

5 The method of steepest decent

The method of steepest decent is a method of searching for the minimum of a function of many variables f . In each iteration of this algorithm a line search is performed in the direction of the steepest decent of the function at the current location. In other words,

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_n \dot{f}(\mathbf{x}_n),$$

where α_n is the nonnegative scalar that minimizes $f(\mathbf{x}_n - \alpha \dot{f}(\mathbf{x}_n))$. It can be shown that relative to the solution set $\{\mathbf{x}^* : \dot{f}(\mathbf{x}^*) = \mathbf{0}\}$, the algorithm is descending and closed, thus converging.

5.1 The rate of convergence in the quadratic case

Assume

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}' Q \mathbf{x} - \mathbf{x}' \mathbf{b} = \frac{1}{2} (\mathbf{x} - \mathbf{x}^*)' Q (\mathbf{x} - \mathbf{x}^*) - \frac{1}{2} \mathbf{x}^{*'} Q \mathbf{x}^*,$$

where Q a positive definite and symmetric matrix and $\mathbf{x}^* = Q^{-1} \mathbf{b}$ is the minimizer of f . Note that in this case $\dot{f}(\mathbf{x}) = Q \mathbf{x} - \mathbf{b}$. and

$$f(\mathbf{x}_n - \alpha \dot{f}(\mathbf{x}_n)) = \frac{1}{2} (\mathbf{x}_n - \alpha \dot{f}(\mathbf{x}_n))' Q (\mathbf{x}_n - \alpha \dot{f}(\mathbf{x}_n)) - (\mathbf{x}_n - \alpha \dot{f}(\mathbf{x}_n))' \mathbf{b},$$

which is minimized at

$$\alpha_n = \frac{\dot{f}(\mathbf{x}_n)' \dot{f}(\mathbf{x}_n)}{\dot{f}(\mathbf{x}_n)' Q \dot{f}(\mathbf{x}_n)}.$$

It follows that

$$\begin{aligned} \frac{1}{2} (\mathbf{x}_{n+1} - \mathbf{x}^*)' Q (\mathbf{x}_{n+1} - \mathbf{x}^*) = \\ \left\{ 1 - \frac{(\dot{f}(\mathbf{x}_n)' \dot{f}(\mathbf{x}_n))^2}{\dot{f}(\mathbf{x}_n)' Q \dot{f}(\mathbf{x}_n) \dot{f}(\mathbf{x}_n)' Q^{-1} \dot{f}(\mathbf{x}_n)} \right\} \times \frac{1}{2} (\mathbf{x}_n - \mathbf{x}^*)' Q (\mathbf{x}_n - \mathbf{x}^*). \end{aligned}$$

Theorem 5.1 (Kantorovich inequality). *Let Q be a positive definite and symmetric matrix and let $0 < a = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n = A$ be the eigenvalues. Then*

$$\frac{(\mathbf{x}' \mathbf{x})^2}{(\mathbf{x}' Q \mathbf{x})(\mathbf{x}' Q^{-1} \mathbf{x})} \geq \frac{4aA}{(a + A)^2}.$$

Proof: By a change of variables Q is diagonal. We assume it is. In which case

$$\frac{(\mathbf{x}'\mathbf{x})^2}{(\mathbf{x}'Q\mathbf{x})(\mathbf{x}'Q^{-1}\mathbf{x})} = \frac{(\sum_{i=1}^d x_i^2)^2}{(\sum_{i=1}^d \lambda_i x_i^2)(\sum_{i=1}^d x_i^2/\lambda_i)}.$$

Denoting $\xi_i = x_i^2 / \sum_{j=1}^d x_j^2$ the above becomes

$$= \frac{1/\sum_{i=1}^d \xi_i \lambda_i}{\sum_{i=1}^d (\xi_i/\lambda_i)} = \frac{\phi(\xi_1, \dots, \xi_d)}{\psi(\xi_1, \dots, \xi_d)}.$$

The numerator is a point on the curve $1/x$. The denominator is a convex combination of points from that curve. The minimal ratio is achieved for some $\lambda = \xi_1 \lambda_1 + \xi_d \lambda_d$, where $\xi_1 + \xi_d = 1$. Hence,

$$\frac{\phi(\xi_1, \dots, \xi_d)}{\psi(\xi_1, \dots, \xi_d)} \geq \min_{\lambda_1 \leq \lambda \leq \lambda_d} \frac{(1/\lambda)}{(\lambda_1 + \lambda_d - \lambda)/(\lambda_1 \lambda_d)}.$$

The minimum is achieved when $\xi_1 = \xi_d = 1/2$, and the proof follows.

Theorem 5.2. *For the quadratic case*

$$\frac{1}{2}(\mathbf{x}_{n+1} - \mathbf{x}^*)'Q(\mathbf{x}_{n+1} - \mathbf{x}^*) \leq \left(\frac{A-a}{A+a}\right)^2 \frac{1}{2}(\mathbf{x}_n - \mathbf{x}^*)'Q(\mathbf{x}_n - \mathbf{x}^*).$$

Proof:

$$1 - \frac{4aA}{(a+A)^2} = \left(\frac{A-a}{A+a}\right)^2.$$

5.2 Applying the method in R

Let us investigate the application of the steepest decent method in the quadratic setting. Consider the matrix Q and the vector \mathbf{b} :

```
> Q <- matrix(c(0.78,-0.02,-0.12,-0.14,-0.02,0.86,-0.04,0.06,
+   -0.12,-0.04,0.72,-0.08,-0.14,0.06,-0.08,0.74),
+   4,4,byrow=TRUE)
> Q
      [,1] [,2] [,3] [,4]
[1,] 0.78 -0.02 -0.12 -0.14
[2,] -0.02 0.86 -0.04 0.06
[3,] -0.12 -0.04 0.72 -0.08
[4,] -0.14 0.06 -0.08 0.74
> b <- c(0.76,0.08,1.12,0.68)
```

We define the quadratic function `quad` using the matrix and the vector and find the minimum of the function using R's built-in general purpose optimizer `optim`:

```
> quad <- function(x,Q,b) t(x)%*%Q%*%x/2 - sum(x*b)
>
> optim(rep(1,4),quad,Q=Q,b=b)
$par
[1] 1.5349646 0.1220379 1.9751225 1.4129248

$value
[1] -2.174660

$count
function gradient
      217      NA

$convergence
[1] 0

$message
NULL

> solve(Q)%*%b
      [,1]
[1,] 1.5349650
[2,] 0.1220096
[3,] 1.9751564
[4,] 1.4129555
```

In its default application the function `optim` takes as argument a function to be minimized and an initial point and produces the minimizing value and the value of the target function at that value. Extra arguments to the function in the argument may be passed on using the “...” argument. Observe that the function produces a minimizing value which is almost identical to the theoretical value.

Next we want to program the algorithm of steepest decent. For that we need a function that computes the gradient:

```
> quad.grad <- function(x,Q,b) Q%*%x - b
>
```

```

> steepest.decent <- function(fun,grad,x,...)
+ {
+   g <- function(a,x,...) fun(x - a*grad(x,...),...)
+   opt <- optimize(g,c(0,10),x=x,...)
+   alpha <- opt$minimum
+   objective <- opt$objective
+   x <- x - alpha*grad(x,...)
+   return(list(x=x,alpha=alpha,objective=objective))
+ }
> steepest.decent(quad,quad.grad,rep(1,4),Q=Q,b=b)
$x
      [,1]
[1,] 1.331422136
[2,] 0.005733593
[3,] 1.815808334
[4,] 1.127470052

$alpha
[1] 1.274701

$objective
      [,1]
[1,] -2.128281

```

Observe that functions can be treated as any other argument. Observe also that we apply the line-search procedure `optimize` in order to find the minimizing value in the direction determined by the gradient.

Let us apply several iterations of the algorithm:

```

> xn <- x <- rep(1,4)
> fn <- quad(x,Q,b)
> for (i in 1:10)
+ {
+   out <- steepest.decent(quad,quad.grad,x,Q=Q,b=b)
+   x <- out$x
+   xn <- cbind(xn,x)
+   fn <- c(fn,out$obj)
+ }
> fn
[1] -1.430000 -2.128281 -2.171504 -2.174440 -2.174644 -2.174658 -2.174659
[8] -2.174660 -2.174660 -2.174660 -2.174660

```

```

> xn
      xn
[1,]  1 1.331422136 1.4813488 1.5235040 1.5302238 1.5342309 1.5345837 1.5349140
[2,]  1 0.005733593 0.1700348 0.1144736 0.1250617 0.1215216 0.1222107 0.1219760
[3,]  1 1.815808334 1.9125511 1.9623510 1.9713649 1.9741493 1.9749043 1.9750791
[4,]  1 1.127470052 1.4000565 1.3926856 1.4122049 1.4114599 1.4129046 1.4128446

[1,] 1.5349357 1.5349614 1.5349628
[2,] 0.1220236 0.1220072 0.1220106
[3,] 1.9751387 1.9751506 1.9751551
[4,] 1.4129518 1.4129472 1.4129552

```

We can see that the algorithm converged to the minimizing value after 8 iterations.

An assessment of the rate of convergence of the algorithm can be obtained via the examination of the extreme eigenvalues:

```

> eigen(Q)
$values
[1] 0.94 0.88 0.76 0.52

$vectors
      [,1]      [,2]      [,3]      [,4]
[1,] 5.773503e-01 5.773503e-01 9.313285e-16 5.773503e-01
[2,] -5.773503e-01 5.773503e-01 -5.773503e-01 7.530868e-16
[3,] 8.592302e-17 -5.773503e-01 -5.773503e-01 5.773503e-01
[4,] -5.773503e-01 9.761208e-16 5.773503e-01 5.773503e-01

> U <- eigen(Q)$vector
> L <- diag(eigen(Q)$val)
> U%*%t(U)
      [,1]      [,2]      [,3]      [,4]
[1,] 1.000000e+00 -2.918210e-17 -1.397266e-16 -1.099923e-16
[2,] -2.918210e-17 1.000000e+00 -1.318634e-16 -3.651837e-16
[3,] -1.397266e-16 -1.318634e-16 1.000000e+00 -1.780802e-16
[4,] -1.099923e-16 -3.651837e-16 -1.780802e-16 1.000000e+00
> t(U)%*%U
      [,1]      [,2]      [,3]      [,4]
[1,] 1.000000e+00 3.255039e-16 -2.019327e-16 2.976677e-16
[2,] 3.255039e-16 1.000000e+00 7.358552e-17 -5.778077e-18
[3,] -2.019327e-16 7.358552e-17 1.000000e+00 1.847481e-16

```

```

[4,] 2.976677e-16 -5.778077e-18 1.847481e-16 1.000000e+00
> L
      [,1] [,2] [,3] [,4]
[1,] 0.94 0.00 0.00 0.00
[2,] 0.00 0.88 0.00 0.00
[3,] 0.00 0.00 0.76 0.00
[4,] 0.00 0.00 0.00 0.52
> U%*%L%*%t(U) - Q
      [,1] [,2] [,3] [,4]
[1,] 1.110223e-16 -2.428613e-17 -1.387779e-17 -5.551115e-17
[2,] -2.775558e-17 -8.881784e-16 -2.914335e-16 -5.342948e-16
[3,] -8.326673e-17 -3.608225e-16 -3.330669e-16 -2.498002e-16
[4,] -1.110223e-16 -5.551115e-16 -2.498002e-16 -3.330669e-16
> ((0.94 - 0.52)/(0.94 + 0.52))^2
[1] 0.08275474

```

5.3 Homework

1. Investigate the the performance of the steepest decent algorithm for

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2.$$

- (a) Compute and program the gradient of the function.
- (b) Apply the function `steepest.decent` in order to find the minimum.
- (c) Plot the function and the points produced by the algorithm.
- (d) Why doesn't the steepest decent algorithm converge?