Statistical Computing (36-350)

Lecture 17: Optimization I: Unconstrained, Deterministic Optimization

Cosma Shalizi

29 October 2012

Agenda

- Gradient descent and Newton's method
- Coordinate descent and Nelder-Mead
- Optimizing statistical functionals
- `optim`

Optimization Problems

Given an **objective function** $f : \mathcal{D} \mapsto R$, find

$$\theta^* = \underset{\theta}{\operatorname{argmin}} f(\theta)$$

Basics: maximizing f is minimizing $-f$:

$$\underset{\theta}{\operatorname{argmin}} -f(\theta) = \underset{\theta}{\operatorname{argmax}} f(\theta)$$

If h is strictly increasing (e.g., log), then

$$\underset{\theta}{\operatorname{argmin}} f(\theta) = \underset{\theta}{\operatorname{argmin}} h(f(\theta))$$

Examples of Optimization Problems

Minimize mean-squared error of regression surface (Gauss, c. 1800)
Maximize log-likelihood of distribution (Fisher, c. 1918)
Maximize output of plywood from given supplies and factories (Kantorovich, 1939)
Maximize output of tanks from given supplies and factories; minimize number of bombing runs to destroy factory (c. 1941–1945)
Maximize return of portfolio for given volatility (Markowitz, 1950s)
Minimize cost of airline flight schedule (Kantorovich...)
Maximize reproductive fitness of an organism (Maynard Smith)

Considerations

Approximation: How close can we get to θ^*, and/or $f(\theta^*)$?
Time complexity: How many computer steps does that take?
Will depend on precision of approximation, niceness of f, size of $\mathcal{D}$, size of data, method...
Big-O notation: write $h(x) = O(g(x))$ if $\lim_{x\to\infty} \frac{h(x)}{g(x)} = c$
e.g., $x^2 - 5000x + 123456778 = O(x^2)$
Useful to look at over-all scaling, hiding details
Most optimization algorithms use **successive approximation**, so distinguish number of iterations from cost of each iteration

As you remember from calculus. . .

Suppose domain $\mathcal{D}$ is $\mathbb{R}^p$, or some part of it
If θ^* is an **interior minimum** and f is differentiable,

$$\nabla f(\theta^*) = 0$$

If f is twice-differentiable,

$$\nabla^2 f(\theta^*) \geq 0$$

meaning for any vector v,

$$v^T \nabla^2 f(\theta^*) v \geq 0$$

$\nabla^2 f =$ the **Hessian, H**
Reverse is *not* true in general: even if $\nabla f(\theta) = 0$, $\mathbf{H}(\theta) \geq 0$, θ might only be a **local minimum**

Gradient Descent

1. Start with initial guess for θ, step-size η
2. While ((not too tired) and (making adequate progress))
    1. Find gradient $\nabla f(\theta)$
    2. Set $\theta \leftarrow \theta - \eta \nabla f(\theta)$
3. Return final θ as approximate θ^*

Variations: adaptively adjust η to make sure of improvement or search along the gradient direction for minimum

Pros and Cons of Gradient Descent

Pro:

- Moves in direction of greatest immediate improvement
- If η is small enough, gets to a local minimum eventually, and then stops
- For nice f, $f(\theta) \leq f(\theta^*) + \epsilon$ in $O(\epsilon^{-2})$ iterations
  For *very* nice f, only $O(\log \epsilon^{-1})$ iterations
- To get $\nabla f(\theta)$, take p derivatives, so each iteration costs $O(p)$

Cons:

- "Sufficiently small" η can be really, really small
- Slow progress or zig-zagging if components of ∇f are very different sizes
- Taking derivatives can slow down as data grows — really $O(np)$ per iteration

Newton's Method

Use a Taylor expansion:

$$f(\theta^*) \approx f(\theta) + (\theta^* - \theta)\nabla f(\theta) + \frac{1}{2}(\theta^* - \theta)^T \mathbf{H}(\theta)(\theta^* - \theta)$$

Take gradient with respect to θ^* and set to zero:

$$\begin{aligned} 0 &= \nabla f(\theta) + \mathbf{H}(\theta)(\theta^* - \theta) \\ \theta^* &= \theta - (\mathbf{H}(\theta))^{-1}\nabla f(\theta) \end{aligned}$$

Works *exactly* if f is quadratic

so that $\mathbf{H}^{-1}$ exists, etc.

If f isn't quadratic, keep pretending it is until we get close to θ^*, when it will be nearly true

Newton's Method: The Algorithm

1. Start with guess for θ
2. While ((not too tired) and (making adequate progress))
    1. Find gradient $\nabla f(\theta)$ and Hessian $\mathbf{H}(\theta)$
    2. Set $\theta \leftarrow \theta - \mathbf{H}(\theta)^{-1}\nabla f(\theta)$
3. Return final θ as approximation to θ^*

Like gradient descent, but with inverse Hessian giving the step-size

"This is about how far you can go with that gradient"

Advantages and Disadvantages of Newton's Method

Pro:

- Step-sizes chosen adaptively through 2nd derivatives, much harder to get zig-zagging, over-shooting, etc.
- Also guaranteed to need $O(\epsilon^{-2})$ steps to get within ϵ of optimum
- Only $O(\log\log\epsilon^{-1})$ for very nice functions
- Typically many fewer iterations than gradient descent

Cons:

- Hopeless if $\mathbf{H}$ doesn't exist or isn't invertible
- Need to take $O(p^2)$ second derivatives *plus* p first derivatives
- Need to solve $\mathbf{H}\theta_{\text{new}} = \mathbf{H}\theta_{\text{old}} - \nabla f(\theta_{\text{old}})$ for θ_{new}

  inverting $\mathbf{H}$ is $O(p^3)$, but cleverness gives $O(p^2)$ for solving

Getting Around the Hessian

Want to use the Hessian to improve convergence
Don't want to have to keep computing the Hessian at each step
Approaches

- Use knowledge of the system to get some approximation to the Hessian, use that instead of taking derivatives ("Fisher scoring")
- Use only diagonal entries (p unmixed 2nd derivatives)
- Use $\mathbf{H}(\theta)$ at initial guess, hope $\mathbf{H}$ changes *very* slowly with θ
- Re-compute $\mathbf{H}(\theta)$ every k steps, $k > 1$
- Fast, approximate updates to the Hessian at each step (BFGS)

Coordinate Descent

Gradient methods adjust all coordinates at once
Try this instead:

1. Start with initial guess θ
2. While ((not too tired) and (making adequate progress))
   - For $i \in (1:p)$
     1. do 1D optimization over i^{th} coordinate of θ, holding the others fixed
     2. Update i^{th} coordinate to this optimal value
3. Return final value of θ

Needs a good 1D optimizer, and can bog down for very tricky functions, but can also be extremely fast and simple

Nelder-Mead, a.k.a. the Simplex Method

Try to cage θ^* with a **simplex** of $p+1$ points
Order the trial points, $f(\theta_1) \leq f(\theta_2) \ldots \leq f(\theta_{p+1})$
θ_{p+1} is the worst guess — try to improve it
$\theta_0 = \frac{1}{n}\sum_{i=1}^{n} \theta_i$ = center of the not-worst

- **Reflection**: Try $x_0 - (x_{p+1} - x_0)$, the opposite side of the center from x_{p+1}
  - if it's better than x_p but not than x_1, replace the old x_{p+1} with it
  - **Expansion**: if the reflect point is better than the best, try $x_0 - 2(x_{p+1} - x_0)$; replace the old x_{p+1} with the better of the reflected and the expanded point
- **Contraction**: If the reflected point is worse that x_p, try $x_0 + \frac{x_{p+1} - x_0}{2}$; if the contracted value is better, replace x_{p+1} with it
- **Reduction**: If all else fails, $x_i \leftarrow \frac{x_1 + x_i}{2}$

Making Sense of Nedler-Mead

The Moves:

- Reflection: try the opposite of the worst point
- Expansion: if that really helps, try it some more
- Contraction: see if we overshot when trying the opposite
- Reduction: if all else fails, try being more like the best point

Pros:

- Each iteration ≤ 4 evaluations of f, plus sorting (at most $O(p \log p)$, usually much better)
- No derivatives used, can even work for dis-continuous f

Con:

- Can need *many* more iterations than gradient methods

Optimizing Statistical Functionals

Optimizing for statistics is funny: we know our objective function is noisy

Have $\hat{f}_n$ (sample objective) but want to minimize f (population objective)

Why optimize $\hat{f}_n$ to $\pm 10^{-6}$ when $\hat{f}$ only matches f to ± 1?

If $\hat{f}_n$ is an average over data points, then (law of large numbers)

$$\mathbb{E}\left[\hat{f}_n(\theta)\right] = f(\theta)$$

and (central limit theorem)

$$\hat{f}_n(\theta) - f(\theta) = O(n^{-1/2})$$

Can use probability theory to analyze how closely the sample optimum matches the population optimum

Statistical Theory in Two Slides

$$\begin{aligned}
\hat{\theta}_n &= \operatorname*{argmin}_{\theta} \hat{f}_n(\theta) \\
\nabla \hat{f}_n(\hat{\theta}_n) &= 0 \\
&\approx \nabla \hat{f}_n(\theta^*) + \widehat{\mathbf{H}}_n(\theta^*)(\hat{\theta}_n - \theta^*) \\
\hat{\theta}_n &\approx \theta^* - \widehat{\mathbf{H}}_n^{-1}(\theta^*)\nabla \hat{f}_n(\theta^*)
\end{aligned}$$

Opposite expansion to Newton's method

When does $\widehat{\mathbf{H}}_n^{-1}(\theta^*)\nabla \hat{f}_n(\theta^*) \to 0$?

$$\begin{aligned}
\widehat{\mathbf{H}}_n(\theta^*) &\to \mathbf{H}(\theta^*) \text{ (by LLN)} \\
\nabla \hat{f}_n(\theta^*) - \nabla f(\theta^*) &= O(n^{-1/2}) \text{ (by CLT) but } \nabla f(\theta^*) = 0 \\
\therefore \nabla \hat{f}_n(\theta^*) &= O(n^{-1/2}) \\
\text{Var}\left[\nabla \hat{f}_n(\theta^*)\right] &\to n^{-1}\mathbf{K}(\theta^*) \text{ (CLT again)}
\end{aligned}$$

How much noise is there in $\hat{\theta}_n$?

$$\begin{aligned}
\mathrm{Var}\left[\hat{\theta}_n\right] &= \mathrm{Var}\left[\hat{\theta}_n - \theta^*\right] \\
&= \mathrm{Var}\left[\widehat{\mathbf{H}}_n^{-1}(\theta^*)\nabla\hat{f}_n(\theta^*)\right] \\
&= \widehat{\mathbf{H}}_n^{-1}(\theta^*)\mathrm{Var}\left[\nabla\hat{f}_n(\theta^*)\right]\widehat{\mathbf{H}}_n^{-1}(\theta^*) \\
&\to n^{-1}\mathbf{H}^{-1}(\theta^*)\mathbf{K}(\theta^*)\mathbf{H}^{-1}(\theta^*) = O(pn^{-1})
\end{aligned}$$

How much noise is there in $f(\hat{\theta}_n)$?

$$\begin{aligned}
f(\hat{\theta}_n) - f(\theta^*) &\approx \frac{1}{2}(\hat{\theta}_n - \theta^*)^T\mathbf{H}(\theta^*)(\hat{\theta}_n - \theta^*) \\
\mathrm{Var}\left[f(\hat{\theta}_n) - f(\theta^*)\right] &\approx \mathrm{tr}\left(\mathbf{H}(\theta^*)\mathrm{Var}\left[\hat{\theta}_n - \theta^*\right]\mathbf{H}(\theta^*)\mathrm{Var}\left[\hat{\theta}_n - \theta^*\right]\right) \\
&\to n^{-2}\,\mathrm{tr}\left(\mathbf{K}(\theta^*)\mathbf{H}^{-1}(\theta^*)\mathbf{K}(\theta^*)\mathbf{H}^{-1}(\theta^*)\right) \\
&= O(pn^{-2})
\end{aligned}$$

What You Need to Remember

If everything works out ideally (maximum likelihood, correct model) $\mathbf{K} = \mathbf{H}$, and

$$\begin{aligned} \hat{\theta}_n &\approx \theta^* - \widehat{\mathbf{H}}_n^{-1}(\theta^*)\nabla \hat{f}_n(\theta^*) \\ \text{Var}\left[\hat{\theta}_n\right] &\approx n^{-1}\mathbf{H}^{-1}(\theta^*) \approx n^{-1}\mathbf{H}(\hat{\theta}_n) \\ \text{Var}\left[f(\hat{\theta}_n) - f(\theta^*)\right] &\approx n^{-2}p \end{aligned}$$

If $\mathbf{K} \neq \mathbf{H}$, do the algebra and deal with more noise

$\therefore$ Little point to optimizing $\hat{f}_n$ *much* more precisely than $\pm\sqrt{p/n^2}$

Optimization in R: `optim`

```
optim(par,fn, gr, method, control, hessian)
```

`fn`	function to be minimized; mandatory
`par`	initial parameter guess; mandatory
`gr`	gradient function; only needed for some methods
`method`	defaults to Nelder-Mead, could be BFGS (Newton-ish)
`control`	optional list of control settings (maximum iterations, scaling, tolerance for convergence, etc.)
`hessian`	should the final Hessian be returned? default FALSE

Return contains the location (`$par`) and the value (`$val`) of the optimum, diagnostics, possibly `$hessian`

```
mse <- function(theta) { mean((gmp$pcgmp - theta[1]*gmp$pop^theta[2])^2) }
grad.mse <- function(theta) { grad(func=mse,x=theta) }
theta0=c(5000,0.15)
fit1 <- optim(theta0,mse,hessian=TRUE) # Nelder-Mead
fit2 <- optim(theta0,mse,grad.mse,method="BFGS",hessian=TRUE)
```

Let's compare the two attempts at optima

fit1: Derivative-free simplex method

Run-time: 0.013 seconds

```
> fit1
$par
[1] 6492.7390560    0.1276986

$value
[1] 61853983

$counts
function gradient
     203       NA

$convergence
[1] 0

$message
NULL

$hessian
              [,1]          [,2]
[1,] 5.250983e+01       4422941
[2,] 4.422941e+06 375813287390
```

fit2: Newton-ish BFGS method

Run-time: 0.027 seconds

```
> fit2
$par
[1] 6493.2563738    0.1276921

$value
[1] 61853983

$counts
function gradient
      63       11

$convergence
[1] 0

$message
NULL

$hessian
             [,1]          [,2]
[1,] 5.25021e+01       4422070
[2,] 4.42207e+06 375729087977
```

Summary

1. Trade-offs: complexity of iteration vs. number of iterations vs. precision of approximation
   - Simplex: very robust, each iteration simple, doesn't take advantage of smoothness
   - Gradient descent: more complex iterations, more guarantees, more adaptive
   - Newton: even more complex iterations, but few of them for good functions
2. Noise limits how much optimization is worth doing
3. Start with pre-built code like `optim`, implement your own only if needed