim()`, only method L-BFGS-B can deal with bounds. `nlminb()` also handles bounds. Assume variables `lb` and `ub` have either single values or else as many values as `x` to define lower and upper bounds respectively.

```r
result <- optim(x0, f [, grad], method = "B-BFGS-B",
                lower = lb, upper = ub)
result <- nlminb(x0, f [, grad], lower = lb, upper = ub)
```

- From a practical viewpoint, methods often fail to find appropriate solutions when lower and upper bounds are specified close together.
- Bounds constrained optimization without `grad` may fail when the gradient approximation tries to evaluate the function outside the bounds.

Linearly constrained optimization

The function `constrOptim` will approximate linearly constrained optimization problems, where there are k linear constraints such that `A %*% x ≥ b` with `A` a k-by-n matrix, `n` the number of parameters and `b` a length k vector of constants.

```r
result <- constrOptim(x0, f [, grad], ui=A, ci=b[, method])
```

- `constrOptim` uses an adaptive barrier method in conjunction with `optim()`, so it has the same strengths and weaknesses as `optim()`.
- Be careful that the dimensions of `A` and `b` are set up correctly.
- The default method is `BFGS`, `L-BFGS-B` is likely more efficient.

NOTE: In Base R there are no optimizers for nonlinear inequality constraints or even for equality constraints. See the Optimization Task View for CRAN packages solving such problems.

Nonlinear least squares

If `f()` is a sum of squared terms, then special methods can be used. Base R has the `nls()` function for such problems. It has many options. Here we will just give an example of the most common task, which is modeling.

Suppose we have a data frame (or list) `Data` with columns `x` and (the dependent) `y`. The model is given as a formula `y ~ a + b * e^(c * x)` with parameters `a`, `b`, `c` to be determined from the data. The call to `nls()` looks likely

```r
nls(y ~ a + b*exp(c*x), data = DATA, trace = FALSE,
    start = c(a = 1.0, b = 2.0, c = 0.05),
    [lower = c(0, 0, 0), upper = c(5.0, 5.0, 1.0),
    [control = list(), algorithm = "port"])
```

If there are bounds constraints, the algorithm must be `port`, else it can be `plinear` or `NULL` (the default), a Gauss-Newton approach. `start` will be a named vector with starting values for all parameters to be determined.

- `nls()` is a powerful tool in the hands of an experienced user, but often leads to “singular gradient” errors, that means it has been unable to find a solutions. Packages `nlsr` and `minpack.lm` should be considered.

See also

CRAN Task View on Optimization:

https://cran.r-project.org/web/views/Optimization.html