Appendix O

Optimization (March 10, 2003)

In this chapter we review various kinds of optimization. These arise naturally as model-fitting problems.

O.1 Model Fitting

O.1.1 Nonlinear Least Squares

Least squares problems ask to find coefficients $\mathbf{a} = (a_1, \ldots, a_p)$ corresponding to a formal expression

$$y \;=\; \sum_{k=1}^{p} a_k \, x_k$$

that minimize the squared error over the input data

$$L(\mathbf{a}) \;=\; \sum_{i=1}^{n} \left(y_i \,-\, \sum_{k=1}^{p} a_k \, x_{ik} \right)^2 .$$

This is a convex optimization problem (defined below).

Nonlinear least squares generalizes this to seek parameters $\mathbf{a} = (a_1, \ldots, a_p)$ for the formal expression

$$y \;=\; \sum_{k=1}^{p} \varphi(\, a_k, \, x_k \,)$$

where $\varphi(a, x)$ can be essentially any function; here we require it only to be differentiable. Again, the parameters should minimize the squared error

$$L(\mathbf{a}) \;=\; \sum_{i=1}^{n} \left(y_i \,-\, \sum_{k=1}^{p} \varphi(\, a_k, \, x_{ik} \,) \right)^2 .$$

This is sometimes, but not always, a convex optimization problem.

O.1.2 Neural Networks

A **perceptron** is a function

$$f(x_1, \ldots, x_p) \;=\; \sigma\left(\sum_{k=1}^{p} w_k \, x_k \right)$$

where σ is a '*sigmoid*' function, such as

$$\begin{aligned} \sigma(x) &\;=\; \tanh(c\,x) \;=\; \frac{1 - e^{-2\,c\,x}}{1 + e^{-2\,c\,x}} \\ \sigma(x) &\;=\; \operatorname{logit}(c\,x) \;=\; \frac{1}{1 + e^{-c\,x}} \end{aligned}$$

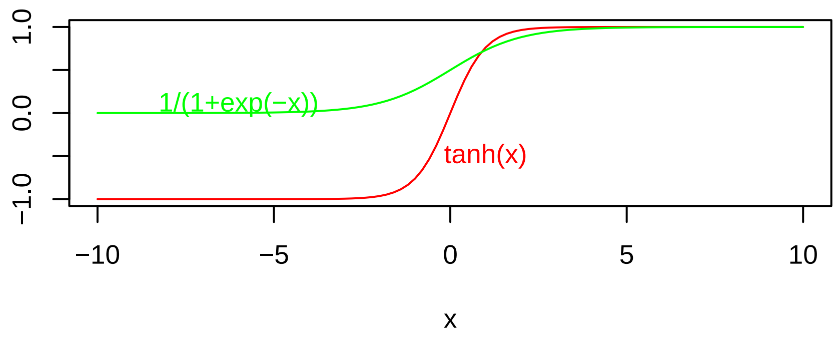

Figure O.1: Two sigmoid functions

where c is a constant, typically near 1. These functions are differentiable approximations of the signum function $\sigma(x) \simeq sign(x)$. The two functions are shown in Figure O.1.

Perceptrons can only recognize linearly-separable sets:

$$y = \begin{cases} +1 & \langle \mathbf{w}, \mathbf{x} \rangle > 0 \\ -1 & \langle \mathbf{w}, \mathbf{x} \rangle < 0 \end{cases}.$$

We can define **neural networks** as multilayer networks of perceptrons. For example, a two-layer neural network has the form

$$y = \sigma\Big(\sum_{k=1}^{q} w_k \, \sigma\big(\sum_{j=1}^{p_k} W_{kj} \, x_j \big) \Big)$$

$$L(\mathbf{w}, \mathbf{W}) = \sum_{i=1}^{n} \left(y_i - \sigma\Big(\sum_{k=1}^{q} w_k \, \sigma\big(\sum_{j=1}^{p_k} W_{kj} \, x_{ij} \big) \Big) \right)^2 .$$

$$\frac{\partial L(\mathbf{w}, \mathbf{W})}{\partial w_k} = 2 \sum_{i=1}^{n} \left(y_i - \sigma\Big(\sum_{k=1}^{q} w_k \, \sigma\big(\sum_{j=1}^{p_k} W_{kj} \, x_{ij} \big) \Big) \right) \sigma'\Big(\sum_{k=1}^{q} w_k \, \sigma\big(\sum_{j=1}^{p_k} W_{kj} \, x_{ij} \big) \Big) \; \sigma\Big(\sum_{j=1}^{p_k} W_{kj} \, x_{ij} \Big)$$

$$\frac{\partial L(\mathbf{w}, \mathbf{W})}{\partial W_{kj}} = 2 \sum_{i=1}^{n} \left(y_i - \sigma\Big(\sum_{k=1}^{q} w_k \, \sigma\big(\sum_{j=1}^{p_k} W_{kj} \, x_{ij} \big) \Big) \right) \sigma'\Big(\sum_{k=1}^{q} w_k \, \sigma\big(\sum_{j=1}^{p_k} W_{kj} \, x_{ij} \big) \Big) \; \sigma'\Big(\sum_{j=1}^{p_k} W_{kj} \, x_{ij} \Big) \; x_{ij}$$

This simple example shows how the derivatives heavily use the chain rule. Finding optimal values for the weights by gradient methods is known as **back propagation**.

O.2 Convex Optimization

A **convex program** is a statement

$$\boxed{\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{such that} & \mathbf{x} \in S \end{array}}$$

where f is a convex function and S is a convex subset of $\Re^n$. Much of life is convex, and that is good, since it turns out that convex programs can be solved effectively.

Recall that a function $f : S \to \Re$ is a **convex function** if for all real λ in $[0, 1]$, and all $\mathbf{x}$, $\mathbf{y}$ in S,

$$f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y}).$$

When the inequality is strict, we call f **strictly convex**. When $-f$ is convex, we call f **concave**.

Theorem 1 *Let f be differentiable, and S be a convex subset of $\Re^n$. Then f is convex on S iff*

$$f(\mathbf{x}) \geq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle.$$

Proof
If f is convex, then for all $0 \le \lambda \le 1$, $f(\mathbf{y} + \lambda(\mathbf{x} - \mathbf{y})) \ \le \ f(\mathbf{y}) + \lambda(f(\mathbf{x}) - f(\mathbf{y}))$, or in other words

$$\frac{f(\mathbf{y} + \lambda(\mathbf{x} - \mathbf{y})) \ - \ f(\mathbf{y})}{\lambda} \quad \le \quad f(\mathbf{x}) - f(\mathbf{y})$$

so that, letting λ tend to zero $\langle\, \nabla f(\mathbf{y}),\, \mathbf{x} - \mathbf{y} \,\rangle \ \le \ f(\mathbf{x}) - f(\mathbf{y})$, which gives the 'if' direction.

Conversely, suppose $f(\mathbf{x}) \ \ge \ f(\mathbf{y}) + \langle\, \nabla f(\mathbf{y}),\, \mathbf{x} - \mathbf{y} \,\rangle$ for all $\mathbf{x},\, \mathbf{y} \in S$. Select any two points $\mathbf{u},\ \mathbf{v} \in S$, and any $\lambda \in [0, 1]$. Because S is a convex set, $(\lambda\mathbf{u} + (1 - \lambda)\mathbf{v}) \in S$ also. Hence if $\mathbf{y} = \lambda\mathbf{u} + (1 - \lambda)\mathbf{v}$,

$$\begin{array}{rcl} f(\mathbf{u}) & \ge & f(\mathbf{y}) \ + \ \langle\, \nabla f(\mathbf{y}),\, \mathbf{u} - \mathbf{y} \,\rangle \\ f(\mathbf{v}) & \ge & f(\mathbf{y}) \ + \ \langle\, \nabla f(\mathbf{y}),\, \mathbf{v} - \mathbf{y} \,\rangle. \end{array}$$

So, combining these inequalities and using the linearity of the scalar product,

$$\lambda f(\mathbf{u}) \ + \ (1 - \lambda) f(\mathbf{v}) \ \ \ge \ \ f(\mathbf{y}) \ + \ \langle\, \nabla f(\mathbf{y}),\, (\lambda\mathbf{u} + (1 - \lambda)\mathbf{v}) - \mathbf{y} \,\rangle \ \ = \ \ f(\mathbf{y}) \ + \ 0 \ \ = \ \ f(\boldsymbol{\lambda}\mathbf{u} + \mathbf{(1 - \boldsymbol{\lambda})}\mathbf{v}),$$

which establishes the 'only if' direction.

Theorem 2 *f is convex and twice differentiable on S (a convex subset of $\Re^n$) iff the $n \times n$* **Hessian matrix**

$$H(f, \mathbf{x}) \quad = \quad (\ \frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x})\ )$$

is positive semidefinite for all $\mathbf{x}$ in S.

Proof
Expanding f in a Taylor-series around $\mathbf{y}$, we get:

$$f(\mathbf{x}) \quad = \quad f(\mathbf{y}) \ + \ \langle\, \nabla f(\mathbf{y}),\, \mathbf{x} - \mathbf{y} \,\rangle \ + \ \langle\, \mathbf{x} - \mathbf{y},\, H(f, \mathbf{z})(\mathbf{x} - \mathbf{y}) \,\rangle$$

where $\mathbf{z} = \lambda\mathbf{x} + (1 - \lambda)\mathbf{y})$ for some $0 \le \lambda \le 1$. Since the Hessian is continuous for twice-differentiable f, we can assume $\mathbf{x}$ is in the interior of S.

If $H(f, \mathbf{y})$ is not positive semidefinite, then because H is continuous there is a neighborhood of points $\mathbf{x}$ around $\mathbf{y}$ in which $\langle\, \mathbf{x} - \mathbf{y},\, H(f, \mathbf{z})(\mathbf{x} - \mathbf{y}) \,\rangle < 0$. But then in this neighborhood

$$f(\mathbf{x}) \quad < \quad f(\mathbf{y}) \ + \ \langle\, \nabla f(\mathbf{y}),\, \mathbf{x} - \mathbf{y} \,\rangle$$

which contradicts the convexity of f.

Theorem 3 *If f is convex on S (a convex subset of $\Re^n$), then the subset of values $\mathbf{x} \in S$ on which $f(\mathbf{x})$ is minimized is convex, and any local minimum of f is a global minimum.*

Proof
Recall that for each real b, the set

$$S_b \quad = \quad \{\ \mathbf{x} \in S \ \mid \ f(\mathbf{x}) \le b\ \}$$

is convex, since $f(\mathbf{x}) \ \le \ b$ is true for any convex combination of points $\mathbf{x} \ \in \ S_b$. Furthermore, if $\mathbf{x}^*$ gives a local minimum of f, but there is another value $\mathbf{x}$ such that $f(\mathbf{x}) < f(\mathbf{x}^*)$, then for any point $\lambda\mathbf{x} + (1 - \lambda)\mathbf{x}^*$ along the line between $\mathbf{x}$ and $\mathbf{x}^*$ we have

$$f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{x}^*) \quad \le \quad \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{x}^*) \quad \le \quad f(\mathbf{x}^*),$$

contradicting the local minimality of f at $\mathbf{x}^*$.

When f is differentiable, we can use its gradient to find a minimum:

Theorem 4 *Suppose f is convex and differentiable on S (a convex subset of $\Re^n$). If there is a point $\mathbf{x}^* \in S$ such that $\langle\, \nabla f(\mathbf{x}^*),\, \mathbf{x} - \mathbf{x}^* \,\rangle \ \ge \ 0$ for all $\mathbf{x} \in S$, then $f(\mathbf{x}^*)$ is a global minimum for f over S.*

Proof
Immediate, since

$$f(\mathbf{x}) \quad \ge \quad f(\mathbf{x}^*) \ + \ \langle\, \nabla f(\mathbf{x}^*),\, \mathbf{x} - \mathbf{x}^* \,\rangle \quad \ge \quad f(\mathbf{x}^*).$$

O.3 Procedures for Finding Minima

Theorem 4 shows that, for quadratic problems, we can follow gradients to a global minimum. However, it does not show how to find minima for non-quadratic problems, and it does not give a procedure for doing this.

How do we go about finding a minimum? Typically, we resort to some kind of 'hill decent' method: iteration, Newton's method, Steepest descent, Conjugate-gradient, etc.

To find the vector of coefficients **a**, we can use gradients. Defining $f(x_1, \ldots, x_p) \; = \; \sum_{k=1}^{p} \; \varphi(\, a_k,\, x_k\,)$,

$$L(\mathbf{a}) \;\;=\;\; \sum_{i=1}^{n} \; (\; y_i \;-\; f(x_1, \ldots, x_p)\;)^2 \;\;=\;\; \sum_{i=1}^{n} \left(\; y_i \;-\; \sum_{k=1}^{p} \varphi(\, a_k,\, x_{ik}\,) \right)^2 .$$

Thus

$$g_k(\mathbf{a}) \;\;=\;\; \frac{\partial L(\mathbf{a})}{\partial a_k} \;\;=\;\; -\,2 \sum_{i=1}^{n} \; (y_i \;-\; f(\mathbf{x}_i)) \;\; \frac{\partial \varphi(a_k, x_i)}{\partial a_k}.$$

So

$$\mathbf{g}(\mathbf{a}) \;\;=\;\; (g_1(\mathbf{a}), \cdots, g_p(\mathbf{a})),$$

is the **gradient**, or **Jacobian**, of the squared error $L(\mathbf{a})$ with respect to **a**.

One technique for minimizing this error is to pursue downward gradients of $L(\mathbf{a})$, hoping that doing this will lead us to a minimum. The **gradient descent** method finds a sequence of **a** values

$$\mathbf{a}^{(t+1)} \;\;=\;\; \mathbf{a}^{(t)} \;\;-\;\; c \;\; \mathbf{g}(\mathbf{a}^{(t)})$$

where c is some constant (typically near 1) determining the length of steps down these gradients.

Minimizing $L(\mathbf{a})$ requires finding **a** such that $\mathbf{g}(\mathbf{a}) = 0$. We can solve $\mathbf{g}(\mathbf{a}) = 0$ using Newton's method. Here, Newton's method takes the form

$$\mathbf{a}^{(t+1)} \;\;=\;\; \mathbf{a}^{(t)} \;\;-\;\; H(\mathbf{a}^{(t)})^{-1} \;\; \mathbf{g}(\mathbf{a}^{(t)}).$$

The derivative

$$H(\mathbf{a}) \;\;=\;\; \left(\frac{\partial g_i(\mathbf{a})}{\partial a_j} \right) \;\;=\;\; \left(\frac{\partial^2 L(\mathbf{a})}{\partial a_i \; \partial a_j} \right)$$

is the **Hessian** of L.

O.3.1 Levenberg-Marquardt

The two methods just shown — gradient descent and Newton's method — have complementary advantages and disadvantages. One is slow and safe, and the other is fast and risky. The Levenberg-Marquardt method combines both of these into a single method.

The two iterations

$$\begin{aligned} \mathbf{a}^{(t+1)} \;\;&=\;\; \mathbf{a}^{(t)} \;-\; c\, \mathbf{g}(\mathbf{a}^{(t)}) \\ \mathbf{a}^{(t+1)} \;\;&=\;\; \mathbf{a}^{(t)} \;\;-\;\; H(\mathbf{a}^{(t)})^{-1} \;\; \mathbf{g}(\mathbf{a}^{(t)}) \end{aligned}$$

can be interpolated into

$$\mathbf{a}^{(t+1)} \;\;=\;\; \mathbf{a}^{(t)} \;\;-\;\; M(\mathbf{a}^{(t)})^{-1} \;\; \mathbf{g}(\mathbf{a}^{(t)})$$

where

$$M(\mathbf{a}) \;\;=\;\; \frac{1}{2} \; (\; H(\mathbf{a}) \;+\; \lambda \; diagH(\mathbf{a}) \;)$$

and $diagH(\mathbf{a})$ is the diagonal matrix containing the principal diagonal of $H(\mathbf{a})$, and λ is a constant (something like $2/c$). For large λ this resembles gradient descent, while for $\lambda = 0$ it reduces to Newton's method.

The Levenberg-Marquardt method begins with a large value of λ, and after every step increases λ if the error increases, and decreases λ if the error decreases. It finds a local minimum near the starting point relatively quickly, but because of its 'safety' the residual error can still be large.

O.3.2 Conjugate Gradient

A limitation of gradient descent methods is that often they make many small steps, particularly in 'gentle valleys'.

Conjugate gradient methods are based on the insight that, instead of following the gradient (which will be mostly perpendicular to the valley floor), we can go in a direction that is conjugate to the gradient (and all directions in which we have gone so far).

O.3.3 Nelder-Mead

We can avoid using derivatives by generalizing on the 'golden mean' search used to find roots in one-dimensional problems.

For minimizing $g(\mathbf{a})$ over a d-dimensional space of vectors $\mathbf{a}$ we can evaluate g at all points of a **simplex** — a simple polyhedron with $d+1$ vertices — and repeatedly move downhill the vertex having the highest value of g. This algorithm, developed by Nelder and Mead, is sometimes called the **downhill simplex algorithm**.

For example, when $d = 3$ the simplex is a tetrahedron. The Nelder-Mead algorithm repeatedly tries to improve the value of g at the worst point on this tetrahedron.

The algorithm works as follows. At each iteration, the vertices $\mathbf{a}_1, \ldots, \mathbf{a}_{d+1}$ are ordered by their g-values

$$g(\mathbf{a}_1) \;\geq\; \cdots \;\geq\; g(\mathbf{a}_{d+1})$$

so that $\mathbf{a}_1$ corresponds to the largest g-value. We then define the average of

$$\bar{\mathbf{a}} \;=\; \left(\sum_{i=2}^{d+1} \mathbf{a}_i \right)$$

and replace point $\mathbf{a}_1$ by its reflection through this average

$$\mathbf{a}_1' \;=\; \bar{\mathbf{a}} \;+\; (\bar{\mathbf{a}} \;-\; \mathbf{a}_1) \;=\; 2\,\bar{\mathbf{a}} \;-\; \mathbf{a}_1.$$

This move looks like it ought to take us downhill, but of course it may not.

We then proceed as follows:

1. if $g(\mathbf{a}_1') \;<\; g(\mathbf{a}_{d+1})$, then the move will be accepted. We then also try doubling the length of this move

$$\mathbf{a}_1' \;=\; \bar{\mathbf{a}} \;+\; 2\,(\bar{\mathbf{a}} \;-\; \mathbf{a}_1) \;=\; 3\,\bar{\mathbf{a}} \;-\; 2\,\mathbf{a}_1$$

   and see if it also has this property.

2. if $g(\mathbf{a}_1') \;>\; g(\mathbf{a}_2)$, so $\mathbf{a}_1'$ is still the worst vertex, we try halving the length of this move

$$\mathbf{a}_1' \;=\; \bar{\mathbf{a}} \;+\; \frac{1}{2}\,(\bar{\mathbf{a}} \;-\; \mathbf{a}_1) \;=\; \frac{3}{2}\,\bar{\mathbf{a}} \;-\; \frac{1}{2}\,\mathbf{a}_1.$$

   (a) if $g(\mathbf{a}_1') \;<\; g(\mathbf{a}_2)$, then the move is accepted.

   (b) if $g(\mathbf{a}_1') \;>\; g(\mathbf{a}_2)$, we abandon the idea of using a reflection, and merely interpolate towards the average:

$$\mathbf{a}_1' \;=\; \bar{\mathbf{a}} \;-\; \frac{1}{2}\,(\bar{\mathbf{a}} \;-\; \mathbf{a}_1) \;=\; \frac{1}{2}\,\bar{\mathbf{a}} \;+\; \frac{1}{2}\,\mathbf{a}_1$$

   i. If this gives a better vertex, we accept it.

   ii. Otherwise, we contract the simplex – by moving all vertices toward the best one:

$$\mathbf{a}_i' \;=\; \mathbf{a}_i \;-\; \frac{1}{2}\,(\mathbf{a}_i \;-\; \mathbf{a}_{d+1}) \;=\; \frac{1}{2}\,\mathbf{a}_i \;+\; \frac{1}{2}\,\mathbf{a}_{d+1} \qquad i \;=\; 1, \ldots, d.$$

O.4 Examples

O.4.1 The Legendary Banana Function

The legendary banana function

$$f(\mathbf{x}) \;=\; 100\,(x_2 - x_1^2)^2 \;+\; (1 - x_1)^2$$

is used in a MATLAB demo because it is confusing to some optimization procedures.

```
> with(linalg):
> #
> # Rosenbrock's banana function  [MATLAB demo]
> #
> f := 100*(x2-x1^2)^2 + (1-x1)^2;
                                          2 2            2
                       f := 100 (x2 - x1 )  + (1 - x1)
```

```
> grad_f := map(simplify, vector(2, [diff(f,x1),  diff(f,x2)]) );

                  [                   3                             2]
         grad_f := [-400 x1 x2 + 400 x1  - 2 + 2 x1, 200 x2 - 200 x1 ]


> #
> # find points (x1,x2) where f is minimized
> #
> solve( {grad_f[1] = 0, grad_f[2] = 0}, {x1, x2} );

                               {x1 = 1, x2 = 1}

> subs( {x1=1, x2=1}, f );
                                       0

> subs( {x1=2, x2=4}, f );
                                       1

> subs( {x1 = -1.9, x2 = 2}, f );  #  where the MATLAB demo starts the search
                                   267.6200

> with(plots):  countourplot(f, -2..2, -1..3);

> hessian_f := matrix(2,2, (i,j) -> diff(diff(f,x.i),x.j) );

                             [       2                         ]
                hessian_f := [1200 x1  - 400 x2 + 2    -400 x1]
                             [                                 ]
                             [       -400 x1             200  ]

> det( hessian_f );
                                    2
                          80000 x1  - 80000 x2 + 400
```

From this determinant, we see that the Hessian is not always positive definite.

```
> #
> #  To find the root of grad_f(x) = g(x), we can use the Newton iteration
> #
> #        x  :=  x  -  inverse(H(x)) * g(x)
> #
> #  where  H(x) = g'(x) = hessian_f(x)  and  x = [x1,x2]:
> #
> NewtonNextStep := map(simplify,
>            evalm( vector(2,[x1,x2]) - inverse(hessian_f) &* grad_f ));

NewtonNextStep :=

    [       3                                         3          ]
    [-200 x1  + 200 x1 x2 - 1  x1 (200 x1 x2 - 200 x1  - 2 + x1)]
    [------------------------, --------------------------------]
    [        2                              2                    ]
    [ -200 x1  + 200 x2 - 1          -200 x1  + 200 x2 - 1      ]

> # convert the vector above to C
> readlib(C):
> C( NewtonNextStep, optimized );

      t1 = x1*x1;
      t2 = t1*x1;
      t3 = x1*x2;
      t6 = 1/(-200.0*t1+200.0*x2-1.0);
      NewtonNextStep[0] = (-200.0*t2+200.0*t3-1.0)*t6;
      NewtonNextStep[1] = x1*(200.0*t3-200.0*t2-2.0+x1)*t6;
```

O.4.2 Old Faithful (well — Statistically Faithful)

A classic dataset includes a set of geyser eruption times (in minutes), and waiting times between successive eruptions, produced by *Old Faithful*. The dataset looks like this:

eruption time (min.)	waiting time (min.)
3.600	79
1.800	54
3.333	74
2.283	62
4.533	85
2.883	55
⋮	⋮

Plotting the histogram of eruption times shows a bimodal distribution, shown in Figure O.2.

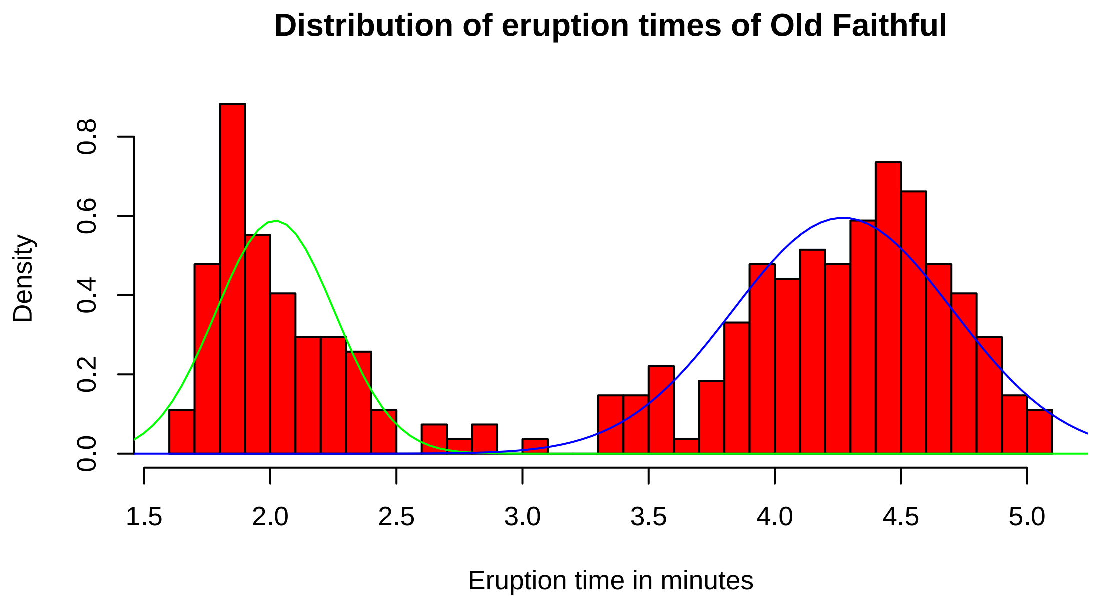

Figure O.2: Histogram of eruption times for Old Faithful, with superimposed maximum likelihood model.

We can try to fit a **mixture model** to this distribution, of the form

$$\text{Pr}[\text{ eruption time} = x \mid \boldsymbol{\theta}\ ] \quad = \quad (1-p)\ \text{normal}(x, \mu_1, \sigma_1) \quad + \quad p\ \text{normal}(x, \mu_2, \sigma_2)$$

where normal is the normal density

$$\text{normal}(x, \mu, \sigma) \quad = \quad \frac{1}{\sqrt{2\pi}\sigma}\ \exp(\,-\frac{1}{2}\ \big((x-\mu)^2/\sigma^2\big)\,).$$

This is a model with five parameters: p, μ_1, σ_1, μ_2, σ_2. If we define a parameter vector

$$\boldsymbol{\theta} \quad = \quad (p,\ \mu_1,\ \sigma_1,\ \mu_2,\ \sigma_2)$$

then we wish to find the value of $\boldsymbol{\theta}$ that best fits the data.

Undoubtedly there are many ways to find such a value, but one way is to maximize the **likelihood** of the parameters being the right ones, given the data. If the set of eruption times is $D = \{\ t_i \mid i = 1, \ldots, n\ \}$, then the

$$\textit{likelihood}(D, \boldsymbol{\theta}) \quad = \quad \text{Pr}[\ D \mid \boldsymbol{\theta}\ ] \quad = \quad \prod_{i=1}^{n} \text{Pr}[\text{ eruption time} = t_i \mid \boldsymbol{\theta}\ ]$$

and thus the log likelihood to be maximized is

$$\textit{log-likelihood}(D, \boldsymbol{\theta}) \quad = \quad \sum_{i=1}^{n} \log\,(\ \ (1-p)\ \text{normal}(t_i, \mu_1, \sigma_1) \quad + \quad p\ \text{normal}(t_i, \mu_2, \sigma_2)\ \ )\,.$$

Optimizing this objective function finds a maximum at:

$$\begin{aligned} p &= 0.652 \\ \mu_1 &= 2.02 \\ \mu_2 &= 4.27 \\ \sigma_1 &= 0.236 \\ \sigma_2 &= 0.437. \end{aligned}$$