Numerical Optimization

Zhentao Shi

Jan 30, 2019

Numerical Optimization

Optimization is a key step to implement extremum estimators in econometrics. 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. However, there is no guarantee to locate the global minimum when multiple local minima exist.

Optimization without the equality and/or inequality constraints is called an *unconstrained* problem; otherwise it is called a *constrained* problem. The constraints can be incorporated into the criterion function via Lagrangian.

Methods

There are many optimization algorithms in the field of operational research; they are variants of a small handful of fundamental principles.

The essential idea for optimizing a twice-differentiable objective function is the Newton's method. A necessary condition for optimization is the first-order condition $s(\theta) = \partial f(\theta)/\partial \theta = 0$.

At an initial trial value θ_0, if $s(\theta_0) \neq 0$, the search is updated by

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

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) and/or $|\theta_{t+1} - \theta_t|/|\theta_t| < \epsilon$ (relative criterion), where ϵ is a small positive number chosen as a 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 verify whether a root to $s(\theta)$ is associated with a minimizer, a maximizer or a saddle point. If there are multiple minima, we compare the value at each to decide the global minimum.

It is clear that Newton's method requires 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 computationally expensive 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

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 algorithms.

Example

We use `optimx` to solve pseudo Poisson maximum likelihood estimation (PPML). If y_i is a continuous random variable, it obviously does not follow a Poisson 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 Poisson regression to obtain a consistent estimator of the parameter β even if y_i does not follow a conditional Poisson distribution.

If Z follows a Poisson distribution with mean λ, the probability mass function

$$\Pr(Z = k) = \frac{\mathrm{e}^{-\lambda}\lambda^k}{k!}, \text{ for } k = 0, 1, 2, \ldots,$$

so that

$$\log \Pr(Y = y|x) = -\exp(x'\beta) + y \cdot x'\beta - \log k!$$

Since the last term is irrelevant to the parameter, the log-likelihood function is

$$\ell(\beta) = \log \Pr(\mathbf{y}|\mathbf{x}; \beta) = -\sum_{i=1}^{n} \exp(x_i'\beta) + \sum_{i=1}^{n} y_i x_i'\beta.$$

In addition, it is easy to write the gradient

$$s(\beta) = \frac{\partial \ell(\beta)}{\partial \beta} = -\sum_{i=1}^{n} \exp(x_i'\beta)x_i + \sum_{i=1}^{n} y_i x_i.$$

and verify that the Hessian

$$H(\beta) = \frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta'} = -\sum_{i=1}^{n} \exp(x_i'\beta)x_i x_i'$$

is negative definite. Therefore, $\ell(\beta)$ is strictly concave in β.

In operational reserach, the default optimization is minimization, although economics has utility maximization and cost minimization and statistics has maximum likelihood estimation and minimal least squared estimation. To follow this convention in operational research, here we formulate the *negative* log-likelihood.

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

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 provided as additional arguments, these arguments must be explicit. (In constrast, in `Matlab` the parameter must be the sole argument for the function to be optimized, and data can only be injected through a nested function.)

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

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

## 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(
    reltol = 1e-7,
    abstol = 1e-7
  )
)
print(b.hat)
```

```
##                   p1          p2    value fevals gevals niter convcode  kkt1
## BFGS        1.177411 -0.09994234 261.1141     99     21    NA        0  TRUE
## Nelder-Mead 1.167261 -0.09703975 261.1317     53     NA    NA        0 FALSE
##             kkt2 xtime
## BFGS        TRUE 0.026
## Nelder-Mead TRUE 0.001
```

Given the conditional mean model, nonlinear least squares (NLS) is also consistent in theory. NLS minimizes

$$\sum_{i=1}^{n}(y_i - \exp(x_i\beta))^2$$

A natural question is: why do we prefer PPML to NLS? My argument is that, PPML's optimization for the linear index is globally convex, while NLS is not. It implies that the numerical optimization of PPML is easier and more robust than that of NLS. I leave the derivation of the non-convexity of NLS as an exercise.

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 is a useful tool to visualize the function surface/manifold in a low dimension.

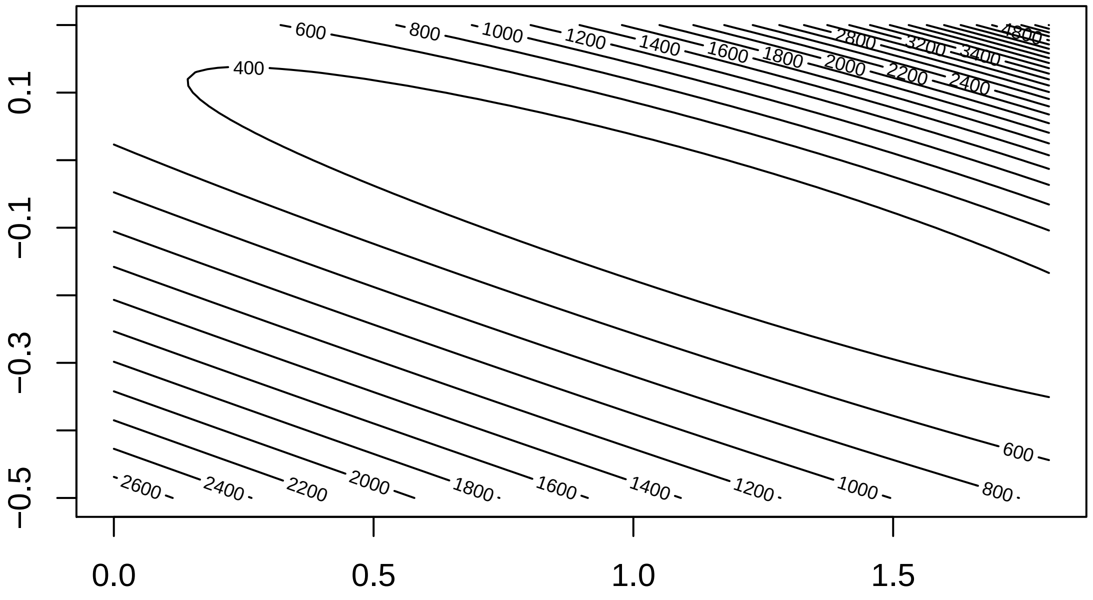
Example

```
x.grid <- seq(0, 1.8, 0.02)
x.length <- length(x.grid)
y.grid <- seq(-.5, .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 commercial Windows software.

`NLopt` offers an extensive list of algorithms.

Example

We first carry out the Nelder-Mead algorithm in NLOPT.

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

opts <- list(
  "algorithm" = "NLOPT_LN_NELDERMEAD",
  "xtol_rel" = 1.0e-7,
  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 2.4.2
##
## 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
```

```r
# "SLSQP" is indeed the BFGS algorithm in NLopt,
# though "BFGS" doesn't appear in the name
opts <- list("algorithm" = "NLOPT_LD_SLSQP", "xtol_rel" = 1.0e-7)
```

To invoke BFGS in NLOPT, we must code up the gradient $s(\beta)$, as in the function poisson.log.grad() below.

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

We compare the analytical gradient with the numerical gradient to make sure the function is correct.

```r
# check the numerical gradient and the analytical gradient
b <- c(0, .5)
grad(poisson.loglik, b)
```

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

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

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

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

```r
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 2.4.2
##
## 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.
- Some algorithms in `nloptr`, for example, `NLOPT_LD_SLSQP`, can handle nonlinear constrained problems.
- New packages to be elaborated next year: ROI General R optimization infrastructure

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

Normal MLE. The (negative) log-likelihood is

$$\ell(\beta, \sigma) = \log \sigma + \frac{1}{\sigma^2} \sum_{i=1}^{n} (y_i - x_i'\beta)^2$$

This is not a convex problem, because $\log \sigma$ is concave. But if we reparameterize the criterion function by $\gamma = 1/\sigma$ and $\alpha = \beta/\sigma$, then

$$\ell(\alpha, \gamma) = -\log \gamma + \sum_{i=1}^{n} (\gamma y_i - x_i'\alpha)^2$$

This function in convex in α, γ.

Indeed, many MLE estimators in the first-year econometrics are convex.

- Lasso (Su, Shi, and Phillips 2016)
- Relaxed empirical likelihood (Shi 2016).
- A companion paper can be found here (Gao and Shi 2020).

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

```r
lo1 <- list()
lo1$sense <- "max"
lo1$c <- c(3, 1, 5, 1)
lo1$A <- Matrix::Matrix(c(
  3, 1, 2, 0,
  2, 1, 3, 1,
  0, 2, 0, 3
),
nrow = 3, byrow = TRUE, sparse = TRUE
)
lo1$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
r <- Rmosek::mosek(lo1) # as today (Feb 10, 2019), the latest version of
# Rmosek is not functional. I can make it work on R3.4, but not the latest R3.5
```

- Convex solvers: CLEPX, MOSEK, Gurubi, (Commercial) ECOS, SDPT3, (free)
- `CVXR` (Fu, Narasimhan, and Boyd 2019) is a convex modeling language in R.

Relaxed empirical likelihood (REL) (Shi 2016)

Consider a model with a "true" parameter β_0 satisfying the moment condition $\mathrm{E}\left[g\left(Z_i,\beta_0\right)\right]=0_m$, where $\{Z_i\}_{i=1}^n$ is the observed data, $\beta \in \mathcal{B} \subset \mathbb{R}^D$ is a finite dimensional vector in the parameter space $\mathcal{B}$, and g is an $\mathbb{R}^m$-valued moment function. Empirical likelihood (EL) (Owen 1988) (Qin and Lawless 1994) solves

$$\max_{\beta\in\mathcal{B},\pi\Delta_n} \sum_{i=1}^n \log\left(\pi_i\right) \text{ s.t. } \sum_{i=1}^n \pi_i g\left(Z_i,\beta\right)=0_m$$

where $\Delta_n=\{\pi\in[0,1]^n : \sum_{i=1}^n \pi_i = 1\}$ is the n-dimensional probability simplex.

To handle the high-dimensional case, i.e., $m>n$, REL is defined as the solution to

$$\max_{\beta\in\mathcal{B}} \max_{\pi\in\Delta_n^\lambda(\beta)} \sum_{i=1}^n \log \pi_i$$

where

$$\Delta_n^\lambda\left(\beta\right)=\left\{\pi\in\Delta_n : \left|\sum_{i=1}^n \pi_i h_{ij}\left(\beta\right)\right| \le \lambda,\ j=1,2,\cdots,m\right\}$$

is a relaxed simplex, $\lambda \ge 0$ is a tuning parameter, $h_{ij}\left(\beta\right)=g_j\left(Z_i,\beta\right)/\hat{\sigma}_j\left(\beta\right)$, $g_j\left(Z_i,\beta\right)$ is the j-th component of $g\left(Z_i,\beta\right)$, and $\hat{\sigma}_j\left(\beta\right)$ is the sample standard deviation of $\{g_j\left(Z_i,\beta\right)\}_{i=1}^n$. The scale-normalization ensures that the estimation is invariant to a scale change of g.

Similar to standard EL, REL's optimization involves an inner loop and an outer loop. The outer loop for β is a general low-dimensional nonlinear optimization, which can be solved by Newton-type methods. With the linear constraints and the logarithm objective, the inner loop is convex in $\pi=\left(\pi_i\right)_{i=1}^n$. By introducing auxiliary variable, t_i, the logarithm objective can be formulated as a linear objective function $\sum_{i=1}^n t_i$ and n exponential conic constraints, $\left(\pi_i,1,t_i\right)\in\mathcal{K}_{\exp}=\{\left(x_1,x_2,x_3\right) : x_1\ge x_2\exp\left(x_3/x_2\right), x_2>0\}\cup\{\left(x_1,0,x_3\right) : x_1\ge 0, x_3\le 0\}$, $i=1,2,\cdots,n$. For each β, the inner problem can be then formulated as a conic programming problem,

$$\begin{aligned}
&\max_{\pi,t}\sum_{i=1}^n t_i\\
\text{s.t. } &\begin{bmatrix}1\\-\lambda\\\vdots\\-\lambda\end{bmatrix} \le \begin{bmatrix}1 & 1 & \cdots & 1\\ h_{11}(\beta) & h_{21}(\beta) & \cdots & h_{n1}(\beta)\\ \vdots & \vdots & \ddots & \vdots\\ h_{1m}(\beta) & h_{2m}(\beta) & \cdots & h_{nm}(\beta)\end{bmatrix}\begin{bmatrix}\pi_1\\\pi_2\\\vdots\\\pi_n\end{bmatrix} \le \begin{bmatrix}1\\\lambda\\\vdots\\\lambda\end{bmatrix}\\
&\left(\pi_i,1,t_i\right)\in\mathcal{K}_{\exp}, 0\le\pi_i\le 1,\ \text{for each } i=1,2,\cdots,n
\end{aligned}$$

and it is readily solvable in Rmosek by translating the mathematical expression into computer code.

```r
innerloop <- function(b, y, X, Z, tau) {
  n <- nrow(Z)
  m <- ncol(Z)

  # Generate moment condition
  H <- MomentMatrix(y, X, Z, b)

  # Initialize the mosek problem
  Prob <- list(sense = "max")

  # Prob$dparam$intpnt_nl_tol_rel_gap <- 1e-5;
  Prob$dparam <- list(INTPNT_CO_TOL_REL_GAP = 1e-5)

  # Linear coefficients of the objective
  Prob$c <- c(rep(0, n), rep(1, n), rep(0, n))

  # Linear constraints
  H_tilde <- Matrix(rbind(rep(1, n), H), sparse = TRUE)
  A <-
    rbind(cbind(H_tilde, Matrix(0, m + 1, 2 * n, sparse = TRUE)),
          cbind(Matrix(0, n, 2 * n, sparse = TRUE), Diagonal(n)))
  Prob$A <- A
  Prob$bc <-
    rbind(c(1, rep(-tau, m), rep(1, n)), c(1, rep(tau, m), rep(1, n)))
  Prob$bx <- rbind(c(rep(0, n), rep(-Inf, n), rep(1, n)),
                   c(rep(1, n), rep(0, n), rep(1, n)))

  # Exponential Cones
  NUMCONES <- n
  Prob$cones <- matrix(list(), nrow = 2, ncol = NUMCONES)
  rownames(Prob$cones) <- c("type", "sub")
  for (i in 1:n) {
    Prob$cones[, i] <- list("PEXP", c(i, 2 * n + i, n + i))
  }

  # Invoke Mosek
  mosek.out <- mosek(Prob, opts = list(verbose = 0, soldetail = 1))


  if (mosek.out$sol$itr$solsta == "OPTIMAL") {
    # Since the default of NLOPTR is to do minimization, need to set it as negative
    return(-mosek.out$sol$itr$pobjval)
  } else{
    warning("WARNING: Inner loop not optimized")
    return(Inf)
  }
}
```

The innerloop of the REL can also be solved by `CVXR` and `nloptr`, details see Gao and Shi (2020).

References

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

Fu, Anqi, Balasubramanian Narasimhan, and Stephen Boyd. 2019. "CVXR: An R Package for Disciplined Convex Optimization." *Journal of Statistical Software.*

Gao, Zhan, and Zhentao Shi. 2020. "Two Examples of Convex-Programming-Based High-Dimensional Econometric Estimators." *arXiv Preprint arXiv:1806.10423.*

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

Owen, Art B. 1988. "Empirical Likelihood Ratio Confidence Intervals for a Single Functional." *Biometrika* 75 (2). [Oxford University Press, Biometrika Trust]: 237–49.

Qin, Jin, and Jerry Lawless. 1994. "Empirical Likelihood and General Estimating Equations." *The Annals of Statistics* 22 (1). JSTOR: 300–325.

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.