MATH 680 Computation Intensive Statistics

October 23, 2018

Gradient Descent

Contents

1 Overview of Algorithms

Many algorithms to compute a local or global minimizer of an objective function $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X} \in \mathbb{R}^p$ is open, involve solving a sequence of univariate subproblems. Let $x^{(k)} \in \mathcal{X}$ be the current iterate. The next iterate is

$$x^{(k+1)} = x^{(k)} + t_k d^{(k)},$$

where $d^{(k)} \in \mathbb{R}^p$ is some direction and $t_k \in \mathbb{R}$ is some step-size. Given $d^{(k)}$, one possible choice is to use the exact line search for the step size

$$t_k = \arg\min_{x \in \mathbb{R}} g(t), \tag{1}$$

where $g(t) = f(x^{(k)} + td^{(k)})$. The feasible set in (1) could be replaced by $\{t \in \mathbb{R} : x^{(k)} + td^{(k)} \in \mathcal{X}\}$. Choosing t_k in this way ensures that $f(x^{(k+1)}) \leq f(x^{(k)})$. Approximations are used when computing (1) is difficult. The following are some named algorithms within this framework:

- Cyclical coordinate descent uses the columns of I_p for the $d^{(k)}$'s, e.g. $d^{(1)} = (1, 0, \ldots, 0)'$, $d^{(2)} = (0, 1, 0, , \ldots, 0)'$, etc.
- Blockwise coordinate descent. e.g. $d^{(1)} = (1, 1, 0, 0)'$, $d^{(2)} = (0, 0, 1, 1)'$, $d^{(3)} = (1, 1, 0, 0)'$, $d^{(4)} = (0, 0, 1, 1)'$.
- Steepest descent uses $d^{(k)} = -\nabla f(x^{(k)})$, and solves t_k in (1) with $t_k > 0$.
- Newton's method uses $d^{(k)} = -\{\nabla^2 f(x^{(k)})\}^{-1} \nabla f(x^{(k)})$ and $t_k = 1$.
- Quasi-Newton methods uses $d^{(k)} = -H_k^{-1} \nabla f(x^{(k)})$ where $H_k \in \mathbb{S}_+^p$ is chosen so that $d^{(k)} \approx -\{\nabla^2 f(x^{(k)})\}^{-1} \nabla f(x^{(k)})$.

2 Line Search Methods

2.1 Convexity of the univariate line search problem

Recall the univariate subproblem in 1, where the univariate objective function is defined by $g(t) = f(x^{(k)} + td^{(k)})$. If f is convex, then so is g. This is because for any two points y_1, y_2 in the domain of g and any $\lambda \in [0, 1]$,

$$
\begin{aligned}
g(\lambda y_1 + (1-\lambda) y_2) &= f(x^{(k)} + \{\lambda y_1 + (1-\lambda) y_2\} d^{(k)}) \\
&= f(\lambda \{x^{(k)} + y_1 d^{(k)}\} + (1-\lambda)\{x^{(k)} + y_2 d^{(k)}\}) \\
&\leq \lambda f(x^{(k)} + y_1 d^{(k)}) + (1-\lambda) f(x^{(k)} + y_2 d^{(k)}) \\
&= \lambda g(y_1) + (1-\lambda) g(y_2).
\end{aligned}
$$

2.2 Backtracking line search

```
backsearch <- function(f, grad.f, b, beta, alpha, quiet = FALSE,
    ...) {
    t = 1
    fk = f(b = b, ...)
    dk = grad.f(b = b, ...)
    while (1) {
        v_left = f(b = b - t * dk, ...)
        v_right = fk - alpha * t * crossprod(dk, dk)
        if (!quiet) {
            # print out condition check and current stepsize
            cat("f(x-t*dx)=", round(v_left, 2), "\n")
            cat("f(x)-at||dx||^2=", round(v_right, 2), "\n")
            cat("The current t is ", round(t, 2), "\n")
        }
        # check Armijo-Goldstein condition
        if (v_left <= v_right)
            break
        # shrink stepsize
        t = t * beta
    }
    return(t)
```

```
}
```

3 Gradient Descent

Algorithm 1 Gradient descent.

Pick an initial iterate $x^{(0)} \in \mathcal{X}$, pick a convergence tolerance $\tau > 0$, and set $k = 0$.

1. Compute $d^{(k)} = -\nabla f(x^{(k)})$.
2. If $d^{(k)} = 0$, then stop because $x^{(k)}$ is a stationary point. Otherwise, compute the step size
$$t_k = \arg\min_{t \in \mathbb{R}_+} g(t), \tag{2}$$
where $g(t) = f(x^{(k)} + td^{(k)})$.
3. Compute $x^{(k+1)} = x^{(k)} + t_k d^{(k)}$.
4. If $\|x^{(k+1)} - x^{(k)}\| < \tau \|x^{(0)}\|$, then stop. Otherwise, replace k by $k+1$ and go to step 1.

3.1 Why it is also called steepest descent

Definition 1. The directional derivatives of $f(x) = f(x_1, x_2, \ldots, x_n)$ at x along a vector $d = (d_1, d_2, \ldots, d_n)$ is

$$\nabla_d f(x) = \lim_{t \to 0_+} \frac{f(x + td) - f(x)}{t},$$

provided that this limit exists.

Note: If the function f is differentiable at x, then the directional derivative exists along any vector d, and one has

$$\nabla_d f(x) = \nabla f(x)' d,$$

where ∇ on the right hand side denotes the gradient vector and $\nabla_d f(x) = \nabla f(x)' d$ is the inner product. Intuitively, the directional derivative of f at a point x represents the rate of change of f when moving past x along direction d.

Proposition 1. *Let $\mathcal{X} \subset \mathbb{R}^p$ be open. Suppose that $f : \mathcal{X} \to \mathbb{R}$ is differentiable at x and there exists a $d \in \mathbb{R}^p$ such that $\nabla_d f(x) = \nabla f(x)' d < 0$. Then for all $t > 0$ sufficiently small, $f(x + td) < f(x)$. We call this d a descent direction.*

Proof. can be easily proved by combining the definition of $\nabla_d f(x)$ and the fact that $\nabla_d f(x) = \nabla f(x)' d$ when f is differentiable. □

Now to understand why the gradient descent algorithm is also called steepest descent, consider our current iterate $x^{(k)}$. Any direction $d \in \mathbb{R}^p$ is a descent direction if $\nabla_d f(x) = \nabla f(x^{(k)})' d < 0$ because by Proposition 1 we have $f(x^{(k)} + td) < f(x^{(k)})$ for all $t > 0$ sufficiently small. The steepest descent direction should give the smallest value in $\nabla_d f(x) = \nabla f(x^{(k)})' d$.

Suppose that $\|d\| = 1$ and $\nabla f(x^{(k)}) \neq 0$, then the unit-length steepest descent direction should return the smallest directional derivative along d, i.e.

$$\bar{d} = \arg \min_{\{d \in \mathbb{R}^p : \|d\| = 1\}} \nabla f(x^{(k)})' d. \tag{3}$$

We now show that the unit-length gradient descent direction d^*

$$d^* = -\frac{\nabla f(x^{(k)})}{\|\nabla f(x^{(k)})\|}$$

is a minimizer of (3), hence $\bar{d} = d^*$. From the Cauchy–Schwartz inequality, we can obtain a lower bound for the objective function in (3): for all $d \in \mathbb{R}^p$ with $\|d\| = 1$,

$$\nabla f(x^{(k)})' d \geq -\|\nabla f(x^{(k)})\| \|d\| = -\|\nabla f(x^{(k)})\|.$$

We see that the objective function in (3) evaluated at d^*, is equal to this this lower bound, so $d^* = \bar{d}$ is a global minimizer of (3). Therefore the unit-length direction of gradient descent is the unit-length direction of steepest descent.

3.2 Convergence of steepest descent

Proposition 2. *Let $\mathcal{X} \subset \mathbb{R}^p$ be open and suppose that $f : \mathcal{X} \to \mathbb{R}$ is differentiable on $\mathcal{X}$ and ∇f is continuous on the level set $S(x_0) = \{x \in \mathcal{X} : f(x) \leq f(x_0)\}$, which we assume is closed and bounded. Let $\{x^{(k)}\}$ be the sequence of points generated by the gradient descent algorithm. Then all limit points $\{x^{(k)}\}$ are stationary points of f.*

Proof. Since $f(x^{(k+1)}) \leq f(x^{(k)})$, we have that $\{x^{(k)}\} \subset S(x_0)$, so by the Weierstrass theorem, there exists a convergent subsequence $\{x_{j(k)}\} \to \bar{x} \in S(x_0)$. Assume that $\nabla f(\bar{x}) \neq 0$. Then there exists a $\bar{t} > 0$ such that

$$\delta = f(\bar{x}) - f(\bar{x} - \bar{t}\nabla f(\bar{x})) > 0$$

and

$$\bar{x} - \bar{t}\nabla f(\bar{x}) \in \text{interior}(S(x_0)).$$

Since we assumed ∇f was continuous on $S(x_0)$,

$$\lim_{k\to\infty} \{x_{j(k)} - \bar{t}\nabla f(x_{j(k)})\} = \bar{x} - \bar{t}\nabla f(\bar{x}).$$

So for k sufficiently large,

$$\begin{aligned} f(x_{j(k)} - \bar{t}\nabla f(x_{j(k)})) &\leq f(\bar{x} - \bar{t}\nabla f(\bar{x})) + \delta/2 \\ &= f(\bar{x}) - \delta + \delta/2 \\ &= f(\bar{x}) - \delta/2, \end{aligned}$$

but

$$f(\bar{x}) \leq f(x_{j(k)} - t_{j(k)}\nabla f(x_{j(k)})) \leq f(x_{j(k)} - \bar{t}\nabla f(x_{j(k)})) \leq f(\bar{x}) - \delta/2,$$

which is impossible because $\delta > 0$. $\square$

Remark: Gradient method does not handle non-differentiable problems. For example in Figure 1,

$$f(x) = \sqrt{x_1^2 + \gamma x_2^2} \quad (|x_2| < x_1), \qquad f(x) = \frac{x_1 + \gamma|x_2|}{\sqrt{1+\gamma}} \quad (|x_2| > x_1)$$

with exact line search, $x^{(0)} = (\gamma, 1)$, gradient method converges to non-optimal point.

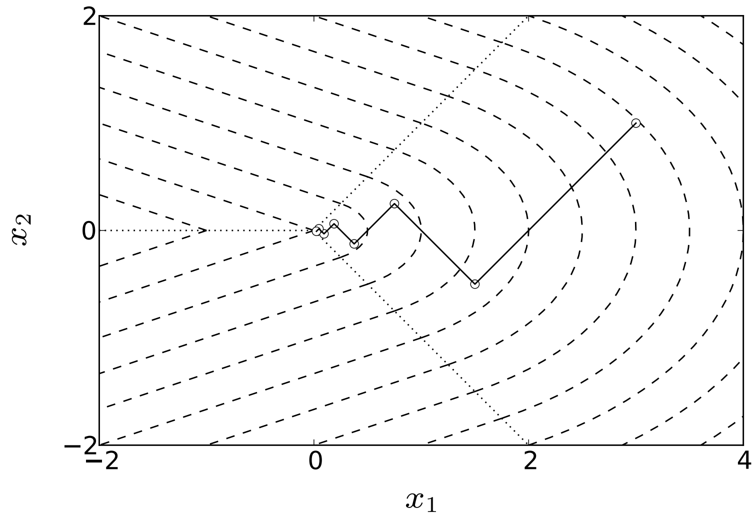


Figure 1: Gradient method does not handle non-differentiable problems.

3.3 Example: an alternative to the residual sum of squares function

In a linear regression model, let $y = (y_1, \ldots, y_n)'$ be the measured responses for the n cases and let $X \in \mathbb{R}^{n \times p}$ be the design matrix, where its ith row $x_i' = (1, x_{i2}, , \ldots, x_{ip})' \in \mathbb{R}^p$ has the values of the $p-1$ explanatory variables for the ith case $(i = 1, \ldots, n)$. The model assumes that y is realization of

$$Y = X\beta_* + \epsilon,$$

where $\beta_* = (\beta_{*1}, \ldots, \beta_{*p})' \in \mathbb{R}^p$ is the unknown regression coefficient vector and $\epsilon = (\epsilon_1, \ldots, \epsilon_n)'$ has $\epsilon_1, \ldots, \epsilon_n$ iid with an unspecified distribution having mean zero and unknown variance σ_*^2. Consider the estimator of β_* defined by

$$\hat{\beta}_{(\delta)} = \arg\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{n} |y_i - \beta' x_i|^{\delta},$$

where $\delta > 1$ is user-selected. Let $f : \mathbb{R}^p \to \mathbb{R}$ be the objective function. Then

$$\nabla f(\beta) = \delta \sum_{i=1}^{n} |y_i - \beta' x_i|^{\delta - 1} \text{sign}(\beta' x_i - y_i) x_i.$$

To use gradient descent in R to compute $\hat{\beta}_{(\delta)}$, we first define the gradient of f and the gradient of g (the univariate subproblem's objective function)

```
## define the objective function
f = function(b, y, X, delta) {
    val = sum(abs(y - X %*% b)^delta)
    return(val)
}


## define the gradient of the objective function
grad.f = function(b, y, X, delta) {
    val = delta * crossprod(X, abs(y - X %*% b)^(delta - 1) *
        sign(X %*% b - y))
    return(val)
}
```

```
## define the gradient of the univariate subproblem's
## objective function
grad.g = function(u, cur.pt, direc, ...) {
    val = sum(grad.f(cur.pt + u * direc, ...) * direc)
    return(val)
}
```

Here `grad.g` is defined for a general direction `direc`. In the steepest descent algorithm, `direc` will be $-\nabla f($`cur.pt`$)$. Let's define a function that performs steepest descent to minimize f

```
## a function that minimizes f by steepest descent
fit.delta.lm.sd = function(y, X, delta, b.start = NULL, tol = 1e-07,
    L = 1e-07, max.u = 1000, quiet = FALSE) {
    p = dim(X)[2]
    if (is.null(b.start))
        bk = rep(0, p) else bk = b.start

    k = 0
    iterating = TRUE
    while (iterating) {
        k = k + 1

        ## compute the gradient of f at our current iterate
```

```
        gf.at.bk = grad.f(b = bk, y = y, X = X, delta = delta)

        ## check if we have converged
        if (sum(abs(gf.at.bk)) < tol) {
            iterating = FALSE
        } else {
            ## the direction of steepest descent is
            direc = -1 * gf.at.bk

            ## compute the best step size in this direction
            uhat = bsearch(dg = grad.g, a0 = 0, b0 = max.u, L = L,
                quiet = TRUE, cur.pt = bk, direc = direc, y = y,
                X = X, delta = delta)

            ## update our current iterate by taking this step
            bk = bk + uhat * direc
        }
        if (!quiet) {
            cat("k=", k, "uhat=", uhat, "gf.at.bk=", gf.at.bk,
                "\n", "\t bk=", bk, "\n")
        }
    }
    return(list(b = bk, k = k))
```

```
}
```

Here is an example dataset

```
set.seed(680)
reps = 10000
n = 10
p = 4
sigma.star = 1/2

## create the true beta
beta.star = c(1, 0, 0, 2)

## randomly generate the design matrix
X = cbind(1, matrix(rnorm(n * (p - 1)), nrow = n, ncol = (p -
    1)))
y = X %*% beta.star + sigma.star * rnorm(n)

## ordinary least-squares estimate of beta.star
beta.hat = qr.coef(qr(x = X), y = y)

## alternative estimates of beta.star
delta = 2
quiet = TRUE
```

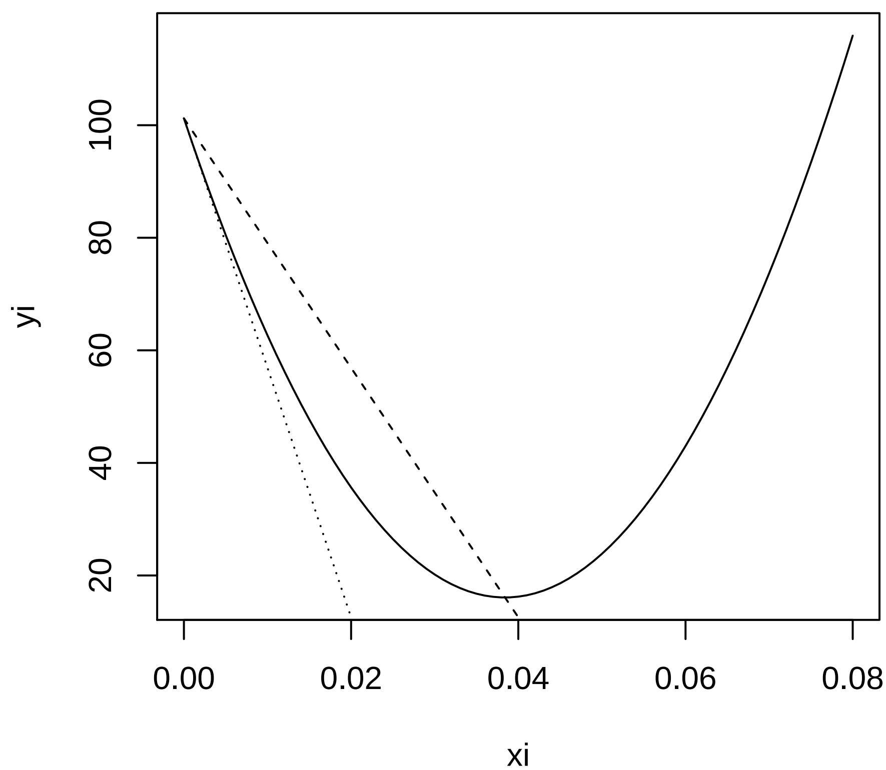
```
## computed with steepest descent
system.time(expr = (fitsd = fit.delta.lm.sd(y = y, X = X, delta = delta,
    L = 1e-10, quiet = quiet)))

##    user  system elapsed
##   0.029   0.000   0.029

## compare the alternative estimate to ordinary least-squares:
cbind(fitsd$b, beta.hat)

##         [,1]    [,2]
## [1,]  1.0794  1.0794
## [2,]  0.0072  0.0072
## [3,] -0.3383 -0.3383
## [4,]  1.8673  1.8673
```

Now we consider gradient descent using backtracking line search.

```
## define the objective function
f = function(b, y, X, delta) {
    val = sum(abs(y - X %*% b)^delta)
    return(val)
}


## define the gradient of the objective function
```

```r
grad.f = function(b, y, X, delta) {
    val = delta * crossprod(X, abs(y - X %*% b)^(delta - 1) *
        sign(X %*% b - y))
    return(val)
}

## make a demostrative plot for backtracking line ## search
bt_plot <- function(b, f = f, grad.f = grad.f, alpha, ...) {
    xi <- seq(0, 0.08, by = 0.001)
    yi <- rep(NA, length(xi))
    zi <- rep(NA, length(xi))
    ti <- rep(NA, length(xi))

    for (i in 1:length(xi)) {
        gk <- grad.f(b, y, X, delta)
        yi[i] <- f(b - xi[i] * gk, y, X, delta)
        zi[i] <- f(b, y, X, delta) - alpha * xi[i] * crossprod(gk,
            gk)
        ti[i] <- f(b, y, X, delta) - xi[i] * crossprod(gk, gk)
    }
    plot(xi, yi, type = "l")
    lines(xi, zi, lty = 2)
    lines(xi, ti, lty = 3)
```

```
}


bt_plot(b = c(2, 2, 2, 2), f = f, grad.f = grad.f, alpha = 0.5,
    y = y, X = X, delta = 2)
```




```
## a function that minimizes f by gradient descent ## using
## backtracking line search.
fit.delta.lm.bt = function(y, X, delta, b.start = NULL, tol = 1e-07,
```

```
L = 1e-07, max.u = 1000, quiet = FALSE) {
p = dim(X)[2]
if (is.null(b.start))
    bk = rep(0, p) else bk = b.start

k = 0
iterating = TRUE
while (iterating) {
    k = k + 1

    ## compute the gradient of f at our current iterate
    gf.at.bk = grad.f(b = bk, y = y, X = X, delta = delta)

    ## check if we have converged
    if (sum(abs(gf.at.bk)) < tol) {
        iterating = FALSE
    } else {
        ## compute the best step size in this direction
        uhat = backsearch(f = f, grad.f = grad.f, b = bk,
            beta = 0.7, alpha = 0.5, y = y, X = X, delta = delta,
            quiet = TRUE)

        ## update our current iterate by taking this step
```

```
            bk = bk - uhat * gf.at.bk
        }
        if (!quiet) {
            cat("k=", k, "uhat=", uhat, "gf.at.bk=", gf.at.bk,
                "\n", "\t bk=", bk, "\n")
        }
    }
    return(list(b = bk, k = k))
}

## computed with steepest descent
system.time(expr = (fitsd1 = fit.delta.lm.bt(y = y, X = X, delta = delta,
    L = 1e-10, quiet = quiet)))

##    user  system elapsed
##   0.024   0.000   0.024

## compare the alternative estimate to ordinary least-squares:
out = cbind(fitsd1$b, fitsd$b, beta.hat)
colnames(out) <- c("Backtracking", "Steepest", "OLS")
out

##      Backtracking Steepest     OLS
## [1,]       1.0794   1.0794  1.0794
## [2,]       0.0072   0.0072  0.0072
```

```
## [3,]      -0.3383  -0.3383 -0.3383
## [4,]       1.8673   1.8673  1.8673
```

A Exact line search

We now discuss three simple algorithms for *exactly* solving the univariate subproblem in (1).

A.1 Dichotomous search

Dichotomous search algorithm minimizes a potentially non-differentiable univariate function over a closed interval.

Algorithm 2 Dichotomous search.

Suppose that the univariate function $g : \mathbb{R} \to \mathbb{R}$ to be minimized is strictly quasi-convex over $[a_0, b_0]$. Given the initial interval of uncertainty $[a_0, b_0]$, the maximum width of the final interval of uncertainty L, and the parameter $\epsilon < L/2$, set $k = 0$.

1. If $b_k - a_k < L$ then stop.
2. Compute
$$\lambda = \frac{a_k + b_k}{2} - \epsilon, \qquad \mu = \frac{a_k + b_k}{2} + \epsilon.$$
3. If $g(\lambda) < g(\mu)$, then set $a_{k+1} = a_k$ and $b_{k+1} = \mu$.
Otherwise, if $g(\lambda) > g(\mu)$, then set $a_{k+1} = \lambda$ and $b_{k+1} = b_k$.
Otherwise, if $g(\lambda) = g(\mu)$, then set $a_{k+1} = \lambda$ and $b_{k+1} = \mu$.
4. Replace k by $k + 1$ and go to step 1.

Figure 2 demonstrates the algorithm. After K iterations, the length of the interval of uncertainty is $0.5^K(b_0 - a_0) + 2\epsilon(1 - 0.5^K)$. It is important to chose $L > 2\epsilon$.

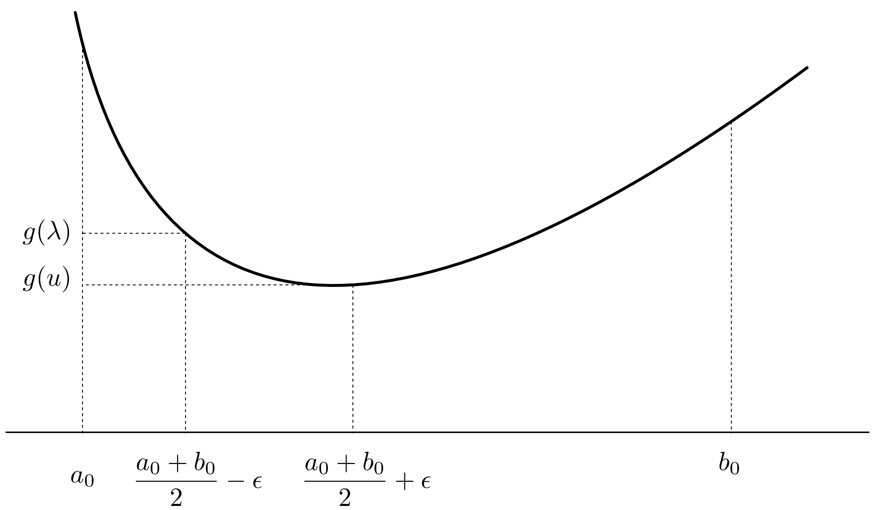


Figure 2: Dichotomous search

The following R code defines a function that does Dichotomous search for a minimizer of a univariate function.

```
## Dichotomous search Minimize a univariate strictly
## quasiconvex function over the interval [a0,b0] Arguments g,
## the function to minimize, where g(u, ...) is the function
## evaluated at u.  a0, left endpoint of the initial interval
## of uncertainty.  b0, right endpoint of the initial interval
## of uncertainty.  L, the maximum length of the final
## interval of uncertainty.  eps, search parameter, must be
## less than L/2.  quiet, should the function stay pouuiet?
## ..., additional argument specifications for g Returns the
## midpoint of the final interval of uncertainty
dsearch = function(g, a0, b0, L = 1e-07, eps = (L/2.1), quiet = FALSE,
    ...) {
    mm = mean(c(a0, b0))
    while (b0 - a0 > L) {
        lam = mm - eps
        mu = mm + eps
        g.at.lam = g(lam, ...)
        g.at.mu = g(mu, ...)
        if (g.at.lam < g.at.mu) {
            b0 = mu
        } else if (g.at.lam > g.at.mu) {
```

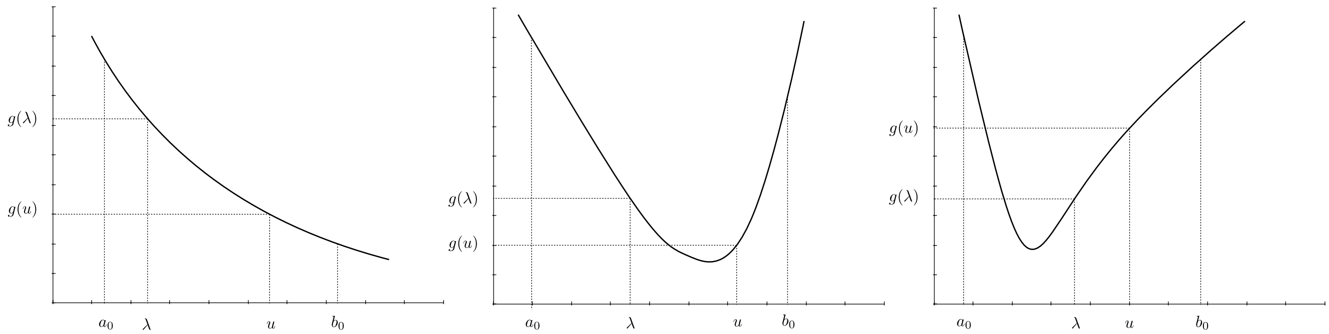


Figure 3: Dichotomous search: case 1, case 2 and case 3.

```
            a0 = lam
        } else {
            b0 = mu
            a0 = lam
        }
        if (!quiet)
            cat("new interval is", a0, b0, "\n")
        mm = mean(c(a0, b0))
    }
    return(mm)
}
```

A.2 Dichotomous search example: estimating the "center" of a distribution.

Suppose that measurements of a response $x_1, \ldots, x_n$ are a realization of n independent copies of the random variable $X \sim F_X$. Consider the estimate of the center of F defined by

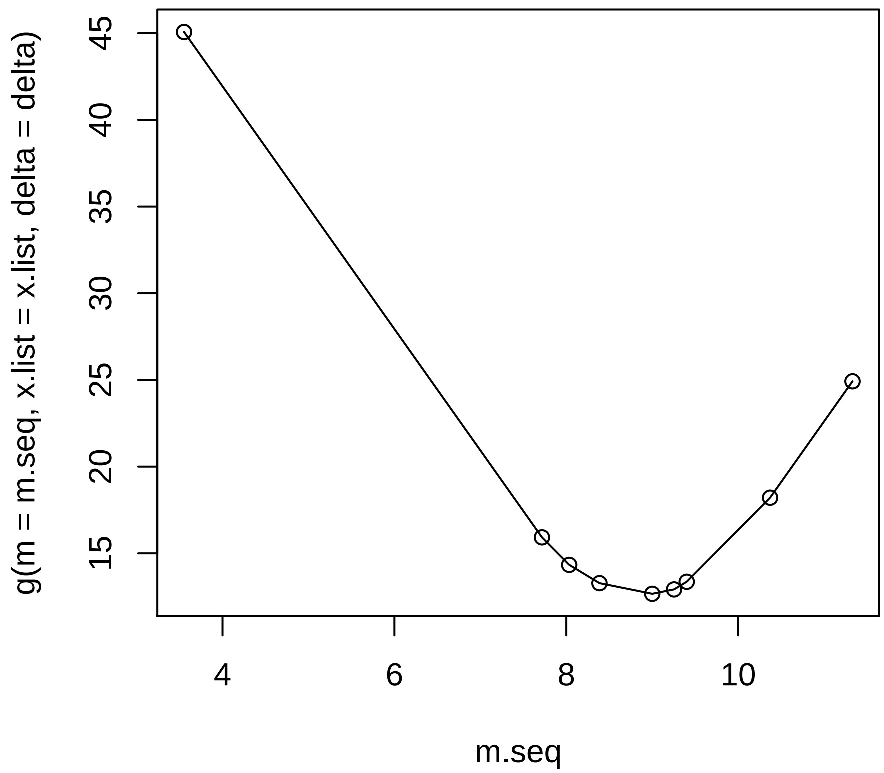
$$\hat{m}_\delta = \arg\min_{m \in \mathbb{R}} \sum_{i=1}^{n} \mid x_i - m \mid^{\delta},$$

where $\delta > 0$. Let $g(m; \delta, x_1, \ldots, x_n) : \mathbb{R} \to \mathbb{R}$ be the objective function, $g(m; \delta, x_1, \ldots, x_n) = \sum_{i=1}^{n} \mid x_i - m \mid^{\delta}$. If $\delta \geq 1$, then $g(m; \delta, x_1, \ldots, x_n)$ is convex. Also, if $\delta > 1$ then g is differentiable. We will use the dichotomous search algorithm to compute $\hat{m}_\delta$ when $\delta = 1$, $n = 9$, and $X = 10 + Z$, where Z has the t-distribution with $df = 3$.

```
## Example: Given x_1,...,x_n, Minimize g( ;x_1,...x_n): R ->
## R, where g(m; x_1,...,x_n)= sum_{i=1}^n |x_i - m|^(delta).
set.seed(680)
## generate a realization of an iid sample from a heavy tailed
## distribution with mean 10.
n = 9
mu.star = 10
x.list = mu.star + rt(n, df = 3)



## Objective function to minimize
g = function(m, x.list, delta) {
```

```
    len.m = length(m)
    if (len.m > 1) {
        ## this case is when we want to return a vector with ith entry
        ## g(m[i], x.list)
        mat = x.list %*% t(rep(1, len.m)) - rep(1, length(x.list)) %*%
            t(m)
        val = apply(abs(mat)^delta, 2, sum)
    } else {
        val = sum(abs(x.list - m)^delta)
    }
    return(val)
}

delta = 1

## Minimize g with Dichotomous search
mhat.1 = dsearch(g = g, a0 = min(x.list), b0 = max(x.list), quiet = TRUE,
    x.list = x.list, delta = delta)
mhat.1

## [1] 9

## this is the same as
median(x.list)
```

```
## [1] 9

## graph g
m.seq = seq(from = min(x.list), to = max(x.list), length.out = 1000)
plot(m.seq, g(m = m.seq, x.list = x.list, delta = delta), t = "l")
points(x.list, g(m = x.list, x.list = x.list, delta = delta))
```



A.3 Bisection search

Algorithm 3 Bisection search.

Given the initial interval of uncertainty $[a_0, b_0]$, and the width of the final interval of uncertainty L, set $k = 0$.

1. If $b_k - a_k < L$ then stop.
2. Compute
$$\lambda = \frac{a_k + b_k}{2}.$$
3. If $\nabla g(\lambda) > 0$, then set $a_{k+1} = a_k$ and $b_{k+1} = \lambda$.
Otherwise, if $\nabla g(\lambda) < 0$, then set $a_{k+1} = \lambda$ and $b_{k+1} = b_k$.
Otherwise, if $\nabla g(\lambda) = 0$, then stop because λ is the stationary point hence a global minimizer of g, since g is pseudo-convex.
4. Replace k by $k + 1$ and go to step 1.

After K iterations, the length of the interval of uncertainty is $0.5K(b_0 - a_0)$. The following R code defines a function that does Bisection search for a minimizer of a univariate function.

```
## Bisection search Minimize a univariate pseduconvex function
## over the interval [a0,b0] Arguments dg, the derivative of
## function to minimize, where dg(u, ...) is this derivative
## at u.  a0, left endpoint of the initial interval of
## uncertainty.  b0, right endpoint of the initial interval of
## uncertainty.  L, the maximum length of the final interval
## of uncertainty.  quiet, should the function stay quiet?
## Returns the midpoint of the final interval of uncertainty
```

```r
bsearch = function(dg, a0, b0, L = 1e-07, quiet = FALSE, ...) {
    mm = mean(c(a0, b0))
    ## compute gradient at the midpoint
    while (b0 - a0 > L) {
        dgm = dg(mm, ...)
        if (dgm < 0) {
            ## function is decreasing at mm new interval is [mm, b0]
            a0 = mm
        } else if (dgm > 0) {
            ## function is increasing at mm new interval is [a0, mm]
            b0 = mm
        } else {
            ## mm is a stationary point
            b0 = mm
            a0 = mm
        }
        if (!quiet)
            cat("new interval is", a0, b0, "\n")
        mm = mean(c(a0, b0))
    }
    return(mm)
}
```

A.4 Bisection search example: estimating the "center" of a distribution continued

We continue the example in section A.2. Recall that we are minimizing $g(m; \delta, x_1, \ldots, x_n) : \mathbb{R} \to \mathbb{R}$, where $g(m; \delta, x_1, \ldots, x_n) = \sum_{i=1}^{n} |x_i - m|^\delta$ and we must pick a $\delta > 1$ so that g is differentiable. Using the chain rule,

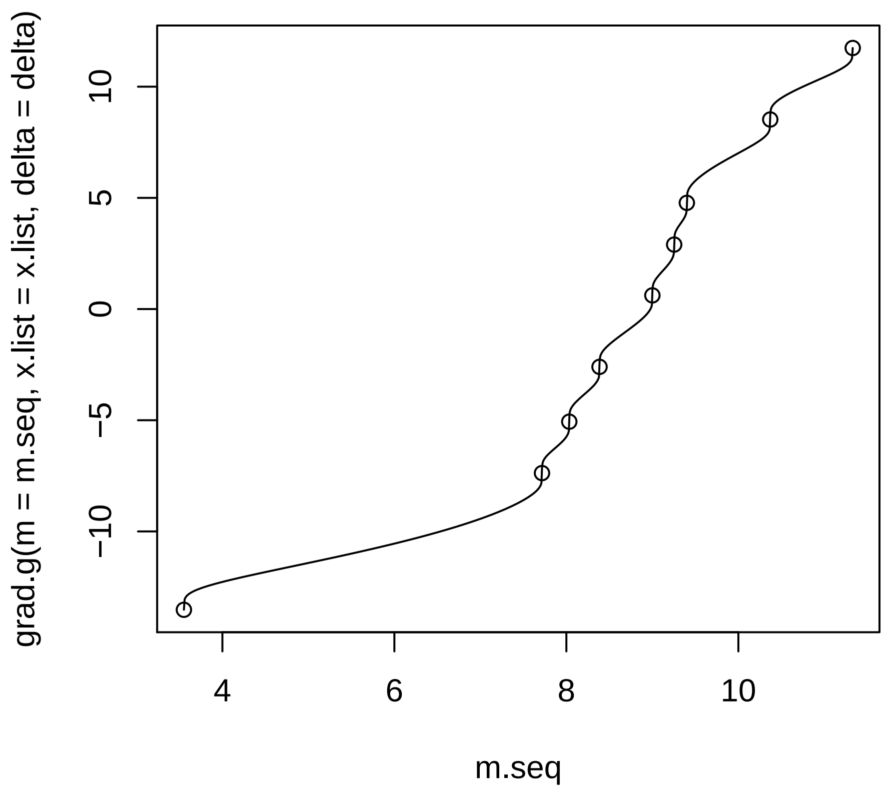
$$\nabla g(m) = \delta \sum_{i=1}^{n} \mid m - x_i \mid^{\delta - 1} \operatorname{sign}(m - x_i).$$

We will compare the Bisection search and dichotomous search algorithms to compute $\hat{m}_\delta$ when $\delta = 1.2$ using the same `x.list` generated in section A.2.

```
## gradient of objective function to minimize
grad.g = function(m, x.list, delta) {
    len.m = length(m)
    if (len.m > 1) {
        ## this case is when we want to return a vector with ith entry
        ## grad.g(m[i], x.list, delta)
        mat = rep(1, length(x.list)) %*% t(m) - x.list %*% t(rep(1,
            len.m))
        val = delta * apply(abs(mat)^(delta - 1) * sign(mat),
            2, sum)
    } else {
        val = delta * sum(abs(m - x.list)^(delta - 1) * sign(m -
```

```
            x.list))
    }
    return(val)
}


## Minimize the objective function with Bisection search
delta = 1.2
system.time(expr = (mhat.1.2 = bsearch(dg = grad.g, a0 = min(x.list),
    b0 = max(x.list), quiet = TRUE, x.list = x.list, delta = delta)))

##    user  system elapsed
##   0.018   0.000   0.017

mhat.1.2

## [1] 9

## compare to this minimizer from Dichotomous search
system.time(expr = (mhat.1.2d = dsearch(g = g, a0 = min(x.list),
    b0 = max(x.list), quiet = TRUE, x.list = x.list, delta = delta)))

##    user  system elapsed
##   0.000   0.000   0.001

mhat.1.2d
```

```
## [1] 9

## graph grad.g
m.seq = seq(from = min(x.list), to = max(x.list), length.out = 1000)
plot(m.seq, grad.g(m = m.seq, x.list = x.list, delta = delta),
    t = "l")
points(x.list, grad.g(m = x.list, x.list = x.list, delta = delta))
```



A.5 Golden section search

Reference:

`http://ezekiel.vancouver.wsu.edu/~cs330/lectures/minimization/gold/golden-search.pdf`