A First Course in Statistical Programming with R

W. John Braun and Duncan J. Murdoch

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press
The Edinburgh Building, Cambridge CB2 8RU, UK

Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9780521872652

First published 2007

Printed in the United Kingdom at the University Press, Cambridge

A catalogue record for this publication is available from the British Library

ISBN 978-0-521-87265-2 hardback
ISBN 978-0-521-69424-7 paperback

7

Numerical optimization

In many areas of statistics and applied mathematics one has to solve the following problem: given a function $f(\cdot)$, which value of x makes $f(x)$ as large or as small as possible?

For example, in financial modeling $f(x)$ might be the expected return from a portfolio, with x being a vector holding the amounts invested in each of a number of possible securities. There might be constraints on x (e.g. the amount to invest must be positive, the total amount invested must be fixed, etc.)

In statistical modeling, we may want to find a set of parameters for a model which minimize the expected prediction errors for the model. Here x would be the parameters and $f(\cdot)$ would be a measure of the prediction error.

Knowing how to do minimization is sufficient. If we want to maximize $f(x)$, we simply change the sign and minimize $-f(x)$. We call both operations "numerical optimization." Use of derivatives and simple algebra often lead to the solution of such problems, but not nearly always. Because of the wide range of possibilities for functions $f(\cdot)$ and parameters x, this is a rich area of computing.

7.1 The golden section search method

The golden section search method is a simple way of finding the minimizer of a single-variable function which has a single minimum on the interval $[a, b]$.

Consider minimizing the function

$$f(x) = |x - 3.5| + (x - 2)^2$$

on the interval $[0, 5]$. This function is not differentiable at $x = 3.5$, so some care must be taken to find the minimizer. We can write an R function to evaluate $f(x)$ as follows:

```
> f <- function(x) {
+     abs(x - 3.5) + (x - 2)^2
+ }
```

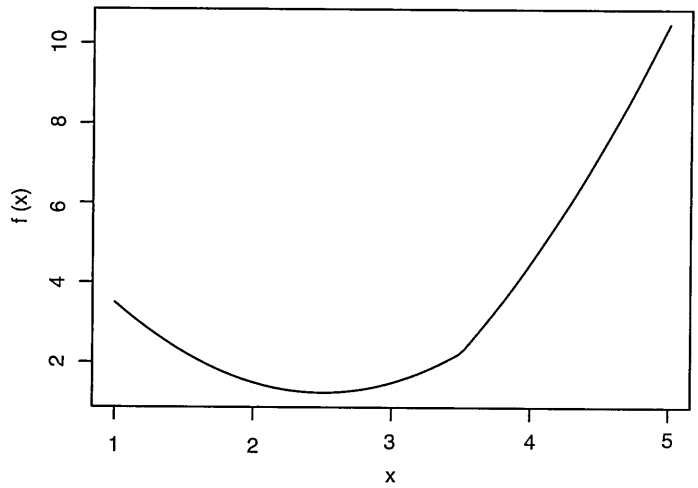

Fig. 7.1 The function $f(x) = |x - 3.5| + (x - 2)^2$.

To check that this function has a single minimum in the interval we use the `curve()` function to plot it:

```
> curve(f, from = 1, to = 5)
```

The curve is displayed in Figure 7.1, where we can see that the minimizer is located near $x = 2.5$.

The golden section search method is an iterative method, which may be outlined as follows:

1. Start with the interval $[a, b]$, known to contain the minimizer.
2. Repeatedly shrink it, finding smaller and smaller intervals $[a', b']$ which still contain the minimizer.
3. Stop when $b' - a'$ is small enough, i.e. when the interval length is less than a pre-set tolerance.

When the search stops, the midpoint of the final interval will serve as a good approximation to the true minimizer, with a maximum error of $(b' - a')/2$.

The shrinkage step 2 begins by evaluating the function at two points $x_1 < x_2$ in the interior of the interval $[a, b]$. (How the points are chosen will be described below.) Because we have assumed that there is a unique minimum, we know that if $f(x_1) > f(x_2)$, then the minimum must lie to the right of x_1, i.e. in the interval $[a', b'] = [x_1, b]$. If $f(x_1) < f(x_2)$, the minimum must lie in $[a', b'] = [a, x_2]$ (see Figure 7.2). (What if the values are exactly equal? We will consider that case later.) Then new values of $x_1, f(x_1), x_2$, and $f(x_2)$ are computed, and the method is repeated until the tolerance criterion is satisfied.

The choice of the points between a and b makes use of properties of the golden ratio $\phi = (\sqrt{5}+1)/2$. The golden ratio (which we saw in Chapter 3 in the context of Fibonacci numbers) has a number of interesting algebraic properties. We make use of the fact that $1/\phi = \phi - 1$ and $1/\phi^2 = 1 - 1/\phi$ in the following. (Some authors call the value $\Phi = 1/\phi$ the "silver ratio," but we'll stick with ϕ in our formulas.)

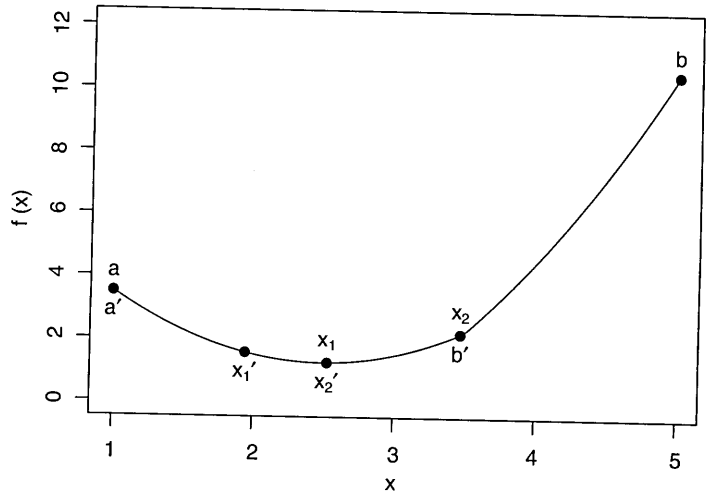

Fig. 7.2 One iteration of the golden section search, applied to the test function $f(x) = |x - 3.5| + (x - 2)^2$.

We locate the interior points at $x_1 = b-(b-a)/\phi$ and $x_2 = a+(b-a)/\phi$. The reason for this choice is as follows. After one iteration of the search, it is possible that we will throw away a and replace it with $a' = x_1$. Then the new value to use as x_1 will be

$$\begin{aligned} x_1' &= b - (b - a')/\phi \\ &= b - (b - x_1)/\phi \\ &= b - (b - a)/\phi^2 \\ &= a + (b - a)/\phi \\ &= x_2, \end{aligned}$$

i.e. we can re-use a point we already have, we do not need a new calculation to find it, and we don't need a new evaluation of $f(x_1')$, we can re-use $f(x_2)$. Similarly, if we update to $b' = x_2$, then $x_2' = x_1$, and we can re-use that point.

We put this together into the following R function:

```
> golden <- function (f, a, b, tol = 0.0000001)
+ {
+     ratio <- 2 / (sqrt(5) + 1)
+     x1 <- b - ratio * (b - a)
+     x2 <- a + ratio * (b - a)
+
+     f1 <- f(x1)
+     f2 <- f(x2)
+
+     while(abs(b - a) > tol) {
+
+         if (f2 > f1) {
+             b <- x2
+             x2 <- x1
```

```
+             f2 <- f1
+             x1 <- b - ratio * (b - a)
+             f1 <- f(x1)
+         } else {
+             a <- x1
+             x1 <- x2
+             f1 <- f2
+             x2 <- a + ratio * (b - a)
+             f2 <- f(x2)
+         }
+     }
+     return((a + b) / 2)
+ }
```

We test and see that `golden()` works, at least on one function:

```
> golden(f, 1, 5)
[1] 2.5
```

Exercises

1 Apply the golden section minimization technique to the following functions:

(a) $f(x) = |x - 3.5| + |x - 2| + |x - 1|$

(b) $f(x) = |x - 3.2| + |x - 3.5| + |x - 2| + |x - 1|$.

For the second function, check the graph to see that the minimizer is not unique. Show that the minimizer found by `golden()` depends on the initial interval supplied to the function.

2 For an odd number of data values $x_1, x_2, \ldots, x_n$, the minimizer of the function

$$f(x) = \sum_{i=1}^{n} |x - x_i|$$

is the sample median of the data values. (Exercise 1(a) is an example of this.) Verify this result for the following data sets:

(a) 3, 7, 9, 12, 15

(b) 3, 7, 9, 12, 15, 18, 21.

Describe, in words, what happens when the number of observations is even.

3 Write a function that would find the maximizer of a function using the golden section search.

7.2 | Newton–Raphson

If the function to be minimized has two continuous derivatives and we know how to evaluate them, we can make use of this information to give a faster algorithm than the golden section search.

We want to find a minimizer x^* of the function $f(x)$ in the interval $[a, b]$. Provided the minimizer is not at a or b, x^* will satisfy $f'(x^*) = 0$. This is a necessary condition for x^* to be a minimizer of $f(x)$, but it is not sufficient: we must check that x^* actually minimizes $f(x)$. Other solutions of $f'(x^*) = 0$ are maximizers and points of inflection. One sufficient condition to guarantee that our solution is a minimum is to check that $f''(x^*) > 0$.

Now, if we have a guess x_0 at a minimizer, we use the fact that $f''(x)$ is the slope of $f'(x)$ and approximate $f'(x)$ using a Taylor series approximation:

$$f'(x) \approx f'(x_0) + (x - x_0)f''(x_0).$$

Finding a zero of the right-hand side should give us an approximate solution to $f'(x^*) = 0$.

We implement this idea as follows, using the Newton–Raphson algorithm to approximate a solution to $f'(x^*) = 0$. Start with an initial guess x_0, and compute an improved guess using the solution

$$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}.$$

This gives a new guess at the minimizer. Then use x_1 in place of x_0, to obtain a new update x_2. Continue with iterations of the form

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}.$$

This iteration stops when $f'(x_n)$ is close enough to 0. Usually, we set a tolerance ε and stop when $|f'(x_n)| < \varepsilon$.

It can be shown that the Newton–Raphson method is guaranteed to converge to a local minimizer, provided the starting value x_0 is close enough to the minimizer. As with other numerical optimization techniques, where there are multiple minimizers, Newton–Raphson won't necessarily find the best one. However, when $f''(x) > 0$ everywhere, there will be only one minimizer.

In actual practice, implementation of Newton–Raphson can be tricky. We may have $f''(x_n) = 0$, in which case the function looks locally like a straight line, with no solution to the Taylor series approximation to $f'(x^*) = 0$. In this case a simple strategy is to move a small step in the direction which decreases the function value, based only on $f'(x_n)$.

In other cases where x_n is too far from the true minimizer, the Taylor approximation may be so inaccurate that $f(x_{n+1})$ is actually larger than $f(x_n)$. When this happens one may replace x_{n+1} with $(x_{n+1} + x_n)/2$ (or some other value between x_n and x_{n+1}) in the hope that a smaller step will produce better results.

Finally, there is always the possibility that the code to calculate $f'(x)$ or $f''(x)$ may contain bugs: it is usually worthwhile to do careful checks to make sure this is not the case.

Example 7.1

We wish to find the minimizer of $f(x) = e^{-x} + x^4$. By inspection, we can guess that the minimizer is somewhere to the right of zero, because e^{-x} is a decreasing function, and x^4 has a minimum at zero. We start by plotting the function to find an initial guess (Figure 7.3):

```
> f <- function(x) exp(-x) + x^4
> curve(f, from=-1, to=4)
```

From the figure, we can see that the minimizer is somewhere near $x_0 = 0.5$; we will use that as our starting value. Because of the difficulties mentioned above, we will not attempt to write a general Newton–Raphson implementation. Instead, we will simply evaluate several updates to see whether it converges or not.

```
> f <- function(x) exp(-x) + x^4
> fprime <- function(x) -exp(-x) + 4 * x^3
> fprimeprime <- function(x) exp(-x) + 12 * x^2
> x <- c(0.5, rep(NA, 6))
> fval <- rep(NA, 7)
> fprimeval <- rep(NA, 7)
> fprimeprimeval <- rep(NA, 7)
> for (i in 1:6) {
+     fval[i] <- f(x[i])
+     fprimeval[i] <- fprime(x[i])
+     fprimeprimeval[i] <- fprimeprime(x[i])
+     x[i + 1] <- x[i] - fprimeval[i] / fprimeprimeval[i]
+ }
> data.frame(x, fval, fprimeval, fprimeprimeval)
          x      fval     fprimeval fprimeprimeval
1 0.5000000 0.6690307 -1.065307e-01       3.606531
2 0.5295383 0.6675070  5.076129e-03       3.953806
3 0.5282544 0.6675038  9.980020e-06       3.938266
4 0.5282519 0.6675038  3.881429e-11       3.938235
5 0.5282519 0.6675038  0.000000e+00       3.938235
6 0.5282519 0.6675038  0.000000e+00       3.938235
7 0.5282519        NA            NA             NA
```

We see that convergence was very rapid, with the derivative numerically equal to zero by the fourth update. The second derivative is positive there, confirming that this is a local minimum. In fact, since $f''(x) = e^{-x} + 12x^2$, the second derivative is positive everywhere, and we can be sure that this is a global minimum.

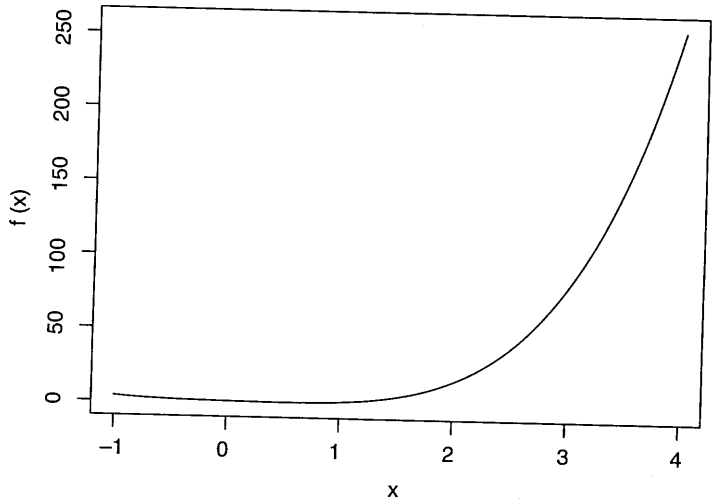

Fig. 7.3 The function $f(x) = e^{-x} + x^4$.

7.3 The Nelder–Mead simplex method

In the previous sections, we have talked about two different methods for optimizing a function of one variable. However, when a function depends on multiple inputs, optimization becomes much harder. It is hard even to visualize the function once it depends on more than two inputs.

The Nelder–Mead simplex algorithm is one method for optimization of a function of several variables. In p dimensions, it starts with $p + 1$ points $x_1, \ldots, x_{p+1}$, arranged so that when considered as vertices of a p-dimensional solid (a "simplex"), they enclose a nonzero volume. For example, in two dimensions the three points would not be allowed to all lie on one line so they would form a triangle, and in three dimensions the four points would form a proper tetrahedron.

The points are labeled in order from smallest to largest values of $f(x_i)$, so that $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{p+1})$. The idea is that to minimize $f(x)$, we would like to drop x_{p+1} and replace it with a point that gives a smaller value. We do this by calculating several proposed points z_i from the existing points. There are four kinds of proposals, illustrated in Figure 7.4 in two dimensions. The first three refer to the midpoint of $x_1, \ldots, x_p$ which we calculate as $x_{\text{mid}} = (x_1 + \cdots + x_p)/p$.

1 Reflection: reflect x_{p+1} through x_{mid} to z_1.
2 Reflection and expansion: reflect x_{p+1} through x_{mid}, and double its distance, giving z_2.
3 Contraction 1: contract x_{p+1} halfway towards x_{mid} to give z_3.
4 Contraction 2: contract all points halfway towards x_1, giving $z_4, \ldots, z_{p+3}$.

We consider each of these choices of simplex in order, based on the values of $f(z_i)$. It is helpful to consider the line shown in Figure 7.5 as you

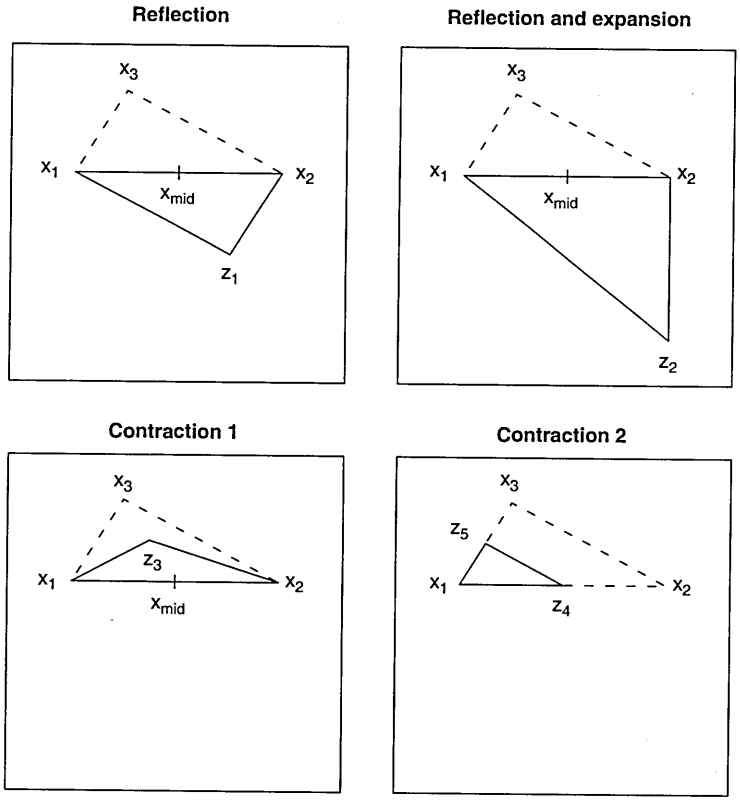

Fig. 7.4 The four types of proposals of the Nelder–Mead algorithm, illustrated in two dimensions.

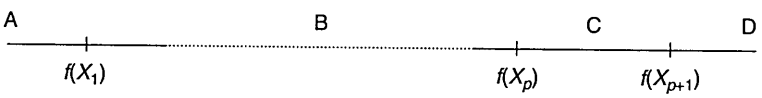

Fig. 7.5 $f(z_1)$ will fall in region A, B, C, or D in the Nelder–Mead algorithm.

read through the following pseudocode outline of the decision process for one update of the simplex:

```
Initialization:
    Place the initial points in a matrix x, so that point i is in
      x[i,]
    For i in 1:(p + 1) calculate f(x[i,])
    Relabel the points so that
      f(x[1,]) <= f(x[2,]) <= ... <= f(x[p + 1,])
    Calculate the midpoint xmid = (x[1,] + x[2,] + ... + x[p,]) / p

Trials:
    Calculate z1 by reflection:  z1 <- xmid - (x[p + 1,] - xmid)
    If f(z1) < f(x[1,]) {                          # Region A
        Calculate z2 by reflection and expansion:
          z2 <- xmid - 2 * (x[p + 1,] - xmid)
        If f(z2) < f(z1) return(z2)
        else return(z1)
    } else {
```

```
        If f(z1) < f(x[p,]) return(z1)          # Region B
        If f(z1) < f(x[p + 1,]) {
            Swap z1 with x[p + 1,]              # Region C
        }
    }

    At this point we know f(z1) is in region D.

    Try contraction 1, giving z3.
    If f(z3) < f(x[p + 1,]) return(z3)          # Region A, B, or C

    At this point nothing has worked, so we use contraction 2 to move
      everything towards x[1,]
```

Example 7.2
In this example we try to minimize the function

```
> f <- function(x, y) ((x - y)^2 + (x - 2)^2 + (y - 3)^4) / 10
```

using the Nelder–Mead algorithm. We start by drawing a contour plot of the function, in order to get approximate starting values. After some experimentation, we obtain the plot shown in Figure 7.6 using the following code:

```
> x <- seq(0, 5, len=20)
> y <- seq(0, 5, len=20)
> z <- outer(x, y, f)
> contour(x, y, z)
```

We implemented the Nelder–Mead update algorithm in an R function with header `neldermead(x, f)`, where `x` is our matrix in the pseudocode, and `f` is the function. The output of `neldermead(x, f)` is an updated copy of the matrix `x`. The following log shows the output of nine Nelder–Mead updates. Figure 7.7 shows the steps the algorithm took in this demonstration.

```
> x <- matrix(c(0, 0, 2, 0, 2, 0), 3, 2)
> polygon(x)
> for (i in 1:9) {
+   cat(i,":") +   x <- neldermead(x,f) +   polygon(x) +
text(rbind(apply(x, 2, mean)), labels=i) + }
1 :Accepted reflection, f(z1)= 3.3
2 :Swap z1 and x3
Accepted contraction 1, f(z3)= 3.25
3 :Accepted reflection and expansion, f(z2)= 0.31875
4 :Accepted reflection, f(z1)= 0.21875
5 :Accepted contraction 1, f(z3)= 0.21875
6 :Accepted contraction 1, f(z3)= 0.1
```

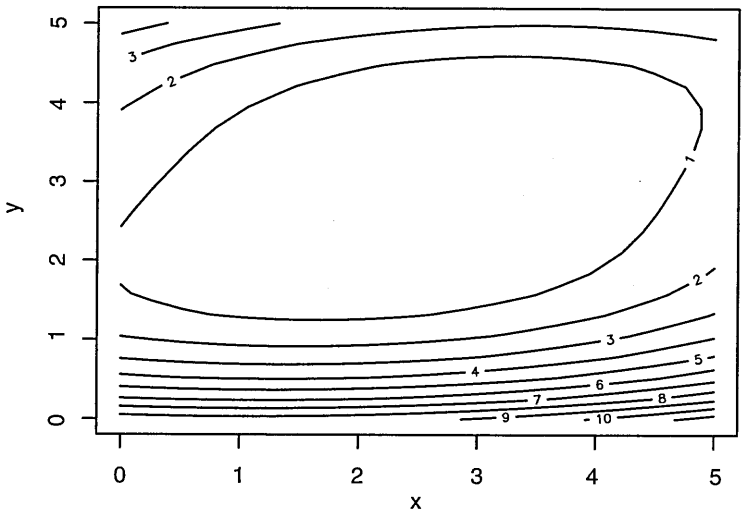

Fig. 7.6 Contour plot of $f(x,y) = [(x-y)^2+(x-2)^2+(y-3)^4]/10$.

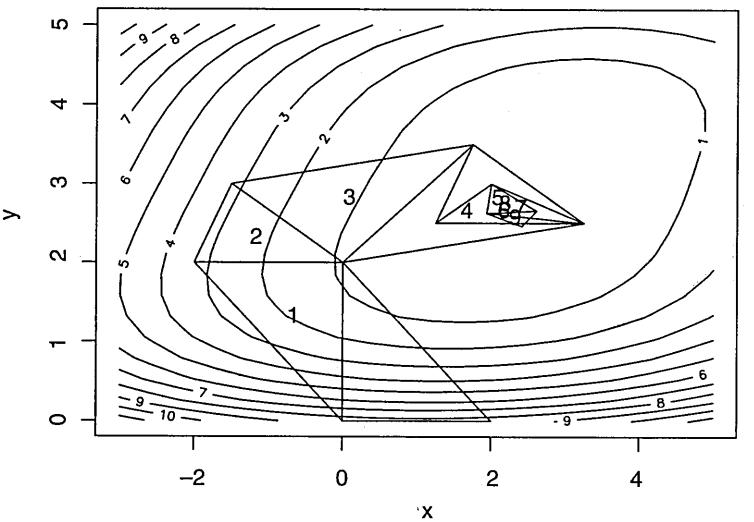

Fig. 7.7 Nine Nelder–Mead updates for $f(x,y) = [(x-y)^2+(x-2)^2+(y-3)^4]/10$.

```
7 :Accepted contraction 1, f(z3)= 0.04963379
8 :Accepted contraction 1, f(z3)= 0.03874979
9 :Swap z1 and x3
Accepted contraction 1, f(z3)= 0.02552485
> x
         [,1]     [,2]
[1,] 2.609375 2.656250
[2,] 1.937500 2.625000
[3,] 2.410156 2.460938
```

At the end of these nine steps, we see that x should be around 1.9–2.6, and y should be around 2.4–2.7. A further 50 updates narrows these down to the true minimum at $(x, y) = (2.25, 2.5)$.

7.4 Built-in functions

There are several general purpose optimization functions in R.

For one-dimensional optimization, the `optimize()` function performs a variation on the golden section search we described earlier. There are also multi-dimensional optimizers. The first of these is the `optim()` function. `optim()` is a general purpose wrapper for several different optimization methods, including Nelder–Mead, variations on Newton–Raphson, and others that we haven't discussed.

Syntax

```
optim(par, fn, ...)
```

The `par` parameter to `optim()` gives starting values for the parameters. Besides telling `optim()` where to begin, these indicate how many parameters will vary in its calls to `fn`, the second parameter. `fn` is an R function which evaluates the function to be minimized. Its first argument should be a vector of the same length as `par`; `optim()` will call it repeatedly, varying the value of this parameter, in order to find the minimum. It should return a scalar value. The `optim()` function has a number of optional parameters described on its help page. Besides those, the optional parameters in the `...` list could include additional parameters to pass to `fn`.

There are other functions in R for general function optimization: `nlm()` and `nlminb()`. In most cases `optim()` is preferred because it offers more flexibility, but there may be instances where one of the others performs better. The `constrOptim()` function is aimed at cases where there are linear inequalities expressing constraints on the parameters.

Exercises

1 Use the `optimize()` function to minimize the following functions:
   (a) $f(x) = |x - 3.5| + |x - 2| + |x - 1|$
   (b) $f(x) = |x - 3.2| + |x - 3.5| + |x - 2| + |x - 1|$.
2 Use `nlm()` and `optim()` to minimize the function

$$f(a,b) = (a-1) + 3.2/b + 3\log(\Gamma(a)) + 3a\log(b).$$

   Note that $\Gamma(a)$ is the gamma function which can be evaluated in R using `gamma(a)`.
3 Re-do the previous exercise using `nlminb()`, noting that a and b should be restricted to being nonnegative.

7.5 Linear programming

We often need to minimize (or maximize) a function subject to constraints. When the function is linear and the constraints can be expressed as linear equations or inequalities, the problem is called a *linear programming* problem.

The so-called standard form for the minimization problem in linear programming is

$$\min_{x_1,x_2,\ldots,x_k} C(x) = c_1x_1 + \cdots + c_kx_k,$$

subject to the *constraints*

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1k}x_k &\geq b_1 \\ a_{21}x_1 + \cdots + a_{2k}x_k &\geq b_2 \\ &\cdots \\ a_{m1}x_1 + \cdots + a_{mk}x_k &\geq b_m, \end{aligned}$$

and the *nonnegativity conditions* $x_1 \geq 0, \ldots, x_k \geq 0$.

The idea is to find values of the *decision variables* $x_1, x_2, \ldots, x_n$ which minimize the *objective function* $C(x)$, subject to the constraints and nonnegativity conditions.

Example 7.3

A company has developed two procedures for reducing sulfur dioxide and carbon dioxide emissions from its factory. The first procedure reduces equal amounts of each gas at a per unit cost of \$5. The second procedure reduces the same amount of sulfur dioxide as the first method, but reduces twice as much carbon dioxide gas; the per unit cost of this method is \$8.

The company is required to reduce sulfur dioxide emissions by 2 million units and carbon dioxide emissions by 3 million units. What combination of the two emission procedures will meet this requirement at minimum cost?

Let x_1 denote the amount of the first procedure to be used, and let x_2 denote the amount of the second procedure to be used. For convenience, we will let these amounts be expressed in millions of units.

Then the cost (in millions of dollars) can be expressed as

$$C = 5x_1 + 8x_2.$$

Since both methods reduce sulfur dioxide emissions at the same rate, the number of units of sulfur dioxide reduced will then be

$$x_1 + x_2.$$

Noting that there is a requirement to reduce the sulfur dioxide amount by 2 million units, we have the constraint

$$x_1 + x_2 \geq 2.$$

The carbon dioxide reduction requirement is 3 million units, and the second method reduces carbon dioxide twice as fast as the first method, so we have the second constraint

$$x_1 + 2x_2 \geq 3.$$

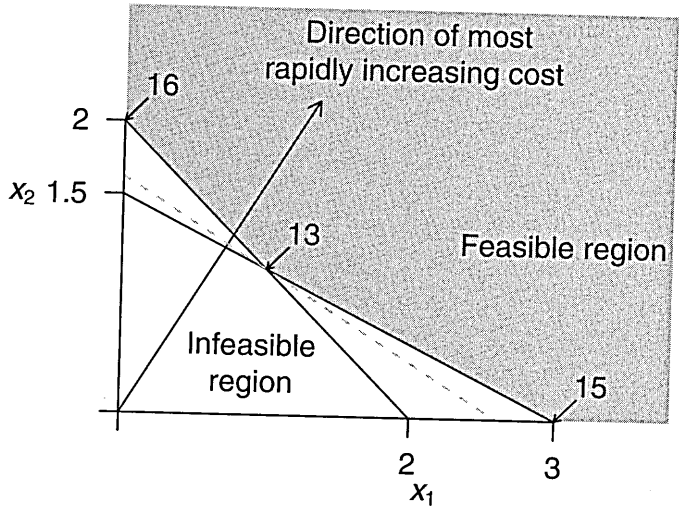

Fig. 7.8 A graphical interpretation of the pollution emission linear programming example. The grey region corresponds to values of x_1 and x_2 which satisfy all of the constraints. The dashed grey line corresponds to values of x_1 and x_2 which give the minimum cost (13); note that this line intersects the feasible region at exactly one point – the optimal solution to the problem (1, 1).

Finally, we note that x_1 and x_2 must be nonnegative, since we cannot use negative amounts of either procedure. Thus, we obtain the linear programming problem:

$$\min C = 5x_1 + 8x_2,$$

subject to the constraints

$$x_1 + x_2 \geq 2$$

$$x_1 + 2x_2 \geq 3,$$

and

$$x_1, x_2 \geq 0.$$

These relations are graphed in Figure 7.8. The region shaded in grey is the *feasible region*; this is the set of all possible (x_1, x_2) combinations which satisfy the constraints. The unshaded area contains those combinations of values where the constraints are violated.

The gradient of the function $C(x)$ is $(5, 8)$, so this vector gives the direction of most rapid increase for that function. The level sets or contours of this function are perpendicular to this vector. One of the level sets is indicated as a dashed line in Figure 7.8. The solution of the minimization problem lies at the intersection of the first contour which intersects the feasible region. If this happens at a single point, we have a *unique* minimizer. In this example, this intersection is located at the point $(1, 1)$.

It can be shown that the only possible minimizers for such linear programming problems must be at the intersections of the constraint boundaries, as in the above example. The points of intersection of the constraints are called *basic solutions*. If these intersection points lie in the feasible region, they are called *basic feasible solutions*. If there is at least one basic

feasible solution, then one of them will be an *optimal solution*. In the above example, the point (1, 1) is the optimal solution.

7.5.1 Solving linear programming problems in R

There is more than one linear programming function available in R, but we believe the `lp()` function in the `lpSolve` package may be the most stable version currently available. It is based on the *revised simplex method*; this method intelligently tests a number of extreme points of the feasible region to see whether they are optimal.

The `lp()` function has a number of parameters; the following are needed to solve minimization problems like the one in the earlier example:

- `objective.in` – the vector of coefficients of the objective function
- `const.mat` – a matrix containing the coefficients of the decision variables in the left-hand side of the constraints; each row corresponds to a constraint
- `const.dir` – a character vector indicating the direction of the constraint inequalities; some of the possible entries are `>=`, `==` and `<=`
- `const.rhs` – a vector containing the constants given on the right-hand side of the constraints.

Example 7.4

To solve the minimization problem set out in Example 7.3, type

```
> library(lpSolve)
> eg.lp <- lp(objective.in=c(5, 8), const.mat=matrix(c(1, 1, 1, 2),
+             nrow=2), const.rhs=c(2, 3), const.dir=c(">=", ">="))
> eg.lp
Success: the objective function is 13
> eg.lp$solution
[1] 1 1
```

The output tells us that the minimizer is at $x_1 = 1, x_2 = 1$, and the minimum value of the objective function is 13.

7.5.2 Maximization and other kinds of constraints

The `lp()` function can handle maximization problems with the use of the `direction="max"` parameter. Furthermore, the `const.dir` parameter allows for different types of inequalities.

Example 7.5

We will solve the problem:

$$\max C = 5x_1 + 8x_2,$$

subject to the constraints

$$x_1 + x_2 \leq 2$$
$$x_1 + 2x_2 = 3,$$

and

$$x_1, x_2 \geq 0.$$

In R, this can be coded as

```
> eg.lp <- lp(objective.in=c(5, 8),
+             const.mat=matrix(c(1, 1, 1, 2), nrow=2),
+             const.rhs=c(2, 3),
+             const.dir=c("<=", "="), direction="max")
> eg.lp$solution
[1] 1 1
```

The solution is (1, 1), giving a maximum value of 13.

7.5.3 Special situations

Multiple optima

It sometimes happens that there are multiple solutions for a linear programming problem.

Example 7.6

A slight modification of the pollution emission example (Example 7.3) is

$$\min C = 4x_1 + 8x_2,$$

subject to the constraints

$$x_1 + x_2 \geq 2$$

$$x_1 + 2x_2 \geq 3,$$

and

$$x_1, x_2 \geq 0.$$

This problem has a solution at (1, 1) as well as at (3, 0). All points on the line joining these two points are solutions as well. Figure 7.9 shows this graphically.

The `lp()` function does not alert the user to the existence of multiple minima. In fact, the output from this function for the modified pollution emission example is the solution $x_1 = 3, x_2 = 0$.

Degeneracy

For a problem with m decision variables, degeneracy arises when more than m constraint boundaries intersect at a single point. This situation is quite rare, but it has potential to cause difficulties for the simplex method, so it is important to be aware of this condition. In very rare circumstances,

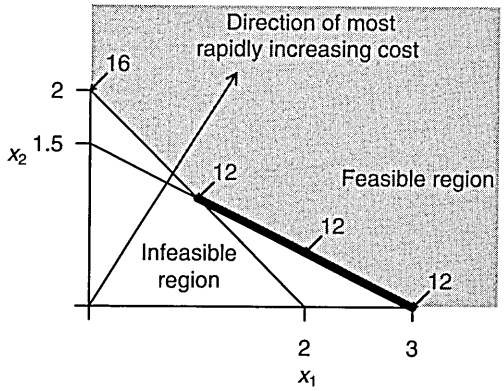

Fig. 7.9 A plot of the gradient of the objective function and the constraint boundaries for Example 7.6. The points on the heavy black segment are all optimal for this problem.

degeneracy can prevent the method from converging to the optimal solution; most of the time, however, there is little to worry about.

Example 7.7

The following problem has a point of degeneracy which is not at the optimum; however, the `lp()` function still finds the optimum without difficulty.

$$\min C = 3x_1 + x_2,$$

subject to the constraints

$$x_1 + x_2 \geq 2$$

$$x_1 + 2x_2 \geq 3$$

$$x_1 + 3x_2 \geq 4$$

$$4x_1 + x_2 \geq 4,$$

and

$$x_1, x_2 \geq 0.$$

The constraint boundaries are plotted in Figure 7.10.

This problem can be solved easily:

```
> degen.lp <- lp(objective.in=c(3, 1),
+                const.mat=matrix(c(1, 1, 1, 4, 1, 2, 3, 1), nrow=4),
+                const.rhs=c(2, 3, 4, 4), const.dir=rep(">=", 4))
> degen.lp

Success: the objective function is 3.333333
> degen.lp$solution
[1] 0.6666667 1.3333333
```

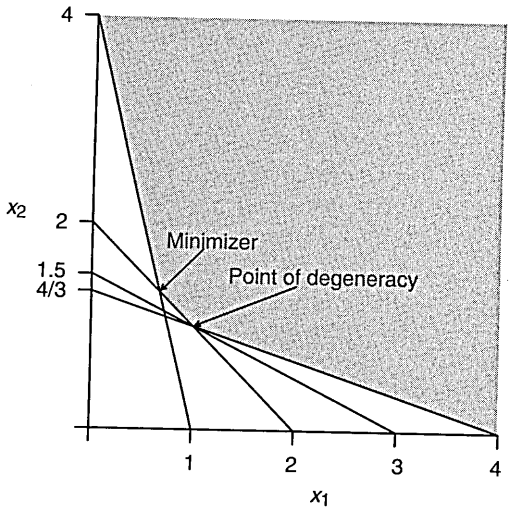

Fig. 7.10 A plot of four constraint boundaries, one of which is redundant, leading to degeneracy. The feasible region is shaded.

Infeasibility

Infeasibility is a more common problem. When the constraints cannot simultaneously be satisfied there is no feasible region. Then no feasible solution exists.

Example 7.8

In the following example, it is obvious that the constraints cannot simultaneously be satisfied.

$$\min C = 5x_1 + 8x_2,$$

subject to the constraints

$$x_1 + x_2 \geq 2$$

$$x_1 + x_2 \leq 1,$$

and

$$x_1, x_2 \geq 0.$$

Here is the output from the `lp()` function:

```
> eg.lp <- lp(objective.in=c(5, 8),
+             const.mat=matrix(c(1, 1, 1, 1), nrow=2),
+             const.rhs=c(2, 1), const.dir=c(">=", "<="))
> eg.lp
Error: no feasible solution found
```

Unboundedness

In rare instances, the constraints and objective function give rise to an unbounded solution.

Example 7.9
A trivial example of unboundedness arises when solving the problem

$$\max C = 5x_1 + 8x_2,$$

subject to the constraints

$$x_1 + x_2 \geq 2$$
$$x_1 + 2x_2 \geq 3,$$

and

$$x_1, x_2 \geq 0.$$

The feasible region for this problem is the same as for Example 7.3 and is plotted in Figure 7.8. However, instead of trying to minimize the objective function, we are now maximizing, so we follow the direction of increasing the objective function this time. We can make the objective function as large as we wish, by taking x_1 and x_2 arbitrarily large.

Here is what happens when `lp()` is applied to this problem:

```
> eg.lp <- lp(objective.in=c(5, 8),
+             const.mat=matrix(c(1, 1, 1, 2), nrow=2),
+             const.rhs=c(2, 3), const.dir=c(">=", ">="),
+             direction="max")
> eg.lp
Error: status 3
```

The condition of unboundedness will most often arise when constraints and/or the objective function have not been formulated correctly.

7.5.4 Unrestricted variables

Sometimes a decision variable is not restricted to being nonnegative. The `lp()` function is not set up to handle this case directly. However, a simple device gets around this difficulty.

If x is unrestricted in sign, then x can be written as $x_1 - x_2$, where $x_1 \geq 0$ and $x_2 \geq 0$. This means that every unrestricted variable in a linear programming problem can be replaced by the difference of two nonnegative variables.

Example 7.10
We will solve the problem:

$$\min C = x_1 + 10x_2,$$

subject to the constraints

$$x_1 + x_2 \geq 2$$
$$x_1 - x_2 \leq 3,$$

and

$$x_1 \geq 0.$$

Noting that x_2 is unrestricted in sign, we set $x_2 = x_3 - x_4$ for nonnegative x_3 and x_4. Plugging these new variables into the problem gives

$$\min C = x_1 + 10x_3 - 10x_4,$$

subject to the constraints

$$x_1 + x_3 - x_4 \geq 2$$

$$x_1 - x_3 + x_4 \leq 3,$$

and

$$x_1 \geq 0, x_3 \geq 0, x_4 \geq 0.$$

Converting this to R code, we have

```
> unres.lp <- lp(objective.in=c(1, 10, -10),
+       const.mat=matrix(c(1, 1, 1, -1, -1, 1), nrow=2),
+       const.rhs=c(2, 3), const.dir=c(">=", "<="))
> unres.lp
Success: the objective function is -2.5
> unres.lp$solution
[1] 2.5 0.0 0.5
```

The solution is given by $x_1 = 2.5$ and $x_2 = x_3 - x_4 = -0.5$.

7.5.5 Integer programming

Decision variables are often restricted to be integers. For example, we might want to minimize the cost of shipping a product by using one, two, or three different trucks. It is not possible to use a fractional number of trucks, so the number of trucks must be integer-valued.

Problems involving integer-valued decision variables are called *integer programming* problems. Simple rounding of a non-integer solution to the nearest integer is *not* good practice; the result of such rounding can be a solution which is quite far from the optimal solution.

The `lp()` function has a facility to handle integer-valued variables using a technique called the *branch and bound algorithm*. The `int.vec` argument can be used to indicate which variables have integer values.

Example 7.11

Find nonnegative x_1, x_2, x_3, and x_4 to minimize

$$C(x) = 2x_1 + 3x_2 + 4x_3 - x_4,$$

subject to the constraints

$$x_1 + 2x_2 \geq 9$$

$$3x_2 + x_3 \geq 9,$$

and

$$x_2 + x_4 \leq 10.$$

Furthermore, x_2 and x_4 can only take integer values. To set up and solve this problem in R, type

```
> integ.lp <- lp(objective.in=c(2, 3, 4, -1),
+     const.mat=matrix(c(1, 0, 0, 2, 3, 1, 0, 1, 0, 0, 0, 1), nrow=3),
+     const.dir=c(">=", ">=", "<="), const.rhs=c(9, 9, 10),
+     int.vec=c(2, 4))
> integ.lp
Success: the objective function is 8
> integ.lp$solution
[1] 1 4 0 6
```

Thus, the best solution when x_2 and x_4 are integer-valued is $x_1 = 1, x_2 = 4$, $x_3 = 0$, and $x_4 = 6$.

Here is what happens when the integer variables are ignored:

```
> wrong.lp <- lp(objective.in=c(2, 3, 4, -1),
+     const.mat=matrix(c(1, 0, 0, 2, 3, 1, 0, 1, 0, 0, 0, 1), nrow=3),
+     const.dir=c(">=", ">=", "<="), const.rhs=c(9, 9, 10))
> wrong.lp
Success: the objective function is 8
> wrong.lp$solution
[1] 0.0 4.5 0.0 5.5
```

Rounding the solution to the nearest integer will lead to a violation of the first constraint (if x_2 is taken to be 4) or to a minimum value of the objective function that is larger than 8 (if $x_2 = 5$).

7.5.6 Alternatives to `lp()`

The `lp()` function provides an interface to code written in C. There is another function in the `linprog` package called `solveLP()` which is written entirely in R; this latter function solves large problems much more slowly than the `lp()` function, but it provides more detailed output. We note also the function `simplex()` in the `boot` package.

It should also be noted that, for very large problems, the simplex method might not converge quickly enough; better procedures, called *interior point methods*, have been discovered recently, and are implemented in other programming languages, but not yet in R.

7.5.7 Quadratic programming

Linear programming problems are a special case of optimization problems in which a possibly nonlinear function is minimized subject to constraints. Such problems are typically more difficult to solve and are beyond the scope of this text; an exception is the case where the objective function is quadratic and the constraints are linear. This is a problem in *quadratic programming*.

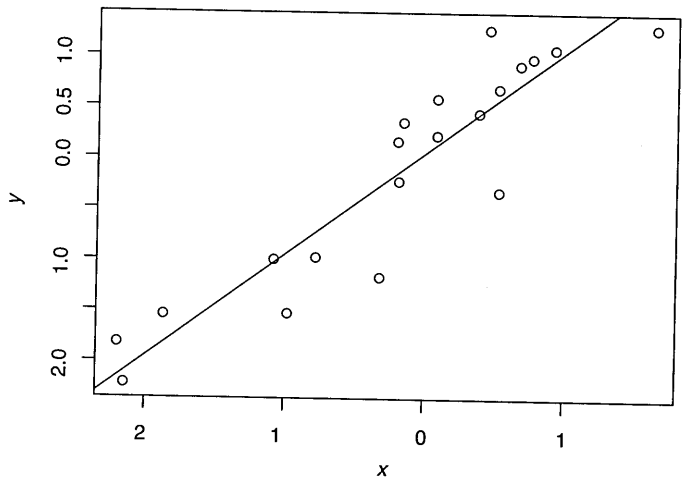

Fig. 7.11 A scatterplot of the 20 observations with a line of slope 1 and intercept 0.05 overlaid.

A quadratic programming problem with k constraints is often of the form

$$\min_{\beta} \frac{1}{2}\beta^T D\beta - d^T\beta,$$

subject to constraints $A^T\beta \geq b$. Here β is a vector of p unknowns, D is a positive definite $p \times p$ matrix, d is vector of length p, A is a $p \times k$ matrix, and b is a vector of length k.

Example 7.12
Consider the following 20 pairs of observations on the variables x and y. A scatterplot is displayed in Figure 7.11.

```
> x <- c(0.45,  0.08, -1.08,  0.92,  1.65,  0.53, 0.52, -2.15, -2.20,
+        -0.32, -1.87, -0.16, -0.19, -0.98, -0.20, 0.67,  0.08,  0.38,
+         0.76, -0.78)
> y <- c(1.26,  0.58, -1.00,  1.07,  1.28, -0.33, 0.68, -2.22, -1.82,
+        -1.17, -1.54,  0.35, -0.23, -1.53,  0.16, 0.91,  0.22,  0.44,
+         0.98, -0.98)
```

Our problem is to pass a line of "best-fit" through these data. We seek a line of the form

$$y = \beta_0 + \beta_1 x,$$

where β_0 is the y-intercept and β_1 is the slope. However, we have additional background information about these data that indicate that the slope β_1 of the required line is at least 1.

The line we want is the one that minimizes the sum of the squared vertical distances between the observed points and the line itself:

$$\min_{\beta_0,\beta_1} \sum_{i=1}^{20} (y_i - \beta_0 - \beta_1 x_i)^2.$$

Our extra information about the slope tells us that this minimization is subject to the constraint $\beta_1 \geq 1$.

This is an example of a *restricted least-squares* problem and is equivalent to

$$\min_{\beta} \ \beta^T X^T X \beta - 2y^T X \beta,$$

subject to

$$A\beta \geq b,$$

where $A = [0 \ \ 1]$, $\beta = [\beta_0 \ \ \beta_1]^T$, y is a column vector consisting of the 20 y measurements, and X is a matrix consisting of two columns, where the first column contains only 1's and the second column contains the 20 x observations:

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdots & \cdots \\ 1 & x_n \end{bmatrix} = \begin{bmatrix} 1 & 0.45 \\ 1 & 0.08 \\ \cdots & \cdots \\ 1 & -0.78 \end{bmatrix}.$$

We then have

$$X^T X = \begin{bmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{bmatrix} = \begin{bmatrix} 20 & -3.89 \\ -3.89 & 21.4 \end{bmatrix},$$

$$y^T X = \left[\sum_{i=1}^n y_i \ \ \sum_{i=1}^n x_i y_i \right] = [-2.89 \ \ 20.7585].$$

This is a quadratic programming problem with $D = X^T X$ and $d = y^T X$.

Linear programming methods have been adapted to handle quadratic programming problems. The `solve.QP()` function is in the `quadprog` package. It solves minimization problems, and the following are parameters which are required:

- `Dmat` – a matrix containing the elements of the matrix (D) of the quadratic form in the objective function
- `dvec` – a vector containing the coefficients of the decision variables in the objective function
- `Amat` – a matrix containing the coefficients of the decision variables in the constraints; each row of the matrix corresponds to a constraint
- `bvec` – a vector containing the constants given on the right-hand side of the constraints
- `mvec` – a number indicating the number of equality constraints. By default, this is 0. If it is not 0, the equality constraints should be listed ahead of the inequality constraints.

The output from this function is a list whose first two elements are the vector that minimizes the function and the minimum value of the function.

Example 7.13
For the restricted least squares problem of Example 7.12, we must first set up the matrices D and A as well as the vectors b and d. Here, $D = X^T X$ and $d = X^T y$.

```
> library(quadprog)
> X <- cbind(rep(1, 20), x)
> XX <- t(X) %*% X
> Xy <- t(X) %*% y
> A <- matrix(c(0, 1), ncol=1)
> b <- 1
> solve.QP(Dmat=XX, dvec=Xy, Amat=A, bvec=b)
$solution
[1] 0.05 1.00

$value
[1] -10.08095
$unconstrainted.solution
[1] 0.04574141 0.97810494

$iterations
[1] 2 0

$iact
[1] 1
```

From the output, we see that the required line is

$$\hat{y} = 0.05 + x.$$

The rest of the output is indicating that the constraint is *active*. If the unconstrained problem had yielded a slope larger than 1, the constraint would have been *inactive*, and the solution to the unconstrained problem would be the same as the solution to the constrained problem.

Note that the decision variables in the above example were restricted in sign. If needed, nonnegativity conditions must be explicitly set when using the `solve.QP()` function. Also, it should be noted that inequality constraints are all of the form `>=`. If your problem contains some inequality constraints with `<=`, then the constraints should be multiplied through by -1 to convert them to the required form.

It should be noted that there are more efficient ways to solve restricted least squares problems in other computing environments. The matrix D in the preceding example is a diagonal matrix, and this special structure can be used to reduce the computational burden. The following example involves a full matrix. This example also places a restriction on the sign of the decision variables.

Example 7.14

Quadratic programming can be applied to the problem of finding an optimal portfolio for an investor who is choosing how much money to invest in each of a set of n stocks. A simple model for this problem boils down to maximizing

$$x^T\beta - \frac{k}{2}\beta^T D\beta,$$

subject to the constraints $\sum_{i=1}^{n} \beta_i = 1$ and $\beta_i \geq 0$ for $i = 1, \ldots, n$.

The ith component of the β vector represents the fraction of the investor's fortune that should be invested in the ith stock. Note that each element of this vector must be nonnegative, since the investor cannot allocate a negative fraction of her portfolio to a stock.[1] The vector x contains the average daily *returns* for each stock; the daily return value for a stock is the difference in closing price for the stock from one day to the next. Therefore, $x^T\beta$ represents the average daily return for the investor.

Most investors do not want to take large risks; the second term in the objective function takes this fact into account. The factor k quantifies the investor's tolerance for risk. If the investor's goal is purely to maximize the average daily return without regard for the risk, then $k = 0$. The value of k is larger for an investor who is concerned about taking risks. The D matrix quantifies the underlying variability in the returns; it is called a *covariance* matrix. The diagonal elements of the D matrix are the variances of the returns for each of the stocks. An off-diagonal element (i,j) is the covariance between returns of the ith and jth stocks; this is a simple measure of relation between the two returns.

For a specific example, we consider three stocks and set $k = 4$ and

$$D = \begin{bmatrix} 0.010 & 0.002 & 0.002 \\ 0.002 & 0.010 & 0.002 \\ 0.002 & 0.002 & 0.010 \end{bmatrix}.$$

We assume the mean daily returns for the three stocks are 0.002, 0.005, and 0.01, respectively, so $x^T = [0.002\ 0.005\ 0.01]$.

The requirement that $\beta_1 + \beta_2 + \beta_3 = 1$ and the nonnegativity restrictions on the β variables can be written as

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \begin{matrix} = \\ \geq \\ \geq \\ \geq \end{matrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Therefore, we take

$$A^T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

[1] Such behaviour is called *shorting* a stock, and we do not allow it here.

To set this up in R, we note first that the maximization problem is equivalent to minimizing the negative of the objective function, subject to the same constraints. This fact enables us to employ `solve.QP()`.

```
> A <- cbind(rep(1, 3), diag(rep(1, 3)))
> D <- matrix(c(0.01, 0.002, 0.002, 0.002, 0.01, 0.002, 0.002, 0.002, 0.01),
+             nrow=3)
> x <- c(0.002, 0.005, 0.01)
> b <- c(1, 0, 0, 0)
> # meq specifies the number of equality constraints;
> # these are listed before the inequality constraints
> solve.QP(2 * D, x, A, b, meq=1)
$solution
[1] 0.1041667 0.2916667 0.6041667

$value
[1] -0.002020833

$unconstrainted.solution
[1] -0.02678571  0.16071429  0.47321429

$iterations
[1] 2 0

$iact
[1] 1
```

The optimal investment strategy (for this investor) is to put 10.4% of her fortune into the first stock, 29.2% into the second stock, and 60.4% into the third stock.

The optimal value of the portfolio is 0.0020 (from `$value` above). (Recall that the negative sign appears in the output, because we were minimizing the negative of the objective function.)

Exercises

1 (a) Find nonnegative x_1, x_2, x_3, and x_4 to minimize

$$C(x) = x_1 + 3x_2 + 4x_3 + x_4,$$

subject to the constraints

$$x_1 - 2x_2 \geq 9$$
$$3x_2 + x_3 \geq 9,$$

and

$$x_2 + x_4 \geq 10.$$

(b) Will the solution change if there is a requirement that any of the variables should be integers? Explain.

(c) Suppose the objective function is changed to

$$C(x) = x_1 - 3x_2 + 4x_3 + x_4.$$

What happens to the solution now?

2 Find nonnegative x_1, x_2, x_3, and x_4 to maximize

$$C(x) = x_1 + 3x_2 + 4x_3 + x_4,$$

subject to the constraints

$$x_1 - 2x_2 \leq 9$$
$$3x_2 + x_3 \leq 9,$$

and

$$x_2 + x_4 \leq 10.$$

Chapter exercises

1 Consider the data of Example 7.12. Calculate the slope and intercept for a line of "best-fit" for these data for which the intercept is at least as large as the slope.

2 Re-do the calculation in the portfolio allocation example using $k = 1$. How does being less risk-averse affect the investor's behavior?

3 Often, there are upper bounds on the proportion that can be invested in a particular stock. Re-do the portfolio allocation problem with the requirement that no more than 50% of the investor's fortune can be tied up in any one stock.

4 Duncan's Donuts Inc. (DDI) and John's Jeans Ltd. (JJL) are two stocks with mean daily returns of 0.005 and 0.010, respectively. What is the optimal portfolio for a completely risk-loving investor (i.e. risk-tolerance constant $k = 0$) who invests only in these two stocks? (Hint: this question does not require any computations.)

5 Suppose the daily returns for DDI and JJL are independent, but $\sigma^2_{\text{DDI}} = 0.01$ and $\sigma^2_{\text{JJL}} = 0.04$. What is the optimal allocation for an investor with a risk tolerance constant (a) $k = 1$? (b) $k = 2$?
You can use the fact that

$$D = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.04 \end{bmatrix}.$$

6 Repeat the preceding question under the assumption that the covariance between the returns for DDI and JJL is 0.01. You can use the fact that

$$D = \begin{bmatrix} 0.01 & 0.01 \\ 0.01 & 0.04 \end{bmatrix}.$$

Appendix

Review of random variables and distributions

Suppose an experiment is conducted in which a number of different outcomes are possible. Each outcome has a certain probability of occurrence.

Consider a cancer treatment that will be tested on 10 patients. The number of patients who show an increase in their white-blood cell count at the end of 5 weeks of treatment cannot be predicted exactly at the beginning of the trial, so this number, which we might label N, is thought of as a *random variable*. N is an example of a *discrete* random variable since it only takes values from a discrete set, i.e. $\{0, 1, 2, \ldots, 10\}$. The time, T, until death could also be measured for one of the patients; again, T cannot be predicted exactly in advance, so it is also an example of a random variable; since it can take a continuum of possible values, it is referred to as a *continuous* random variable.

A random variable is characterized by its distribution. This specifies the probability that the variable will take one or more values. If X denotes the number of heads obtained in two independent tosses of a fair coin, we might write

$$P(X \leq 1) = 0.75$$

to indicate that the probability of 0 or 1 head in two tosses is 0.75. In general, the function

$$F(x) = P(X \leq x)$$

is called the distribution function of the random variable X. If $F(x)$ has a derivative, we can define the probability density function of X as

$$f(x) = F'(x).$$

This is often possible with continuous random variables X. Note that, in this case,

$$F(y) = \int_{-\infty}^{y} f(x)\,\mathrm{d}x.$$

Among other things, note that the area under the curve specified by $f(x)$ is 1.

The expected value of a random variable is also an important concept. For continuous random variables, we can write

$$E[X] = \int_{-\infty}^{\infty} x f(x)\,dx.$$

This is the mean value of the density function $f(x)$. It is often denoted by the symbol μ. We also can take expectations of functions of random variables using the formula

$$E[g(X)] = \int_{-\infty}^{\infty} g(x) f(x)\,dx.$$

An important example of this is the variance. The variance of a random variable gives an indication of the unpredictability in a random variable. Its formula is

$$\mathrm{Var}(X) = E[(X-\mu)^2] = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)\,dx.$$

Another important concept is that of *quantile*: this is the value of x for which $F(x)$ takes on a particular value. When the inverse function $F^{-1}(y)$ is defined, the α quantile of X is given by $F^{-1}(\alpha)$. For example, the 0.95 quantile is the value of x for which $F(x) = 0.95$; in other words, x is the 95th percentile of the distribution. Frequently used quantiles are the median $\tilde{x}$ which satisfies $F(\tilde{x}) = 0.5$, and the upper and lower quartiles which satisfy $F(x) = 0.75$ and $F(x) = 0.25$, respectively.

The following tables summarize properties of some commonly used univariate distributions:

Distribution name	$f(x)$	$F(x)$	$E[X]$	$\mathrm{Var}(X)$
Uniform (a,b)	$\frac{1}{b-a}$, $a < x < b$	x	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
Exponential (λ)	$\lambda e^{-\lambda x}$, $x > 0$	$1 - e^{-\lambda x}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
Normal (μ, σ^2)	$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	$\int_{-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}\,dy$	μ	σ^2

Distribution name	$P(X = x)$	$E[X]$	$\mathrm{Var}(X)$
Binomial (n,p)	$\binom{n}{x} p^x (1-p)^{n-x}$	np	$np(1-p)$
Poisson (λ)	$\frac{\lambda^x e^{-\lambda}}{x!}$	λ	λ

We conclude this review with some brief comments about bivariate distributions. In particular, suppose X and Y are continuous random variables having joint probability density $f(x, y)$. We can define expectations using double integrals:

$$E[g(X, Y)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) f(x, y)\, dx\, dy$$

for functions $g(x, y)$. In particular, setting $g(X, Y) = I(X \leq u)I(Y \leq v)$ gives

$$E[I(X \leq u)I(Y \leq v)] = \int_{-\infty}^{u} \int_{-\infty}^{v} f(x, y)\, dx\, dy,$$

which implies that, for any u and v,

$$P(X \leq u, Y \leq v) = \int_{-\infty}^{u} \int_{-\infty}^{v} f(x, y)\, dx\, dy.$$

Here, $I()$ denotes the indicator function which takes on the value 1, when its argument is true, and 0, when its argument is false.

The marginal density of X is obtained by integrating over all values of y:

$$f_X(x) = \int_{-\infty}^{\infty} f(x, y)\, dy,$$

and similarly, the marginal density of Y is obtained by integrating over all values of x:

$$f_Y(y) = \int_{-\infty}^{\infty} f(x, y)\, dx.$$

X and Y are stochastically independent if

$$f(x, y) = f_X(x) f_Y(y).$$

Among other things, this implies that $P(X \leq u, Y \leq v) = P(X \leq u) P(Y \leq v)$, and, by the definition of *conditional probability*,

$$P(X \leq u | Y \leq v) = \frac{P(X \leq u, Y \leq v)}{P(Y \leq v)} = P(X \leq u).$$

The term on the left denotes the conditional probability that $X \leq u$, given that $Y \leq v$. Intuitively, the above statement means that knowledge of the value of Y does not give us any additional information with which to predict the value of X.