Multivariate Optimization

Multivariate Optimization

- We have seen Golden Section and Newton Raphson searches for one dimensional optima.

- It is rarely the case that we only want to optimize one thing.
-

Exa
mixt

$$\sum_{i=1}^{n} \log \frac{1}{2}\left(\phi(X_i - \mu_1) + \phi(X_i - \mu_2)\right)$$

Where $\phi(x)$ is the normal density.

On Maximum Likelihood Estimation

General principle of obtaining parameter estimates

Choose the parameters that make the data as probable as possible.

In the c
data, i
two m

$$P(X) = \frac{1}{2}(\phi(X - \mu_1) + \phi(X - \mu_2))$$

with centers around 2 and 4.5
(Probably smaller standard
deviations, too, but lets keep it
simple for now).

But we'd like to use the X_i to decide on μ_1 and μ_2.

Log Likelihood

Probability distribution for any X is

$$P(X) = \frac{1}{2}(\phi(X - \mu_1) + \phi(X - \mu_2))$$

so tha

Usua

of parameters μ_1 and μ_2 (doesn't change where the maximum is).

$$l(\mu_1, \mu_2) = \sum_{i=1}^{n} \log P(x_i) = \sum_{i=1}^{n} \log \frac{1}{2} \ldots))$$

This is the log likelihood that we want to maximize for μ_1 and μ_2.

Note: we could also fit standard deviations, proportions for each normal, but that won't plot so well.

Co-ordinate Ascent

Just optimize one parameter at a time.

1. Start with $(\mu_{0,1}, \mu_{0,2})$
2. .

3. Run an optimizer to choose μ_2 ke value

$$\mu_{1,2} = \underset{\mu_2}{\operatorname{argmax}}$$

4. Iterate until the updates do not move the solution much.

Some Coding Notes

- Use, say, `GoldenSection` from Lecture 8 to do the 1d optimization.

- But this assumes a function with only one input.
- We'll re-write this so the function can take a vector of inputs
-
- And it still needs upper and lower starting valu m].
- We've also got data X to add to the fun add an extra argument throughout.

- This is fairly specialized to the problem; similar things apply to Newton's method functions (but we also need vectors of derivatives).
- See code for this lecture for details.

In Code

Using a modified `GoldenSection`:


```
CoordinateAscent = function(mu,mul,mur,fn,X,tol=1e-8,maxit=1000)
{
  iter = 0              # Initialization
  tol.met = FALSE
  mu

  wh

    for(dim in 1:length(mu)){    # But we'll update t
      iter = iter +              # at each coordinate
        mu = GoldenSection(fn,mul,mur,m
        muhist = rbind(muhist,mu)
    }

    if(max(abs(mu - oldmu))<tol | iter>maxit){tol.met = TRUE}
    else{ oldmu = mu }
  }
```

Results

We need to redefine our function to take a vector (μ_1, μ_2):

```
fn = function(mu,X){ return(sum(log(0.5*(dnorm(X-mu[1])+dnorm(X-mu[2]))))) }
```


And define a box of values containing the maximum

```
mul = c(
mur = c(
mu  = c(3
```



```
opt = CoordinateAscent(mu,mul,mur,fn,X)
```

Some Notes

- Using GoldenSection requires a box for the optimal value as well as a starting point.
-
- of zig-zag lines).
- Convergence criterion: $\|\mu_{k+1} - \mu$ over the last iteration? Does not guarantee t found a good minimum.

Steepest Ascent

Perhaps we can work out a better direction to move in?

- We'd like to move uphill as quickly as possible.
- asy

- x_k) for some α.
- To choose α: 1-dimensional optimizat

$$f^*(\alpha) = f(x_k + \alpha \nabla f(x_k))$$

Our Example

We need derivatives of the log likelihood

$$\frac{dl(\mu_1, \mu_2)}{d\mu_1} = \sum_{i=1}^{n} \frac{(X_i - \mu_1)\phi(X_i - \mu_1)}{\phi(X_i - \mu_1) + \phi(X_i - \mu_2)}$$

n

Visu

Steepest Ascent Continued

Three-step method at iteration k with current estimate x_k

1. Calculate the gradient $g_k = \nabla f(x_k)$.
2. $+ \alpha g)$.
3.

Line s

- Requires either specialized code (as in ve, or defining a function within a function (see ex

- Should guarantee that $f(x_{k+1}) >$
- Known starting point, but no known limits.

GoldenSection Line Search

Consider a golden section search over $f(x_k + \alpha g)$.

- We we know that $f(x_k)$ is increasing in the direction of g, so we can require $\alpha > 0$. That is the left end point is $a_l = 0$.

  - Start with a_r small (say a some multi
  - Keep doubling a_r until $f^*(a_r)$ d be between the last three values.
  - To prevent going on forever, set a maximu

Things are actually not so complicated with a Newton-Raphson method, but you need more derivatives.

Line Search Function


```
GoldenSectionLineSearch = function(fns,X,tol,maxtry)
{
    # We need to work out where to put our limits for the Golden Section
    # search.

    ar = c(0,2*tol,4*tol)              # Start near tolerance
    f

    t                                             # Keep doubling
    w

      try = try+1
      fval = c(fval,fns(ar[try],X))
    }
    if(try == maxtry){ al = ar[try-1]; ar = ar[try] }  # Last two poin
    else{ al = ar[try-2]; ar = ar[try] }    # if not at a hump, otherwise
                                            # last three.

    # Now call GoldenSection for the line search

    return( GoldenSection(fns,al,ar,al,1,X,tol=tol,maxit=maxit) )
}
```

Putting It Together


```
SteepestAscent = function(mu,fn,dfn,X,tol=1e-8,maxit=1000,maxtry=25)
{
  tol.met=FALSE; iter = 0; iterhist = mu

  while(!tol.met){
    iter = iter+1; oldmu = mu; g = dfn(mu,X)

    # D
    # t
    f

    # Conduct the line search
    linesearch = GoldenSectionLineSearch(fns,

    # And update the estimate
    mu = mu + linesearch$xm*g
    iterhist = rbind(iterhist,mu)

    if( max(abs( mu-oldmu )) < tol | iter > maxit){ tol.met=TRUE }
  }
  return(list(mu=mu,iter=iter,iterhist=iterhist))
}
```

Some Notes

- More direct than coordinate ascent.

- $f(x_k)$ at each step.

- Still some zig-zagging across ridges.

Multivariate Newton-Raphson

More derivatives can help. Take a Taylor expansion in p dimensions

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

wher

$$H(x^*) = \begin{bmatrix} \frac{d^2 f(x^*)}{dx_1 dx_1} & \cdots & \frac{d^2 f(x^*)}{dx_1 dx_p} \\ \vdots & \ddots & \vdots \\ \frac{d^2 f(x^*)}{dx_p dx_1} & \cdots & \frac{d^2 f(x^*)}{dx_p dx_p} \end{bmatrix}$$

And try to maximize the quadratic approximation to f.

Multivariate Newton-Raphson

Want to maximize the approximation

$$\tilde{f}(x) = f(x^*) + (x - x^*)^T \nabla f(x^*) + \frac{1}{2}(x - x^*)^T H(x^*)(x - x^*)$$

Setti

or

$$x = x^* - H(x^*)^{-}$$

This gives the Newton-Raphson update (with iteration rather than dimension)

$$x_{k+1} = x_k - H(x_k)^{-1} \nabla f(x_k)$$

1-dimensional Newton Raphson obtained exactly the same way.

A First Multivariate Newton-Raphson Function


```
NewtonRaphson2 = function(mu,dfn,d2fn,X,tol=1e-8,maxit=100)
{
  tol.met=FALSE; iter = 0; iterhist = mu

  while(!tol.met){
    iter = iter + 1
    o

    g = d                    # Grad
    H = d                    # He
    m
    iterhist = rbind(iterhist,mu)

    if( (max(abs( mu-oldmu )) < tol & max(abs(g)) < tol) | iter > m
        { tol.met=TRUE }
  }
  return(list(mu=mu,iter=iter,iterhist=iterhist))
}
```


Convergence: requires both last step *and* the gradient to be sufficiently small.

Some Notes and Results

- Nominally very fast convergence.
-

- Can have problems with singular Hessians.

- Various fixes (eg checking that $f(x)$ increases each iteration)

Green = Newton Raphson,
Red = Steepest Descent.

Multi-Modal Objectives

When there are multiple local minima that in your surface, you end up in one of them.

Newton-Raphson can end up in a saddle point, too.

Usually, you start your optimization in multiple places (often chosen at random) and pick the best end-point.

Least Squares and Gauss-Newton Methods

Regression is one of the most commonly used statistical methods.

Instead of a linear regression, what if we had some non-linear function we wanted to fit to data?

Eg: th

$$Y_i = \frac{}{\theta_2 + x_i} + \epsilon$$

Estimate θ_1 and θ_2 by minimizing squared

$$-\frac{1}{2}\sum_{i=1}^{n}\left(Y_i - \frac{\theta_1 x_i}{\theta_2 + x_i}\right)^2$$

More generally, we model $Y_i = f(x_i, \theta) + \epsilon_i$.

Example: Puromycin Data

Data from an enzymatic reaction that saturates as the concentration of Puromycin increases.

Blue line gives model of increased response subject to saturation.

A Bit of Notation

Using the model $Y_i = g(x_i, \theta) + \epsilon_i$, with vector θ of interest.

In vector form

$$Y = f(X, \theta) + E$$

and in p

$$\frac{\quad}{d}$$

where the entries are

$$J(X, \theta)_{ij} = \frac{dg(}{d\theta_j}$$

rows = observations, columns = elements of θ.

Gauss-Newton Methods

We have the objective function

$$F(\theta) = \frac{1}{2}\sum_{i=1}^{n}(Y_i - g(x_i,\theta))^2$$

with g

$$\nabla F \quad \sum_{i=1} \quad \frac{\cdots}{d\theta} \quad (X,\theta))$$

and Hessian

$$
\begin{aligned}
H(\theta) &= -\sum_{i=1}^{n} \frac{dg(x_i,\theta)}{d\theta}\frac{dg(x_i,\theta)}{d\theta}^T - \frac{\cdots_i}{d\theta d\theta^T}(Y_i - g(x_i,\theta)) \\
&\approx -J(X,\theta)^T J(X,\theta)
\end{aligned}
$$

Because second term should be small.

Gauss-Newton Iteration

$$\theta_{k+1} = \theta_k + \left[J(\theta_k, X)^T J(\theta_k, X) \right]^{-1} J(\theta_k, X)^T (Y - g(\theta_k, X))$$


```
GaussNewton = function(theta,gn,dgn,Y,x,tol=1e-8,maxit=100)
{
  tol

  whi
    i
    o

    g = gn(theta,x)
    dg = dgn(theta,x)

    theta = theta + solve(t(dg)%*%dg,t(dg)%*%(Y-g
    iterhist = rbind(iterhist,theta)

    if(max(abs( theta-oldtheta )) < tol | iter > maxit)
            { tol.met=TRUE }
  }
  return(list(theta=theta,iter=iter,iterhist=iterhist))
}
```

Setting Up Puromycin

Need functions that return vector $f(X,\theta)$


```
mmgn = function(theta,x){ return( theta[1]*x/(theta[2]+x) )
```

and m

```
dmmg
  re
}
```


We can then call

```
opt = GaussNewton(c(205,0.08),mmgn,   ,1])
```


Note that $J(\theta_k, X)^T J(\theta_k, X)$ is always po

Gauss-Newton tends to have fewer convergence problems than standard Newton-Raphson (but not none).

Other Methods: `optim`

- Optimization is a large and ongoing field of research (probably larger than all of Statistics).

- None of these methods are generally used without modification.
- lized

- function: among the possible met
  - BFGS, L-BFGS-B, CG – like Newton about how Hessians are calculated.

  - Nelder-Meade only uses function v
  - SANN simulated annealing (random searching algorithm – next slide).

Most do *minimization* instead of *maximization* – just put a minus sign out front.

Mixture Model Example

Fit all parameters parameters; $\phi_{\mu,\sigma}(x) = N(\mu,\sigma)$ density

$$-\sum \log\left((1-\theta_3)\phi_{\theta_1,\theta_4}(X_i) + \theta_3\phi_{\theta_2,\theta_5}(X_i)\right)$$


```
fn = function(theta,X)
{
-sum
                                                     =theta[5])))
}
```



```
res = optim(c(2,4.3,0.5,0.5,0.5),fn,metho
```

Some Other Methods

Nelder-Meade Simplex

- Like Golden section in higher dimensions.

- In a plane: take three points forming a triangle.
- Possible reflect the worst point, and expand, otherwise shrink.
-

Sim

-
- Decide to move to that point probabilistical probability for moving downhill.

- Slowly make the size of moves smaller and insi moving uphill.
- Theoretical guarantees that you'll find the "best" maximum (but you'll never know if you have); very computationally expensive.

Some Other Problems

Constrained Optimization

- $x = \text{argmax}\, f(x)$ but $x \geq 0$; or insist on a more complex region.

- Eg, in mixture model (μ_1, μ_2) gives the same value as (μ_2, μ_1)
-

Discrete Optimization

- Sometimes x takes a discrete set of value labels etc.

- Example: search over θ_i either 1 or 0 de include X_i in a regression model.
- Usually multiple dimensions – far too many possible combinations to search over.
- Many heuristic algorithms developed.

Levenberg-Marquardt

Usually applied to Gauss-Newton iteration.

-
-
- On the other hand, steepest-ascent alway i objective function; but can converge slowl

- Perhaps we find a way to trade these off.

A Trade-Off

Recall the Newton update

$$x_{k+1} = x_k - H(x_k)^{-1}\nabla f(x_k)$$

we'll

- When $\lambda_k \approx 0$, we get Newton's method.
- For λ_k very large, our step is

$$x_{k+1} \approx x_k + \frac{1}{\lambda_k}\nabla f(x_k)$$

the steepest ascent direction.

Choosing λ

Some things to keep in mind

- We want to end up in a maximum
  - $H(x_k)$ should be negative definite.
  - i.e. we want all the eigenvalues of $H(x_k)$ to be negative.
  - he ive definite).
- We also want to make sure that f iteration.
- In practice, taking the Eigen-decomposition of $H(x_k)$ is computationally expensive and unnecessary.
- Instead, we will use rules to increase or decrease λ_k at each iteration.

Levenberg-Marquardt PseudoCode

- Choose tolerances ϵ_1, ϵ_2, ϵ_3, maximum iterations
- Start at x_0, calculate $f(x_0)$, $\nabla f(x_0)$, $H(x_0)$
- Set λ_0 to be larger than the maximum positive eigenvalue of

  is neg.

$k+1$ $-$ k 3

  - $\tilde{x}_{k+1} = x_k - (H(x_k) - \lambda_k I)^{-1}\nabla$
  - If $f(\tilde{x}_{k+1}) > f(x_k)$, set $x_{k+1} = \tilde{}$
  - Else, while $f(\tilde{x}_{k+1}) \leq f(x_k)$
    - $\lambda_k = 2\lambda_k$, $\tilde{x}_{k+1} = x_k - (H(x_k) - \lambda_k I) \quad \nabla f(x_k)$

  Set $x_{k+1} = \tilde{x}_{k+1}$, $\lambda_{k+1} = \lambda_k$.

Idea: keep increasing λ until we get an increase in $f(x_k)$; otherwise try to decrease to get back to Newton's method.

LM On Old Faithful Data

- When Levenberg-Marquardt

- Newton Raphson just gets stuck
- But L-M avoids Steepest-Ascent's wiggles.

The E-M Algorithm and Latent Variables

Re-interpretation of the Old Faithful eruption data:

- Draw a binary random number Z with $P(Z = 1) = \pi$.
- If $Z = 1$, draw $X \quad N(\mu_1, \sigma_1^2)$
-

We ha the com i

But we don't see the Z_k; they're latent (si

Strategy below: if we observed the Z_k, t divide observations into groups and estimate parameters). We want to do something like that, but account for the fact that we don't know the Z_k.

Likelihood With Latent Variables

The probability of observing X_i (without knowing Z_i) is

$$P(X_i) = P(X_i|Z_i = 1)P(Z_i = 1) + P(X_i|Z_i = 0)P(Z_i = 0)$$
$$= \pi\phi_{\mu_1,\sigma_1}(X_i) + (1-\pi)\phi_{\mu_2,\sigma_2}(X_i)$$

Mor

$P(X$

– ie, take the expectation $P(X|Z)$ wit

In this case there are 5 parameters $\theta =$

likelihood

$$l(\pi, \mu_1, \mu_2, \sigma_1, \sigma_2) = \sum_{i=1}^{n} \log\left(\pi\phi_{\mu_1,\sigma_1}(X_i) + (1-\pi)\phi_{\mu_2,\sigma_2}(X_i)\right)$$

Must be maximized numerically.

EM In Mixture Models

First, let's consider the distribution of Z_i *given* X_i:

$$P(Z_i = 1 | X_i, \theta) = \frac{P(X_i | Z_i = 1) P(Z_i = 1)}{P(X_i | Z_i = 1) P(Z_i = 1) + P(X_i | Z_i = 0) P(Z_i = 0)}$$

$$= \frac{\pi \phi_{\mu_1, \sigma_1^2}(X_i)}{\text{[illegible]}}$$

This is f [illegible]

Now use these to update parameters using weighted estimates:

$$\pi_{k+1} = \frac{1}{n} \sum_{i=}^{n} \text{[illegible]}$$

and

$$\mu_{1,k+1} = \frac{\sum_{i=1}^n p_i X_i}{\sum_{i=1}^n p_i}, \quad \sigma^2_{1,k+1} = \frac{\sum_{i=1}^n p_i (X_i - \mu_{1,k+1})^2}{\sum_{i=1}^n p_i}$$

Finally, updates for μ_2, σ_2 are analogous, but with weights $(1 - p_i)$.

EM For Mixture Models

- Start with θ_0, tolerance $\epsilon > 0$, maxit
- While max $|\theta_{k+1} - \theta_k| > \epsilon$ and $k <$ maxit, iterate:
  - Update probabilities

$$\frac{\pi_k \phi_{\mu_1, \sigma^2}(X_i)}{\quad}$$

$$\pi_{k+1} = \frac{1}{n}\sum_{i=1}^{n} p_i, \quad \mu_{1,k+1} = \frac{\sum_{i=1}^{n} p_i X_i}{\sum_{i=1}^{n} p_i}, \quad \frac{\sum_{i}^{n} p_i (X_i - \mu_{1,k+1})^2}{\quad}$$

$$\mu_{2,k+1} = \frac{\sum_{i=1}^{n}(1 - p_i)X_i}{\sum_{i=1}^{n}(1 - p_i)}, \quad \sigma \quad \frac{\quad \mu_{2,k+1})^2}{\sum_{i=1}(1 - p_i)}$$

Note that R functions `dnorm`, `weighted.mean`, `cov.wt` can make code very easy.

Application to Old Faithful

See code in notes:

```
           pi      mu1       sig1      mu2      sig2
0          0.5000000 2.200000 0.5000000 4.500000 0.5000000
1          0.3749820 2.103842 0.3927174 4.318084 0.3838773
2          0.3614952 2.0
3          0.3563379 2.0
4          0.3536141 2.0
...
33         0.3485451 2.018942 0.2374124 4.273651 0.4378
```

Agrees with results from `optim` (Lectu

Slow convergence, but for many components (i.
avoids derivatives, high-dimensional optimization.

Sometimes the only thing that can be done.

Part of a more general recipe (beyond scope of course).

Summary

- Multivariate Optimization considerably more tricky that
-

- Requires some care with coding and checking that the answer you get really is reasonable.

- Many sophisticated methods available.
- Next: integration.