Date Science with R and Python

Numerical Optimization

Peng Zhang

School of Mathematical Sciences, Zhejiang University

2023/07/09

Agenda

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

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** $f: \mathrm{D} \mapsto R$, find

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

Basics: maximizing f is minimizing $-f$:

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

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 $f(\theta^*)$?
- Time complexity: How many computer steps does that take?
  Varies with precision of approximation, niceness of f, size of D, size of data, method…
- Most optimization algorithms use **successive approximation**, so distinguish number of iterations from cost of each iteration

You remember calculus, right?

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,

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

You remember calculus, right?

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**, $\mathbf{H}$

θ might just be a **local** minimum

Gradients and Changes to f

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

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

Locally, the function looks linear; to minimize a linear function, move down the slope

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 θ, step-size η
2. While ((not too tired) and (making adequate progress))
   - Find gradient $\nabla f(\theta)$
   - 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

Cons:

- "small enough" 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 $c \neq 0$

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

e.g., $e^x/(1 + e^x) = O(1)$

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?

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

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:

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0) \cdot \nabla f(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T \mathbf{H}(\theta_0)(\theta - \theta_0)$$

Minimizing a Quadratic

If we know

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

we minimize exactly:

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

If

$$f(x) = \frac{1}{2}a(x - x_0)^2 + b(x - x_0) + c$$

then

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

Newton's Method

Taylor-expand for the value *at the minimum* θ^*

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

Take gradient, set to zero, solve for θ^*:

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

Works *exactly* if f is quadratic
and $\mathbf{H}^{-1}$ 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))
    - Find gradient $\nabla f(\theta)$ and Hessian $\mathbf{H}(\theta)$
    - 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

Pros:

- 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 for θ_{new}

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)

Other Methods

- Lots!
- See bonus slides at end for for "Nedler-Mead", a.k.a. "the simplex method", which doesn't need any derivatives
- See bonus slides for the meta-method "coordinate descent"

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:

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

"Robust" curve fitting:

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

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`

Optimization in R: optim()

fit1: Newton-ish BFGS method

```
gmp <- read.table("data/gmp.dat")
gmp$pop <- gmp$gmp/gmp$pcgmp
library(numDeriv)
mse <- function(theta) { mean((gmp$pcgmp - theta[1]*gmp$pop^theta[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[1:3]
```

```
## $par
## [1] 6493.2563726    0.1276921
##
## $value
## [1] 61853983
##
## $counts
## function gradient
##       63       11
```

```r
fit1[4:6]
```

```
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##              [,1]         [,2]
## [1,] 5.25021e+01      4422070
## [2,] 4.42207e+06 375729088303
```

nls

`optim` is a general-purpose optimizer

So is `nlm` --- try them both if one doesn't work

`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

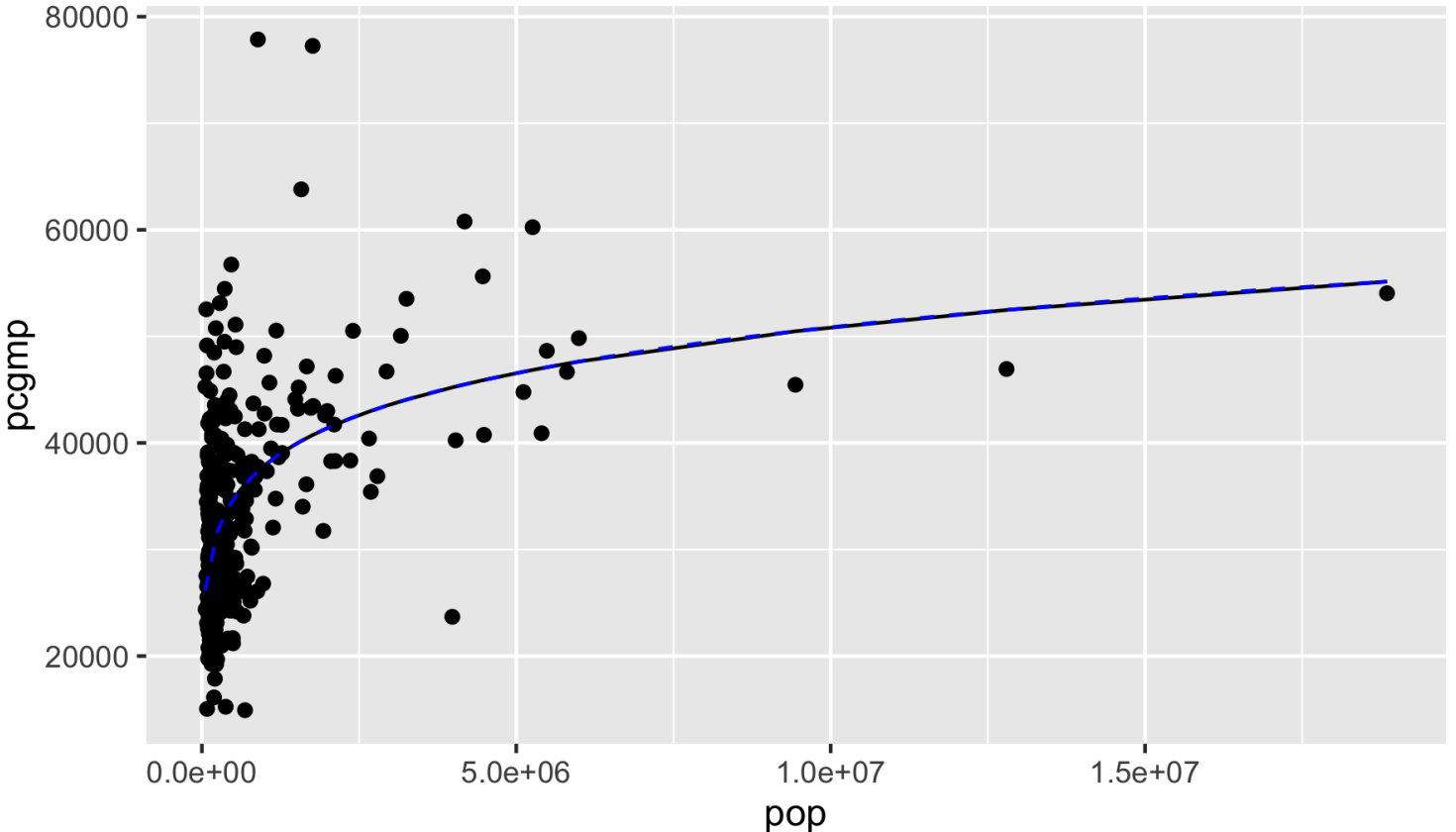
fit2: Fitting the Same Model with nls()

```
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: 1.738e-07
```

fit2: Fitting the Same Model with nls()

```
myfunc <- function(x) fit1$par[1]*x^fit1$par[2]
gmp %>% mutate(fitt = fitted(fit2)) %>% ggplot()+
  geom_point(aes(x = pop, y = pcgmp))+ geom_line(aes(x = pop, y = fit
  stat_function(fun = myfunc, col='blue', linetype = 2)
```



Summary

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

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

Center of the not-worst = $\theta_0 = \frac{1}{n} \sum_{i=1}^{n} \theta_i$

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

Try to improve the worst guess θ_{p+1}

1. **Reflection**: Try $\theta_0 - (\theta_{p+1} - \theta_0)$, across the center from θ_{p+1}
   - if it's better than θ_p but not than θ_1, replace the old θ_{p+1} with it
   - **Expansion**: if the reflected point is the new best, try $\theta_0 - 2(\theta_{p+1} - \theta_0)$; replace the old θ_{p+1} with the better of the reflected and the expanded point
2. **Contraction**: If the reflected point is worse that θ_p, try $\theta_0 + \frac{\theta_{p+1} - \theta_0}{2}$; if the contracted value is better, replace θ_{p+1} with it
3. **Reduction**: If all else fails, $\theta_i \leftarrow \frac{\theta_1 + \theta_i}{2}$
4. Go back to (1) until we stop improving or run out of time

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 making each point more like the best point

Pros:

- Each iteration ≤ 4 values of f, plus sorting
  (and sorting is 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

Coordinate Descent

Gradient descent, Newton's method, simplex, etc., adjust all coordinates of θ at once --- gets harder as the number of dimensions p grows

Coordinate descent: never do more than 1D optimization

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

Coordinate Descent

Cons:

- Needs a good 1D optimizer
- Can bog down for very tricky functions, especially with lots of interactions among variables

Pros:

- Can be extremely fast and simple

Constrained & Penalized Optimization

- Optimization under constraints
- Lagrange multipliers
- Penalized optimization
- Statistical uses of penalied optimization

Maximizing a multinomial likelihood

I roll dice n times; $n_1, \ldots n_6$ count the outcomes

Likelihood and log-likelihood:

$$L(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) = \frac{n!}{n_1! n_2! n_3! n_4! n_5! n_6!} \prod_{i=1}^{6} \theta_i^{n_i}$$

$$\ell(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) = \log \frac{n!}{n_1! n_2! n_3! n_4! n_5! n_6!} + \sum_{i=1}^{6} n_i \log \theta_i$$

Optimize by taking the derivative and setting to zero:

$$\frac{\partial \ell}{\partial \theta_1} = \frac{n_1}{\theta_1} = 0$$
$$\therefore \theta_1 = \infty$$

Maximizing a multinomial likelihood

We forgot that $\sum_{i=1}^{6} \theta_i = 1$

We could use the constraint to eliminate one of the variables

$$\theta_6 = 1 - \sum_{i=1}^{5} \theta_i$$

Then solve the equations

$$\frac{\partial \ell}{\partial \theta_i} = \frac{n_1}{\theta_i} - \frac{n_6}{1 - \sum_{j=1}^{5} \theta_j} = 0$$

BUT eliminating a variable with the constraint is usually messy

Lagrange Multipliers

$$g(\theta) = c \;\Leftrightarrow\; g(\theta) - c = 0$$

Lagrangian:

$$\mathcal{L}(\theta, \lambda) = f(\theta) - \lambda(g(\theta) - c)$$

$= f$ when the constraint is satisfied

Now do *unconstrained* minimization over θ and λ:

$$\nabla_\theta \mathcal{L}|_{\theta^*, \lambda^*} = \nabla f(\theta^*) - \lambda^* \nabla g(\theta^*) = 0$$
$$\left.\frac{\partial \mathcal{L}}{\partial \lambda}\right|_{\theta^*, \lambda^*} = g(\theta^*) - c = 0$$

optimizing **Lagrange multiplier** λ enforces constraint

More constraints, more multipliers

Lagrange Multipliers

Try the dice again:

$$\mathcal{L} = \log \frac{n!}{\prod_i n_i!} + \sum_{i=1}^{6} n_i \log(\theta_i) - \lambda \left(\sum_{i=1}^{6} \theta_i - 1 \right)$$

$$\left. \frac{\partial \mathcal{L}}{\partial \theta_i} \right|_{\theta_i = \theta_i^*} = \frac{n_i}{\theta_i^*} - \lambda^* = 0$$

$$\frac{n_i}{\lambda^*} = \theta_i^*$$

$$\sum_{i=1}^{6} \frac{n_i}{\lambda^*} = \sum_{i=1}^{6} \theta_i^* = 1$$

$$\lambda^* = \sum_{i=1}^{6} n_i \;\Rightarrow \theta_i^* = \frac{n_i}{\sum_{i=1}^{6} n_i}$$

Thinking About the Lagrange Multipliers

Constrained minimum value is generally higher than the unconstrained

Changing the constraint level c changes θ^*, $f(\theta^*)$

$$\begin{aligned}\frac{\partial f(\theta^*)}{\partial c} &= \frac{\partial \mathcal{L}(\theta^*, \lambda^*)}{\partial c} \\ &= [\nabla f(\theta^*) - \lambda^* \nabla g(\theta^*)] \frac{\partial \theta^*}{\partial c} - [g(\theta^*) - c] \frac{\partial \lambda^*}{\partial c} + \lambda^* = \lambda^*\end{aligned}$$

$\lambda^* =$ Rate of change in optimal value as the constraint is relaxed

$\lambda^* =$ "Shadow price": How much would you pay for minute change in the level of the constraint

Inequality Constraints

What about an *inequality* constraint?

$$h(\theta) \leq d \;\Leftrightarrow\; h(\theta) - d \leq 0$$

The region where the constraint is satisfied is the **feasible set**

Roughly two cases:

1. Unconstrained optimum is inside the feasible set $\Rightarrow$ constraint is **inactive**
2. Optimum is outside feasible set; constraint **is active**, **binds** or **bites**; *constrained* optimum is usually on the boundary

Add a Lagrange multiplier; $\lambda \neq 0 \Leftrightarrow$ constraint binds

Mathematical Programming

Older than computer programming...

Optimize $f(\theta)$ subject to $g(\theta) = c$ and $h(\theta) \leq d$

> Give us the best deal on f, keeping in mind that we've only got d to spend, and the books have to balance

Linear programming (Kantorovich, 1938)

1. f, h both linear in θ
2. θ^* always at a corner of the feasible set

Back to the Factory

Revenue: 13k per car, 27k per truck

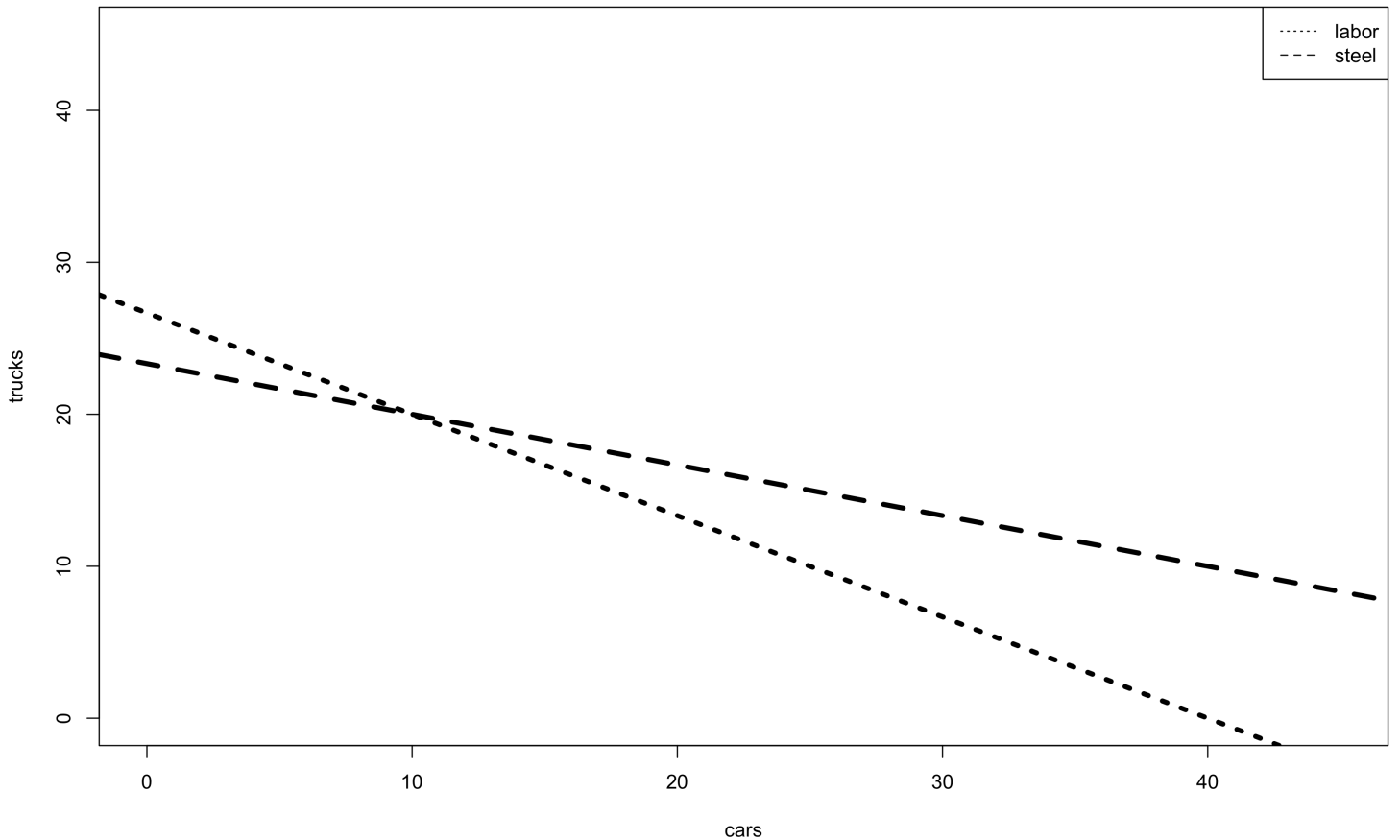
Constraints:

$$40 * \text{cars} + 60 * \text{trucks} < 1600\text{hours}$$
$$1 * \text{cars} + 3 * \text{trucks} < 70\text{tons}$$

Find the revenue-maximizing number of cars and trucks to produce

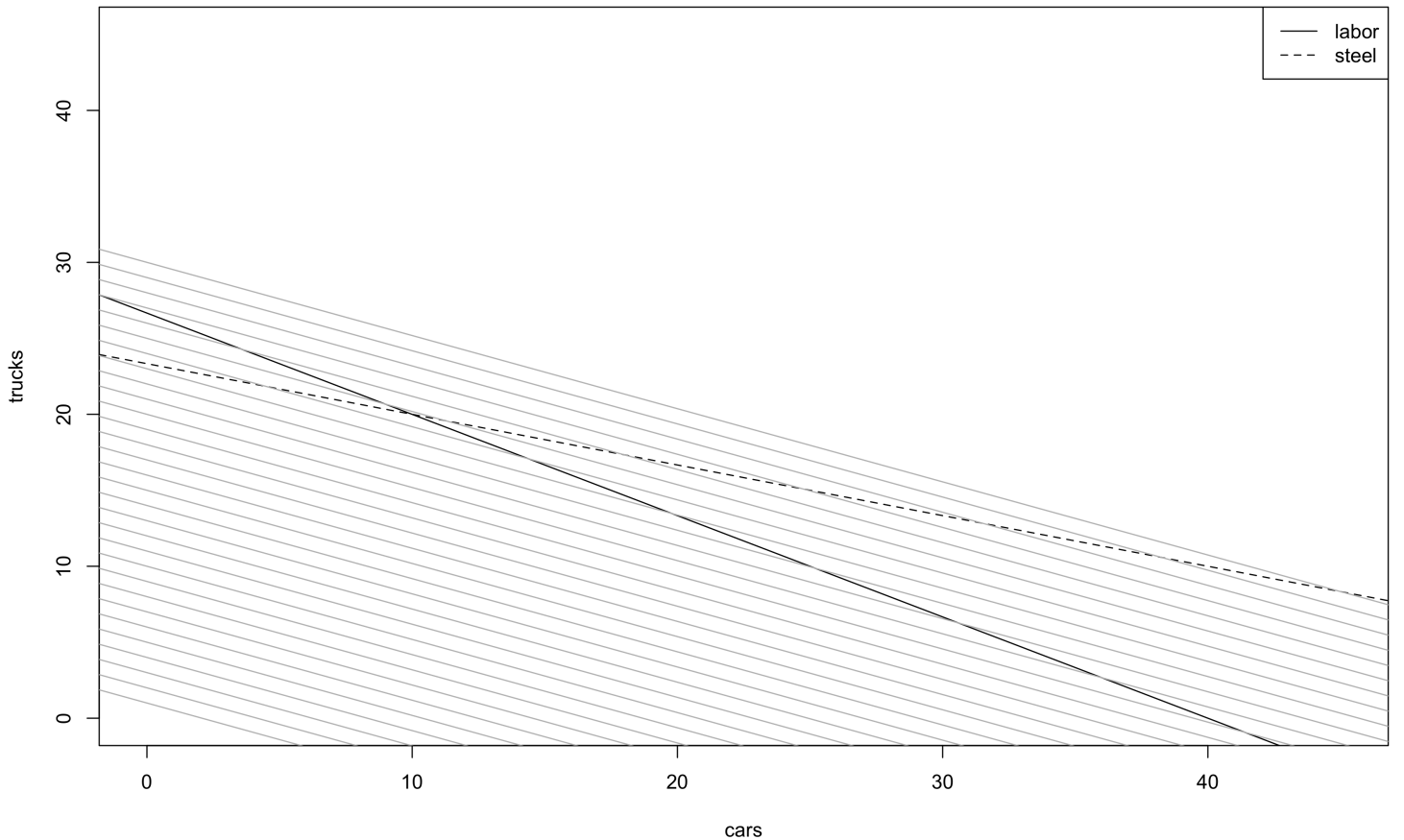
Back to the Factory

The feasible region:



Back to the Factory

The feasible region, plus lines of equal profit



The Equivalent Capitalist Problem

... *is* that problem

> the Walrasian model [of economics] is essentially about allocations and only tangentially about markets --- as one of us (Bowles) learned when he noticed that the graduate microeconomics course that he taught at Harvard was easily repackaged as 'The Theory of Economic Planning' at the University of Havana in 1969. (S. Bowles and H. Gintis, "Walrasian Economics in Retrospect", *Quarterly Journal of Economics*, 2000)

The Slightly More Complex Financial Problem

Given: expected returns $r_1, \ldots r_p$ among p financial assets, their $p \times p$ matrix of variances and covariances Σ

Find: the portfolio shares $\theta_1, \ldots \theta_n$ which maximizes expected returns

Such that: total variance is below some limit, covariances with specific other stocks or portfolios are below some limit
e.g., pension fund should not be too correlated with parent company

Expected returns $f(\theta) = r \cdot \theta$

Constraints: $\sum_{i=1}^{p} \theta_i = 1$, $\theta_i \geq 0$ (unless you can short)
Covariance constraints are linear in θ
Variance constraint is quadratic, over-all variance is $\theta^T \Sigma \theta$

Barrier Methods

(a.k.a. "interior point", "central path", etc.)

Having constraints switch on and off abruptly is annoying
especially with gradient methods

Fix $\mu > 0$ and try minimizing

$$f(\theta) - \mu \log (d - h(\theta))$$

"pushes away" from the barrier --- more and more weakly as $\mu \to 0$

Barrier Methods

1. Initial θ in feasible set, initial μ
2. While ((not too tired) and (making adequate progress))
   a. Minimize $f(\theta) - \mu \log(d - h(\theta))$
   b. Reduce μ
3. Return final θ

R implementation

`constrOptim` implements the barrier method

Try this:

```
factory <- matrix(c(40,1,60,3),nrow=2,
  dimnames=list(c("labor","steel"),c("car","truck")))
available <- c(1600,70); names(available) <- rownames(factory)
prices <- c(car=13,truck=27)
revenue <- function(output) { return(-output %*% prices) }
plan <- constrOptim(theta=c(5,5),f=revenue,grad=NULL,
  ui=-factory,ci=-available,method="Nelder-Mead")
plan$par
```

```
## [1]  9.999896 20.000035
```

`constrOptim` only works with constraints like $\mathbf{u}\theta \geq c$, so minus signs

Constraints vs. Penalties

$$\operatorname*{argmin}_{\theta: h(\theta) \leq d} f(\theta) \Leftrightarrow \operatorname*{argmin}_{\theta, \lambda} f(\theta) - \lambda(h(\theta) - d)$$

d doesn't matter for doing the second minimization over θ

We could just as well minimize

$$f(\theta) - \lambda h(\theta)$$

Constrained optimization	**Penalized optimization**
Constraint level d	Penalty factor λ

"A fine is a price"

Statistical Applications of Penalization

Mostly you've seen unpenalized estimates (least squares, maximum likelihood)

Lots of modern advanced methods rely on penalties

- For when the direct estimate is too unstable
- For handling high-dimensional cases
- For handling non-parametrics

Ordinary Least Squares

No penalization; minimize MSE of linear function $\beta \cdot x$:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta \cdot x_i)^2 = \underset{\beta}{\operatorname{argmin}} \, MSE(\beta)$$

Closed-form solution if we can invert matrices:

$$\hat{\beta} = (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y}$$

where $\mathbf{x}$ is the $n \times p$ matrix of x vectors, and $\mathbf{y}$ is the $n \times 1$ matrix of y values.

Ridge Regression

Now put a penalty on the *magnitude* of the coefficient vector:

$$\tilde{\beta} = \underset{\beta}{\operatorname{argmin}}\, MSE(\beta) + \mu \sum_{j=1}^{p} \beta_j^2 = \underset{\beta}{\operatorname{argmin}}\, MSE(\beta) + \mu \|\beta\|_2^2$$

Penalizing β this way makes the estimate more *stable*; especially useful for

- Lots of noise
- Collinear data ($\mathbf{x}$ not of "full rank")
- High-dimensional, $p > n$ data (which implies collinearity)

This is called **ridge regression**, or **Tikhonov regularization**

Closed form:

$$\tilde{\beta} = (\mathbf{x}^T\mathbf{x} + \mu I)^{-1}\mathbf{x}^T\mathbf{y}$$

The Lasso

Put a penalty on the sum of coefficient's absolute values:

$$\beta^\dagger = \underset{\beta}{\text{argmin}}\, MSE(\beta) + \lambda \sum_{j=1}^{p} |\beta_j| = \underset{\beta}{\text{argmin}}\, MSE(\beta) + \lambda \|\beta\|_1$$

This is called **the lasso**

- Also stabilizes (like ridge)
- Also handles high-dimensional data (like ridge)
- Enforces **sparsity**: it likes to drive small coefficients exactly to 0

No closed form, but very efficient interior-point algorithms (e.g., `lars` package)

Spline Smoothing

"Spline smoothing": minimize MSE of a smooth, nonlinear function, plus a penalty on curvature:

$$\hat{f} = \underset{f}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \int (f''(x))^2 dx$$

This fits smooth regressions without assuming any specific functional form

- Lets you check linear models
- Makes you wonder why you bother with linear models

Many different R implementations, starting with `smooth.spline`

How Big a Penalty?

Rarely know the constraint level or the penalty factor λ from on high

Lots of ways of picking, but often **cross-validation** works well:

- Divide the data into parts
- For each value of λ, estimate the model on one part of the data
- See how well the models fit the other part of the data
- Use the λ which extrapolates best on average

Summary

- We use Lagrange multipliers to turn constrained optimization problems into unconstrained but penalized ones
  - Optimal multiplier values are the prices we'd pay to weaken the constraints
- The nature of the penalty term reflects the sort of constraint we put on the problem
  - Shrinkage
  - Sparsity
  - Smoothness

Hypothesis Testing

Test the hypothesis that the data are distributed $\sim P$ against the hypothesis that they are distributed $\sim Q$

$P =$ noise, $Q =$ signal

Want to maximize **power**, probability the test picks up the signal when it's present

Need to limit false alarm rate, probability the test claims "signal" in noise

Hypothesis Testing

Say "signal" whenever the data falls into some set S

Power $= Q(S)$

False alarm rate $= P(S) \leq \alpha$

$$\max_{S:P(S)\leq\alpha} Q(S)$$

With Lagrange multiplier,

$$\max_{S,\lambda} Q(S) - \lambda(P(S) - \alpha)$$

Looks like we have to do ugly calculus over set functions...

Hypothesis Testing

Pick any point x: should we add it to S?

Marginal benefit = $dQ/dx = q(x)$

Marginal cost = $\lambda dP/dx = \lambda p(x)$

Keep expanding S until marginal benefit = marginal cost so $q(x)/p(x) = \lambda$

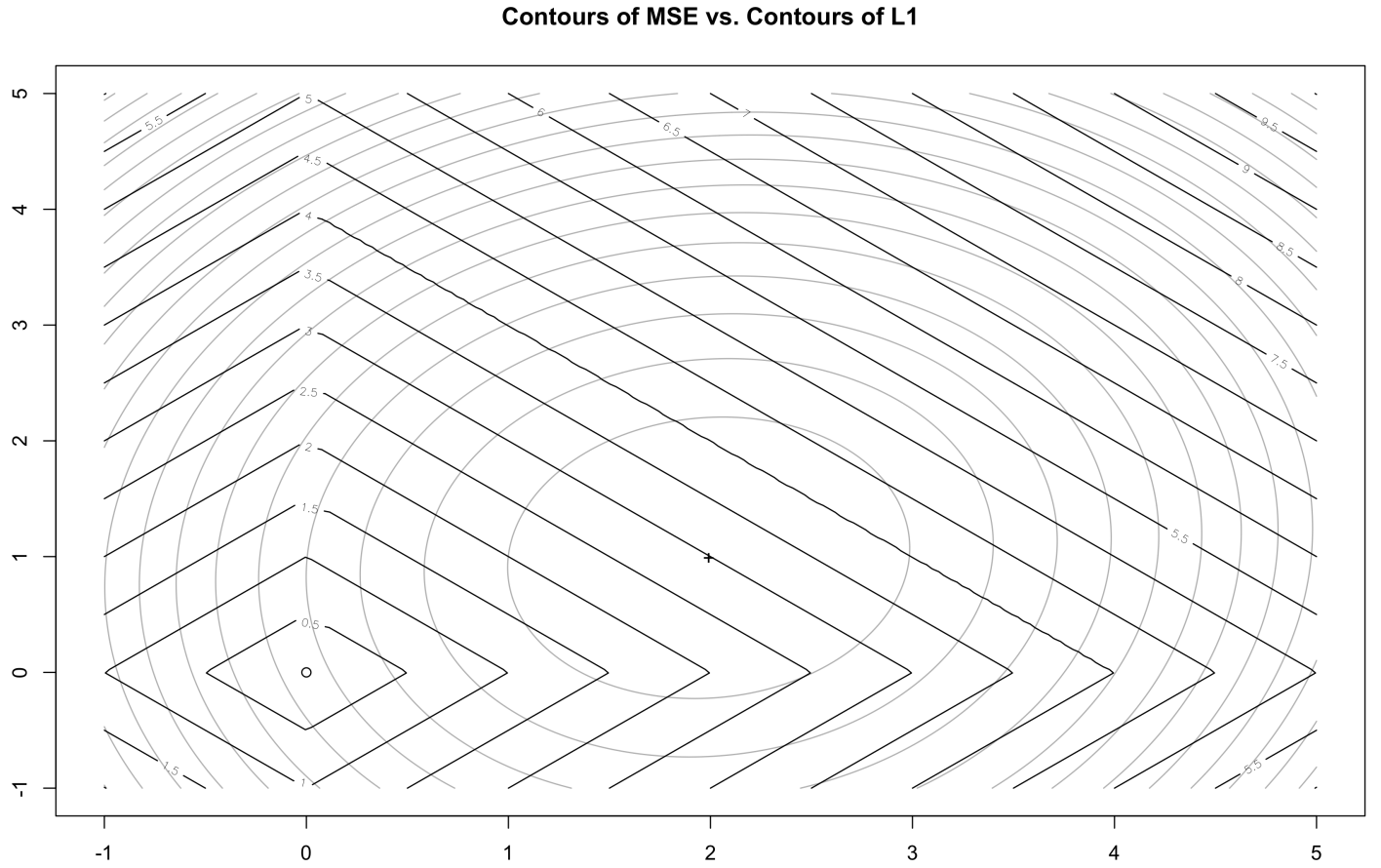
$q(x)/p(x) =$ **likelihood ratio**; optimal test is **Neyman-Pearson test**

$\lambda =$ critical likelihood ratio $=$ shadow price of power

Lasso Example

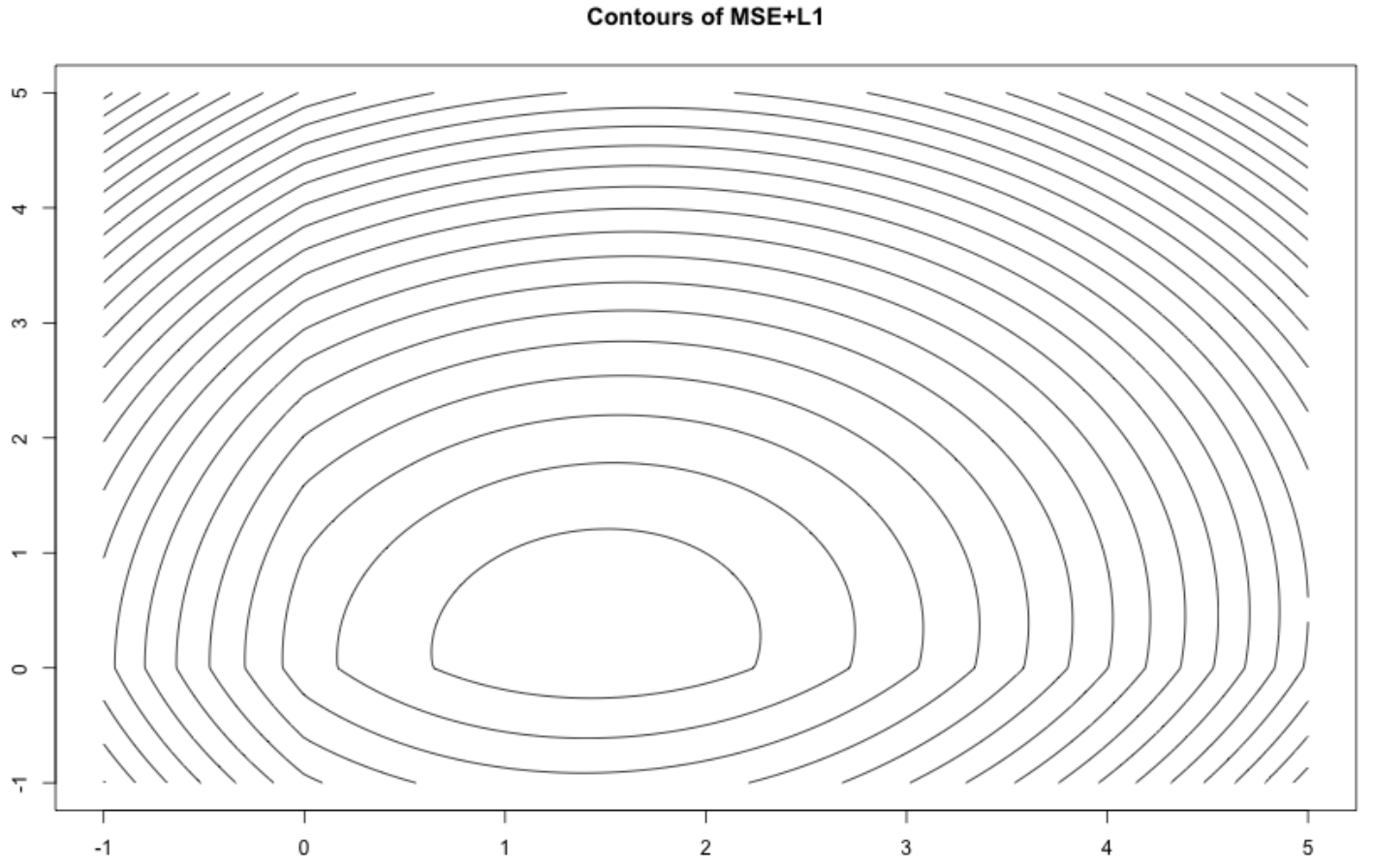
```
x <- matrix(rnorm(200),nrow=100)
y <- (x %*% c(2,1))+ rnorm(100,sd=0.05)
mse <- function(b1,b2) {mean((y- x %*% c(b1,b2))^2)}
coef.seq <- seq(from=-1,to=5,length.out=200)
m <- outer(coef.seq,coef.seq,Vectorize(mse))
l1 <- function(b1,b2) {abs(b1)+abs(b2)}
l1.levels <- outer(coef.seq,coef.seq,l1)
ols.coefs <- coefficients(lm(y~0+x))
```

```
contour(x=coef.seq,y=coef.seq,z=m,drawlabels=FALSE,nlevels=30,col="gr
  main="Contours of MSE vs. Contours of L1")
contour(x=coef.seq,y=coef.seq,z=l1.levels,nlevels=20,add=TRUE)
points(x=ols.coefs[1],y=ols.coefs[2],pch="+")
points(0,0)
```



Lasso Example

```
contour(x=coef.seq,y=coef.seq,z=m+l1.levels,drawlabels=FALSE,nlevels=
  main="Contours of MSE+L1")
```



Augmented Lagrangian Methods

A simple trick for constrained optimization: minimize

$$f(\theta) + r(g(\theta) - c)^2$$

over and over, letting $r \to \infty$

Drawback: really unstable when r is huge

Augmented Lagrangian trick: fix r and a guess at λ, then minimize

$$f(\theta) + \lambda(g(\theta) - c) + r(g(\theta) - c)^2$$

Now crank up r and update λ by an amount that reflects how badly the constraint was violated

Often converges at finite r

Augmented Lagrangian Methods

Same ideas work for inequality constraints

Unlike interior-point methods, initial guess needn't be in feasible set

R impelementation: `alabama` package

Stochastic Optimization

- Why there's such a thing as too much data
- How to make noise our friend
- How to make slop our friend

Optional reading: Bottou and Bosquet, "The Tradeoffs of Large Scale Learning"

Problems with Big Data

- Typical statistical objective function, mean-squared error:

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - m(x_i, \theta))^2$$

- Getting a value of f is $O(n)$, ∇f is $O(np)$, $\mathbf{H}$ is $O(np^2)$
  - worse still if m slows down with n
- Not bad when $n = 100$ or even $n = 10^4$, but if $n = 10^9$ or $n = 10^{12}$ we don't even know which way to move

Sampling, an Alternative to Sarcastic Gradient Descent

- Pick *one* data point I at random (uniform on $1:n$)
- Loss there, $(y_I - m(x_I, \theta))^2$, is random, but

$$\mathbb{E}\left[(y_I - m(x_I, \theta))^2\right] = f(\theta)$$

- Generally, if $f(\theta) = n^{-1}\sum_{i=1}^{n} f_i(\theta)$ and f_i are well-behaved,

$$\begin{aligned}\mathbb{E}\left[f_I(\theta)\right] &= f(\theta)\\ \mathbb{E}\left[\nabla f_I(\theta)\right] &= \nabla f(\theta)\\ \mathbb{E}\left[\nabla^2 f_I(\theta)\right] &= \mathbf{H}(\theta)\end{aligned}$$

$\therefore$ Don't optimize with all the data, optimize with random samples

Stochastic Gradient Descent

Draw lots of one-point samples, let their noise cancel out:

1. Start with initial guess θ, learning rate η
2. While ((not too tired) and (making adequate progress))
   - At t^{th} iteration, pick random I uniformly
   - Set $\theta \leftarrow \theta - t^{-1}\eta\nabla f_I(\theta)$
3. Return final θ

Shrinking step-size by $1/t$ ensures noise in each gradient dies down

(Variants: put points in some random order, only check progress after going over each point once, adjust $1/t$ rate, average a couple of random data points ("mini-batch"), etc.)

Stochastic Gradient Descent

```
stoch.grad.descent <- function(f,theta,df,max.iter=1e6,rate=1e-6) {
  for (t in 1:max.iter) {
    g <- stoch.grad(f,theta,df)
    theta <- theta - (rate/t)*g
  }
  return(x)
}

stoch.grad <- function(f,theta,df) {
  stopifnot(require(numDeriv))
  i <- sample(1:nrow(df),size=1)
  noisy.f <- function(theta) { return(f(theta, data=df[i,])) }
  stoch.grad <- grad(noisy.f,theta)
  return(stoch.grad)
}
```

Stochastic Newton's Method

a.k.a. 2nd order stochastic gradient descent

1. Start with initial guess θ
2. While ((not too tired) and (making adequate progress))
   - At t^{th} iteration, pick uniformly-random I
   - $\theta \leftarrow \theta - t^{-1}\mathbf{H}_I^{-1}(\theta)\nabla f_I(\theta)$
3. Return final θ

+ all the Newton-ish tricks to avoid having to recompute the Hessian

Stochastic Gradient Methods

- Pros:
  - Each iteration is fast (and constant in n)
  - Never need to hold all data in memory
  - Does converge eventually
- Cons:
  - Noise *does* reduce precision --- more iterations to get within ϵ of optimum than non-stochastic GD or Newton

Often low computational cost to get within *statistical* error of the optimum

How Precise?

- We're minimizing f and aiming at $\hat{\theta}$
- f is a function of the data, which are full of useless details
  - e.g., $f(\theta) = n^{-1} \sum_{i=1}^{n} (y_i - m(x_i; \theta))^2$
- We hope there's some true f_0, with minimum θ_0
  - e.g., $f_0(\theta) = \mathbb{E}\left[(Y - m(X;\theta))^2\right] = \int (y - m(x;\theta))^2 p(x,y) dx dy$
- but we know $f \neq f_0$
  - because we've only got a finite sample, which includes noise
- Past some point, getting a better $\hat{\theta}$ isn't helping is find θ_0

(why push optimization to $\pm 10^{-6}$ if f only matches f_0 to ± 1?)

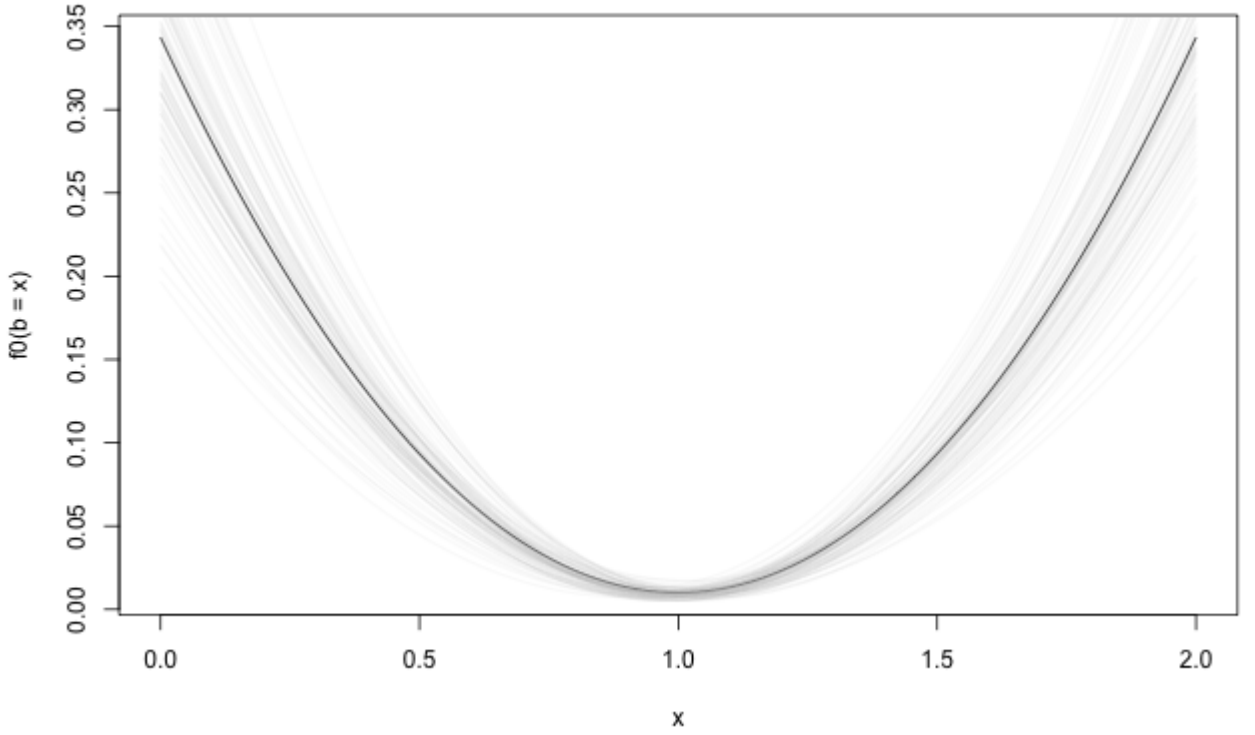
How Precise?

- An example
  - true model: $Y = X+$ Gaussian noise with mean 0, s.d. 0.1, X uniform between 0 and 1
  - Try regression through the origin ($Y=bX$, b unknown)
- We'll plot the population objective function, and a bunch of sample objective functions from different $n = 30$ draws

```
f0 <- function(b) { 0.1^2+ (1/3)*(b-1)^2 }
f <- Vectorize(FUN=function(b,df) { mean((df$y - b*df$x)^2) }, vecto
simulate_df <- function(n) {
  x <- runif(n)
  y<-x+rnorm(n,0,0.1)
  return(data.frame(x=x,y=y))
}
```

How Precise?

```
curve(f0(b=x), from=0,to=2,)
replicate(100,curve(f(b=x,df=simulate_df(30)),
  add=TRUE,col="grey",lwd=0.1))
```



How Precise?

- Pretty generally:
  - $\hat{\theta} - \theta_0 = O(1/\sqrt{n})$ at best
  - $f(\hat{\theta}) - f(\theta_0) = O(1/n)$ at best
  - see bonus slides for gory details
- Not much point in pushing an optimization past those limits
  - More work gets a better approximation to $\hat{\theta}$...
  - but the numerical precision is swamped by the statistical noise...
  - so the answer isn't really any more accurate or precise than if we'd stopped sooner

How Precise?

- We shouldn't care about differences in θ much smaller than $O(1/\sqrt{n})$...
  - the `tol` of the optimization
- but what's the constant multiplying $1/\sqrt{n}$?
- Could do some heroic calculus + linear algebra
  - "sandwich covariance matrix" (see bonus slides; the answer involves the Hessian)
- Can use tricks like the jackknife to get an idea of what the statistical uncertainty is
  - though the bootstrap is usually better than the jackknife (take 36-402)

Summary

- Stochastic gradient descent etc. deliberately use samples of the data, rather than all of it at once
  - Gives up precision for speed and memory
  - Make fitting models to enormous data sets computationally tractable
- Since the function we're optimizing is noisy anyway, don't bother pushing the numerical precision much below the statistical uncertainty
  - Can estimate uncertainty by statistical theory
  - or by re-sampling methods

Asymptotics of Optimization

Have f (sample objective) but want to minimize f_0 (population objective)

If f is an average over data points, then (law of large numbers)

$$\mathbb{E}\left[f(\theta)\right] = f_0(\theta)$$

and (central limit theorem)

$$f(\theta) - f_0(\theta) = O(n^{-1/2})$$

Asymptotics of Optimization

Do the opposite expansion to the one we used to derive Newton's method:

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

Asymptotics of Optimization

$$\hat{\theta}_n \approx \theta^* - \widehat{\mathbf{H}}_n^{-1}(\theta^*)\nabla f(\theta^*)$$

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

$$\begin{aligned} \widehat{\mathbf{H}}_n(\theta^*) &\to \mathbf{H}(\theta^*) \text{ (by LLN)} \\ \nabla f(\theta^*) - \nabla f(\theta^*) &= O(n^{-1/2}) \text{ (by CLT)} \end{aligned}$$

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

$$\begin{aligned} \therefore \nabla f(\theta^*) &= O(n^{-1/2}) \\ \operatorname{Var}\left[\nabla f(\theta^*)\right] &\to n^{-1}\mathbf{K}(\theta^*) \text{ (CLT again)} \end{aligned}$$

Asymptotics of Optimization

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

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

so $\hat{\theta}_n - \theta^* = O(1/\sqrt{n})$

Asymptotics of Optimization

How much noise is there in $f_0(\hat{\theta}_n)$?

$$
\begin{aligned}
f_0(\hat{\theta}_n) - f_0(\theta^*) &\approx \frac{1}{2}(\hat{\theta}_n - \theta^*)^T \mathbf{H}(\theta^*)(\hat{\theta}_n - \theta^*) \\
\mathbb{E}\left[f(\hat{\theta}_n) - f(\theta^*)\right] &\approx \frac{1}{2}\text{tr}\left(\text{Var}\left[(\hat{\theta}_n - \theta^*)\right]\mathbf{H}(\theta^*)\right) \\
&+ \frac{1}{2}\mathbb{E}\left[\hat{\theta}_n - \theta^*\right]^T \mathbf{H}(\theta^*)\mathbb{E}\left[\hat{\theta}_n - \theta^*\right] \\
&= \frac{1}{2}\text{tr}\left(n^{-1}\mathbf{H}^{-1}(\theta^*)\mathbf{K}(\theta^*)\mathbf{H}^{-1}(\theta^*)\mathbf{H}(\theta^*)\right) \\
&= \frac{1}{2}n^{-1}\text{tr}\left(\mathbf{H}^{-1}(\theta^*)\mathbf{K}(\theta^*)\right) \\
\text{Var}\left[f_0(\hat{\theta}_n) - f_0(\theta^*)\right] &\approx \text{tr}\left(\left(\mathbf{H}(\theta^*)\text{Var}\left[\hat{\theta}_n - \theta^*\right]\mathbf{H}(\theta^*)\text{Var}\left[\hat{\theta}_n - \theta^*\right]\right)\right) \\
&\to n^{-2}\text{tr}\left(\left(\mathbf{K}(\theta^*)\mathbf{H}^{-1}(\theta^*)\mathbf{K}(\theta^*)\mathbf{H}^{-1}(\theta^*)\right)\right) \\
&= O(pn^{-2})
\end{aligned}
$$

Asymptotics of Optimization

The ideal case is well-specified maximum likelihood: then $\mathbf{K} = \mathbf{H}$, and

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