Nonlinear regression

Model: $y_k = h(x_k, \theta) + \varepsilon_k$; $\varepsilon_k \sim$ iid $\mathrm{N}(0, \sigma^2)$; h known.

Data: (x_k, y_k) for $k = 1, \ldots, n$

Goal: Estimate the p-vector θ.

MLE: $\hat{\theta} = \arg\min_\theta S(\theta)$; $S(\theta) = \sum_k \{y_k - h(x_k, \theta)\}^2$

Residuals: $f_k(\theta) = y_k - h(x_k, \theta)$

Jacobian (an $n \times p$ matrix): $\{J(\theta)\}_{kj} = \frac{\partial}{\partial \theta_j} h(x_k, \theta) = -\frac{\partial}{\partial \theta_j} f_k(\theta)$

Under quite general regularity conditions, the MLE $\hat{\theta}$ is asympototically $\mathrm{N}(\theta, \Sigma)$ where $\Sigma = \sigma^2 [J(\theta)' J(\theta)]^{-1}$.

The Jacobian matrix J thus plays the role of the X matrix. Note that in the special case where x_k is a p-vector and $h(x_k, \theta) = x_k' \theta$, we have $J = X$, where the rows of X are the row vectors x_k'.

A natural estimate of Σ is $\hat{\Sigma} = s^2 [J(\hat{\theta})' J(\hat{\theta})]^{-1}$ where $s^2 = \mathrm{RSS}/(n-p) = f(\hat{\theta})' f(\hat{\theta})/(n-p)$.

Towards the algorithm

We wish to minimize $S(\theta) = \sum_k \{f_k(\theta)\}^2 = f(\theta)' f(\theta)$

Gradient: $g_j(\theta) = \frac{\partial}{\partial \theta_j} S(\theta) = 2 \sum_k f_k(\theta) \frac{\partial}{\partial \theta_j} f_k(\theta)$,
so that $g(\theta) = -2J(\theta)' f(\theta)$

Hessian: $G_{ij}(\theta) = \frac{\partial^2}{\partial \theta_i \partial \theta_j} S(\theta) =$
$2 \sum_k \{f_k(\theta) \frac{\partial^2}{\partial \theta_i \partial \theta_j} f_k(\theta) + \frac{\partial}{\partial \theta_i} f_k(\theta) \frac{\partial}{\partial \theta_j} f_k(\theta)\}$

We could use the previously-discussed Newton-Raphson approach, but it is simpler to do the following.

Simplifying assumption:
Suppose $h(x_k, \theta)$ is approximately linear around θ_0.

Then $y_k \approx h(x_k, \theta_0) + (\theta - \theta_0)' \nabla h(x_k, \theta_0) + \varepsilon_k$.

It follows that $f(\theta_0) \approx J(\theta_0) (\theta - \theta_0) + \varepsilon$.

This suggests the **Gauss-Newton algorithm** (next page).

Gauss-Newton algorithm

1. Find a good starting point $\hat{\theta}^{(0)}$.
2. At step $s+1$,
   (a) Form the residuals $f(\hat{\theta}^{(s)})$
   (b) Form the Jacobian matrix $J(\hat{\theta}^{(s)})$
   (c) Use a standard linear regression routine to obtain $\delta^{(s)} = [J(\hat{\theta}^{(s)})' J(\hat{\theta}^{(s)})]^{-1} \; J(\hat{\theta}^{(s)})' f(\hat{\theta}^{(s)})$
   (d) Obtain the new estimate $\hat{\theta}^{(s+1)} = \hat{\theta}^{(s)} + \delta^{(s)}$.

Problem: Need good starting values.

Solution: Similar to previous solutions to Newton-Raphson (especially step-halving).

Note: We can do all of this with weights w_k or a known $n \times n$ covariance matrix V.

Example: Michaelis-Menton equation

Consider the kinetics of an enzyme-catalyzed reaction converting a substrate S into a product P.

$$E + S \rightleftharpoons E \cdot S \rightarrow E + P$$

Let $V_0 =$ initial velocity of the reaction
$[S] =$ concentration of the substrate
$V_{\max} =$ maximum initial velocity
$K_m =$ Michael-Menton rate constant

We are interested in estimating $V_{\max}$ and K_m (properties of the enzyme) from data on $[S]$ (under experimental control) and V_0 (measureable).

A reasonable kinetic model results in the Michaelis-Menton equation:

$$V_0 = \frac{V_{\max}\,[S]}{K_m + [S]}$$

We might imagine that the measurement error in V_0 is normally distributed with constant variance, leading to a nonlinear regression type model.
Biologists usually look at $1/V_0$ vs $1/[S]$, resulting in the Lineweaver-Burk equation:

$$1/V_0 = 1/V_{\max} + (K_m/V_{\max})(1/[S])$$

```
h.mm <- function(theta,x) x*theta[1]/(theta[2]+x)

hg.mm <- function(theta,x)
  cbind(x/(theta[2]+x),-x*theta[1]/(theta[2]+x)^2)

nls2 <-
function(h, g, x, y, start=c(33,0.05),
         tol1=1e-8, tol2=1e-6, nit=1000, nhalf=20)
{
  theta.old <- start
  r <- y - h(theta.old,x); rss.old <- sum(r^2)

  for(i in 1:nit) {
    X <- g(theta.old,x)
    step <- lm(r ~ -1 + X)$coef
    theta <- theta.old + step

    for(j in 0:nhalf) {     # step-halving
      rn <- y - h(theta,x)
      if(rss.old > (rss <- sum(rn^2))) break
      step <- step/2
      theta <- theta.old + step
    }
    r <- rn
    if(all(abs(theta-theta.old) <
        tol1*(abs(theta.old)+tol2))) break
    theta.old <- theta; rss.old <- rss
  }

  X <- g(theta,x)
  sigmasq <- sum(r^2)/(length(r)-length(theta))
  V <- solve(t(X) %*% X)

  list(est=theta, var=sigmasq*V)
}
```

```
# my function
o <- nls2(h.mm, hg.mm, data$x, data$y,
          start=c(33,0.05))
o$est

# non-linear regression
library(nls)
o2 <- nls(y~V*x/(k+x), data=data,
          start=list(V=33,k=0.05))
summary(o2)

# general optimizer
f <- function(theta,x,y)
      sum((y-h.mm(theta,x))^2)

o3 <- nlm(f,c(33,0.05), x=data$x, y=data$y,
          steptol=1e-9, gradtol=1e-9)
o3$est
```

	$V_{\max}$	K_m
NLR	33.1 (0.4)	0.046 (0.002)
LR	32.5 (0.6)	0.043 (0.002)

Example results

Good starting points:

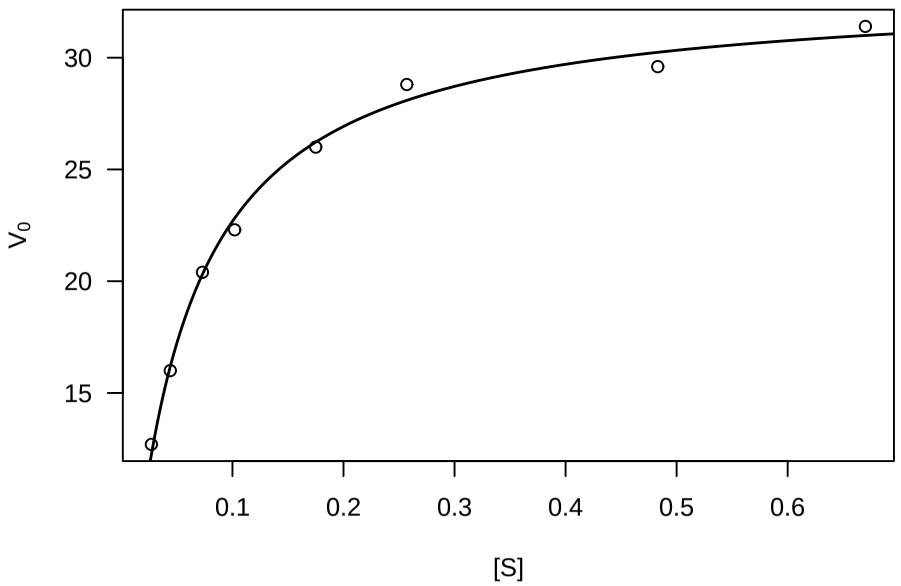
iteration	stephalves	$V_{\max}$	K_m
start	—	32.50	0.0400
1	0	33.08	0.0456
2–5	0	33.12	0.0461

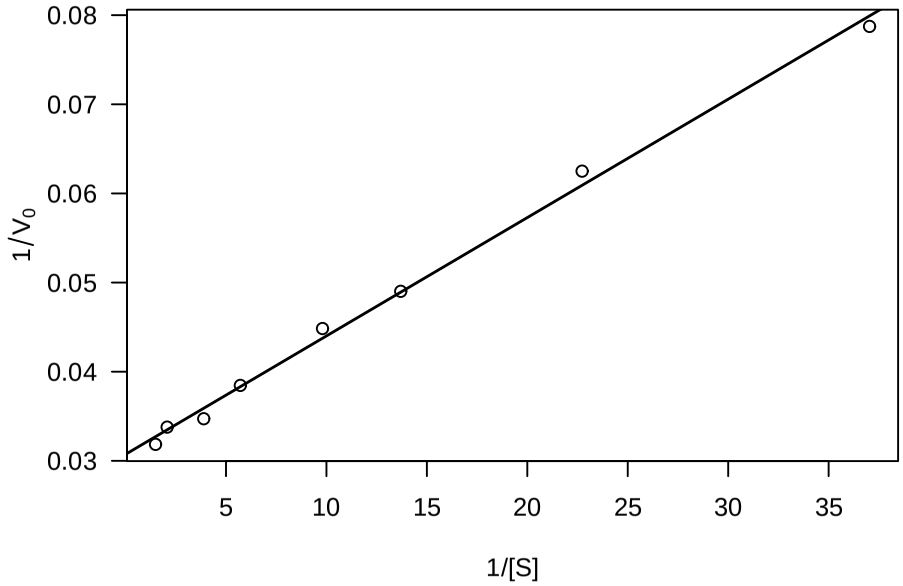
Bad starting points:

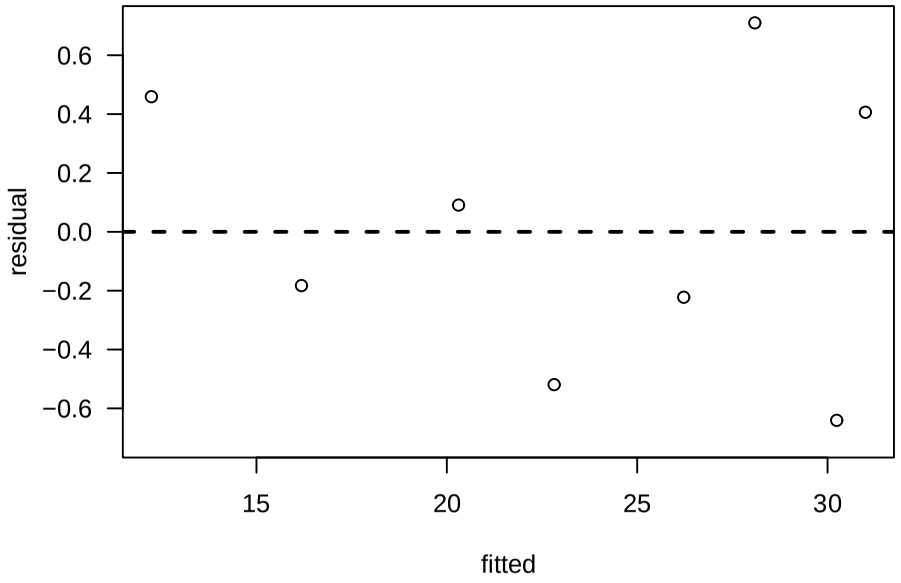
start	—	1	2
1	0	–444	–1328
2	0	$-\infty$	$-\infty$

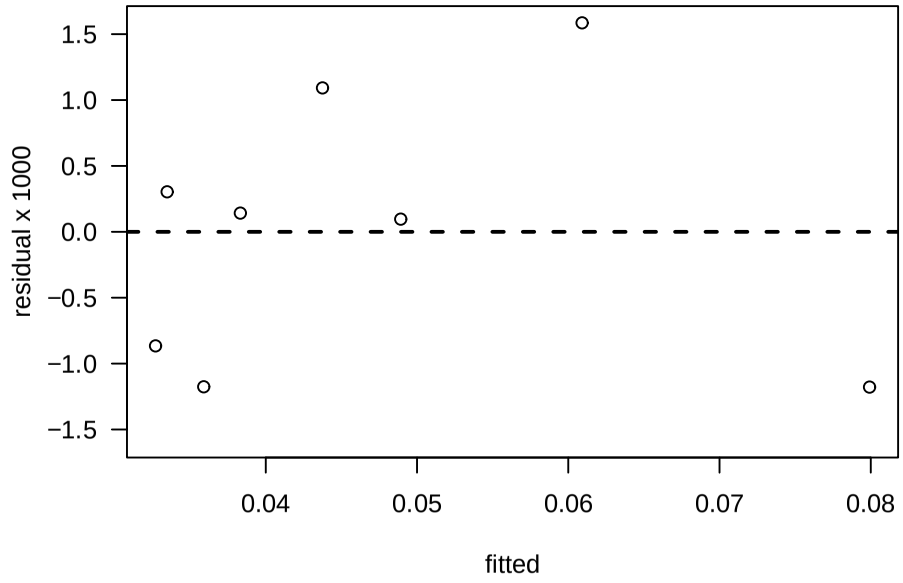
Bad starting points; use step halving:

start	—	1.00	2.0000
1	11	0.57	0.7014
2	9	0.39	–0.0036
3	0	28.85	1.2215
4	5	17.79	0.3966
5	3	14.47	–0.0117
6	0	26.14	0.0103
⋮	⋮	⋮	⋮
10–12	0	33.12	0.0461


V_0
30
25
20
15
0.1
0.2
0.3
0.4
0.5
0.6
[S]


$1/V_0$
0.08
0.07
0.06
0.05
0.04
0.03
5
10
15
20
25
30
35
1/[S]


residual
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
15
20
25
30
fitted


residual x 1000
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.04
0.05
0.06
0.07
0.08
fitted

Generalized linear models

Data: $(y, x)_i$ for $i = 1, \ldots, n$

Mean: $\mu = \mathsf{E}(y|x)$

Link, g: $\eta = g(\mu) = x'\beta$

Var func: $\mathsf{var}(y|x) = \phi\, V(\mu)$

Log likelihood: $l(\theta, \phi; y) = \{y\ \theta - b(\theta)\}/a(\phi) + c(y, \phi)$

Score function: $\partial l/\partial\theta = \{y - b'(\theta)\}/a(\phi)$

Neg obs info: $\partial^2 l/\partial\theta^2 = -b''(\theta)/a(\phi)$

Note:

$$\mu = b'(\theta)$$
$$\mathsf{var}(y) = b''(\theta)a(\phi)$$

Canonical link: g such that $g(\mu) = \theta$ (*ie*, $g^{-1} = b'$)

Generally we have $a(\phi) = \phi/w$ (in that case, ϕ will drop out of the following).

Examples

Model	Normal	Poisson	Binomial/m	Gamma
ϕ	σ^2	1	$1/m$	$1/\nu$
$b(\theta)$	$\theta^2/2$	$\exp(\theta)$	$\log(1+e^\theta)$	$-\log(-\theta)$
$\mu(\theta)$	θ	$\exp(\theta)$	$e^\theta/(1+e^\theta)$	$-1/\theta$
canon link	id	$\log$	logit	reciprocal
var fctn	1	μ	$\mu(1-\mu)$	μ^2

Iteratively reweighted least squares

$\hat{\eta}_0 =$ current estimate of the linear predictor
$\hat{\mu}_0 =$ corresponding fitted value ($\hat{\eta}_0 = g(\hat{\mu}_0)$)

Linearize $g(y)$ around $\hat{\mu}_0$:

$$\begin{aligned} z_0 &= g(\hat{\mu}_0) + (y - \hat{\mu}_0)\, g'(\hat{\mu}_0) \\ &= \hat{\eta}_0 + (y - \hat{\mu}_0)\, g'(\hat{\mu}_0) \end{aligned}$$

Estimated weights:

$$W_0^{-1} = \hat{\text{var}}(z_0) = \{g'(\hat{\mu}_0)\}^2 V(\hat{\mu}_0)$$

IRLS algorithm:

1. Start with initial estimates, generally $\hat{\mu}_0 = y$ and $\hat{\eta}_0 = g(\hat{\mu}_0)$
2. Form the z_0 and the weights W_0.
3. Estimate the parameters β_1 by regressing z_0 on the covariates x with weights W_0.
4. Form the linear predictors $\hat{\eta}_1 = x'\beta$ and the fitted values $\hat{\mu}_1 = g^{-1}(\hat{\eta})$ and return to step 2.

It turns out that IRLS is equivalent to Fisher scoring. (See McCullagh and Nelder, section 2.5.) In the case of the canonical link, this is also equivalent to Newton-Raphson.

Examples

Normal: just least squares

Poisson with log link:

$\eta = \log(\mu)$; $\mu = e^{\eta}$

$g'(\mu) = 1/\mu$; $V(\mu) = \mu$

$z_0 = \hat{\eta}_0 + (y - e^{\hat{\eta}_0})e^{-\hat{\eta}_0}$; $W_0 = \hat{\mu}_0 = e^{\hat{\eta}_0}$

Binomial with logit link:

$\eta = \mathsf{logit}(\mu) = \log[\mu/(1-\mu)]$; $\mu = e^{\eta}/(1+e^{\eta})$

$g'(\mu) = 1/[\mu(1-\mu)]$; $V(\mu) = \mu(1-\mu)$

$z_0 = \hat{\eta}_0 + (y - \hat{\mu}_0)/[\hat{\mu}_0(1-\hat{\mu}_0)]$

$W_0 = \hat{\mu}_0(1-\hat{\mu}_0)$

Gamma with reciprocal link:

$\eta = 1/\mu$; $\mu = 1/\eta$

$g'(\mu) = -1/\mu^2$; $V(\mu) = \mu^2$

$z_0 = \hat{\eta}_0 - (y - 1/\hat{\eta}_0)\hat{\eta}_0^2$

$W_0 = \hat{\eta}_0^2$

Miscellaneous things

Dispersion parameter: When we don't take $\phi = 1$, the usual estimate is via the method of moments:

$$\hat{\phi} = \frac{1}{n-p} \sum \frac{(y - \hat{\mu})^2}{V(\hat{\mu})}$$

Standard errors: $\hat{\text{var}}(\hat{\beta}) = \hat{\phi}(X'\hat{W}X)^{-1}$

Quasi likelihood: Pick a link and a variance function and don't worry about the model. In other words, IRLS is a good thing!

IRLS vs nonlinear minimization:
Consider the identity link with $V(\mu) = \mu$, and compare:

IRLS: pick $\hat{\beta}^{(0)}$; find $\hat{\beta}^{(1)}$ minimizing
$\sum (y - \hat{\beta}^{(1)} x)^2 / (\hat{\beta}^{(0)} x)$

NL min: $\hat{\beta}^{(1)}$ minimizing $\sum (y - \hat{\beta}^{(1)} x)^2 / (\hat{\beta}^{(1)} x)$

Further miscellanea

More complex variance functions:

Often you may wish to use var$(y_i) = \delta_i V(\mu_i) + \gamma_i$, where the δ_i and γ_i are known. *This is easy!*

Things get much harder if you want to estimate parameters in the variance function. (See McCullagh and Nelder.)

Nonlinear parameters in the link or covariates:

Consider $g(\mu) = (\mu^\lambda - 1)/\lambda$ where λ is to be estimated, or $\mu = \beta_0 + \beta_1 \exp(\gamma_1 x_1) + \beta_2 \exp(\gamma_2 x_2)$, where the β's and γ's are to be estimated.

This is not too hard: linearize as at the beginning of the lecture.

Major point: You want to get to know what's going on in `nls` and `glm` to some extent, because you may need to make things more general.

GLM in R

```
o <- glm(n.dead/n.mites ~ dose, data=mites,
         family=binomial(link=logit),
         weights=n.mites,
         epsilon=1e-7, maxit=100, trace=T)
summary(o)$coef[,1:2]
                Est    SE
(Intercept)  -0.80  0.10
dose          1.22  0.10

o2 <- glm(n.dead/n.mites ~ offset(dose) + dose,
          family=binomial(link=logit),
          weights=n.mites, data=mites)
summary(o2)$coef[,1:2]
                Est    SE
(Intercept)  -0.80  0.10
dose          0.22  0.10

# this should work, but doesn't
summary(o,dispersion=0)$coef[,1:2]

# what to do
r <- resid(o,type="pearson")
print(phi <- sum(r^2)/(nrow(mites)-2))
1.97

# revised  SEs
summary(o2)$coef[,2] * sqrt(phi)
0.14 0.14
```