Advanced Topics in Computing Lectures

1 Generalized Linear Models

1.1 Exponential Dispersion Models

We write

$$f(y|\theta,\phi) = \exp\left(\frac{y\theta - b(\theta)}{\phi} + c(y,\phi)\right)$$

(alternatively, write $a(\phi)$ instead of ϕ in the denominator): this is an *exponential dispersion model*; for *fixed* ϕ, this is an exponential family in θ.

Write m for the reference measure for the densities.

Clearly,

$$\int f(y|\theta,\phi)\,dm(y) = 1.$$

Differentiation with respect to θ, assuming that interchanging integration and differentiation is justified:

$$\begin{aligned} 0 &= \frac{\partial}{\partial\theta}\int f(y|\theta,\phi)\,dm(y) \\ &= \int \frac{\partial f(y|\theta,\phi)}{\partial\theta}\,dm(y) \\ &= \int \frac{y - b'(\theta)}{\phi} f(y|\theta,\phi)\,dm(y) \\ &= \frac{E_{\theta,\phi}(y) - b'(\theta)}{\phi} \end{aligned}$$

so that

$$E_{\theta,\phi}(y) = b'(\theta)$$

which does not depend on ϕ. We can thus write

$$\mathbb{E}(y) = \mu = b'(\theta).$$

Differentiating once more, we get

$$\begin{aligned} 0 &= \int \left(-\frac{b''(\theta)}{\phi} + \left(\frac{y - b'(\theta)}{\phi}\right)^2\right) f(y|\theta,\phi)\,dm(y) \\ &= -\frac{b''(\theta)}{\phi} + \frac{V_{\theta,\phi}(y)}{\phi^2} \end{aligned}$$

so that

$$V_{\theta,\phi}(y) = \phi b''(\theta).$$

If $\mu = b'(\theta)$ defines a one-to-one relation between μ and θ, we can write $b''(\theta) = V(\mu)$ where V is the *variance function* of the family. Formally,

$$V(\mu) = b''((b')^{-1}(\mu))$$

With this,

$$\mathrm{var}(y) = \phi b''(\theta) = \phi V(\mu)$$

1.2 Generalized Linear Models

For a GLM, we then have for $i = 1, \ldots, n$ responses y_i and covariates (explanatory variables) x_i so that for $\mathbb{E}(y_i) = \mu_i = b'(\theta_i)$ we have

$$g(\mu_i) = \beta' x_i = \eta_i,$$

where g is the *link function* and η_i is the *linear predictor*. Alternatively,

$$\mu_i = h(\beta' x_i) = h(\eta_i),$$

where h is the *response function* (and g and h are inverses of each other if invertible).

What this precisely means is that for fixed ϕ_i, conditional on x_i the distribution of y_i has density $f(y_i|\theta_i, \phi_i)$ where $g(b'(\theta_i)) = \beta' x_i$:

$$y_i|x_i \sim f(y_i|\theta_i, \phi_i), \qquad g(b'(\theta_i)) = \beta' x_i.$$

1.3 Maximum Likelihood Estimation

The log-likelihood thus equals

$$\ell = \sum_{i=1}^{n} \left(\frac{y_i \theta_i - b(\theta_i)}{\phi_i} + c(y_i, \phi_i) \right).$$

To obtain the score function (partials of the log-likelihood with respect to the β_j), we first differentiate $g(b'(\theta_i)) = \beta' x_i$ with respect to β_j to get

$$x_{ij} = \frac{\partial \beta' x_i}{\partial \beta_j} = \frac{\partial g(b'(\theta_i))}{\partial \beta_j} = g'(b'(\theta_i)) b''(\theta_i) \frac{\partial \theta_i}{\partial \beta_j} = g'(\mu_i) V(\mu_i) \frac{\partial \theta_i}{\partial \beta_j}$$

from which

$$\frac{\partial \theta_i}{\partial \beta_j} = \frac{x_{ij}}{g'(\mu_i) V(\mu_i)}$$

and

$$\frac{\partial \ell}{\partial \beta_j} = \sum_{i=1}^{n} \frac{y_i - b'(\theta_i)}{\phi_i} \frac{x_{ij}}{g'(\mu_i) V(\mu_i)} = \sum_{i=1}^{n} \frac{y_i - \mu_i}{\phi_i V(\mu_i)} \frac{x_{ij}}{g'(\mu_i)}.$$

Maximum likelihood estimation is typically performed by solving the likelihood equations (i.e., setting the scores to zero). For Newton's method one needs to compute the Hessian $H_\ell(\beta) = [\partial^2\ell/\partial\beta_i\partial\beta_j]$. As $\mu_i = b'(\theta_i)$,

$$\frac{\partial\mu_i}{\partial\beta_j} = b''(\theta_i)\frac{\partial\theta_i}{\partial\beta_j} = V(\mu_i)\frac{x_{ij}}{g'(\mu_i)V(\mu_i)} = \frac{x_{ij}}{g'(\mu_i)}$$

and hence

$$\begin{aligned}
\frac{\partial^2\ell}{\partial\beta_j\beta_k} &= \sum_{i=1}^n \frac{\partial}{\partial\beta_k}\frac{y_i-\mu_i}{\phi_i V(\mu_i)}\frac{x_{ij}}{g'(\mu_i)} \\
&= \sum_{i=1}^n \frac{x_{ij}}{\phi_i}\left(-\frac{\partial\mu_i}{\partial\beta_k}\frac{1}{V(\mu_i)g'(\mu_i)} - \frac{y_i-\mu_i}{(V(\mu_i)g'(\mu_i))^2}\frac{\partial(V(\mu_i)g'(\mu_i))}{\partial\beta_k}\right) \\
&= -\sum_{i=1}^n \frac{x_{ij}x_{ik}}{\phi_i V(\mu_i)g'(\mu_i)^2} \\
&\quad -\sum_{i=1}^n \frac{(y_i-\mu_i)x_{ij}x_{ik}}{\phi_i V(\mu_i)^2 g'(\mu_i)^3}(V'(\mu_i)g'(\mu_i) + V(\mu_i)g''(\mu_i))
\end{aligned}$$

Noting that the second term looks complicated and has expectation zero, only the first term is used for the "Newton-type" iteration, the *Fisher scoring* algorithm. Equivalently, instead of using the observed information matrix (the negative Hessian of the log-likelihood), one uses its expectation (the Fisher information matrix).

Assume that $\phi_i = \phi/a_i$ with *known* case weights a_i. Then the Fisher information matrix is

$$\frac{1}{\phi}\sum_{i=1}^n \frac{a_i}{V(\mu_i)g'(\mu_i)^2}x_{ij}x_{ik} = \frac{X'W(\beta)X}{\phi},$$

where X is the usual regressor matrix (with x_i' as row i) and

$$W(\beta) = \operatorname{diag}\left(\frac{a_i}{V(\mu_i)g'(\mu_i)^2}\right), \qquad g(\mu_i) = x_i'\beta.$$

Similarly, the score function is

$$\frac{1}{\phi}\sum_{i=1}^n \frac{a_i}{V(\mu_i)g'(\mu_i)^2}g'(\mu_i)(y_i-\mu_i)x_{ij} = \frac{X'W(\beta)r(\beta)}{\phi},$$

where $r(\beta)$ is the vector with elements $g'(\mu_i)(y_i-\mu_i)$, the so-called *working residuals.* (Note that from $g(\mu_i) = \eta_i$, $g'(\mu_i)$ is often written as $\partial\eta_i/\partial\mu_i$.) Thus, the Fisher scoring update is

$$\begin{aligned}
\beta_{\text{new}} &\leftarrow \beta + (X'W(\beta)X)^{-1}X'W(\beta)r(\beta) \\
&= (X'W(\beta)X)^{-1}X'W(\beta)(X\beta + r(\beta)) \\
&= (X'W(\beta)X)^{-1}X'W(\beta)z(\beta)
\end{aligned}$$

where $z(\beta)$, the *working response*, has elements

$$\beta' x_i + g'(\mu_i)(y_i - \mu_i), \qquad g(\mu_i) = x_i'\beta.$$

Thus, the update is computed by weighted least squares regression of $z(\beta)$ on X (using the square roots of $W(\beta)$ as weights), and the Fisher scoring algorithm for determining the MLEs is an *iterative weighted least squares* (IWLS) algorithm.

Everything is considerable simplified when using the *canonical link* $g = (b')^{-1}$, for which

$$\eta_i = g(\mu_i) = g(b'(\theta_i)) = \theta_i.$$

As

$$\frac{d}{d\mu}(b')^{-1}(\mu) = \frac{1}{b''((b')^{-1}(\mu))} = \frac{1}{V(\mu)},$$

for the canonical link we have $g'(\mu)V(\mu) \equiv 1$. Hence,

$$\frac{\partial \ell}{\partial \beta_j} = \frac{1}{\phi}\sum_{i=1}^n a_i(y_i - \mu_i)x_{ij}, \qquad \frac{\partial^2 \ell}{\partial \beta_j \partial \beta_k} = -\frac{1}{\phi}\sum_{i=1}^n a_i V(\mu_i) x_{ij} x_{ik}$$

and observed and expected information matrices coincide, so that the IWLS Fisher scoring algorithm is the same as Newton's algorithm.

(Note that all of the above becomes more convenient if we consistently use the response function instead of the link function.)

1.4 Inference

Under suitable conditions, MLE $\hat\beta$ asymptotically

$$N(\beta, I(\beta)^{-1})$$

with expected Fisher information matrix

$$I(\beta) = \frac{1}{\phi} X' W(\beta) X.$$

Thus, standard errors can be computed as the square roots of the diagonal elements of

$$\widehat{\mathrm{cov}}(\hat\beta) = \phi(X' W(\hat\beta) X)^{-1}$$

where $X' W(\hat\beta) X$ is a by-product of the final IWLS iteration.

This needs an estimate of ϕ (unless known). Estimation by MLE is practically difficult: hence, ϕ is usually estimated by the method of moments.

Remembering that $\mathrm{var}(y_i) = \phi_i V(\mu_i) = \phi V(\mu_i)/a_i$, we see that if β was known, unbiased estimate of ϕ would be

$$\frac{1}{n}\sum_{i=1}^n \frac{a_i(y_i - \mu_i)^2}{V(\mu_i)}.$$

Taking into account that β is estimated, estimate employed is

$$\hat{\phi} = \frac{1}{n-p} \sum_{i=1}^{n} \frac{a_i (y_i - \hat{\mu}_i)^2}{V(\hat{\mu}_i)}$$

(where p is the number of β parameters).

1.5 Deviance

The *deviance* is a quality-of-fit statistic for model fitting achieved by Maximum Likelihood, generalizing the idea of using the sum of squares of residuals in ordinary least squares, and (here) defined as

$$D = 2\phi(\ell_{sat} - \ell_{mod})$$

(assuming a common ϕ, perhaps after taking out weights), where the *saturated model* uses separate parameters for each observation so that the data is fitted exactly. Here, this can be achieved via $y_i = \mu_i^* = b'(\theta_i^*)$ which gives zero scores (provided that this also maximizes, of course).

The contribution of observation i to $\ell_{sat} - \ell_{mod}$ is then

$$\frac{y_i \theta_i^* - b'(\theta_i^*)}{\phi_i} - \frac{y_i \hat{\theta}_i - b'(\hat{\theta}_i)}{\phi_i} = \left. \frac{y_i \theta - b(\theta)}{\phi_i} \right|_{\hat{\theta}_i}^{\theta_i^*},$$

where $\hat{\theta}_i$ is obtained from the fitted model (i.e., $g(b'(\hat{\theta}_i)) = \hat{\beta}' x_i$). Now,

$$(y_i \theta - b(\theta))\big|_{\hat{\theta}_i}^{\theta_i^*} = \int_{\hat{\theta}_i}^{\theta_i^*} \frac{d}{d\theta}(y_i \theta - b(\theta))\, d\theta = \int_{\hat{\theta}_i}^{\theta_i^*} (y_i - b'(\theta))\, d\theta.$$

Substituting $\mu = b'(\theta)$ gives $d\mu = b''(\theta)\, d\theta$ or equivalently, $d\theta = V(\mu)^{-1}\, d\mu$, so that

$$\int_{\hat{\theta}_i}^{\theta_i^*} (y_i - b'(\theta))\, d\theta = \int_{\hat{\mu}_i}^{y_i} \frac{y_i - \mu}{V(\mu)}\, d\mu$$

and the deviance contribution of observation i is

$$2\phi_i \left. \frac{y_i \theta - b(\theta)}{\phi_i} \right|_{\hat{\theta}_i}^{\theta_i^*} = 2 \int_{\hat{\mu}_i}^{y_i} \frac{y_i - \mu}{V(\mu)}\, d\mu.$$

Note that this can be taken as the definition for the deviance as well as a basis for defining quasi-likelihood models based on a known variance function V.

1.6 Residuals

Several kinds of residuals can be defined for GLMs:

response $y_i - \hat{\mu}_i$

working from working response in IWLS, i.e., $g'(\hat{\mu}_i)(y_i - \hat{\mu}_i)$

Pearson

$$r_i^P = \frac{y_i - \hat{\mu}_i}{V(\hat{\mu}_i)}$$

so that $\sum_i (r_i^P)^2$ equals the generalized Pearson statistic.

deviance so that $\sum_i (r_i^D)^2$ equals the deviance (see above).

(All definitions equivalent for the Gaussian family.)

1.7 Generalized Linear Mixed Models

We augment the linear predictor by (unknown) random effects b_i:

$$\eta_i = x_i'\beta + z_i'b_i$$

where the b_i come from a suitable family of distributions and the z_i (as well as the x_i, of course) are known covariates. Typically,

$$b_i \sim N(0, G(\vartheta)).$$

Conditionally on b_i, y_i is taken to follow an exponential dispersion model with

$$g(\mathbb{E}(y_i|b_i)) = \eta_i = x_i'\beta + z_i'b_i.$$

The *marginal* likelihood function of the observed y_i is obtained by integrating the joint likelihood of the y_i and b_i with respect to the marginal distribution of the b_i:

$$L(\beta, \phi, \vartheta) = \prod_{i=1}^n \int f(y_i|\beta, \phi, \vartheta, b_i) f(b_i|\vartheta)\, db_i$$

(assuming for simplicity that random effects are independent across observation units).

Note in particular that for non-linear g we generally have

$$\mathbb{E}(y_i) = \mathbb{E}g^{-1}(x_i'\beta + z_i'b_i) \neq g^{-1}(\mathbb{E}(x_i'\beta + z_i'b_i)) = g^{-1}(x_i'\beta).$$

Thus, the β parameters have no simple interpretation in the marginal model for the y_i.

2 Optimization and Root Finding

Clearly, root finding and unconstrained smooth optimization are closely related: solving $f(x) = 0$ can be accomplished via minimizing $\|f(x)\|^2$, and unconstrained optima of f must be critical points, i.e., solve $\nabla f(x) = 0$. (Note that we simply write f in both cases: but f is necessarily scalar for optimization and typically vector-valued for root finding.)

Linear equations and linear least squares problems can be solved “exactly” using techniques from numerical linear algebra. Other problems can typically only be solved “approximately” as limits of iterations $x_{k+1} = g(x_k)$.

Suppose g is smooth and x^* the limit. Then clearly $x^* = g(x^*)$ and

$$x_{k+1} = g(x_k) \approx g(x^*) + J_g(x^*)(x_k - x^*) = x^* + J_g(x^*)(x_k - x^*)$$

where

$$J_g(x) = \left[\frac{\partial g_i}{\partial x_j}(x)\right]$$

is the Jacobian of g at x (sometimes also written as $(\nabla g)'(x)$, with the prime denoting transposition). Thus

$$x_{k+1} - x^* \approx J_g(x^*)(x_k - x^*)$$

and in general for local convergence it is necessary and sufficient that

$$\rho(J_g(x^*)) < 1$$

where $\rho(A) = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } A\}$ is the spectral radius of A, and in this case we get (at least) linear (local) convergence, i.e.,

$$\|x_{k+1} - x^*\| \le C\|x_k - x^*\|$$

for some $0 < C < 1$.

2.1 Root finding

The most well-known method for solving a non-linear equation $f(x) = 0$ is Newton's method. Suppose we have an approximation x_k to x^* and that in a neighborhood of x_k, the linearization

$$L_k(x) = f(x_k) + J_f(x_k)(x - x_k)$$

is a good approximation to f. Then an obvious candidate for a next approximation is obtained by solving $L_k(x) = 0$, i.e.,

$$x_{k+1} = x_k - J_f(x_k)^{-1}f(x_k).$$

This is the mathematical form: the computational one is

$$\text{Solve } J_f(x_k)s_k = -f(x_k) \text{ for } s_k; \qquad x_{k+1} = x_k + s_k.$$

Mathematically, the iteration function g for which $x_{k+1} = g(x_k)$ is

$$g(x) = x - J_f(x)^{-1}f(x)$$

which component-wise can be written as

$$g_i(x) = x_i - \sum_l [J_f(x)^{-1}]_{il} f_l(x).$$

Thus,

$$\frac{\partial g_i}{\partial x_j}(x) = \delta_{ij} - \sum_l \frac{\partial [J_f(x)^{-1}]_{il}}{\partial x_j} f_l(x) - \sum_l [J_f(x)^{-1}]_{il} \frac{\partial f_l}{\partial x_j}(x)$$

As $f(x^*) = 0$,

$$\frac{\partial g_i}{\partial x_j}(x^*) = \delta_{ij} - \sum_l [J_f(x^*)^{-1}]_{il} \frac{\partial f_l}{\partial x_j}(x^*) = \delta_{ij} - [J_f(x^*)^{-1} J_f(x^*)]_{ij} = 0$$

so that $J_g(x^*)$ vanishes! Thus, local convergence is super-linear and can in fact be shown to be at least quadratic.

Thus, Newton's method works very nicely once we get close enough to a root where the above approximations apply. However (with n the dimension of the problem):

- In each step, $O(n^2)$ derivatives needs to be computed (exactly or numerically), which can be costly (in particular if function and/or derivative evaluations are costly);
- In each step, an $n \times n$ linear system needs to be solved, which takes $O(n^3)$ operations.
- For super-linear convergence the exact Jacobian is not needed.

Super-linear convergence is particularly attractive because the inequality

$$|\|x_{k+1} - x_k\| - \|x_k - x^*\|| \le \|x_{k+1} - x^*\|$$

implies that in the case of super-linear convergence where

$$\|x_{k+1} - x^*\| \le \alpha_k \|x_k - x^*\|$$

with $\lim_{k\to\infty} \alpha_k = 0$ we have

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x_k\|}{\|x_k - x^*\|} = 1,$$

suggesting that the relative change in the x_k update is a good approximation for the relative error in the approximation of x^* by x_k (which is commonly used for stopping criteria).

In fact, one can show that for sequences

$$x_{k+1} = x_k - B_k^{-1} f(x_k)$$

with suitable non-singular matrices B_k one has super-linear (local) convergence to x^* with $f(x^*) = 0$ if and only if

$$\lim_{k\to\infty} \frac{\|(B_k - J_f(x^*))(x_{k+1} - x_k)\|}{\|x_{k+1} - x_k\|} = 0,$$

i.e., if B_k converges to $J_f(x^*)$ along the directions $s_k = x_{k+1} - x_k$ of the iterative method.

This suggests investigating iterative "Quasi-Newton" methods with such B_k which are less costly to compute and/or invert. The most famous one is *Broyden's method*, which is obtained as follows. First, we note that Newton's method is based on the approximation

$$f(x_{k+1}) \approx f(x_k) + J_f(x_k)(x_{k+1} - x_k).$$

This suggests considering approximations B_{k+1} which exactly satisfy the *secant equation*

$$f(x_{k+1}) = f(x_k) + B_{k+1}(x_{k+1} - x_k).$$

Of all such B_{k+1}, the one with the least change to B_k seems particularly attractive. Thus, writing

$$y_k = f(x_{k+1}) - f(x_k),$$

one looks for solutions to the problem

$$\|B - B_k\|_F \to \min, \qquad y_k = Bs_k$$

(where $\|\cdot\|_F$ is the Frobenius norm). Now

$$\|B - B_k\|_F^2 = \|\mathrm{vec}(B) - \mathrm{vec}(B_k)\|^2$$

and

$$\mathrm{vec}(Bs_k - y_k) = (s_k' \otimes I)\mathrm{vec}(B) - y_k.$$

So we end up with a convex quadratic optimization problem with linear constraints, which can be solved using geometric ideas involving (orthogonal projections on affine subspaces) or using Lagrange's method: with $b = \mathrm{vec}(B)$, $b_k = \mathrm{vec}(B_k)$ and $A_k = s_k' \otimes I$ we have the Lagrangian

$$\frac{1}{2}\|b - b_k\|^2 + \lambda'(A_k b - y_k)$$

with gradient with respect to b given by $b - b_k + A_k'\lambda$. Hence, the critical point solves $b = b_k - A_k'\lambda$ with

$$y_k = A_k b = A_k b_k - A_k A_k' \lambda$$

from which $\lambda = (A_k A_k')^{-1}(A_k b_k - y_k)$ and

$$b = b_k - A_k'(A_k A_k')^{-1}(A_k b_k - y_k).$$

Now

$$A_k A_k' = (s_k' \otimes I)(s_k \otimes I) = s_k' s_k I$$

and thus

$$\begin{aligned} A_k'(A_k A_k')^{-1}(A_k b_k - y_k) &= \frac{1}{s_k' s_k}(s_k \otimes I)(A_k b_k - y_k) \\ &= \frac{1}{s_k' s_k}\mathrm{vec}((A_k b_k - y_k)s_k') \end{aligned}$$

so that altogether

$$B = B_k + \frac{(y_k - B_k s_k)s_k'}{s_k' s_k}.$$

This gives *Broyden's method* (which generalizes the well-known secant method for $n = 1$: based on a suitable guess x_0 and a suitable full rank approximation B_0 for the Jacobian (e.g., I), iterate using

```
Solve $B_k s_k = -f(x_k)$ for $s_k$
$x_{k+1} = x_k + s_k$
$y_k = f(x_{k+1}) - f(x_k)$
$B_{k+1} = B_k + (y_k - B_k s_k)s_k'/(s_k' s_k)$.
```

Again, this is the mathematical form. Computationally, this can be improved by noticing that the iteration for the B_k performs *rank-one updates* of the form

$$B_{k+1} = B_k + u_k v_k'$$

and that we really need the inverse $H_k = B_k^{-1}$, which can be obtained by the Sherman-Morrison formula

$$(A + uv')^{-1} = A^{-1} - (1/\sigma)A^{-1}uv'A^{-1}, \qquad \sigma = 1 + v'A^{-1}u \neq 0.$$

Thus,

$$\begin{aligned} H_{k+1} &= (B_k + u_k v_k')^{-1} \\ &= B_k^{-1} - \frac{1}{1 + v_k' B_k^{-1} u_k} B_k^{-1} u_k v_k' B_k^{-1} \\ &= H_k - \frac{1}{1 + v_k' H_k u_k} H_k u_k v_k' H_k \end{aligned}$$

With $u_k = (y_k - B_k s_k)$ and $v_k = s_k/(s_k' s_k)$ we have

$$H_k u_k = B_k^{-1} u_k = B_k^{-1} y_k - s_k = H_k y_k - s_k$$

so that

$$1 + v_k' H_k u_k = 1 + \frac{s_k'(H_k y_k - s_k)}{s_k' s_k} = \frac{s_k' H_k y_k}{s_k' s_k}$$

and altogether

$$H_{k+1} = H_k - \frac{(H_k y_k - s_k)s_k' H_k}{s_k' H_k y_k}$$

(provided of course that $s_k' H_k y_k \neq 0$).

So the improved iteration takes

$$s_k = -H_k f(x_k)$$

and then updates H_k as above.

For very large n, so-called *spectral methods* (e.g., Barzilai-Borwein) have become popular. One variant (DF-SANE) for root finding is based on the idea to use very simple B_k or H_k proportional to the identity matrix. Now with $B_k = \beta_k I$ the secant condition $B_{k+1}s_k = y_k$ typically cannot be achieved exactly. Instead considering the least squares problem

$$\|y_k - \beta_{k+1}s_k\|^2 \to \min$$

yields

$$\beta_{k+1} = \frac{y_k' s_k}{s_k' s_k}$$

and hence the iteration (note we need B_k^{-1})

$$x_{k+1} = x_k - \alpha_k f(x_k), \qquad \alpha_{k+1} = \frac{s_k' s_k}{y_k' s_k}.$$

Alternatively, when instead using $H_k = \alpha_k I$, the secant condition $H_{k+1}y_k = s_k$ is replaced by the least squares problem

$$\|s_k - \alpha_{k+1}y_k\|^2 \to \min$$

with solution $\alpha_{k+1} = s_k' y_k / y_k' y_k$, resulting in the iteration

$$x_{k+1} = x_k - \alpha_k f(x_k), \qquad \alpha_{k+1} = \frac{s_k' y_k}{y_k' y_k}.$$

For R, these methods are only available in add-on packages. Package **nleqslv** has function `nleqslv()` providing the Newton and Broyden methods with a variety of global strategies. Package **BB** has function `BBsolve()` providing Barzilai-Borwein solvers, and `multiStart()` for starting the solver from multiple starting points.

As an example, consider the system

$$x_1^2 + x_2^2 = 2, \qquad e^{x_1 - 1} + x_2^3 = 2$$

which clearly has one solution at $x_1 = x_2 = 1$.

We use

```
> fn <- function(x) {
+     c(x[1]^2 + x[2]^2 - 2, exp(x[1] - 1) + x[2]^3 - 2)
+ }
```

and a starting guess of

```
> x0 <- c(2, 0.5)
```

Then

```
> nleqslv::nleqslv(x0, fn, method = "Broyden")
```

```
$x
[1] 1 1

$fvec
[1] 1.499778e-09 2.056389e-09

$termcd
[1] 1

$message
[1] "Function criterion near zero"

$scalex
[1] 1 1

$nfcnt
[1] 12

$njcnt
[1] 1

$iter
[1] 10
```

(which is the default method) and

```
> nleqslv::nleqslv(x0, fn, method = "Newton")

$x
[1] 1 1

$fvec
[1] 6.841629e-10 1.764278e-09

$termcd
[1] 1

$message
[1] "Function criterion near zero"

$scalex
[1] 1 1

$nfcnt
[1] 6

$njcnt
[1] 5
```

```
$iter
[1] 5
```

both find the solution at $(1, 1)$, with Newton's method needed fewer iterations. Note that we did not need to provide the Jacobians, but we could: with

```
> jac <- function(x) {
+     matrix(c(2 * x[1],
+              exp(x[1] - 1),
+              2 * x[2],
+              3 * x[2]^2),
+            2L,
+            2L)
+ }
```

we can use

```
> nleqslv::nleqslv(x0, fn, jac, method = "Broyden")

$x
[1] 1 1

$fvec
[1] 1.499780e-09 2.056391e-09

$termcd
[1] 1

$message
[1] "Function criterion near zero"

$scalex
[1] 1 1

$nfcnt
[1] 12

$njcnt
[1] 1

$iter
[1] 10

> nleqslv::nleqslv(x0, fn, jac, method = "Newton")

$x
[1] 1 1

$fvec
```

```
[1] 6.838969e-10 1.762221e-09

$termcd
[1] 1

$message
[1] "Function criterion near zero"

$scalex
[1] 1 1

$nfcnt
[1] 6

$njcnt
[1] 5

$iter
[1] 5
```

Alternatively, using the spectral Barzilai-Borwein solver:

```
> BB::BBsolve(x0, fn)

  Successful convergence.
$par
[1] 1.0000001 0.9999999

$residual
[1] 8.996546e-08

$fn.reduction
[1] 0.0003023472

$feval
[1] 68

$iter
[1] 6

$convergence
[1] 0

$message
[1] "Successful convergence"

$cpar
method      M     NM
     2     50      1
```

somewhat “surprisingly” also works fine on our (admittedly rather simple) test problem.

2.2 Optimization

We assume that the basic terminology and facts about (continuous) unconstrained and constrained optimization problems are well known.

2.2.1 Theory

Consider an unconstrained minimization problem with objective function f. If f is smooth,

$$f(x^* + s) = f(x^*) + \nabla f(x^*)' s + \frac{1}{2} s' H_f(x^* + \alpha s) s$$

for some $0 < \alpha < 1$, where

$$H_f(x) = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}(x) \right]$$

is the Hessian of f at x. Thus, for a local minimum x^* we necessarily have that $\nabla f(x^*) = 0$, i.e., x^* is a critical point of f (necessary first-order condition), and $H_f(x^*)$ must be non-negative definite (necessary second-order condition). Conversely, if x^* is a critical point of f and $H_f(x^*)$ is positive definite (sufficient second-order condition), then x^* is a local minimum.

For constrained optimization, there is a feasible set X, and local optima are relative to neighborhoods in the feasible set. Consider a suitable “interior” feasible point x^*. Then x^* is a local minimum iff $t \mapsto f(x(t))$ has a local minimum at $t = 0$ for all smooth curves $t \mapsto x(t)$ in X defined in some open interval containing 0 with $x(0) = x^*$. (Alternatively, if $J_g(x^*)$ has full column rank, use the implicit mapping theorem to obtain a local parametrization $x = x(s)$ of the feasible set with, say, $x(0) = x^*$, and consider the conditions for $f(x(s))$ to have an unconstrained local minimum at $s = 0$.)

Now

$$\frac{df(x(t))}{dt} = \sum_j \frac{\partial f}{\partial x_j}(x(t)) \dot{x}_j(t)$$

and

$$\frac{d^2 f(x(t))}{dt^2} = \sum_{j,k} \frac{\partial^2 f}{\partial x_j \partial x_k}(x(t)) \dot{x}_j(t) \dot{x}_k(t) + \sum_j \frac{\partial f}{\partial x_j}(x(t)) \ddot{x}_j(t)$$

so clearly we must have

$$\langle \nabla f(x^*), s \rangle = 0$$

for all feasible directions s at x^*. In the case X is defined by equality constraints $g(x) = [g_1(x), \dots, g_m(x)]' = 0$, we must have $g_i(x(t)) \equiv 0$ and hence

$$\sum_j \frac{\partial g_i}{\partial x_j}(x(t)) \dot{x}_j(t) = 0$$

and

$$\sum_{j,k} \frac{\partial^2 g_i}{\partial x_j \partial x_k}(x(t))\dot{x}_j(t)\dot{x}_k(t) + \sum_j \frac{\partial g_i}{\partial x_j}(x(t))\ddot{x}_j(t) = 0.$$

The first says that the set of feasible directions at x^* is given by $\mathrm{Null}(J_g(x^*))$. The first order necessary condition then translates into

$$\nabla f(x^*) \in (\mathrm{Null}(J_g(x^*)))^\perp = \mathrm{Range}(J_g(x^*)'),$$

i.e., there exist Lagrange multipliers λ such that

$$\nabla f(x^*) = -J_g(x^*)'\lambda.$$

With this λ, adding the sum of λ_i times the second-order constraint identities gives

$$\begin{aligned} \left.\frac{d^2 f(x(t))}{dt^2}\right|_{t=0} &= \sum_{j,k}\left(\frac{\partial^2 f}{\partial x_j \partial x_k}(x^*) + \sum_i \lambda_i \frac{\partial^2 g_i}{\partial x_j \partial x_k}(x^*)\right) s_j s_k \\ &\quad + \sum_j \left(\frac{\partial f}{\partial x_j}(x^*) + \sum_i \lambda_i \frac{\partial g_i}{\partial x_j}(x^*)\right)\ddot{x}_j(0) \\ &= s'\left(H_f(x^*) + \sum_i \lambda_i H_{g_i}(x^*)\right)s \\ &= s'B(x^*,\lambda)s \end{aligned}$$

so we must have that $s'B(x^*,\lambda)s \ge 0$ for all feasible directions s at x^*, which by the above equals $\mathrm{Null}(J_g(x^*)$. Thus, if Z is a basis for this space, for a local minimum we must have that the projected (or reduced) Hessian $Z'B(x^*,\lambda)Z$ is non-negative definite; conversely, one can show that if the first-order conditions are satisfied and $Z'B(x^*,\lambda)Z$ is positive definite, then x^* is a constrained local minimum.

For the Lagrangian

$$L(x,\lambda) = f(x) + \lambda' g(x)$$

we have

$$\nabla L(x,\lambda) = \begin{bmatrix} \nabla f(x) + J_g(x)'\lambda \\ g(x) \end{bmatrix}, \qquad H_L(x,\lambda) = \begin{bmatrix} B(x,\lambda) & J_g(x)' \\ J_g(x) & O \end{bmatrix}$$

and the above conveniently translates into x^* and λ being a critical point of the Lagrangian for with the second-order conditions for the x part (note that (x^*,λ) is a saddle point of L, so H_L cannot be positive definite).

When there are inequality constraints $h_k(x) \le 0$, one (KKT conditions) additionally gets that $\lambda_k^* \ge 0$ for the λ corresponding to h and and $h_k(x^*)\lambda_k = 0$: thus, the k for which $\lambda_k^* > 0$ given the *active set* of inequality constraints which are satisfied as equalities.

2.2.2 Algorithms for unconstrained optimization

For smooth unconstrained minimization, the linear approximation

$$f(x + \alpha s) \approx f(x) + \alpha \nabla f(x)' s$$

suggests looking for good α along the steepest descent direction $s = -\nabla f(x)$, typically performed as

$$\begin{aligned}
&s_k = -\nabla f(x_k) \\
&\text{Choose } \alpha_k \text{ to minimize } f(x_k + \alpha s_k) \\
&x_{k+1} = x_k + \alpha_k s_k.
\end{aligned}$$

(method of steepest descent).

Better performance is obtained using local quadratic approximation for f (equivalently, linear approximation for ∇f) which yields Newton's method

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

with computational form

$$\text{Solve } H_f(x_k) s_k = -\nabla f(x_k) \text{ for } s_k; \qquad x_{k+1} = x_k + s_k.$$

Similarly to the root-finding case, there are many modifications of the basic scheme, the general form

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

referred to as *quasi-Newton* methods: adding the α_k allows to improve the global performance by controlling the step size. (Note however that away from a critical point, the Newton direction s does not necessarily result in a local decrease of the objective function.)

Similar to the root-finding case, one can look for simple updating rules for Hessian approximations B_k subject to the secant constraint

$$B_k s_k = y_k = \nabla f(x_{k+1}) - \nabla f(x_k),$$

additionally taking into account that the B_k should be symmetric (as approximations to the Hessians). This motivates looking for solutions to the problem

$$\|B - B_k\|_F \to \min, \qquad B \text{ symmetric}, \quad y_k = B s_k.$$

which can easily be shown to have the unique solution

$$B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)'}{(y_k - B_k s_k)' s_k}.$$

However, this symmetric rank-one (SR1) update formula (apparently due to Davidon, 1959) has numerical difficulties. One thus considers similar problems with norms different from the simple Frobenius norm, such as weighted Frobenius norms

$$\|B - B_k\|_{F,W} = \|W^{1/2} (B - B_k) W^{1/2}\|_F.$$

One can show: for any (symmetric) W for which $Wy_k = s_k$ (in particular for the inverse of the average Hessian $\int_0^1 H_f(x_k + \tau s_k)\,d\tau$ featuring in Taylor's theorem) which makes the approximation problem non-dimensional, the unique solution to

$$\|B - B_k\|_{F,W} \to \min, \qquad B \text{ symmetric}, \quad y_k = Bs_k.$$

is given by

$$(I - \rho_k y_k s_k')B_k(I - \rho_k s_k y_k') + \rho_k y_k y_k', \qquad \rho_k = \frac{1}{y_k' s_k}.$$

This is the DFP (Davidon-Fletcher-Powell) update formula.

(Note that

$$\begin{aligned}
\nabla f(x+s) - \nabla f(x) &= \nabla f(x + \tau s)\Big|_0^1 \\
&= \int_0^1 \frac{d}{d\tau} \nabla f(x + \tau s)\,d\tau \\
&= \left(\int_0^1 H_f(x + \tau s)\,d\tau\right) s
\end{aligned}$$

and hence,

$$y_k = \nabla f(x_k + s_k) - \nabla f(x_k) = \left(\int_0^1 H_f(x_k + \tau s_k)\,d\tau\right) s_k,$$

ignoring α_k for simplicity.)

The most effective quasi-Newton formula is obtained by applying the corresponding approximation argument to $H_k = B_k^{-1}$, i.e., by using the unique solution to

$$\|H - H_k\|_{F,W} \to \min \qquad H \text{ symmetric}, \quad s_k = Hy_k.$$

with W any matrix satisfying $Ws_k = y_k$ (such as the average Hessian matrix), which is given by

$$(I - \rho_k s_k y_k')H_k(I - \rho_k y_k s_k') + \rho_k s_k s_k', \qquad \rho_k = \frac{1}{y_k' s_k}.$$

This is the BFGS (Broyden-Fletcher-Goldfarb-Shanno) update formula.

Using the Sherman-Morrison-Woodbury formula $(A+UV')^{-1} = A^{-1} - A^{-1}U(I + V'A^{-1}U)^{-1}V'A^{-1}$ and lengthy computations, one can show that the BFGS update for H_k corresponds to

$$B_{k+1} = B_k + \frac{y_k y_k'}{y_k' s_k} - \frac{B_k s_k s_k' B_k}{s_k' B_k s_k}$$

and that the DFP update for B_k corresponds to

$$H_{k+1} = H_k + \frac{s_k s_k'}{s_k' y_k} - \frac{H_k y_k y_k' H_k}{y_k' H_k y_k}$$

(Note the symmetry.)

Hence, all these updates are *rank-two updates.*

An alternative to quasi-Newton methods are *conjugate gradient methods* which have the form

$$x_{k+1} = x_k + \alpha_k s_k, \qquad s_{k+1} = -\nabla f(x_{k+1}) + \beta_{k+1} s_k$$

where the β are chosen such that s_{k+1} and s_k are suitably conjugate. E.g., the Fletcher-Reeves variant uses

$$\beta_{k+1} = \frac{g'_{k+1} g_{k+1}}{g'_k g_k}, \qquad g_{k+1} = \nabla f(x_{k+1}).$$

The methods require no matrix storage (uses only gradients) and are faster than steepest descent, and hence are particularly suitable for large-scale optimization problems.

A particular class of objective functions is

$$\phi(x) = \frac{1}{2} \sum_i r_i(x)^2 = \|r(x)\|^2/2,$$

known (roughly) as non-linear least squares problems, corresponding to minimizing the sum of squares of residuals

$$r_i(x) = t_i - f(e_i, x)$$

(in statistics, we usually write x_i and y_i for the inputs and targets, and θ instead of x for the unknown parameter). Clearly,

$$\nabla \phi(x) = J_r(x)' r(x), \qquad H_\phi(x) = J_r(x)' J_r(x) + \sum_i r_i(x) H_{r_i}(x).$$

The straightforward Newton method is typically replaced by the *Gauss-Newton method*

$$(J_r(x_k)' J_r(x_k)) s_k = -J_r(x_k)' r(x_k)$$

which drops the terms involving the H_{r_i} from the Hessian based on the idea that in the minimum the residuals r_i should be "small". This replaces a non-linear least squares problem by a sequence of linear least-squares problems with the right limit. However, the simplification may lead to ill-conditioned or rank deficient linear problems, in which case using the *Levenberg-Marquardt method*

$$(J_r(x_k)' J_r(x_k) + \mu_k I) s_k = -J_r(x_k)' r(x_k)$$

with suitably chosen $\mu_k > 0$ can be employed.

2.2.3 Algorithms for constrained optimization

Consider a minimization problem with equality constraints. Suppose we use Newton's method to determine the critical points of the Lagrange function, i.e., to solve

$$\nabla L(x, \lambda) = \begin{bmatrix} \nabla f(x) + J_g(x)' \lambda \\ g(x) \end{bmatrix} = 0.$$

This gives the system

$$\begin{bmatrix} B(x_k, \lambda_k) & J_g(x_k)' \\ J_g(x_k) & O \end{bmatrix} \begin{bmatrix} s_k \\ \delta_k \end{bmatrix} = - \begin{bmatrix} \nabla f(x_k) + J_g(x_k)'\lambda_k \\ g(x_k). \end{bmatrix}$$

which gives the first-order conditions for the constrained optimization problem

$$\min_s \frac{1}{2} s' B(x_k, \lambda_k) s + s'(f(x_k) + J_g(x_k)'\lambda_k)$$

subject to

$$J_g(x_k) s + g(x_k) = 0.$$

(Dropping arguments, to solve

$$\frac{1}{2} s' B s + s' u \to \min, \qquad Js + v = 0$$

we form the Lagrangian

$$L(s, \delta) = \frac{1}{2} s' B s + s' u + \delta'(Js + v)$$

for which the first-order conditions are

$$Bs + u + J'\delta = 0, \qquad Js + v = 0$$

or equivalently,

$$\begin{bmatrix} B & J' \\ J & O \end{bmatrix} \begin{bmatrix} s \\ \delta \end{bmatrix} = - \begin{bmatrix} u \\ v \end{bmatrix}.$$

Thus, each Newton step amounts to solving a quadratic optimization problem under linear constraints, so the approach is know as *sequential quadratic programming*. One does not necessarily solve the linear system directly (remember that the Hessian is symmetric but indefinite, so Choleski factorization is not possible).

Another idea is to convert the constrained problem into a sequence of unconstrained problems featuring a *penalty function* to achieve (approximate) feasibility. E.g., for the equality-constrained problem

$$f(x) \to \min, \qquad g(x) = 0$$

one considers

$$\phi_\rho(x) = f(x) + \rho \|g(x)\|^2/2 \to \min;$$

then under appropriate conditions it can be shown that the solutions $x^*(\rho)$ converge to the solution x^* of the original problem for $\rho \to \infty$. However, letting ρ increase without bounds often leads to problems, which motivates to instead use an *augmented Lagrange function*

$$L_\rho(x, \lambda) = f(x) + \lambda' g(x) + \rho \|g(x)\|^2/2$$

for which the Lagrange multiplier estimates can be manipulated in ways to keep them bounded while converging to the constrained optimum.

As an illustration where a fixed $\rho > 0$ suffices, we consider convex minimization with linear constraints, i.e., a primal problem of the form

$$f(x) \to \min, \qquad Ax = b.$$

with f convex. Using convex duality theory, with the Lagrangian $L(x, \lambda) = f(x) + \lambda'(Ax - b)$, the dual function is $\inf_x L(x, \lambda) = -f^*(-A'\lambda) - b'\lambda$ (where f^* is the convex conjugate), and optimality conditions are

$$Ax - b = 0, \qquad \nabla f(x) + A'\lambda = 0$$

(primal and dual feasibility).

The augmented Lagrangian

$$L_\rho(x, \lambda) = f(x) + \lambda'(Ax - b) + (\rho/2)\|Ax - b\|^2$$

can be viewed as unaugmented Lagrangian for the augmented problem

$$f(x) + (\rho/2)\|Ax - b\|^2 \to \min, \qquad Ax = b.$$

Now consider the iteration

$$x_{k+1} = \mathrm{argmin}_x L_\rho(x, \lambda_k), \qquad \lambda_{k+1} = \lambda_k + \rho(Ax_{k+1} - b).$$

By definition, x_{k+1} minimizes $L_\rho(x, \lambda_k)$, hence

$$\begin{aligned} 0 &= \nabla_x L_\rho(x_{k+1}, \lambda_k) \\ &= \nabla_x f(x_{k+1}) + A'(\lambda_k + \rho(Ax_{k+1} - b)) \\ &= \nabla_x f(x_{k+1}) + A'\lambda_{k+1}. \end{aligned}$$

Thus, the iteration ensures dual feasibility, convergence implies primal feasibility, hence altogether optimality.

More general cases use the same idea, but may need increasing $\rho = \rho_k$.

For additional inequality constraints, one can e.g. add slack variables and use bound constraints.

A related idea is using *interior point methods*, also known as *barrier methods*, which use barrier functions which ensure feasibility and increasingly penalize feasible points as they increase the boundary of the feasible set. E.g., with constraints $h(x) \le 0$,

$$\phi_\mu(x) = f(x) - \mu \sum_i \frac{1}{h_i(x)}, \qquad \phi_\mu(x) = f(x) - \mu \sum_i \log(-h_i(x))$$

are the inverse and logarithmic barrier functions, respectively.

For $\mu > 0$, an unconstrained minimum x^*_μ of ϕ_μ is necessarily feasible; as $\mu \to 0$, x^*_μ should converge to the constrained minimum x^*.

Such methods need starting values and keep all approximations in the interior of the feasible set, but allow convergence to points on the boundary.

In the base R distribution:

- `nlm()` does a "Newton-type algorithm" for unconstrained minimization.

- `optim()` provides several methods, including the Nelder-Mead simplex algorithm (default: derivative-free but slow, not discussed above), quasi-Newton (BFGS), conjugate gradient as well as simulated annealing (a stochastic global optimization method, not discussed above).

  The quasi-Newton method can also handle *box constrains* of the form $l_i \le x_i \le u_i$ (where l_i and r_i may be $-\infty$ and ∞, respectively), known as L-BFGS-B.

- `nlminb()`, now documented to be "for historical compatibility", provides code from the PORT library by Gay for unconstrained or box-constrained optimization using trust region methods.

- `constrOptim()` performs minimization under linear inequality constraints using an adaptive (logarithmic) barrier algorithm in combination with the Nelder-Mead or BFGS minimizers. Note that this is an interior point method, so there must be interior points! I.e., linear equality constraints cannot be handled.

In contributed packages:

- Several add-on packages perform linear programming.

- Package **quadprog** performs quadratic programming (strictly convex case with linear equality and/or inequality constrains).

- Package **BB** provides `BBsolve()` which itself wraps around `spg()`, which implements the spectral projected gradient method for large-scale optimization with convex constraints. Needs a function for projecting on the feasible set.

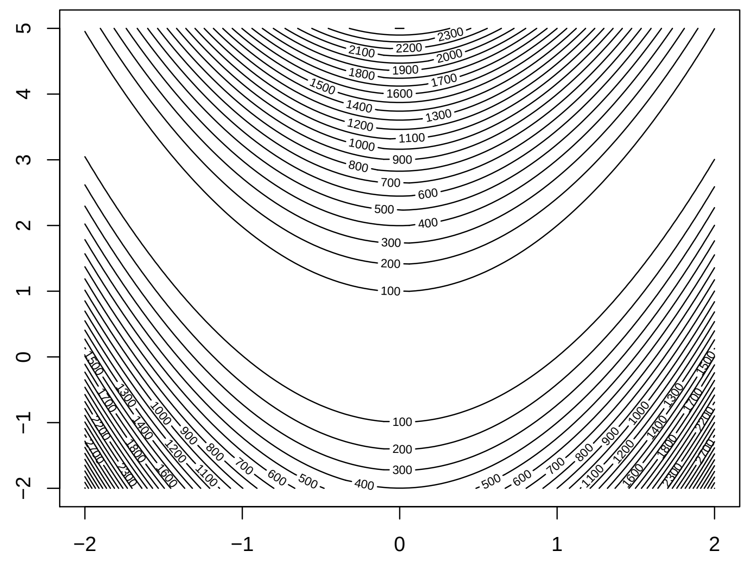
- Package **nloptr** provides an interface to NLopt (`http://ab-initio.mit.edu/nlopt`), a general purpose open source library for non-linear unconstrained and constrained optimization. One needs to consult the NLopt web page to learn about available methods (see `http://ab-initio.mit.edu/wiki/index.php/NLopt_Algorithms`).

- Package **alabama** provides `auglag()`, an Augmented Lagrange routine for smooth constrained optimization, which conveniently does not require feasible starting values.

As an example, consider the Rosenbrock banana function

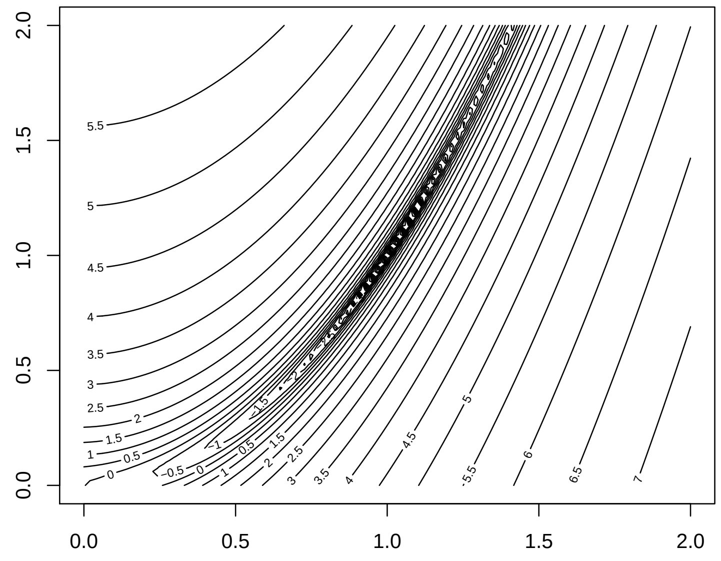
$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

```
> op <- par(mar = c(2, 2, 0, 0) + 0.1)
> x <- seq(-2, 2, length.out = 100)
> y <- seq(-2, 5, length.out = 100)
> z <- outer(x, y, function(x, y) 100 * (y - x^2)^2 + (1 - x)^2)
> contour(x, y, z, nlevels = 30)
> par(op)
```



It is clear that this has a global minimum at $x^* = (1, 1)$, even though this is hard to visualize using contour plots:

```
> op <- par(mar = c(2, 2, 0, 1) + 0.1)
> x <- seq(0, 2, length.out = 100)
> y <- seq(0, 2, length.out = 100)
> z <- outer(x, y, function(x, y) 100 * (y - x^2)^2 + (1 - x)^2)
> contour(x, y, log(z), nlevels = 30)
> par(op)
```



To use L-BFGS from NLopt (where ‘L’ stands for low-storage BFGS), we can use

```
> x0 <- c( -1.2, 1 )
> f <- function(x) {
+     100 * (x[2] - x[1]^2)^2 + (1 - x[1])^2
+ }
```

and also need to explicitly provide the gradient

$$\nabla f(x) = \begin{bmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix}$$

either as

```
> grad_f <- function(x) {
+     c( -400 * x[1] * (x[2] - x[1]^2) - 2 * (1 - x[1]),
+         200 * (x[2] - x[1]^2) )
+ }
> opts <- list(algorithm = "NLOPT_LD_LBFGS", xtol_rel = 1e-8)
> res <- nloptr::nloptr( x0, eval_f = f, eval_grad_f = grad_f, opts = opts)
> res

Call:

nloptr::nloptr(x0 = x0, eval_f = f, eval_grad_f = grad_f, opts = opts)



Minimization using NLopt version 2.4.2

NLopt solver status: 1 ( NLOPT_SUCCESS: Generic success return
value. )

Number of Iterations....: 56
Termination conditions:  xtol_rel: 1e-08
Number of inequality constraints:  0
Number of equality constraints:    0
Optimal value of objective function:  7.35727226897802e-23
Optimal value of controls: 1 1
```

or alternatively via a function returning a list of objective and gradient:

```
> fc <- function(x) list(objective = f(x), gradient = grad_f(x))
> res <- nloptr::nloptr( x0, eval_f = fc, opts = opts)
> res

Call:
nloptr::nloptr(x0 = x0, eval_f = fc, opts = opts)


Minimization using NLopt version 2.4.2
```

```
NLopt solver status: 1 ( NLOPT_SUCCESS: Generic success return
value. )

Number of Iterations....: 56
Termination conditions:  xtol_rel: 1e-08
Number of inequality constraints:  0
Number of equality constraints:    0
Optimal value of objective function:  7.35727226897802e-23
Optimal value of controls: 1 1
```

Note that in both cases using the default relative xtol will work more or less fine, but yield a warning about early stopping, and a non-success status.

Note also that constrained optimization problems are written as

$$f(x) \to \min, \qquad g(x) \le 0, \quad h(x) = 0, \quad x_L \le x \le x_U,$$

i.e., with g giving the inequality constraints and h giving the equality constraints.

A multi-dimensional generalization (e.g., `http://en.wikipedia.org/wiki/Rosenbrock_function`) is the *extended Rosenbrock function*

$$\sum_{i=1}^{2n} (100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2$$

(the sum of n uncoupled 2-d Rosenbrock functions).

```
> f <- function(x) {
+     n <- length(x) / 2
+     i2 <- 2 * seq_len(n)
+     i1 <- i2 - 1
+     sum(100 * (x[i2] - x[i1]^2)^2 + (1 - x[i1])^2)
+ }
```

If for some reason we prefer numerical gradients:

```
> grad_f <- function(x) numDeriv::grad(f, x)
```

Create a hopefully suitable starting value:

```
> set.seed(1234)
> x0 <- rnorm(50)
```

Using **nloptr**:

```
> res <- nloptr::nloptr(x0, eval_f = f, eval_grad_f = grad_f, opts = opts)
> res

Call:
```

```
nloptr::nloptr(x0 = x0, eval_f = f, eval_grad_f = grad_f, opts = opts)
```

```
Minimization using NLopt version 2.4.2

NLopt solver status: 5 ( NLOPT_MAXEVAL_REACHED: Optimization
stopped because maxeval (above) was reached. )

Number of Iterations....: 100
Termination conditions:  xtol_rel: 1e-08
Number of inequality constraints:  0
Number of equality constraints:    0
Current value of objective function:  0.0806509312051749
Current value of controls: 1.109693 1.230888 0.9133261 0.8339653 0.9589956 0.9194043
0.9758473 0.951159 0.967025 0.934232 0.9634624 0.9279126
1.013091 1.024584 0.9706215 0.9416226 0.9668534 0.9339807
0.9923276 0.9865313 0.9971337 0.9940954 1.038696 1.077937
0.9355939 0.875129 0.9738206 0.9469955 0.9771592 0.954593
0.9748558 0.9515763 0.9743042 0.948384 0.9715086 0.9433319
0.8069314 0.6533916 0.9974332 0.9950374 0.9702693 0.939623
1.040058 1.080486 0.9730678 0.9459595 0.9581794 0.9189948
0.9828571 0.9653233
```

Alternatively, using **BB**:

```
> res <- BB::BBoptim(x0, f)
```

```
iter:  0  f-value:  10824.77  pgrad:  5320.046
iter:  10  f-value:  54.40324  pgrad:  52.26724
iter:  20  f-value:  33.13841  pgrad:  2.110108
iter:  30  f-value:  26.40889  pgrad:  2.109646
iter:  40  f-value:  18.79585  pgrad:  2.063969
iter:  50  f-value:  16.09957  pgrad:  2.000223
iter:  60  f-value:  13.9462  pgrad:  2.576766
iter:  70  f-value:  12.70453  pgrad:  2.179719
iter:  80  f-value:  10.25791  pgrad:  2.176441
iter:  90  f-value:  8.034416  pgrad:  12.35637
iter:  100  f-value:  7.331731  pgrad:  2.065253
iter:  110  f-value:  6.600971  pgrad:  1.61036
iter:  120  f-value:  4.842992  pgrad:  1.517246
iter:  130  f-value:  4.273633  pgrad:  1.502324
iter:  140  f-value:  3.519309  pgrad:  17.5416
iter:  150  f-value:  2.483051  pgrad:  2.053522
iter:  160  f-value:  3.74858  pgrad:  35.27227
iter:  170  f-value:  1.095609  pgrad:  0.8585015
iter:  180  f-value:  1.002489  pgrad:  0.4675539
iter:  190  f-value:  0.9069942  pgrad:  0.4363785
iter:  200  f-value:  0.4666699  pgrad:  1.127691
```

```
iter:  210  f-value:  0.3889428  pgrad:  0.2377373
iter:  220  f-value:  0.07270047  pgrad:  2.09311
iter:  230  f-value:  0.01169285  pgrad:  0.0282269
iter:  240  f-value:  0.01127181  pgrad:  1.139276
  Successful convergence.
```

```
> res
```

```
$par
 [1] 0.9999683 0.9999366 0.9999685 0.9999369 0.9999687 0.9999373
 [7] 0.9999684 0.9999368 0.9999685 0.9999369 0.9999685 0.9999369
[13] 0.9999684 0.9999367 0.9999686 0.9999372 0.9999685 0.9999369
[19] 0.9999682 0.9999363 0.9999685 0.9999369 0.9999684 0.9999367
[25] 0.9999685 0.9999369 0.9999685 0.9999369 0.9999685 0.9999369
[31] 0.9999685 0.9999370 0.9999684 0.9999368 0.9999685 0.9999369
[37] 0.9999685 0.9999369 0.9999685 0.9999369 0.9999685 0.9999369
[43] 0.9999684 0.9999368 0.9999685 0.9999369 0.9999685 0.9999369
[49] 0.9999684 0.9999368

$value
[1] 2.488589e-08

$gradient
[1] 1.617719e-06

$fn.reduction
[1] 10824.77

$iter
[1] 242

$feval
[1] 345

$convergence
[1] 0

$message
[1] "Successful convergence"

$cpar
method      M
     2     50
```

Maybe allowing for more iterations helps with **nloptr**?

```
> opts <- c(opts, list(maxeval = 500))
> res <- nloptr::nloptr(x0, eval_f = f, eval_grad_f = grad_f, opts = opts)
> res
```

```
Call:

nloptr::nloptr(x0 = x0, eval_f = f, eval_grad_f = grad_f, opts = opts)



Minimization using NLopt version 2.4.2

NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization
stopped because xtol_rel or xtol_abs (above) was reached. )

Number of Iterations....: 192
Termination conditions:  xtol_rel: 1e-08        maxeval: 500
Number of inequality constraints:  0
Number of equality constraints:    0
Optimal value of objective function:  1.22433068008534e-15
Optimal value of controls: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

> range(res$solution - 1)

[1] -3.748205e-08  3.797778e-08
```

Yes, but the solver is still not happy ...

Using base R:

```
> res <- optim(x0, f, grad_f, method = "BFGS")
> res

$par
 [1] 0.8068806 0.6556558 1.8241596 3.3335912 1.1903147 1.4205120
 [7] 1.1651675 1.3664579 1.4551502 2.1202594 1.0272226 1.0647898
[13] 1.3237797 1.7562876 0.6764855 0.4559492 1.5509184 2.4131344
[19] 0.6042197 0.3674069 1.0699758 1.1456671 1.4462707 2.0939430
[25] 1.2056527 1.4634369 1.1228043 1.2610085 1.1532155 1.3393696
[31] 0.7031416 0.4927555 0.3739166 0.1362803 0.5352724 0.2849364
[37] 0.3895075 0.1503218 0.8940917 0.7991006 1.0741290 1.1654213
[43] 1.0657319 1.1323901 0.8864578 0.7855402 0.9506292 0.9057445
[49] 1.0886982 1.1851489

$value
[1] 3.125207

$counts
function gradient
     287      100

$convergence
```

```
[1] 1

$message
NULL
```

shows that the maximal number of iterations was reached. Trying once more:

```
> res <- optim(x0, f, grad_f, method = "BFGS",
+              control = list(maxit = 500))
> res

$par
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[31] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

$value
[1] 4.761147e-19

$counts
function gradient
     569      218

$convergence
[1] 0

$message
NULL
```

See also the CRAN task view on optimization: `https://CRAN.R-project.org/view=Optimization`.