Numerical Optimization

Zhentao Shi

March 14, 2017

Numerical Optimization

Optimization is the key step to apply econometric extremum estimators. A general optimization problem is formulated as

$$\min_{\theta \in \Theta} f(\theta) \;\text{ s.t. } g(\theta) = 0, h(\theta) \leq 0,$$

where $f(\cdot)$ is a criterion function, $g(\theta) = 0$ is an equality constraint, and $h(\theta) \leq 0$ is an inequality constraint.

Most established numerical optimization algorithms aim at finding a local minimum if it exists. However, there is no guarantee to locate the global minimum when multiple local minima exist.

An optimization problem without the equality and/or inequality constraints is called an *unconstrained* problem; otherwise it is called a constrained problem. We can incorporate the constraints into the criterion function via Lagrangian.

Methods

There are numerous optimization algorithms in the field of operational research, but only a small handful of major ideas stand out.

The fundamental idea for twice-differentiable objective function is the Newton's method. We know what a necessary condition for optimization is that $s(\theta) = \partial f(\theta) / \partial \theta = 0$.

At an initial trial value θ_0, if $s(\theta_0) \neq 0$, we can direct the search by updating

$$\theta_{t+1} = \theta_t + (H(\theta_t))^{-1} s(\theta_t)$$

for $t = 0, 1, \cdots$ where $H(\theta) = \frac{\partial s(\theta)}{\partial \theta}$ is the Hessian matrix. The algorithm iterates until $|\theta_{t+1} - \theta_t| < \epsilon$ (absolute criterion) or $|\theta_{t+1} - \theta_t| / |\theta_t| < \epsilon$ (relative criterion), where ϵ is a small positive number chosen as the tolerance level.

Newton's Method. Newton's method seeks the solution to $s(\theta) = 0$. Recall that the first-order condition is a necessary condition but not a sufficient condition. We still need to verify the second-order condition to identify whether a root to $s(\theta)$ is associated to a minimizer or a maximizer, and we compare the values of the mimima to decide a global minimum.

It is clear from the description of Newton's method that it requires the computation of the gradient $s(\theta)$ and the Hessian $H(\theta)$. Newton's method converges at quadratic rate, which is fast.

Quasi-Newton method. The most well-known quasi-Newton algorithm is BFGS. It avoids explicit calculation of the Hessian matrix. Instead, starting from an initial (inverse) Hessian, it updates the Hessian by an explicit formula motivated from the idea of quadratic approximation.

Derivative-Free Method. Nelder-Mead is a simplex method. It searches a local minimum by reflection, expansion and contraction.

Implementation

In the past, optimization was a weak spot of R. In the recent year, R's optimization infrastructure has been constantly improving. R Optimization Task View gives a survey of the available CRAN packages.

For general-purpose nonlinear optimization, the package `optimx` (Nash 2014) effectively replaces R's default optimization commands. `optimx` provides a unified interface for various widely-used optimization algorithms. Moreover, it facilitates comparison among optimization routines.

Example

We use `optimx` to solve pseudo Poisson maximum likelihood estimation. If y_i is a continuous random variable, it obviously does not follow a possion distribution, whose support consists of non-negative integers. However, if the conditional mean model

$$E[y_i|x_i] = \exp(x_i'\beta),$$

is satisfied, we can still use the possion regression to obtain a consistent estimator of the parameter β even if y_i does not follow a conditional poisson distribution,

To implement optimization in `R`, it is recommended to write the criterion as a function of the parameter. Data can be fed inside or outside of the function. If the data is injected as additional arguments, these arguments must be explicit.

```r
# implement both BFGS and Nelder-Mead for comparison.

library(AER)
library(numDeriv)
library(optimx)

# Poisson likelihood
poisson.loglik = function(b) {
    b = as.matrix(b)
    lambda = exp(X %*% b)
    ell = -sum(-lambda + y * log(lambda))
    return(ell)
}

## prepare the data
data("RecreationDemand")
y = RecreationDemand$trips
X = with(RecreationDemand, cbind(1, income))

## estimation
b.init = c(0, 1)  # initial value
b.hat = optimx(b.init, poisson.loglik, method = c("BFGS", "Nelder-Mead"), control = list(relto
    abstol = 1e-07))
print(b.hat)
```

```
##                    p1          p2    value fevals gevals niter convcode
## BFGS        1.177411 -0.09994234 261.1141     99     21    NA        0
## Nelder-Mead 1.167261 -0.09703975 261.1317     53     NA    NA        0
##              kkt1 kkt2 xtimes
## BFGS         TRUE TRUE   0.01
## Nelder-Mead FALSE TRUE   0.00
```

In practice no algorithm suits all problems. Simulation, where the true parameter is known, is helpful to check the accuracy of one's optimization routine before applying to an empirical problem, where the true parameter is unknown. Contour plot helps visualize the function surface in a low dimension.

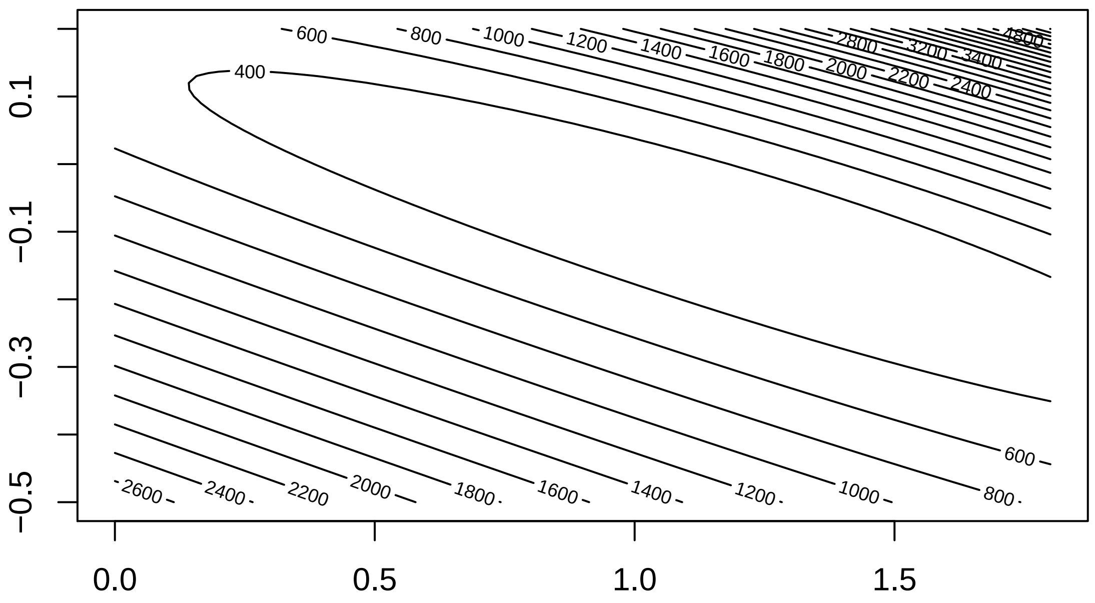
Example

```
x.grid = seq(0, 1.8, 0.02)
x.length = length(x.grid)
y.grid = seq(-0.5, 0.2, 0.01)
y.length = length(y.grid)

z.contour = matrix(0, nrow = x.length, ncol = y.length)

for (i in 1:x.length) {
    for (j in 1:y.length) {
        z.contour[i, j] = poisson.loglik(c(x.grid[i], y.grid[j]))
    }
}

contour(x.grid, y.grid, z.contour, 20)
```



For problems that demand more accuracy, third-party standalone solvers can be invoked via interfaces to R. For example, we can access `NLopt` through the packages `nloptr`. However, standalone solvers usually have to be compiled and configured. These steps are often not as straightforward as installing most of Windows applications.

`NLopt` provides an extensive list of algorithms.

Example

We first carry out Nelder-Mead algorithm in NLOPT.

```r
library(nloptr)
## optimization with NLoptr

opts = list(algorithm = "NLOPT_LN_NELDERMEAD", xtol_rel = 1e-07, maxeval = 500)

res_NM = nloptr(x0 = b.init, eval_f = poisson.loglik, opts = opts)
print(res_NM)
```

```
##
## Call:
## nloptr(x0 = b.init, eval_f = poisson.loglik, opts = opts)
##
##
## Minimization using NLopt version 2.4.0
```

```
##
## NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization stopped because
## xtol_rel or xtol_abs (above) was reached. )
##
## Number of Iterations....: 118
## Termination conditions:  xtol_rel: 1e-07 maxeval: 500
## Number of inequality constraints:  0
## Number of equality constraints:    0
## Optimal value of objective function:  261.114078295329
## Optimal value of controls: 1.177397 -0.09993984
```

```
# 'SLSQP' is indeed the BFGS algorithm in NLopt, though 'BFGS' doesn't
# appear in the name
opts = list(algorithm = "NLOPT_LD_SLSQP", xtol_rel = 1e-07)
```

To invoke BFGS in NLOPT, we must provide the gradient, as in the function `poisson.log.grad`. In this example, we compare it with the numerical gradient to make sure the function is correct.

```
poisson.loglik.grad = function(b) {
    b = as.matrix(b)
    lambda = exp(X %*% b)
    ell = -colSums(-as.vector(lambda) * X + y * X)
    return(ell)
}

# check the numerical gradient and the analytical gradient
b = c(0, 0.5)
grad(poisson.loglik, b)
```

```
## [1]  6542.46 45825.40
```

```
poisson.loglik.grad(b)
```

```
##             income
##  6542.46 45825.40
```

With the function of gradient, we are ready for BFGS.

```
res_BFGS = nloptr(x0 = b.init, eval_f = poisson.loglik, eval_grad_f = poisson.loglik.grad,
    opts = opts)
print(res_BFGS)
```

```
##
## Call:
##
## nloptr(x0 = b.init, eval_f = poisson.loglik, eval_grad_f = poisson.loglik.grad,
##     opts = opts)
##
##
## Minimization using NLopt version 2.4.0
##
```

```
## NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization stopped because
## xtol_rel or xtol_abs (above) was reached. )
##
## Number of Iterations....: 38
## Termination conditions:  xtol_rel: 1e-07
## Number of inequality constraints:  0
## Number of equality constraints:    0
## Optimal value of objective function:  261.114078295329
## Optimal value of controls: 1.177397 -0.09993984
```

Contrained Optimization

- `optimx` can handle simple box-constrained problems.
- `constrOptim` can handle linear constrained problems.
- Some algorithms in `nloptr`, for example, `NLOPT_LD_SLSQP`, can handle nonlinear constrained problems.
- `Rdonlp2` is an alternative for general nonlinear constrained problems. Rdonlp2 is a package offered by `Rmetric` project. It can be installed by `install.packages("Rdonlp2", repos="http://R-Forge.R-project.org")`

Convex Optimization

If a function is convex in its argument, then a local minimum is a global minimum. Convex optimization is particularly important in high-dimensional problems. The readers are referred to Boyd and Vandenberghe (2004) for an accessible comprehensive treatment. They claim that "convex optimization is technology; all other optimizations are arts." This is true to some extent.

Example

- linear regression model MLE
- Lasso (Su, Shi, and Phillips 2016)
- Relaxed empirical likelihood (Shi 2016).

A class of common convex optimization can be reliably implemented in R. `Rmosek` is an interface in R to access `Mosek`. Mosek a high-quality commercial solver dedicated to convex optimization. It provides free academic licenses. (Rtools is a prerequisite to install Rmosek.)

```r
require(Rmosek)
lo1 <- list()
lo1$sense <- "max"
lo1$c <- c(3, 1, 5, 1)
lo1$A <- Matrix(c(3, 1, 2, 0, 2, 1, 3, 1, 0, 2, 0, 3), nrow = 3, byrow = TRUE,
    sparse = TRUE)
lo1$bc <- rbind(blc = c(30, 15, -Inf), buc = c(30, Inf, 25))
lo1$bx <- rbind(blx = c(0, 0, 0, 0), bux = c(Inf, 10, Inf, Inf))
r <- mosek(lo1)
```

References

Boyd, Stephen, and Lieven Vandenberghe. 2004. *Convex Optimization.* Cambridge university press.

Nash, John C. 2014. "On Best Practice Optimization Methods in R." *Journal of Statistical Software* 60 (2): 1–14.

Shi, Zhentao. 2016. "Econometric Estimation with High-Dimensional Moment Equalities." *Journal of Econometrics* 195 (1). Elsevier: 104–19.

Su, Liangjun, Zhentao Shi, and Peter CB Phillips. 2016. "Identifying Latent Structures in Panel Data." *Econometrica* 84 (6). Wiley Online Library: 2215–64.