Statistical Computing (36-350)

Lecture 10: Optimization I: Basics

Cosma Shalizi

30 September 2013

Agenda

- Basics of optimization
- Gradient descent
- Newton's method
- Curve-fitting
- R: `optim`, `nls`

READING: Recipes 13.1 and 13.2 in *The R Cookbook*
OPTIONAL READING: 1.1, 2.1 and 2.2 in *Red Plenty*

Examples of Optimization Problems

Minimize mean-squared error of regression surface (Gauss, c. 1800)
Maximize 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. 1939–1945)
Maximize return of portfolio for given volatility (Markowitz, 1950s)
Minimize cost of airline flight schedule (Kantorovich...)
Maximize reproductive fitness of an organism (Maynard Smith)

Optimization Problems

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

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

Optimization Problems

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

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

Basics: maximizing f is minimizing $-f$:

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

Optimization Problems

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

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

Considerations

Approximation: How close can we get to θ^*, and/or $f(\theta^*)$?

Considerations

Approximation: How close can we get to θ^*, and/or $f(\theta^*)$?
Time complexity: How many computer steps does that take?

Varies with precision of approximation, niceness of f, size of $\mathscr{D}$, size of data, method…

Considerations

Approximation: How close can we get to θ^*, and/or $f(\theta^*)$?
Time complexity: How many computer steps does that take?
Varies with precision of approximation, niceness of f, size of $\mathscr{D}$, size of data, method...
Most optimization algorithms use **successive approximation**, so distinguish number of iterations from cost of each iteration

As you remember from calculus…

Suppose x is one dimensional and f is smooth

As you remember from calculus. . .

Suppose x is one dimensional and f is smooth
If x^* is an **interior** minimum / maximum / extremum point

$$\left.\frac{df}{dx}\right|_{x=x^*} = 0$$

As you remember from calculus…

Suppose x is one dimensional and f is smooth
If x^* is an **interior** minimum / maximum / extremum point

$$\left.\frac{df}{dx}\right|_{x=x^*} = 0$$

If x^* a minimum,

$$\left.\frac{d^2f}{dx^2}\right|_{x=x^*} > 0$$

As you remember from calculus…

This all carries over to multiple dimensions:
At an **interior extremum,**

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

As you remember from calculus…

This all carries over to multiple dimensions:
At an **interior extremum**,

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

At an **interior minimum**,

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

meaning for any vector v,

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

$\nabla^2 f =$ the **Hessian, H**

As you remember from calculus. . .

This all carries over to multiple dimensions:
At an **interior extremum,**

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

At an **interior minimum,**

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

meaning for any vector v,

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

$\nabla^2 f =$ the **Hessian, H**
θ might just be a **local** minimum

Gradients and Changes to f

$$f'(x_0) = \left.\frac{df}{dx}\right|_{x=x_0} = \lim_{x \rightarrow x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

Gradients and Changes to f

$$\begin{aligned} f'(x_0) &= \left.\frac{df}{dx}\right|_{x=x_0} = \lim_{x\to x_0} \frac{f(x)-f(x_0)}{x-x_0} \\ f(x) &\approx f(x_0) + (x-x_0)f'(x_0) \end{aligned}$$

Gradients and Changes to f

$$\begin{aligned} f'(x_0) &= \left.\frac{df}{dx}\right|_{x=x_0} = \lim_{x\rightarrow x_0}\frac{f(x)-f(x_0)}{x-x_0} \\ f(x) &\approx f(x_0)+(x-x_0)f'(x_0) \end{aligned}$$

Locally, the function looks linear

Gradients and Changes to f

$$f'(x_0) = \left.\frac{df}{dx}\right|_{x=x_0} = \lim_{x\to x_0} \frac{f(x)-f(x_0)}{x-x_0}$$

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

Locally, the function looks linear
To minimize a linear function, move down the slope

Gradients and Changes to f

$$\begin{aligned} f'(x_0) &= \left.\frac{df}{dx}\right|_{x=x_0} = \lim_{x\rightarrow x_0} \frac{f(x)-f(x_0)}{x-x_0} \\ f(x) &\approx f(x_0) + (x-x_0)f'(x_0) \end{aligned}$$

Locally, the function looks linear
To minimize a linear function, move down the slope
Multivariate version:

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0)\cdot\nabla f(\theta_0)$$

$\nabla f(\theta_0)$ points in the direction of fastest ascent at θ_0

Gradient Descent

```
1. Start with initial guess for $\theta$, step-size $\eta$
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 $\theta$ as approximate $\theta^*$
```

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

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

Cons:

- "Sufficiently small" η can be really, really small
- Slowness or zig-zagging if components of ∇f are very different sizes

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

Cons:

- "Sufficiently small" η can be really, really small
- Slowness or zig-zagging if components of ∇f are very different sizes

How much work do we need?

Scaling

Big-O notation:

$$h(x) = O(g(x))$$

means

$$\lim_{x \to \infty} \frac{h(x)}{g(x)} = c$$

for some $c \neq 0$

Scaling

Big-O notation:

$$h(x) = O(g(x))$$

means

$$\lim_{x \to \infty} \frac{h(x)}{g(x)} = c$$

for some $c \neq 0$

e.g., $x^2 - 5000x + 123456778 = O(x^2)$

$e^x/(1+e^x) = O(1)$

Scaling

Big-O notation:

$$h(x) = O(g(x))$$

means

$$\lim_{x \to \infty} \frac{h(x)}{g(x)} = c$$

for some $c \neq 0$

e.g., $x^2 - 5000x + 123456778 = O(x^2)$

$e^x/(1+e^x) = O(1)$

Useful to look at over-all scaling, hiding details

Scaling

Big-O notation:

$$h(x) = O(g(x))$$

means

$$\lim_{x \to \infty} \frac{h(x)}{g(x)} = c$$

for some $c \neq 0$

e.g., $x^2 - 5000x + 123456778 = O(x^2)$

$e^x/(1+e^x) = O(1)$

Useful to look at over-all scaling, hiding details

Also done when the limit is $x \to 0$

How Much Work is Gradient Descent?

Pro:

- 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, $\therefore$ each iteration costs $O(p)$

Con:

- Taking derivatives can slow down as data grows — each iteration might really be $O(np)$

Taylor Series

What if we do a quadratic approximation to f?

Taylor Series

What if we do a quadratic approximation to f?

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

Taylor Series

What if we do a quadratic approximation to f?

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0)$$

Special cases of general idea of Taylor approximation

Taylor Series

What if we do a quadratic approximation to f?

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0)$$

Special cases of general idea of Taylor approximation
Simplifies if x_0 is a minimum since then $f'(x_0) = 0$:

$$f(x) \approx f(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0)$$

Near a minimum, smooth functions look like parabolas

Taylor Series

What if we do a quadratic approximation to f?

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0)$$

Special cases of general idea of Taylor approximation
Simplifies if x_0 is a minimum since then $f'(x_0) = 0$:

$$f(x) \approx f(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0)$$

Near a minimum, smooth functions look like parabolas
Carries over to the multivariate case:

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

Minimizing a Quadratic

If we know

$$f(x) = ax^2 + bx + c$$

we minimize exactly:

$$\begin{aligned} 2ax^* + b &= 0 \\ x^* &= \frac{-b}{2a} \end{aligned}$$

Minimizing a Quadratic

If we know

$$f(x) = ax^2 + bx + c$$

we minimize exactly:

$$\begin{aligned} 2ax^* + b &= 0 \\ x^* &= \frac{-b}{2a} \end{aligned}$$

If

$$f(x) = a(x - x_0)^2 + b(x - x_0) + c$$

then

$$x^* = x_0 - \frac{1}{2}\frac{b}{a}$$

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:

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

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.

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 $\theta$
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 $\theta$ as approximation to $\theta^*$
```

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

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 $\mathbf{H}\theta_{\mathrm{new}} = \mathbf{H}\theta_{\mathrm{old}} - \nabla f(\theta_{\mathrm{old}})$ 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

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)

Curve-Fitting by Optimizing

We have data $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
We also have possible curves, $r(x; \theta)$

Curve-Fitting by Optimizing

We have data $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
We also have possible curves, $r(x; \theta)$
e.g., $r(x) = x \cdot \theta$

Curve-Fitting by Optimizing

We have data $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
We also have possible curves, $r(x; \theta)$
e.g., $r(x) = x \cdot \theta$
e.g., $r(x) = \theta_1 x^{\theta_2}$

Curve-Fitting by Optimizing

We have data $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
We also have possible curves, $r(x;\theta)$
e.g., $r(x) = x \cdot \theta$
e.g., $r(x) = \theta_1 x^{\theta_2}$
e.g., $r(x) = \sum_{j=1}^{q} \theta_j b_j(x)$ for fixed "basis" functions b_j

Curve-Fitting by Optimizing

We have data $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
We also have possible curves, $r(x;\theta)$
e.g., $r(x) = x \cdot \theta$
e.g., $r(x) = \theta_1 x^{\theta_2}$
e.g., $r(x) = \sum_{j=1}^{q} \theta_j b_j(x)$ for fixed "basis" functions b_j
Least-squares curve fitting:

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \frac{1}{n} \sum_{i=1}^{n} (y_i - r(x_i;\theta))^2$$

Curve-Fitting by Optimizing

We have data $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
We also have possible curves, $r(x; \theta)$
e.g., $r(x) = x \cdot \theta$
e.g., $r(x) = \theta_1 x^{\theta_2}$
e.g., $r(x) = \sum_{j=1}^{q} \theta_j b_j(x)$ for fixed “basis” functions b_j
Least-squares curve fitting:

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \frac{1}{n} \sum_{i=1}^{n} (y_i - r(x_i; \theta))^2$$

“Robust” curve fitting:

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \frac{1}{n} \sum_{i=1}^{n} \psi(y_i - r(x_i; \theta))$$

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 a gradient-free method ("Nedler-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,grad.mse,method="BFGS",hessian=TRUE)
```

`fit1`: Newton-ish BFGS method

Run-time: 0.027 seconds

```
> fit1
$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
```

nls

`optim` is a general-purpose optimizer
`nls` is for nonlinear least squares

```
nls(formula, data, start, control, [[many other options]])
```

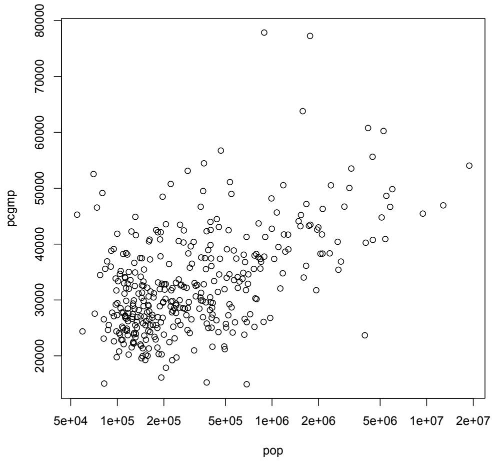
`formula`	Mathematical expression with response variable, predictor variable(s), and unknown parameter(s)
`data`	Data frame with variable names matching `formula`
`start`	Guess at parameters (optional)
`control`	Like with `optim` (optional)

Returns an `nls` object, with fitted values, prediction methods, etc.
The default optimization is a version of Newton's method

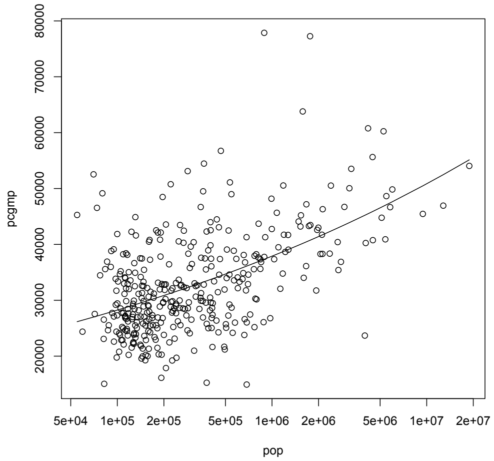
fit2: Fitting the Same Model with nls

```
> fit2 <- nls(pcgmp~y0*pop^a,data=gmp,start=list(y0=5000,a=0.1))
> summary(fit2)

Formula: pcgmp ~ y0 * pop^a

Parameters:
    Estimate Std. Error t value Pr(>|t|)
y0 6.494e+03  8.565e+02   7.582 2.87e-13 ***
a  1.277e-01  1.012e-02  12.612  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7886 on 364 degrees of freedom

Number of iterations to convergence: 5
Achieved convergence tolerance: 1.781e-07
```

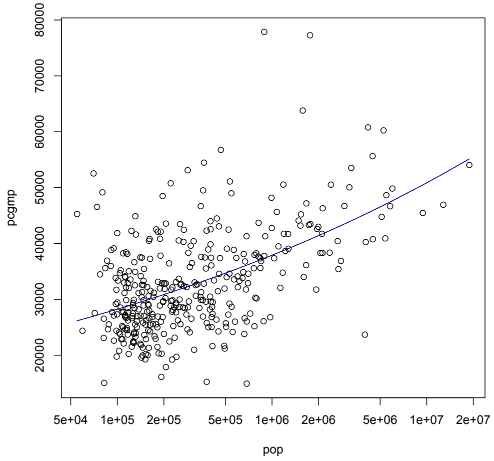
We will see later where all the inferential statistics come from



```
plot(pcgmp~pop,data=gmp)
```



```
plot(pcgmp~pop,data=gmp)
pop.order <- order(gmp$pop)
lines(gmp$pop[pop.order],fitted(fit2)[pop.order])
```



```
plot(pcgmp~pop,data=gmp)
pop.order <- order(gmp$pop)
lines(gmp$pop[pop.order],fitted(fit2)[pop.order])
curve(fit1$par[1]*x^fit1$par[2],add=TRUE,lty="dashed",col="blue")
```

Summary

1. Trade-offs: complexity of iteration vs. number of iterations vs. precision of approximation
   - Gradient descent: more complex iterations, more guarantees, more adaptive
   - Newton: even more complex iterations, but few of them for good functions
2. Start with pre-built code like `optim`, implement your own as needed