CISC 371 Class 9

Second-Order Optimization – Newton's Method and Scaling Methods

Texts: [1] pp. 128–130; [2] pp. 83–88, 58–66

Main Concepts:
- *Local quadratic approximation*
- *Newton's Method*
- *Damped Newton's Method uses backtracking*
- *Scaling the weights is altered descent*

Sample Problem, Computer Vision: How can we find the bottom of a deep and narrow valley?

In Class 8, one function seemed to have slow convergence using the steepest descent algorithm. We do not know whether the main reason for slow convergence is the nature of the function, the small stepsize, or that we chose a less than ideal search direction. It is, of course, possible that all three factors have contributed to the slow convergence.

If we examine the first plot of steepest descent, which is re-plotted in Figure 9.1, we see that the direction of steepest descent does not "point" towards the minimizer, which is at the origin. This suggests that another search direction might be a better choice for some situations.

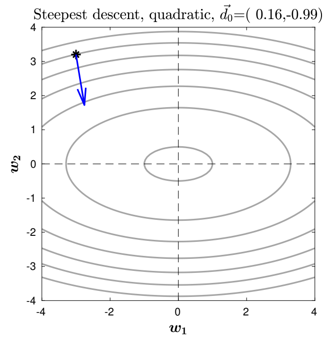

Figure 9.1: Descent direction from $\vec{w}_0$ for the quadratic function in Equation 8.7, showing contours in gray and the direction of steepest descent in blue. The direction of steepest descent does not point directly towards the minimizer, which is at the origin.

9.1 Scaled Descent

A more exaggerated situation is a function that looks like a valley that is narrow, deep, and has a shallow minimum. One example is the quadratic function and its derivatives

$$\begin{aligned} f_1(\vec{w}) &= 100w_1^2 - 10w_1w_2 + w_2^2 + 100 \\ \underline{\nabla} f_1(\vec{w}) &= \begin{bmatrix} 200w_1 - 10w_2 & , & -10w_1 + 2w_2 \end{bmatrix} \\ \underline{\nabla}^2 f_1(\vec{w}) &= \begin{bmatrix} 200 & -10 \\ -10 & 2 \end{bmatrix} \end{aligned} \tag{9.1}$$

The function f_1 in Equation 9.1 is illustrated in Figure 9.2. A simple computation shows that the Hessian matrix – which is constant for all points $\vec{w}$ – has eigenvalues that are approximately $\lambda_1 \approx 1.5$ and $\lambda_2 \approx 200.5$, so the Hessian matrix is not well conditioned.

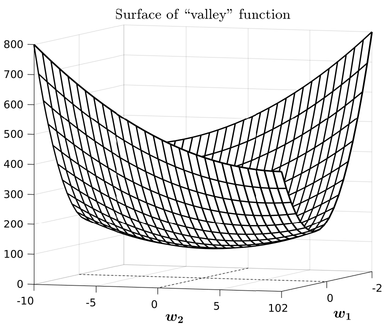

Figure 9.2: Surface plot of the "deep valley" function f_1 of Equation 9.1. At the origin, the global minimum has one relatively flat direction and one relatively curved direction.

For this function, we can try to find the minimizer by steepest descent with a fixed stepsize. Suppose that we use the same starting point as we used in Class 8, and after some experimentation we determine a stepsize that does not exhibit oscillation. The arguments that we can use might be

$$\vec{w}_0 = \begin{bmatrix} 3.0 \\ -3.2 \end{bmatrix} \qquad s = 0.005 \tag{9.2}$$

After 100 iterations of steepest descent, using the arguments in Equation 9.2, we would find that the fixed-stepsize algorithm descended into the valley relatively quickly, and then descended towards the minimizer relatively slowly. The path of the estimated minimizer is shown in Figure 9.3, superimposed on contours of the function f_1.

A fundamental difficulty with the function f_1 is the relative scale of the entries in the weight vector: the function is 10 times more sensitive to entry w_1 than it is to entry w_2. The difficulty also

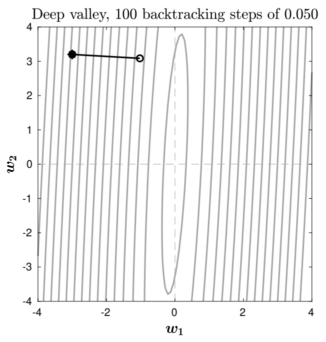

Figure 9.3: Iterative descent of the quadratic function f_1 in Equation 9.1, using steepest descent with a fixed computed fixed stepsize. Contours are shown in gray and the path of descent descent, taking 100 steps, is shown in black. The first step was nearly to the local "bottom" of the valley and the subsequent 99 steps were towards the minimizer.

appears in the gradient, where a step in the w_1 direction will have about 10 times the magnitude of a step in the w_2 direction.

From numerical linear algebra, one way to manage such a disparity is to *precondition*, or *scale*, the variables. Scaling the variables, for the example of f_1 in Equation 9.1, can be performed by thinking of substituting a new variable in place of $\vec{w}$. For example, we might select a variable $\vec{v}$ that reduces the effect of w_1. One way to do this is by having $v_2 = w_2$, and having $v_1 = 10w_1$ or $w_1 = 0.1v_1$. Mathematically, the relationship between the original variable $\vec{w}$ and the new variable $\vec{v}$ would be

$$\vec{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \vec{g}(\vec{v}) = \begin{bmatrix} 0.1v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0.1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = B_M\vec{v} \tag{9.3}$$

In Equation 9.3, we subscript the matrix to indicate that we manually selected the scaling factors. As described in the extra notes for this class, this modifies the method of steepest descent to use the iteration

$$\vec{w}_{k+1} = \vec{w}_k - sB_M[\underline{\nabla} f_1(\vec{w}_k)]^T \tag{9.4}$$

Because we have scaled the problem by a factor of 10, we can multiply the stepsize by 10. This would give us argument values of

$$\vec{w}_0 = \begin{bmatrix} 3.0 \\ -3.2 \end{bmatrix} \qquad s = 0.05 \qquad B_M = \begin{bmatrix} 0.1 & 0 \\ 0 & 1 \end{bmatrix} \tag{9.5}$$

When we perform scaled steepest descent with a fixed stepsize, the optimization is much better. The path of the estimated minimizer and contours of the function f_1 are shown in Figure 9.4.

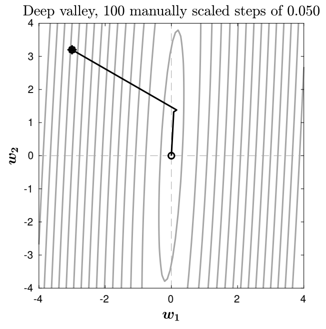

Figure 9.4: Manually scaled steepest descent of the quadratic function f_1 in Equation 9.1, using a fixed stepsize. Contours are shown in gray and the path of descent descent, taking 100 steps, is shown in black. The first step was nearly to the local "bottom" of the valley and the subsequent 99 steps were towards the minimizer.

What if we use a different scaling matrix? As described in the extra notes for this class, we can scale with any symmetric positive definite matrix B and be performing a descent method. Suppose that we use the inverse of the Hessian matrix of Equation 9.1, which is

$$B_H = [\underline{\nabla}^2 f_1(\vec{w})]^{-1} = \begin{bmatrix} \frac{1}{150} & \frac{1}{30} \\ \frac{1}{30} & \frac{2}{3} \end{bmatrix} \tag{9.6}$$

When we use the same values of the arguments as in Equation 9.5, and use B_H instead of B_M, we more rapidly converge to the minimizer. The path of the estimates of the minimizer is shown in Figure 9.5, superimposed on contours of the function f_1.

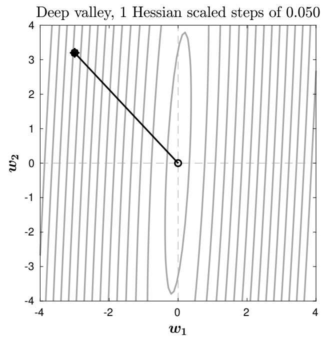

Figure 9.5: Hessian scaled steepest descent of the quadratic function f_1 in Equation 9.1, using a fixed stepsize. Contours are shown in gray and the path of descent descent, taking 100 steps, is shown in black. The steps are small but are steadily directed towards the minimizer.

These attempts at scaling suggest a question: does the symmetric positive definite matrix B need to be fixed for the entire iteration? Or can the matrix be variable, depending on the step B_k? There are many effective ways to select such a matrix. One way can be derived from a more detailed exploration of a Taylor's series for a function, which was an idea that we used when we tried to minimize a function with a scalar argument.

9.2 Newton's Method

For a function with a scalar argument, one way that we used when searched for a local minimum was to use a local quadratic curve to model the function. Let us extend this idea further.

We can use a quadratic model, of a function with a vector argument, to derive a second-order gradient-based method. For historical reasons that are related to zero-finding, using a local quadratic model is called *Newton's Method.*

We can derive Newton's Method by modeling the function as a truncated Taylor series that is expanded around a point $\vec{w}_0$, which is

$$g(\vec{w}) = f(\vec{w}_0) + \underline{\nabla} f(\vec{w}_0)[\vec{w} - \vec{w}_0] + \frac{1}{2}[\vec{w} - \vec{w}_0]^T \underline{\nabla}^2 f(\vec{w}_0)[\vec{w} - \vec{w}_0] \tag{9.7}$$

Before we solve Equation 9.7 for a stationary point, we should recall the material from Class 6. In the extra notes for that class, Equation 6.8 provides a necessary condition for a quadratic form $\vec{v}^T K \vec{v}$ to have a strict local minimizer – the matrix K must be positive definite. Using this result for Equation 9.7 gives us an important condition:

The model function in Equation 9.7 has a single stationary point that is a strict local minimizer if the Hessian matrix, evaluated at $\vec{w}_0$, is positive definite. (9.8)

If Condition 9.8 holds, then we can differentiate Equation 9.7 with respect to $\vec{w}$, which is

$$\frac{\partial}{\partial \vec{w}} g(\vec{w}) = \underline{\nabla} f(\vec{w}_0) + [\vec{w} - \vec{w}_0]^T \underline{\nabla}^2 f(\vec{w}_0) \tag{9.9}$$

The derivative term in Equation 9.9 is, of course, a 1-form. We can solve for the search direction $[\vec{w} - \vec{w}_0]$ by setting the transpose of the derivative to be the zero vector, so

$$\begin{array}{rrcl} & [\underline{\nabla} f(\vec{w}_0) + [\vec{w} - \vec{w}_0]^T \underline{\nabla}^2 f(\vec{w}_0)]^T & = & \vec{0} \\ \rightarrow & [\underline{\nabla} f(\vec{w}_0]^T + \underline{\nabla}^2 f(\vec{w}_0)[\vec{w} - \vec{w}_0] & = & \vec{0} \\ \rightarrow & \underline{\nabla}^2 f(\vec{w}_0)[\vec{w} - \vec{w}_0] & = & -[\underline{\nabla} f(\vec{w}_0]^T \end{array} \tag{9.10}$$

The vector $[\vec{w} - \vec{w}_0]$ in Equation 9.10 is a search direction $\vec{d}$. Because the Hessian matrix at $\vec{w}_0$ is symmetric and positive definite, we know that the linear equation has a solution. Using exact mathematics, we can write the search direction from $\vec{w}_0$ as

$$\vec{d} = [\vec{w} - \vec{w}_0] = -[\underline{\nabla}^2 f(\vec{w}_0)]^{-1}[\underline{\nabla} f(\vec{w}_0]^T \tag{9.11}$$

The search direction $\vec{d}$ is called the *Newton direction*. We can use Equation 9.11 to develop an iteration

$$\begin{aligned} \vec{w}_{k+1} &= \vec{w}_k + \vec{d} \\ &= \vec{w}_k - [\underline{\nabla}^2 f(\vec{w}_k)]^{-1}[\underline{\nabla} f(\vec{w}_k]^T \end{aligned} \tag{9.12}$$

The iteration of Equation 9.12 leads us to develop Newton's Algorithm for gradient descent.

Algorithm 9.1 Newton's Method, fixed stepsize

```
Require: kmax  $> 0$
Require: gmag  $> 0$
  w $\leftarrow$ w0
  fcurr $\leftarrow$ f(w)
  g $\leftarrow$ $\underline{\nabla}$f(w)
  H $\leftarrow$ $\underline{\nabla}^2$f(w)
  d $\leftarrow$ $-$H$\backslash$[g$^T$]
  k $\leftarrow$ 0
  while $\neg$ (converged) do
    w $\leftarrow$ w + d                    $\triangleright$ fixed Newton's step
    fcurr $\leftarrow$ f(w)
    g $\leftarrow$ $\underline{\nabla}$f(w)
    H $\leftarrow$ $\underline{\nabla}^2$f(w)
    d $\leftarrow$ $-$H$\backslash$[g$^T$]
    k $\leftarrow$ k+1
  end while
```

We can implement Algorithm 9.1 and test it on the same functions as we used in Class 8:

$$\begin{aligned} f_2(\vec{w}) &= \vec{w}^T K \vec{w} \\ f_3(\vec{w}) &= -e^{\vec{w}^T K \vec{w}} \end{aligned} \quad \text{with} \quad K = \begin{bmatrix} 1/4 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{starting at} \quad \vec{w}_1 = \begin{bmatrix} -0.6 \\ 0.9 \end{bmatrix} \tag{9.13}$$

The result of a single iteration for f_2 of Equation 9.13, with the same starting point that was used in Class 8, is shown in Figure 9.6. This immediately converges, because the model function exactly matches the objective function. The minimizer of the model is the same as that of the objective.

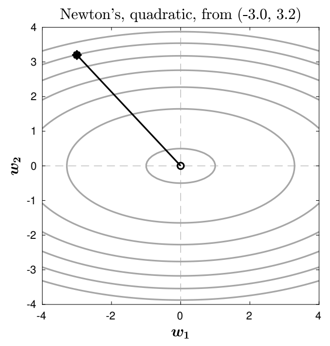

Figure 9.6: Iterative descent of the function f_2 defined in Equation 9.13 using Newton's Method with the computed fixed stepsize. The first step converges within computing tolerance because the model function is exact.

Next, we can try Newton's Method with the computed fixed stepsize on the exponential function f_3 of Equation 9.13. The results of four iterations are shown in Figure 9.7.

The effects of Newton's Method for the exponential function f_3 of Equation 9.13 are qualitatively similar to the effects of steepest descent using an overly large stepsize. The successive estimates of the minimizer, $\vec{w}_k$, oscillate around the true minimizer while slowly converging. Visually, the direction of descent seems to be appropriate; the difficulty appears to be in the choice of the stepsize.

9.3 Damped Newton's Method Using Backtracking

There are many ways to ameliorate the difficulty observed in Figure 9.7. A common method is to *damp* the effect of the magnitude of the computed stepsize in Newton's Method. Instead of using the computed magnitude of the descent vector $\vec{d}$, we can select a different stepsize. An easy way to select a damped stepsize is to use Armijo backtracking, just as we did when minimizing a function with a scalar argument. Pseudocode for a damped Newton's Method is described in Algorithm 9.2.

As shown in Figure 9.8, only three iterations of Algorithm 9.2 are needed to minimize f_3.

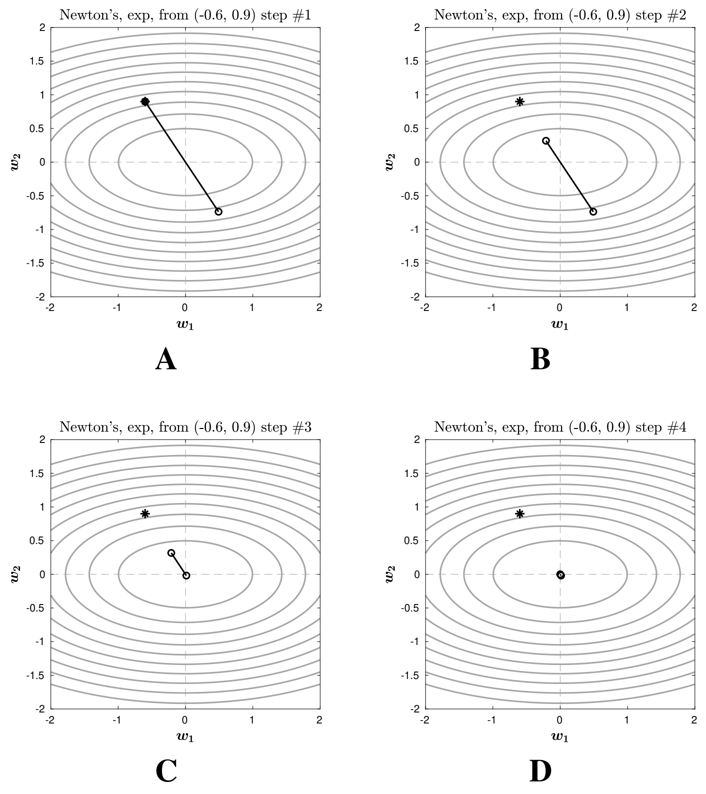

Figure 9.7: Iterative descent of the exponential function f_3 defined in Equation 9.13 using Newton's Method with the computed fixed stepsize. (A) The first step is to the opposite quadrant, closer to the minimizer at the origin. (B) The second step returns to the original quadrant, converging towards the minimizer. (C) The third returns to the opposite quadrant. (D) The fourth step has nearly converged at the minimizer.

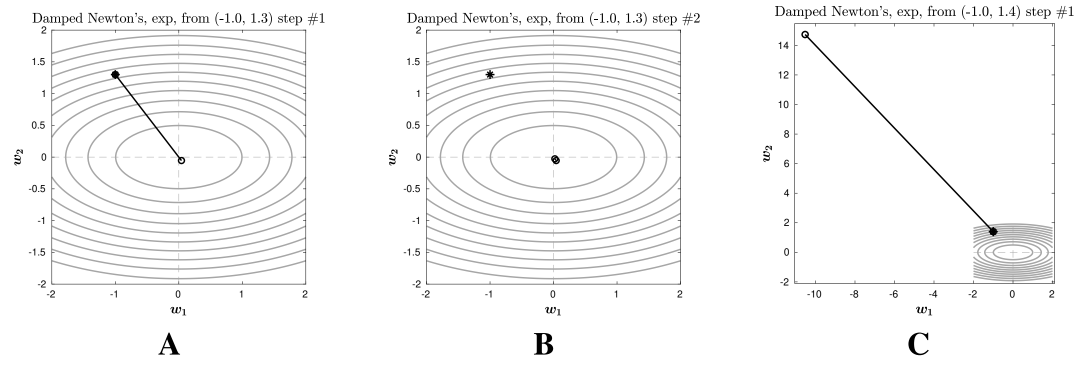

Figure 9.8: Iterative descent of the function f_3 using a damped Newton's Method with backtracking. (A) The first step is close to the minimizer at the origin. (B) The second step stays in the quadrant opposite to $\vec{w}_1$. (C) The third step has nearly converged.

Algorithm 9.2 Damped Newton's Method, using backtracking

```
Require: kmax  $> 0$
Require: gmag  $> 0$
  w $\leftarrow$ w0
  fcurr $\leftarrow$ f(w)
  g $\leftarrow$ $\underline{\nabla}$f(w)
  H $\leftarrow$ $\underline{\nabla}^2$f(w)
  d $\leftarrow$ $-$H$\backslash$[g$^T$]
  $\alpha$ $\leftarrow$ g/2
  k $\leftarrow 0$
  while $\neg$ (converged) do
     s $\leftarrow$ s0
     fest $\leftarrow$ f(w+s$\star$d)
     while fest$>$(fcurr+$\alpha\star$s$\star$d) do
        s $\leftarrow$ $\beta\star$s                  $\triangleright$ Armijo backtracking
        fest $\leftarrow$ f(w+s$\star$d)
     end while
     w $\leftarrow$ w + s$\star$d                 $\triangleright$ damped Newton's step
     fcurr $\leftarrow$ f(w)
     g $\leftarrow$ $\underline{\nabla}$f(w)
     H $\leftarrow$ $\underline{\nabla}^2$f(w)
     d $\leftarrow$ $-$H$\backslash$[g$^T$]
     $\alpha$ $\leftarrow$ g/2
     k $\leftarrow$ k+1
  end while
```

What happens with damped Newton's Method when we alter the starting point slightly? Suppose that we use an initial estimate of

$$\vec{w}_B = \begin{bmatrix} -1.0 \\ 1.4 \end{bmatrix}$$

We will find that the iteration *diverges*, instead of converging to the minimizer. The effect of a single step of the damped Newton's Method, using backtracking, is shown in Figure 9.9.

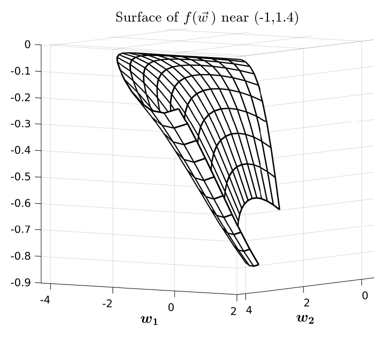

Figure 9.9: A single iteration of the function f_3, using a damped Newton's Method with backtracking. From a slightly altered starting point, the algorithm immediately diverges.

What went wrong? We can try plotting the surface of f_3 near the starting point. As shown in Figure 9.10, the function is *inflected* near the point $\vec{w}_B$. The point $\vec{w}_B$ is not a stationary point, so it does not satisfy the definition of a saddle point; something else has occurred.

Figure 9.10: A surface plot the function f_3 near the point $\vec{w}_B$. The surface is inflected, or curved in "opposite" directions. This is distinct from a saddle point that, by definition, is a stationary point.

When we compute the Hessian matrix $\underline{\nabla}^2 f_3(\vec{w}_B)$ and its eigenvalues, we find that the Hessian matrix is indefinite at $\vec{w}_B$. We have failed to meet a necessary condition of Newton's Method and the consequence is a divergent computation.

Extra Notes

9.4 Extra Notes on Scaled Steepest Descent

There are two basic concepts, using a symmetric positive definite matrix, that are foundational for many of the methods in unconstrained optimization.

Theorem: scaled steepest iteration

For any continuous differentiable $f : \mathbb{R}^n \to \mathbb{R}$, for any $\vec{w} \in \mathbb{R}^n$, for any $s \in \mathbb{R}_{++}$, and for any square full-rank matrix $M \in \mathbb{R}^{n \times n}$, the change of variables $\vec{w} = M\vec{v}$ produces the steepest-descent iteration

$$\vec{w}_{k+1} = \vec{w}_k - s[MM^T][\underline{\nabla} f(\vec{w}_k)]^T \tag{9.14}$$

Proof: Let $\vec{w} = \vec{g}(M\vec{v})$ be a change of variables. Then $\vec{v} = M^{-1}\vec{w}$ because m is assumed to be of full rank, and thus is invertible.

A new matrix $B \in \mathbb{R}^{n\times n}$ can be defined as MM^T. By construction, B is a symmetric matrix. Because M is full rank, $B \succ 0$ is positive definite.

The original function f can be written and differentiated as

$$f(\vec{w}) \;=\; f(\vec{g}(\vec{v}))$$

so

$$\begin{aligned}
\underline{\nabla} f(\vec{g}(\vec{v})) &= \frac{\partial f}{\partial \vec{g}}\frac{\partial \vec{g}}{\partial \vec{v}} \\
&= \frac{\partial f}{\partial \vec{g}} M \\
&= [\underline{\nabla} f(\vec{w})]M \\
\rightarrow \qquad [\underline{\nabla} f(\vec{g}(\vec{w}))]^T &= [[\underline{\nabla} f(\vec{w})M]^T \\
&= M^T[\underline{\nabla} f(\vec{w})]^T
\end{aligned}$$

The iteration for steepest descent, in terms of the changed variable $\vec{v}$, can be written in terms of the original variable $\vec{w}$ as

$$\begin{aligned}
\vec{v}_{k+1} &= \vec{v}_k - s[\underline{\nabla} f(\vec{g}(\vec{v}))]^T \\
&= \vec{v}_k - sM^T[\underline{\nabla} f(\vec{w})]^T \\
\equiv \quad M^{-1}\vec{w}_{k+1} &= M^{-1}\vec{w}_k - sM^T[\underline{\nabla} f(\vec{w})]^T \\
\equiv \quad \vec{w}_{k+1} &= \vec{w}_k - MsM^T[\underline{\nabla} f(\vec{w})]^T \\
&= \vec{w}_k - MsM^T[\underline{\nabla} f(\vec{w})]^T \\
&= \vec{w}_k - sB[\underline{\nabla} f(\vec{w})]^T
\end{aligned}$$

Observation: Theorem 9.14 states that any full-rank change of variables is a scaled steepest descent. A basic matrix factorization is the Cholesky decomposition: for any symmetric positive definite matrix $B \in \mathbb{R}^{n\times n} : B \succ 0$, there is a lower-triangular matrix $L \in \mathbb{R}^{n\times n}$ such that

$$B = LL^T$$

This decomposition can be used to show that a scaled descent is equivalent to a change of variables.

Theorem: scaled steepest descent

For any continuous differentiable $f : \mathbb{R}^n \to \mathbb{R}$*, for any* $\vec{w}_0 \in \mathbb{R}^n$ *such that* $\vec{w}_0$ *is not a stationary point of* f*, and for any symmetric matrix* $B \in \mathbb{R}^{n\times n} : B \succ 0$,

$$-B[\underline{\nabla} f(\vec{w})]^T \text{ is a descent direction at } \vec{w}_0 \tag{9.15}$$

Proof: Let $\vec{u} = B[-\underline{\nabla} f(\vec{w})]^T$ be a scaled version of the steepest descent direction $[\underline{\nabla} f(\vec{w})]^T$.

From Definition 8.2, if $\vec{u}$ is a descent direction of $f(\vec{w}_0)$ then $D_{\vec{u}} f(\vec{w}_0) < 0$. Then

$$\begin{aligned} D_{\vec{u}} f(\vec{w}_0) &= [\underline{\nabla} f(\vec{w}_0)]\vec{u} \\ &= [\underline{\nabla} f(\vec{w}_0)] - [B\underline{\nabla} f(\vec{w})]^T \\ &= -[\underline{\nabla} f(\vec{w}_0)]B[\underline{\nabla} f(\vec{w})]^T \end{aligned}$$

For any positive definite matrix B, and for any $\vec{v} \neq \vec{0}$, the quadratic form is positive:

$$(\vec{v} \neq \vec{0}) \to (\vec{v}^T B \vec{v} > 0)$$

Because $\vec{w}_0$ is assumed to be a nonstationary point, $[\underline{\nabla} f(\vec{w})]^T \neq \vec{0}$, so

$$\begin{aligned} & [\underline{\nabla} f(\vec{w}_0)]B[\underline{\nabla} f(\vec{w})]^T > 0 \\ \equiv\ & -[\underline{\nabla} f(\vec{w}_0)]B[\underline{\nabla} f(\vec{w})]^T < 0 \\ \equiv\ & D_{-B[\underline{\nabla} f(\vec{w})]^T} f(\vec{w}_0) < 0 \end{aligned}$$

Therefore $-B[\underline{\nabla} f(\vec{w})]^T$ is a descent direction at $\vec{w}_0$.

End of Extra Notes

References

[1] Antoniou A, Lu WS: Practical Optimization: Algorithms and Engineering Applications. Springer Science & Business Media, 2007

[2] Beck A: Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with MATLAB. Siam Press, 2014