Lecture 5 - Optimization in one dimension

Björn Andersson (w/ Jianxin Wei)

Department of Statistics, Uppsala University

February 7, 2014

Some notes

- `chol2inv(A)` calculates the inverse of the matrix which has cholesky factorization `A`, i.e. to get the inverse of the matrix B we write `chol2inv(chol(B))`
- In `rank.condition()` in **corpcor**, the numerical rank is determined by comparing the singular values to a certain tolerance level. By default, for a matrix `A`, the tolerance is set to

  ```
  max(dim(A))*max(D)*.Machine$double.eps,
  ```

  where `D` are the singular values of `A`.

Table of Contents

Fundamental concepts of optimization

What are the components of an optimization problem?

- An objective function f to be minimized or maximixed.
- A decision variable $\mathbf{x}$
- Constraints, e.g. $g(\mathbf{x}) = 0$ (equality contraints) or $h(\mathbf{x}) \geq 0$ (inequality constraints)

Optimization in statistics

In a statistical context the decision variable will usually be the paramaters of a model, and f either the model likelihood to be maximized or a measure of discrepancy between data and predictions which is to be minimized.
Statistical examples are:

- Maximum likelihood estimator (MLE)
- Ordinary least squares (OLS)

Global minimum and local minimum

Let f be a continuous function on an interval (a, b).

- If, for a point $x^* \in (a, b)$, $f(x^*) \leq f(x)$ for all $x \in (a, b)$, then x^* is a global minimum
- If, for a point $x^* \in (a, b)$, $f(x^*) \leq f(x)$ for any feasible point in a neighbourhood of x^*, then x^* is a local minimum

Finding the global minimum and local minima

- A global minimum is difficult to find unless the objective function has special properties
- Most optimization methods use local information and are thus designed to only find local minima
- To find the global minimum, one strategy is to try several different starting points scattered throughout the feasible set

Note that maximization and minimization are equivalent: replace f with $-f$.

Unimodality

A function $f : \mathbf{R} \to \mathbf{R}$ is unimodal on an interval $[a, b]$ if there is a unique value $x^* \in [a, b]$ such that

1. $f(x^*)$ is the minimum of f on $[a, b]$
2. $f(x)$ is monotonically decreasing for $x < x^*$ and monotonically increasing for $x > x^*$

Table of Contents

Golden section search

What is the golden section search method?

- An iterative numerical method used in one-dimensional optimization
- The idea is to narrow down the interval that contains the local minimum until the length of the remaining interval is less than a pre-determined tolerance level

Golden section search

Let f be a unimodal function on $[a, b]$. Golden section search algorithm:

1. Choose a tolerance level tol
2. Compare two function values $f(x_1)$ and $f(x_2)$, $x_1, x_2 \in [a, b]$, and discard a subinterval of $[a, b]$
3. If the length of the remaining interval is less than tol, stop the process and provide the middle point of the remaining interval as the solution.

Step 2 in more detail

- We evaluate f at x_1 and x_2 where $x_1, x_2 \in [a, b]$ and $x_1 < x_2$.
  - If $f(x_1) > f(x_2)$, we know that there is not a minimum in $[a, x_1]$ since f is unimodal on $[a, b]$.
  - Likewise, if $f(x_1) < f(x_2)$, then the minimum can not be in $[x_2, b]$.
- Hence the interval is updated to $[a', b'] = [x_1, b]$ or $[a', b'] = [a, x_2]$.

How to choose x_1 and x_2?

At each iteration, we want to evaluate f only one time. Hence, we should select the new points such that one will be equal to the old function evaluation. It is accomplished by selecting x_1 and x_2 such that

$$x_1 = b - \frac{(b-a)}{\phi}$$

and

$$x_2 = a + \frac{(b-a)}{\phi},$$

where $\phi = (\sqrt{5}+1)/2$, the golden ratio.

Golden section search: remarks

- The function f must be unimodal on $[a, b]$.
- Advantages:
  - Guaranteed to converge to the true minimum/maximum
  - No derivatives are needed
- Disadvantage
  - The convergence rate is slow (linear)

Successive parabolic interpolation

- A technique for finding the minimum of a unimodal continuous function by successively fitting parabolas (polynomials of degree 2) to the function at three unique points
- At each iteration one of the old points is replaced with the minimum of the fitted parabola
- The point at which the fitted parabola is minimized is the approximation of the solution

A parabola is the ”simplest” function with an extreme value, the extreme value is attained at $-\frac{b}{2a}$

Parabolic interpolation

- For three points (which do not form a line), there is a unique parabola $ax^2 + bx + c$ that goes through all three points
- If the coordinates of the three points are (x_1, y_1), (x_2, y_2), (x_3, y_3), then the coefficients a, b and c can be found by solving the linear system:

$$\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Parabolic interpolation outline

Let f be a unimodal function on $[a, b]$. Choose a tolerance level tol

1. The function f is evaluated at three points and a quadratic polynomial (parabola) is fit
2. The minimum point of the resulting parabola (if there is a minimum) is taken as a new approximate minimum point of f
3. One of the previous points is replaced by this new minimum and the process is repeated until the selected tolerance level is attained

Parabolic interpolation remarks

- The function f to be minimized must be unimodal on $[a, b]$
- Advantages
  - Given that the method converges to the minimum, it converges quickly. The convergence rate is approximately 1.324 (i.e. superlinear)
  - No derivatives are needed
- Disadvantage
  - Convergence is not guaranteed when using the method in isolation

Brent's method

From *Algorithms for Minimization without Derivatives* (Brent, 1973).

- A method for continuous functions which uses a combination of a golden section search and successive parabolic interpolation. The method does not use derivatives.
- If the function to be optimized has a continuous second derivative which is positive at the minimum the method converges at a rate of approximately 1.324.
- More reliable than using only parabolic interpolation.

Newton's method

Newton's method is a common technique for root finding and optimization. Let f be a twice-differentiable function. The method proceeds by calculating

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}.$$

- In root finding, the local information is used to approximate $f'(x)$ by a line.
- In optimization, the local information is used to approximate $f(x)$ by a parabola

Newton-Raphson

Here we focus on optimization using Newton's method, called Newton-Raphson in the book.
The goal is to find the minimizer x^* of a function $f(x)$ on the interval $[a, b]$. Given that $x^* \neq a$ and $x^* = b$, we have that $f'(x^*) = 0$. This is a necessary but not sufficient condition for a minimum. A sufficient condition is that $f'(x^*) = 0$ and $f''(x^*) > 0$ both hold.

Newton-Raphson

If we have an initial guess x_0, we can approximate $f'(x)$ using a Taylor series expansion:

$$f'(x) \approx f'(x_0) + (x - x_0) \times f''(x_0).$$

If the right-hand side is zero, it provides an approximate solution to $f'(x^*) = 0$. We thus set the right-hand side equal to zero and solve for x:

$$\begin{aligned} f'(x_0) + (x - x_0) \times f''(x_0) &= 0 \\ \iff x \times f''(x_0) &= x_0 \times f''(x_0) - f'(x_0) \\ \iff x &= x_0 - \frac{f'(x_0)}{f''(x_0)}. \end{aligned}$$

Newton-Raphson algorithm for optimization

For a chosen tolerance level ϵ:

- Guess a starting value x_0.
- Compute an improved guess x_1 from

$$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

- Continue for $k = 2, \ldots$ until retrieving $|f'(x_k)| < \epsilon$

Newton-Raphson: remarks

- Newton's method may not converge (at all) or converge to a maximum or a saddle point
- If the starting value x_0 is close enough to the minimum, then the Newton method is guaranteed to converge to a local minimum.
- Advantage:
  - Convergence is fast, with convergence rate 2 provided the starting value is close enough to the solution.
- Disadvantages
  - Not guaranteed to converge to the true value
  - Requires derivatives

Table of Contents

Univariate optimization in R

- The function `optimize()` uses a combination of Golden section search and successive parabolic interpolation (Brent's method) to find the minimum of a unimodal function.
- The function `nlm()` offers a Newton-like method.

optimize()

```
optimize(f = , interval = ,  ..., lower = min(interval),
         upper = max(interval), maximum = FALSE,
         tol = .Machine$double.eps^0.25)
```

Exercise 7.1.1 a:

```
R> func1 <- function(x) return(abs(x-3.5) + abs(x-2)
+      + abs(x-1))
R> optimize(f=func1, interval=c(0, 5))
```

```
$minimum
[1] 2.000005

$objective
[1] 2.500005
```

optimize()

We may pass arguments to the function to optimize: (Exercise 7.1.1 b)

```
R> func2 <- function(x, y) return(abs(x-y[1]) +
+     abs(x-y[2]) + abs(x-y[3])+ abs(x-y[4]))
R> optimize(f=func2, interval=c(0,5), y=c(3.2, 3.5, 2, 1))
```

```
$minimum
[1] 2.553556


$objective
[1] 3.7
```

The function is always optimized with respect to the first argument of the function. Note that `minimum` is the argument value at the minimum.

nlm()

The function can conduct multivariate optimization but only univariate examples are presented here. If no derivatives are supplied, numerical derivatives are calculated and used. Example 7.1:

```
R> func3 <- function(x) return(exp(-x)+x^4)
R> sol3 <- nlm(f=func3, p=2)
R> sol3$minimum

[1] 0.6675038

R> sol3$estimate

[1] 0.5282519
```

Note that `minimum` is the function value at the minimum and `estimate` is the argument value at the minimum.

nlm()

We can add the first and second derivatives as attributes to the output of the function.

```
R> func4 <- function(x){
+      res <- exp(-x)+x^4
+      attr(res, "gradient") <- -exp(-x)+4*x^3
+      attr(res, "hessian") <- exp(-x)+12*x^2
+      return(res)
+  }
R> sol4 <- nlm(func4, 2)
R> sol4$iterations

[1] 7

R> sol3$iterations

[1] 10
```

nlm() vs optimize()

Using derivatives is faster in theory, but how the methods are implemented can make this not so in practice.

```
R> system.time({ for(i in 1:2000) optimize(func3,
+     interval=c(-1000, 1000), tol=1e-6)})

   user  system elapsed
   0.33    0.00    0.33

R> system.time({ for(i in 1:2000) nlm(func4, 2)})

   user  system elapsed
   0.41    0.00    0.41

R> system.time({ for(i in 1:2000) nlm(func4, 0.5)})

   user  system elapsed
   0.28    0.00    0.28
```

A faster Newton-Raphson

```
R> NewtonOpt <- function(x0, func, dfunc, d2func, tol=1e-06,
+      max.iter=100){
+      j <- 0
+      df <- dfunc(x0)
+      ddf <- d2func(x0)
+      for(i in 1:max.iter){
+          if(abs(df)<tol) break
+          x0 <- x0 - df/ddf
+          df <- dfunc(x0)
+          ddf <- d2func(x0)
+          j <- j + 1
+      }
+      return(list(objective=func(x0), minimum=x0, iterations=j))
+  }
```

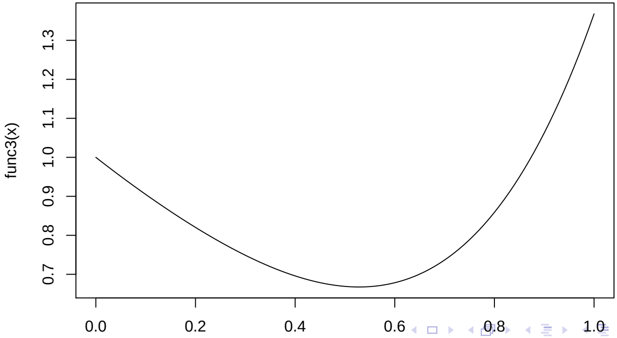
A faster Newton-Raphson function

```
R> NewtonOpt(2, func3, dfunc3, d2func3)
$objective
[1] 0.6675038

$minimum
[1] 0.5282519

$iterations
[1] 7
R> system.time({for(i in 1:2000) NewtonOpt(2, func3,
+     dfunc3, d2func3) })
   user  system elapsed
   0.15    0.00    0.15
```

Plotting tools

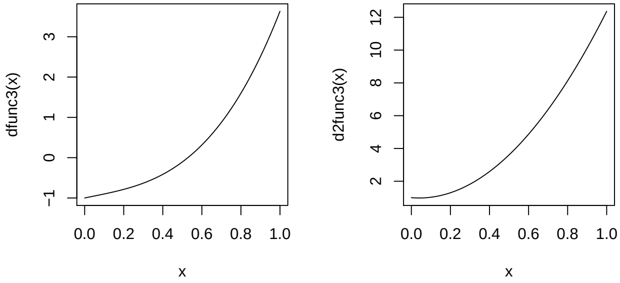
R> curve(func3, from=0, to=1)

Equivalent to `plot(func3, from=0, to=1)`.



Plotting tools

```
R> par(mfrow=c(1, 2))
R> curve(dfunc3, from=0, to=1)
R> curve(d2func3, from=0, to=1)
```


dfunc3(x)
x
d2func3(x)
x