Assignment Project Exam Help

Multivariate Optimization

https://powcoder.com

BTRY/STSCI 4520

Add WeChat powcoder

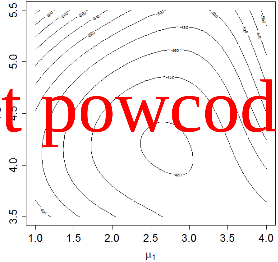
Multivariate Optimization

- 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.
- So we need some strategies for working in multiple dimensions.

Example: means of a normal mixture:

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

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



On Maximum Likelihood Estimation

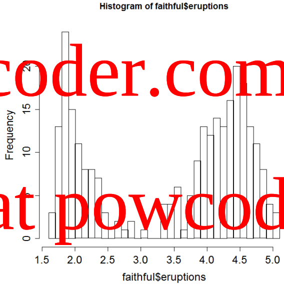
General principle of obtaining parameter estimates

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

In the case of the old-faithful data, it looks like we could use two normal distributions

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

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 that the probability of $X_1, \ldots, X_n$ is $\prod_{i=1}^{n} P(X_i)$.

Usually more convenient to work with log probability as a function of parameters μ_1 and μ_2 (doesn't change where the maximum is).

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

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. Run an optimizer to choose a new μ_1 keeping μ_2 fixed.

$$\mu_{1,1} = \operatorname*{argmax}_{\mu_1} l(\mu_1, \mu_{0,2})$$

3. Run an optimizer to choose μ_2 keeping μ_1 fixed at its current value

$$\mu_{1,2} = \operatorname*{argmax}_{\mu_2} l(\mu_{1,1}, \mu_2)$$

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 `mu` (in this case (μ_1, μ_2)).

- It also needs a dimension (`dim`) to tell it which element to work on.
- And it still needs upper and lower starting values for `mu[dim]`.

- We've also got data `X` to add to the function so we need to 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
  muhist = c()

  while(!tol.met){  # Tolerance will be checked by how much we
    oldmu = mu      # move mu over one cycle

    for(dim in 1:length(mu)){   # But we'll update the history
      iter = iter + 1           # at each coordinate
      mu = GoldenSection(fn,mul,mur,mu,dim,X)$xm
      muhist = rbind(muhist,mu)
    }

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

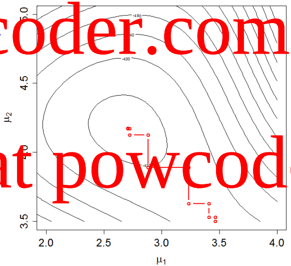
Results

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(1,3.5) # lower bound
mur = c(4,5.5) # upper bound
mu = c(3.6,3.5) # start
```


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

Some Notes

- Using `GoldenSection` requires a box for the optimal value as well as a starting point.
- Alternative: `NewtonRaphson` could be used at each 1-dimensional step (but would also require modification).
- One co-ordinate at a time = very slow convergence (long set of zig-zag lines).
- Convergence criterion: $\|\mu_{k-1} - \mu_k\|$: how much did we move over the last iteration? Does not guarantee that you have 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.
- Define the gradient $\nabla f(x) = (df/dx_1, \ldots, df/dx_p)$ – it is easy to see that this is the direction that f changes fastest.
- So consider an update of the form $x_{k+1} = x_k + \alpha \nabla f(x_k)$ for some α.
- To choose α: 1-dimensional optimization on the function

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

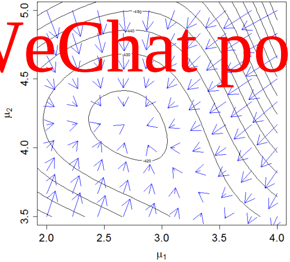
Our Example

We need derivatives of the log likelihood

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

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

Visualized as arrows:



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. Line search: find the α_k that maximizes $f^*(\alpha) = f(x_k + \alpha g)$.
3. Set $x_{k+1} = x_k + \alpha_k g_k$.

Line search: any one-dimensional optimization

- Requires either specialized code (as in `GoldenSection` above, or defining a function within a function (see example later).
- Should guarantee that $f(x_{k+1}) > f(x_k)$.
- 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 $a_l = 0$.
- Right end point is more tricky:; we want to be past the top of the 'bump' in the line search.

- One strategy:
  - Start with a_r small (say a some multiples of tolerance).
  - Keep doubling a_r until $f^*(a_r)$ decreases. The maximum must be between the last three values.
  - To prevent going on forever, set a maximum place to stop.

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
    fval = c(fns(0,X),fns(2*tol,X),fns(4*tol,X))

    try = 3                                          # Keep doubling
    while( fval[try]>fval[try-1] & try < maxtry ){  # the end point
      ar = c(ar,2*ar[try])
      try = try+1
      fval = c(fval,fns(ar[try],X))
    }
    if(try == maxtry){ al = ar[try-1]; ar = ar[try] }   # Last two points
    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

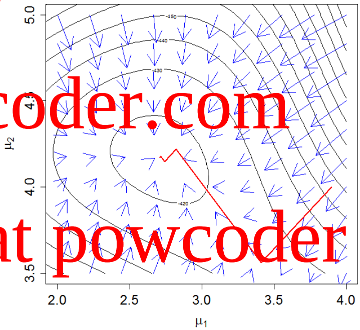

```
SteepestAscent = function(mu,fn,dfn,X,tol=1e-8,maxit=1000,maxtry=25)
{
  tol.met=FALSE; iter = 0; iterhist = mu

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

    # Define a new function -- I'm using the fact that mu and g are going to be
    # taken from the SteepestAscent workspace when fns is called.
    fns = function(alpha,X){ return(fn(mu+alpha*g,X)) }

    # Conduct the line search
    linesearch = GoldenSectionLineSearch(fns,X,tol,maxtry)

    # 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.

- Still guaranteed to increase $f(x_k)$ at each step.

- Still some zig-zagging across ridges.



Multivariate Newton-Raphson

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

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

where $H(x^*)$ is the *Hessian* at x^*

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

And try to maximize the quadratic approximation to f.

Multivariate Newton-Raphson

Want to maximize the approximation

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

Setting the gradient to zero

$$\nabla \tilde{f}(x) = \nabla f(x^*) + H(x^*)(x - x^*) = 0$$

or

$$x = x^* - H(x^*)^{-1} \nabla f(x^*)$$

This gives the Newton-Raphson update (with k now meaning 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.

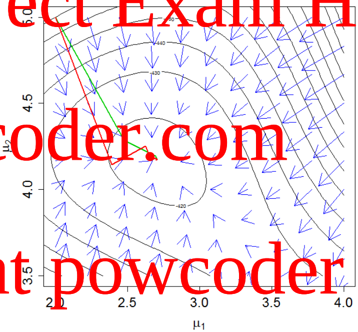
A First Multivariate Newton-Raphson Function

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

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

    g = dfn(mu,X)            # Gradient
    H = d2fn(mu,X)           # Hessian
    mu = mu - solve(H,g)     # Update
    iterhist = rbind(iterhist,mu)

    if( (max(abs( mu-oldmu )) < tol & max(abs(g)) < tol) | iter > maxit)
       { 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

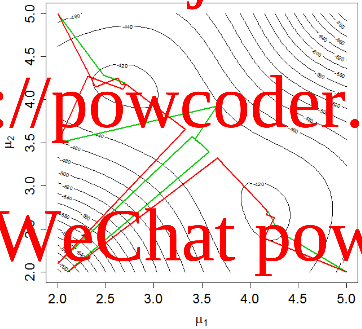
- Nominally very fast convergence.
- But no guarantee that you will find a maximum instead of a minimum (or a saddle).
- Can have problems with singular Hessians.
- Various fixes (eg checking that $f(x)$ increases each iteration)



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: the Michaelis-Menten model

$$Y_i = \frac{\theta_1 x_i}{\theta_2 + x_i} + \epsilon_i$$

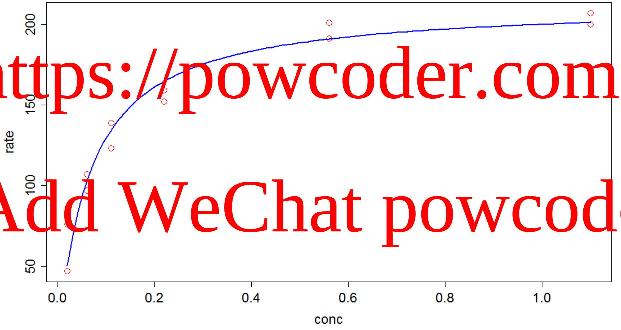
Estimate θ_1 and θ_2 by minimizing squared error, ie maximizing

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

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 particular we will want to look at the matrix

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

where the entries are

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

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 gradient

$$\nabla F(\theta) = \sum_{i=1}^{n}\frac{dg(x_i,\theta)}{d\theta}(Y_i - g(x_i,\theta)) = -J(X,\theta)^T(Y - g(X,\theta))$$

and Hessian

$$\begin{aligned} H(\theta) &= -\sum_{i=1}^{n}\frac{dg(x_i,\theta)}{d\theta}\frac{dg(x_i,\theta)}{d\theta}^T - \frac{d^2 g(x_i,\theta)}{d\theta d\theta^T}(Y_i - g(x_i,\theta)) \\ &\approx -J(X,\theta)^T J(X,\theta) \end{aligned}$$

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.met=FALSE; iter = 0; iterhist = theta

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

    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 matrix $J(X,\theta)$


```
dmmgn = function(theta,x){
  return( cbind(x/(theta[2]+x), -theta[1]*x/(theta[2]+x)^2) )
}
```

We can then call


```
opt = GaussNewton(c(205,0.08),mmgn,dmmgn,Puro[,2],Puro[,1])
```

Note that $J(\theta_k,X)^T J(\theta_k,X)$ is always positive definite; 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.
- Multiple optimization tools in various `R` packages, specialized to different purposes.
- Some general-purpose tools are available in the `optim` function: among the possible `method` options:
  - `BFGS`, `L-BFGS-B`, `CG` – like Newton-Raphson; details differ about how Hessians are calculated.
  - `Nelder-Meade` only uses function values (see next slide).
  - `SANN` simulated annealing (random searching algorithm – next slide).

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

Mixture Model Example

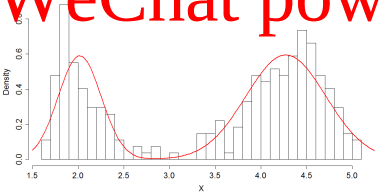
Fit all parameters parameters; $\phi_{\mu,\sigma}(x) = N(\mu,\sigma)$ density

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


```
fn = function(theta,X)
{
-sum( log( (1-theta[3])*dnorm(X,mean=theta[1],sd=theta[4])+
              theta[3]*dnorm(X,mean=theta[2],sd=theta[5])))
}
```



```
res = optim(c(2,4.3,0.5,0.5,0.5),fn,method = "Nelder-Mead",X=X)
```

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.
- Analogs in higher dimensions.

Simulated Annealing

- Start somewhere and randomly try a nearby point.
- Decide to move to that point probabilistically, but with lower probability for moving downhill.

- Slowly make the size of moves smaller and insist more on 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) so constrain $\mu_1 > \mu_2$.
- Multiple ways of incorporating constraints, depending on type of constraint, size of problem etc.

Discrete Optimization

- Sometimes x takes a discrete set of values – integers, category labels etc.
- Example: search over θ_i either 1 or 0 depending on whether to 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.

- We have seen that Newton's method works very well close to the optimum.
- But it can get stuck in local minima instead of maxima, or in saddle points.
- On the other hand, steepest-ascent always increases the objective function; but can converge slowly.
- 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 modify this by adding a diagonal element of $H(x_k)$:

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

- 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.

  Note that if $(e_1, \dots, e_p)$ are the eigenvalues of $H(x_k)$, the eigenvalues of $H(x_k) - \lambda I$ are $(e_1 - \lambda, \dots, e_k - \lambda)$

  $\Rightarrow$ make $\lambda >$ largest positive e_j (so $H(x_k) - \lambda I$ is negative definite).
- We also want to make sure that $f(x_k)$ increases at each 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

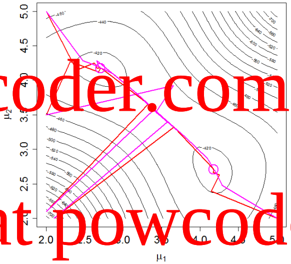
- Choose tolerances ϵ_1, ϵ_2, ϵ_3, maximum iterations
- Start at x_0, calculate $f(x_0)$, $\nabla f(x_0)$, $H(x_0)$
- Set λ_0 to be larger than the maximum positive eigenvalue of $H(x_0)$ plus some tolerance (make it near zero if $H(x_0)$ is neg. def).
- While $\|x_{k+1} - x_k\| > \epsilon_1$, $\|\nabla f(x_{k+1})\| > \epsilon_2$, $f(x_{k+1}) - f(x_k) > \epsilon_3$, $k <$ maxit
  - $\tilde{x}_{k+1} = x_k - (H(x_k) - \lambda_k I)^{-1} \nabla f(x_k)$.
  - If $f(\tilde{x}_{k+1}) > f(x_k)$, set $x_{k+1} = \tilde{x}_{k+1}$, $\lambda_{k+1} = \lambda_k/2$
  - Else, while $f(\tilde{x}_{k+1}) \leq f(x_k)$
    - $\lambda_k = 2\lambda_k$, $\tilde{x}_{k+1} = x_k - (H(x_k) - \lambda_k I)^{-1} \nabla f(x_k)$

    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 hits a saddle-point, the steepest-descent algorithm pushes it towards one of the true maxima.
- 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 $X \sim N(\mu_1, \sigma_1^2)$
- If $Z = 0$, draw $X \sim N(\mu_2, \sigma_2^2)$.

We have $X_1, \ldots, X_k$, pretend there is also $Z_1, \ldots, Z_k$ giving the component that X_i belonged to.

But we don't see the Z_k; they're latent (similar to random effects).

Strategy below: if we observed the Z_k, things would be easy (just 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

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

More generally (any latent variable), if you know the distribution $P(X|Z)$ and the distribution $P(Z)$, then

$$P(X) = \int P(X|Z)dP(Z)$$

– ie, take the expectation $P(X|Z)$ with respect to $P(Z)$.

In this case there are 5 parameters $\theta = (\pi, \mu_1, \mu_2, \sigma_1, \sigma_2)$ with log 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:

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

$$= \frac{\pi \phi_{\mu_1, \sigma_1^2}(X_i)}{\pi \phi_{\mu_1, \sigma_1^2}(X_i) + (1 - \pi)\phi_{\mu_2, \sigma_2^2}(X_i)} = p_i$$

This is fairly easy to calculate with `dnorm`.

Now use these to update parameters using weighted estimates:

$$\pi_{k+1} = \frac{1}{n}\sum_{i=1}^{n} p_i$$

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

$$p_i = \frac{\pi_k \phi_{\mu_{1,k},\sigma^2_{1,k}}(X_i)}{\pi_k \phi_{\mu_{1,k},\sigma^2_1}(X_i) + (1 - \pi_k)\phi_{\mu_{2,k},\sigma^2_{2,k}}(X_i)}$$

  - Update elements of θ_{k+1}:

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

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

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 2.053577 0.2968770 4.299772 0.4005250
3      0.3563379 2.037892 0.2673986 4.290458 0.4125602
4      0.3536141 2.031067 0.2565781 4.284700 0.4211885
...
33     0.3485451 2.018942 0.2374124 4.273651 0.4378409
```

Agrees with results from `optim` (Lecture 10) to 2 decimal places.

Slow convergence, but for many components (i.e. more than 2) 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 univariate.
- Trade-offs between local minima, saddle points and speed of convergence.

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

- Next: integration.