Numerical Optimization: Penn State Math 555 Lecture Notes

Version 1.0.1

Christopher Griffin

With Contributions By:

Simon Miller

Douglas Mercer

Contents

List of Figures

Using These Notes

Stop! This is a set of lecture notes. It is not a book. Go away and come back when you have a real textbook on Numerical Optimization. Okay, do you have a book? Alright, let's move on then. This is a set of lecture notes for Math 555–Penn State's graduate Numerical Optimization course. Since I use these notes while I teach, there may be typographical errors that I noticed in class, but did not fix in the notes. If you see a typo, send me an e-mail and I'll add an acknowledgement. There may be many typos, that's why you should have a real textbook.

The lecture notes are loosely based on Nocedal and Wright's book *Numerical Optimization*, Avriel's text on Nonlinear Optimization, Bazaraa, Sherali and Shetty's book on Nonlinear Programming, Bazaraa, Jarvis and Sherali's book on Linear Programming and several other books that are cited in these notes. All of the books mentioned are good books (some great). The problem is, some books don't cover things in enough depth. The other problem is for students taking this course, this may be the first time they're seeing optimization, so we have to cover some preliminaries. Our Math 555 course should really be two courses: one on theory and the other on practical algorithms. Apparently we're not that interested, so we offer everything in one course.

This set of notes correct some of the problems I mention by presenting the material in a format for that can be used easily in Penn State in Math 555. These notes are probably really inappropriate if you have a strong Operations Research program. You'd be better off reading Nocedal and Wright's book directly.

In order to use these notes successfully, you should know something about multi-variable calculus. It also wouldn't hurt to have had an undergraduate treatment in optimization (in some form). At Penn State, the only prerequisite for this course is Math 456, which is a numerical methods course. That could be useful for some computational details, but I'll review everything that you'll need. That being said, I hope you enjoy using these notes!

One last thing: the algorithms in these notes were coded using Maple. I've also coded most of the algorithms in C++. The code will be posted (eventually – perhaps it's already there). Until then, you can e-mail me if you want the code. I can be reached at `griffin` 'at' `ieee.org`.

CHAPTER 1

Introduction to Optimization Concepts, Geometry and Matrix Properties

1. Optimality and Optimization

Definition 1.1. Let $z : D \subseteq \mathbb{R}^n \to \mathbb{R}$. The point $\mathbf{x}^*$ is a *global maximum* for z if for all $\mathbf{x} \in D$, $z(\mathbf{x}^*) \geq z(\mathbf{x})$. A point $\mathbf{x}^* \in D$ is a *local maximum* for z if there is a neighborhood $S \subseteq D$ of $\mathbf{x}^*$ (i.e., $\mathbf{x}^* \in S$) so that for all $\mathbf{x} \in S$, $z(\mathbf{x}^*) \geq z(\mathbf{x})$. When the foregoing inequalities are strict, $\mathbf{x}^*$ is called a strict global or local maximum.

Remark 1.2. Clearly Definition 1.1 is valid only for domains and functions where the concept of a neighborhood is defined and understood. In general, S must be a topologically connected set (as it is in a neighborhood in $\mathbb{R}^n$) in order for this definition to be used or at least we must be able to define the concept of *neighborhood* on the set.

Definition 1.3 (Maximization Problem). Let $z : D \subseteq \mathbb{R}^n \to \mathbb{R}$; for $i = 1, \dots, m$, $g_i : D \subseteq \mathbb{R}^n \to \mathbb{R}$; and for $j = 1, \dots, l$ $h_j : D \subseteq \mathbb{R}^n \to \mathbb{R}$ be functions. Then the general maximization problem with objective function $z(x_1, \dots, x_n)$ and *inequality constraints* $g_i(x_1, \dots, x_n) \leq b_i$ $(i = 1, \dots, m)$ and *equality constraints* $h_j(x_1, \dots, x_n) = r_j$ is written as:

$$
\left\{
\begin{aligned}
\max \;\; & z(x_1, \dots, x_n) \\
s.t. \;\; & g_1(x_1, \dots, x_n) \leq b_1 \\
& \qquad \vdots \\
& g_m(x_1, \dots, x_n) \leq b_m \\
& h_1(x_1, \dots, x_n) = r_1 \\
& \qquad \vdots \\
& h_l(x_1, \dots, x_n) = r_l
\end{aligned}
\right.
\tag{1.1}
$$

Remark 1.4. Expression 1.1 is also called a *mathematical programming problem.* Naturally when constraints are involved we define the global and local maxima for the objective function $z(x_1, \dots, x_n)$ in terms of the feasible region instead of the entire domain of z, since we are only concerned with values of $x_1, \dots, x_n$ that satisfy our constraints.

Remark 1.5. When there are no constraints (or the only constraint is that $(x_1, \dots, x_n) \in \mathbb{R}^n$), the problem is called an unconstrained maximization problem.

Example 1.6. Let's recall a simple optimization problem from differential calculus (Math 140): Goats are an environmentally friendly and inexpensive way to control a lawn when there are lots of rocks or lots of hills. (Seriously, both Google and some U.S. Navy bases use goats on rocky hills instead of paying lawn mowers!)

Suppose I wish to build a pen to keep some goats. I have 100 meters of fencing and I wish to build the pen in a rectangle with the largest possible area. How long should the sides of the rectangle be? In this case, making the pen *better* means making it have the largest possible area.

We can write this problem as:

$$\left\{\begin{array}{rl} \max & A(x,y) = xy \\ s.t. & 2x + 2y = 100 \\ & x \geq 0 \\ & y \geq 0 \end{array}\right. \tag{1.2}$$

Remark 1.7. It is easy to see that when we replace the objective function in Problem 1.1 with the objective function $-z(x_1, \dots, x_n)$, then the solution to this new problem minimizes $z(x_1, \dots, x_n)$ subject to the constraints of Problem 1.1. It therefore suffices to know how to solve either maximization or minimization problems, since one can be converted easily into the other.

2. Some Geometry for Optimization

Remark 1.8. We'll denote vectors in $\mathbb{R}^n$ in boldface. So $\mathbf{x} \in \mathbb{R}^n$ is an n-dimensional vector and we have $\mathbf{x} = (x_1, \dots, x_n)$. We'll always associate an n-dimensional vectors with a $n \times 1$ matrix (column vector) unless otherwise noted. Thus, when we write $\mathbf{x} \in \mathbb{R}^n$ we also mean $\mathbf{x} \in \mathbb{R}^{n\times 1}$ (the set of $n \times 1$ matrices with entries from $\mathbb{R}$).

Definition 1.9 (Dot Product). Recall that if $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ are two n-dimensional vectors, then the *dot product* (*scalar product*) is:

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i \tag{1.3}$$

where x_i is the i^{th} component of the vector $\mathbf{x}$. Clearly if $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{n\times 1}$, then

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y} \tag{1.4}$$

where $\mathbf{x}^T \in \mathbb{R}^{1\times n}$ is the transpose of $\mathbf{x}$ when treated as a matrix.

Lemma 1.10. *Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and let θ be the angle between $\mathbf{x}$ and $\mathbf{y}$, then*

$$\mathbf{x} \cdot \mathbf{y} = ||\mathbf{x}|| ||\mathbf{y}|| \cos\theta \tag{1.5}$$

□

Remark 1.11. The preceding lemma can be proved using the *law of cosines* from trigonometry. The following small lemma follows and is is proved as Theorem 1 of [MT03]:

Lemma 1.12. *Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. Then the following hold:*

1. *The angle between $\mathbf{x}$ and $\mathbf{y}$ is less than $\pi/2$ (i.e., acute) iff $\mathbf{x} \cdot \mathbf{y} > 0$.*
2. *The angle between $\mathbf{x}$ and $\mathbf{y}$ is exactly $\pi/2$ (i.e., the vectors are orthogonal) iff $\mathbf{x} \cdot \mathbf{y} = 0$.*
3. *The angle between $\mathbf{x}$ and $\mathbf{y}$ is greater than $\pi/2$ (i.e., obtuse) iff $\mathbf{x} \cdot \mathbf{y} < 0$.*

□

Lemma 1.13 (Schwartz Inequality). *Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. Then:*

$$(\mathbf{x}^T \mathbf{y})^2 \leq (\mathbf{x}^T \mathbf{x}) \cdot (\mathbf{y}^T \mathbf{y}) \tag{1.6}$$

This is equivalent to:

$$(\mathbf{x}^T\mathbf{y})^2 \leq ||\mathbf{x}||^2||\mathbf{y}||^2 \tag{1.7}$$

Definition 1.14 (Graph). Let $z : D \subseteq \mathbb{R}^n \to \mathbb{R}$ be function, then the *graph* of z is the set of $n+1$ tuples:

$$\{(\mathbf{x}, z(\mathbf{x})) \in \mathbb{R}^{n+1}|\mathbf{x} \in D\} \tag{1.8}$$

When $z : D \subseteq \mathbb{R} \to \mathbb{R}$, the graph is precisely what you'd expect. It's the set of pairs $(x, y) \in \mathbb{R}^2$ so that $y = z(x)$. This is the graph that you learned about back in Algebra 1.

Definition 1.15 (Level Set). Let $z : \mathbb{R}^n \to \mathbb{R}$ be a function and let $c \in \mathbb{R}$. Then the *level set of value c for function z* is the set:

$$\{\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n | z(\mathbf{x}) = c\} \subseteq \mathbb{R}^n \tag{1.9}$$

Example 1.16. Consider the function $z = x^2 + y^2$. The level set of z at 4 is the set of points $(x, y) \in \mathbb{R}^2$ such that:

$$x^2 + y^2 = 4 \tag{1.10}$$

You will recognize this as the equation for a circle with radius 4. We illustrate this in the following two figures. Figure 1.1 shows the level sets of z as they sit on the 3D plot of the function, while Figure 1.2 shows the level sets of z in $\mathbb{R}^2$. The plot in Figure 1.2 is called a *contour plot*.

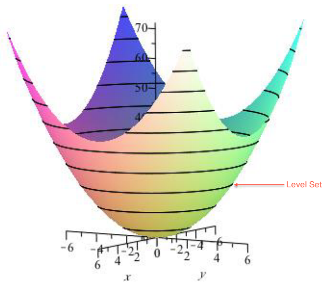


Figure 1.1. Plot with Level Sets Projected on the Graph of z. The level sets existing in $\mathbb{R}^2$ while the graph of z existing $\mathbb{R}^3$. The level sets have been projected onto their appropriate heights on the graph.

Definition 1.17. (Line) Let $\mathbf{x}_0, \mathbf{h} \in \mathbb{R}^n$. Then the *line* defined by vectors $\mathbf{x}_0$ and $\mathbf{h}$ is the function $\mathbf{l}(t) = \mathbf{x}_0 + t\mathbf{h}$. Clearly $l : \mathbb{R} \to \mathbb{R}^n$. The vector $\mathbf{h}$ is called the direction of the line.

Example 1.18. Let $\mathbf{x_0} = (2, 1)$ and let $\mathbf{h} = (2, 2)$. Then the line defined by $\mathbf{x_0}$ and $\mathbf{h}$ is shown in Figure 1.3. The set of points on this line is the set $L = \{(x, y) \in \mathbb{R}^2 : x = 2 + 2t, y = 1 + 2t, t \in \mathbb{R}\}$.

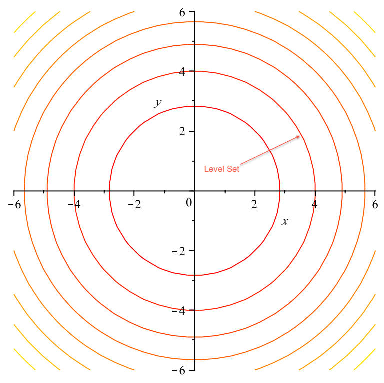


Figure 1.2. Contour Plot of $z = x^2 + y^2$. The circles in $\mathbb{R}^2$ are the level sets of the function. The lighter the circle hue, the higher the value of c that defines the level set.

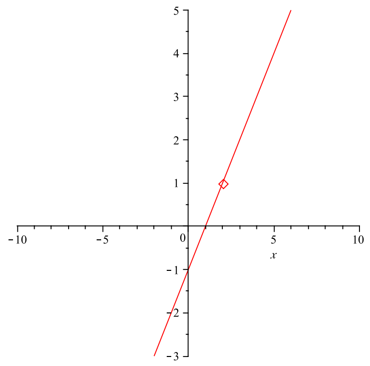


Figure 1.3. A Line Function: The points in the graph shown in this figure are in the set produced using the expression $\mathbf{x}_0 + \mathbf{h}t$ where $\mathbf{x_0} = (2, 1)$ and let $\mathbf{h} = (2, 2)$.

Definition 1.19 (Directional Derivative). Let $z : \mathbb{R}^n \to \mathbb{R}$ and let $\mathbf{h} \in \mathbb{R}^n$ be a vector (direction) in n-dimensional space. Then the directional derivative of z at point $\mathbf{x}_0 \in \mathbb{R}^n$ in the direction of $\mathbf{h}$ is

$$\left.\frac{d}{dt} z(\mathbf{x}_0 + t\mathbf{h})\right|_{t=0} \tag{1.11}$$

when this derivative exists.

Proposition 1.20. *The directional derivative of z at $\mathbf{x}_0$ in the direction $\mathbf{h}$ is equal to:*

$$\lim_{h \to 0} \frac{z(\mathbf{x}_0 + h\mathbf{h}) - z(\mathbf{x}_0)}{h} \tag{1.12}$$

Exercise 1. Prove Proposition 1.20. [Hint: Use the definition of derivative for a univariate function and apply it to the definition of directional derivative and evaluate $t = 0$.]

Definition 1.21 (Gradient). Let $z : \mathbb{R}^n \to \mathbb{R}$ be function and let $\mathbf{x}_0 \in \mathbb{R}^n$. Then the *gradient* of z at $\mathbf{x}_0$ is the vector in $\mathbb{R}^n$ given by:

$$\nabla z(\mathbf{x}_0) = \left(\frac{\partial z}{\partial x_1}(\mathbf{x}_0), \dots, \frac{\partial z}{\partial x_n}(\mathbf{x}_0)\right) \tag{1.13}$$

Gradients are extremely important concepts in optimization (and vector calculus in general). Gradients have many useful properties that can be exploited. The relationship between the directional derivative and the gradient is of critical importance.

THEOREM 1.22. *If $z : \mathbb{R}^n \rightarrow \mathbb{R}$ is differentiable, then all directional derivatives exist. Furthermore, the directional derivative of z at $\mathbf{x}_0$ in the direction of $\mathbf{h}$ is given by:*

$$\nabla z(\mathbf{x}_0) \cdot \mathbf{h} \tag{1.14}$$

where $\cdot$ denotes the dot product of two vectors.

PROOF. Let $\mathbf{l}(t) = \mathbf{x}_0 + \mathbf{h}t$. Then $\mathbf{l}(t) = (l_1(t), \dots, l_n(t))$; that is, $\mathbf{l}(t)$ is a vector function whose i^{th} component is given by $l_i(t) = \mathbf{x}_{0_i} + \mathbf{h}_i t$.

Apply the chain rule:

$$\frac{dz(\mathbf{l}(t))}{dt} = \frac{\partial z}{\partial l_1}\frac{dl_1}{dt} + \cdots + \frac{\partial z}{\partial l_n}\frac{dl_n}{dt} \tag{1.15}$$

Thus:

$$\frac{d}{dt}z(\mathbf{l}(t)) = \nabla z \cdot \frac{d\mathbf{l}}{dt} \tag{1.16}$$

Clearly $d\mathbf{l}/dt = \mathbf{h}$. We have $\mathbf{l}(0) = \mathbf{x}_0$. Thus:

$$\left.\frac{d}{dt}z(\mathbf{x}_0 + t\mathbf{h})\right|_{t=0} = \nabla z(\mathbf{x}_0) \cdot \mathbf{h} \tag{1.17}$$

□

We now come to the two most important results about gradients, (i) the fact that they always point in the direction of steepest ascent with respect to the level curves of a function and (ii) that they are perpendicular (normal) to the level curves of a function. We can exploit this fact as we seek to maximize (or minimize) functions.

THEOREM 1.23. *Let $z : \mathbb{R}^n \rightarrow \mathbb{R}$ be differentiable, $\mathbf{x}_0 \in \mathbb{R}^n$. If $\nabla z(\mathbf{x}_0) \neq 0$, then $\nabla z(\mathbf{x}_0)$ points in the direction in which z is increasing fastest.*

PROOF. Recall $\nabla z(\mathbf{x}_0) \cdot \mathbf{h}$ is the directional derivative of z in direction $\mathbf{h}$ at $\mathbf{x}_0$. Assume that $\mathbf{h}$ is a unit vector. We know that:

$$\nabla z(\mathbf{x}_0) \cdot \mathbf{h} = ||\nabla z(\mathbf{x}_0)|| \cos\theta \tag{1.18}$$

(because we assumed $\mathbf{h}$ was a unit vector) where θ is the angle between the vectors $\nabla z(\mathbf{x}_0)$ and $\mathbf{h}$. The function $\cos\theta$ is largest when $\theta = 0$, that is when $\mathbf{h}$ and $\nabla z(\mathbf{x}_0)$ are parallel vectors. (If $\nabla z(\mathbf{x}_0) = 0$, then the directional derivative is zero in all directions.) □

THEOREM 1.24. *Let $z : \mathbb{R}^n \rightarrow \mathbb{R}$ be differentiable and let $\mathbf{x}_0$ lie in the level set S defined by $z(\mathbf{x}) = k$ for fixed $k \in \mathbb{R}$. Then $\nabla z(\mathbf{x}_0)$ is normal to the set S in the sense that if $\mathbf{h}$ is a tangent vector at $t = 0$ of a path $\mathbf{c}(t)$ contained entirely in S with $\mathbf{c}(0) = \mathbf{x}_0$, then $\nabla z(\mathbf{x}_0) \cdot \mathbf{h} = 0$.*

REMARK 1.25. Before giving the proof, we illustrate this theorem in Figure 1.4. The function is $z(x, y) = x^4 + y^2 + 2xy$ and $\mathbf{x}_0 = (1, 1)$. At this point $\nabla z(\mathbf{x}_0) = (6, 4)$. We include the tangent line to the level set at the point (1,1) to illustrate the normality of the gradient to the level curve at the point.

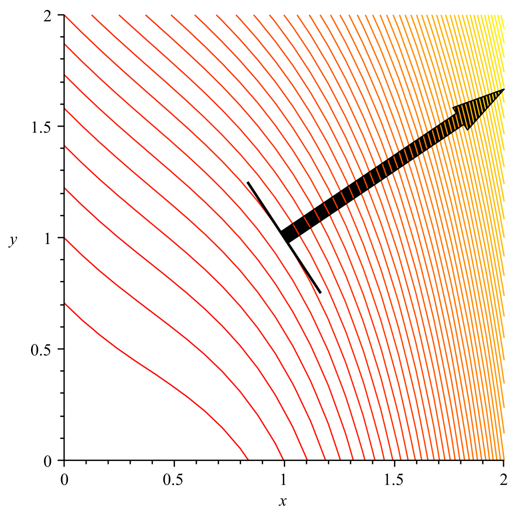


Figure 1.4. A Level Curve Plot with Gradient Vector: We've scaled the gradient vector in this case to make the picture understandable. Note that the gradient is perpendicular to the level set curve at the point $(1, 1)$, where the gradient was evaluated. You can also note that the gradient is pointing in the direction of steepest ascent of $z(x, y)$.

Proof. As stated, let $\mathbf{c}(t)$ be a curve in S. Then $\mathbf{c} : \mathbb{R} \to \mathbb{R}^n$ and $z(\mathbf{c}(t)) = k$ for all $t \in \mathbb{R}$. Let $\mathbf{h}$ be the tangent vector to $\mathbf{c}$ at $t = 0$; that is:

$$\left.\frac{d\mathbf{c}(t)}{dt}\right|_{t=0} = \mathbf{h} \tag{1.19}$$

Differentiating $z(\mathbf{c}(t))$ with respect to t using the chain rule and evaluating at $t = 0$ yields:

$$\left.\frac{d}{dt}z(\mathbf{c}(t))\right|_{t=0} = \nabla z(\mathbf{c}(0)) \cdot \mathbf{h} = \nabla z(\mathbf{x}_0) \cdot \mathbf{h} = 0 \tag{1.20}$$

Thus $\nabla z(\mathbf{x}_0)$ is perpendicular to $\mathbf{h}$ and thus normal to the set S as required. □

3. Matrix Properties for Optimization

Definition 1.26 (Definiteness). A matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ is *positive semi-definite* if for all $\mathbf{x} \in \mathbb{R}^n$ with $\mathbf{x} \neq \mathbf{0}$, $\mathbf{x}^T\mathbf{M}\mathbf{x} \geq 0$. The matrix $\mathbf{M}$ is *positive definite* if for all $\mathbf{x} \in \mathbb{R}^n$ with $\mathbf{x} \neq \mathbf{0}$, $\mathbf{x}^T\mathbf{M}\mathbf{x} > 0$. The matrix M is negative definite if $-M$ is positive definite and negative semi-definite if $-M$ is positive semi-definite. If $\mathbf{M}$ satisfies none of these properties, then $\mathbf{M}$ is indefinite.

Remark 1.27. We note that this is not the most general definition of matrix definiteness. In general, matrix definiteness can be defined for complex matrices and has specialization to Hermitian matrices.

Remark 1.28. Positive semi-definiteness is also called non-negative definiteness and negative semi-definiteness is also called non-positive definiteness.

Lemma 1.29. *A matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ is positive definite if and only if for every vector $\mathbf{x} \in \mathbb{R}^n$, there is an $\alpha \in \mathbb{R}_+$ (that is $\alpha > 0$) such that $\mathbf{x}^T\mathbf{M}\mathbf{x} > \alpha\mathbf{x}^T\mathbf{x}$*[1].

[1]Thanks to Doug Mercer for point out that α needs to be positive.

Exercise 2. Prove Lemma 1.29.

Lemma 1.30. *Suppose that* $\mathbf{M}$ *is positive definite. If* $\lambda \in \mathbb{R}$ *is a real eigenvalue of* $\mathbf{M}$ *with real eigenvector* $\mathbf{x}$, *then* $\lambda > 0$.

Exercise 3. Prove Lemma 1.30.

Lemma 1.31. *Prove that* $\mathbf{M} \in \mathbb{R}^{n\times n}$ *has eigenvalue* λ *if and only if* $\mathbf{M}^{-1}$ *has eigenvalue* $1/\lambda$.

Exercise 4. Prove Lemma 1.31.

Remark 1.32. The theory of eigenvalues and eigenvectors of matrices is deep and well understood. A substantial part of this theory should be covered in Math 436, for those interested. The following two results are proved in several different sources. One very nice proof is in Chapter 8 of [GR01]. Unfortunately, the proofs are well outside the scope of the class.

Theorem 1.33 (Spectral Theorem for Real Symmetric Matrices). *Let* $\mathbf{M} \in \mathbb{R}^{n\times n}$ *be a symmetric matrix. Then the eigenvalues of* $\mathbf{M}$ *are all real.* □

Theorem 1.34 (Principle Axis Theorem). *Let* $\mathbf{M} \in \mathbb{R}^{n\times n}$ *be a symmetric matrix. Then* $\mathbb{R}^n$ *has an orthongonal basis consisting of eigenvectors of* $\mathbf{M}$. □

Lemma 1.35. *Let* $\mathbf{M} \in \mathbb{R}^{n\times n}$ *be a symmetric matrix and suppose that* $\lambda_1 \geq \cdots \geq \lambda_n$ *are the eigenvalues of* $\mathbf{M}$. *If* $\mathbf{x} \in \mathbb{R}^n$ *is a vector with* $||\mathbf{x}|| = 1$, *then* $\mathbf{x}^T\mathbf{M}\mathbf{x} \leq \lambda_1$ *and* $\mathbf{x}^T\mathbf{M}\mathbf{x} \geq \lambda_n$.

Proof. From the Principle Axis Theorem, we know that that there is a basis $\mathcal{B} = \{\mathbf{v}_1, \dots \mathbf{v}_n\}$ consisting of eigenvectors of $\mathbf{M}$. For any $\mathbf{x} \in \mathbb{R}^n$, we can write:

$$\mathbf{x} = \left(\mathbf{x}^T\mathbf{v}_1\right)\mathbf{v}_1 + \cdots + \left(\mathbf{x}^T\mathbf{v}_n\right)\mathbf{v}_n \tag{1.21}$$

That is, in the coordinates of $\mathcal{B}$:

$$\mathbf{x} = \left[\left(\mathbf{x}^T\mathbf{v}_1\right), \dots, \left(\mathbf{x}^T\mathbf{v}_n\right)\right]^T$$

It follows then that:

$$\mathbf{M}\mathbf{x} = \left(\mathbf{x}^T\mathbf{v}_1\right)\mathbf{M}\mathbf{v}_1 + \cdots + \left(\mathbf{x}^T\mathbf{v}_n\right)\mathbf{M}\mathbf{v}_n = \left(\mathbf{x}^T\mathbf{v}_1\right)\lambda_1\mathbf{v}_1 + \cdots + \left(\mathbf{x}^T\mathbf{v}_n\right)\lambda_n\mathbf{v}_n = \\ \lambda_1\left(\mathbf{x}^T\mathbf{v}_1\right)\mathbf{v}_1 + \cdots + \lambda_n\left(\mathbf{x}^T\mathbf{v}_n\right)\mathbf{v}_n$$

Pre-multiplying by $\mathbf{x}^T$ yields:

$$\mathbf{x}^T\mathbf{M}\mathbf{x} = \lambda_1\left(\mathbf{x}^T\mathbf{v}_1\right)\mathbf{x}^T\mathbf{v}_1 + \cdots + \lambda_n\left(\mathbf{x}^T\mathbf{v}_n\right)\mathbf{x}^T\mathbf{v}_n$$

From Equation 1.21 and our assumption on $||\mathbf{x}||$, we see that[2]:

$$1 = ||\mathbf{x}||^2 = \left(\mathbf{x}^T\mathbf{v}_1\right)^2 + \cdots + \left(\mathbf{x}^T\mathbf{v}_n\right)^2$$

Since $\lambda_1 \geq \cdots \geq \lambda_n$, we have:

$$\mathbf{x}^T\mathbf{M}\mathbf{x} = \lambda_1\left(\mathbf{x}^T\mathbf{v}_1\right)^2 + \cdots + \lambda_n\left(\mathbf{x}^T\mathbf{v}_n\right)^2 \leq \lambda_1\left(\mathbf{x}^T\mathbf{v}_1\right)^2 + \cdots + \lambda_1\left(\mathbf{x}^T\mathbf{v}_n\right)^2 = \lambda_1$$

The fact that $\mathbf{x}^T\mathbf{M}\mathbf{x} \geq \lambda_n$ follows by a similar argument. This completes the proof. □

[2]If this is not obvious, note that the length of a vector is an invariant. That is, it is the same no matter in which basis the vector is expressed.

THEOREM 1.36. *Suppose* $\mathbf{M} \in \mathbb{R}^{n\times n}$ *is symmetric. If every eigenvalue of* $\mathbf{M}$ *is positive, then* $\mathbf{M}$ *is positive definite.*

PROOF. By the spectral theorem, all the eigenvalues of $\mathbf{M}$ are real and thus we can write them in order as: $\lambda_1 \geq \cdots \geq \lambda_n$. By Lemma 1.35, we know that $\mathbf{x}^T\mathbf{M}\mathbf{x} \geq \lambda_n$. By our assumption $\lambda_n > 0$ and thus $\mathbf{x}^T\mathbf{M}\mathbf{x} > 0$ for all $\mathbf{x} \in \mathbb{R}^n$. ☐

COROLLARY 1.37. *If* $\mathbf{M}$ *is a positive definite matrix, then its inverse is also positive definite.*

PROOF. Applying Lemma 1.31 and the preceding theorem yields the result. ☐

REMARK 1.38. The proofs of the preceding two results can be found in [**Ant94**] (and most likely later editions). These are fairly standard proofs though and are available in most complete linear programming texts.

REMARK 1.39. Thus we've proved that a symmetric real matrix is positive definite if and only if every eigenvalue is positive. We can use this fact to obtain a simple test for positive definiteness[3].

DEFINITION 1.40 (Gerschgorin Disc). Let $\mathbf{A} \in \mathbb{C}^{n\times n}$ and let

$$R_i = \sum_{j\neq i} |A_{ij}|$$

That is, the sum of the off-diagonal entries of row i. Let $D(\mathbf{A}_{ii}, R_i)$ be a closed disc centered at $\mathbf{A}_{ii}$ with radius R_i. This is a Gerschgorin disc. When $\mathbf{A} \in \mathbb{R}^{n\times n}$, this disc is a closed interval.

REMARK 1.41. The following theorem was first proved in [**Ger31**] and is now a classical matrix theorem, used frequently in control theory.

THEOREM 1.42 (Gerschgorin Disc Theorem). *Let* $\mathbf{A} \in \mathbb{C}^{n\times n}$. *Every eigenvalue of* $\mathbf{A}$ *lies in a Gerschgorin disc.* ☐

COROLLARY 1.43. *Let* $\mathbf{M} \in \mathbb{R}^{n\times n}$ *be symmetric. If*

$$\mathbf{A}_{ii} > \sum_{j\neq i} |A_{ij}| \tag{1.22}$$

then $\mathbf{M}$ *is positive definite.*

EXERCISE 5. Prove Corollary 1.43.

EXERCISE 6. Construct an example in which the converse of Corollary 1.43 does not hold or prove that it must hold.

REMARK 1.44. The last theorem of this section, which we will not prove, shows that real symmetric positive definite matrices have a special unique decomposition (called the Cholesky decomposition). (Actually, positive definite Hermetian matrices have a unique Cholesky decomposition and this generalizes the result on real symmetric matrices that are positive definite.) A proof can be found in Chapter 2 of [**Dem97**].

THEOREM 1.45. *A* $\mathbf{M} \in \mathbb{R}^{n\times n}$ *is symmetric positive definite if and only if there is a (unique) lower triangular matrix* $\mathbf{H} \in \mathbb{R}^{n\times n}$ *such that* $\mathbf{M} = \mathbf{H}\mathbf{H}^T$.

[3]Thanks for Xiao Chen for pointing out we needed the matrix to be symmetric.

Definition 1.46. The previous matrix factorization is called the *Cholesky Decomposition*. It serves as a test for positive definiteness in symmetric matrices that is less computationally complex than computing the eigenvalues of the matrix.

Remark 1.47. We note that Theorem 1.45 could be written to say there is an upper triangular matrix $\mathbf{H}$ so that $\mathbf{M} = \mathbf{H}\mathbf{H}^T$. Some numerical systems implement this type of Cholesky decomposition, e.g., Mathematica. The example and code we provide computes the lower triangular matrix variation.

Example 1.48 (The Cholesky Decomposition). There is a classic algorithm for computing the Cholesky decomposition, which is available in [Dem97] (Algorithm 2.11). Instead of using the algorithm, we illustrate the algorithm with a more protracted example.

Consider the matrix:

$$\mathbf{M} = \begin{bmatrix} 8 & -2 & 7 \\ -2 & 10 & 5 \\ 7 & 5 & 12 \end{bmatrix}$$

We can construct the LU decomposition using Gaussian elimination without pivoting to obtain:

$$\begin{bmatrix} 8 & -2 & 7 \\ -2 & 10 & 5 \\ 7 & 5 & 12 \end{bmatrix} = \begin{bmatrix} 8 & 0 & 0 \\ -2 & 19/2 & 0 \\ 7 & \frac{27}{4} & \frac{41}{38} \end{bmatrix} \cdot \begin{bmatrix} 1 & -1/4 & \frac{7}{8} \\ 0 & 1 & \frac{27}{38} \\ 0 & 0 & 1 \end{bmatrix}$$

Notice the determinant of the U matrix is 1. The L matrix can be written as $L = L_1 D$ where D is a diagonal matrix whose determinant is equal to the determinant of M and L_1 is a lower triangular matrix with determinant 1. This yields:

$$\begin{bmatrix} 8 & -2 & 7 \\ -2 & 10 & 5 \\ 7 & 5 & 12 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ \frac{7}{8} & \frac{27}{38} & 1 \end{bmatrix} \cdot \begin{bmatrix} 8 & 0 & 0 \\ 0 & 19/2 & 0 \\ 0 & 0 & \frac{41}{38} \end{bmatrix} \cdot \begin{bmatrix} 1 & -1/4 & \frac{7}{8} \\ 0 & 1 & \frac{27}{38} \\ 0 & 0 & 1 \end{bmatrix}$$

Notice $L_1^T = U$ and more importantly, notice that the diagonal of $\mathbf{D}$ is composed of ratios of the principal minors of $\mathbf{M}$. That is:

$$\begin{aligned} 8 &= |8| \\ 76 &= \begin{vmatrix} 8 & -2 \\ -2 & 10 \end{vmatrix} \\ 82 &= \begin{vmatrix} 8 & -2 & 7 \\ -2 & 10 & 5 \\ 7 & 5 & 12 \end{vmatrix} \end{aligned}$$

And we have $76/8 = 19/2$ and $82/76 = 41/38$. Thus, if we multiply the diagonal elements of the matrix $\mathbf{D}$ we easily see that this yields 82, the determinant of $\mathbf{M}$. Finally, define:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ \frac{7}{8} & \frac{27}{38} & 1 \end{bmatrix} \cdot \begin{bmatrix} 2\sqrt{2} & 0 & 0 \\ 0 & 1/2\sqrt{38} & 0 \\ 0 & 0 & 1/38\sqrt{1558} \end{bmatrix} =$$

$$\begin{bmatrix} 2\sqrt{2} & 0 & 0 \\ -1/2\sqrt{2} & 1/2\sqrt{38} & 0 \\ 7/4\sqrt{2} & \frac{27}{76}\sqrt{38} & 1/38\sqrt{1558} \end{bmatrix}$$

This is the Cholesky factor, where the second matrix in the product is obtained by taking the square root of the diagonal elements of $\mathbf{D}$. It is easy to see now that $\mathbf{M} = \mathbf{H}\mathbf{H}^T$. The fact that the determinants of the principal minors of $\mathbf{M}$ are always positive is a direct result of the positive definiteness of $\mathbf{M}$.

This is naturally not the most efficient way to compute the Cholesky decomposition, but it is illustrative. The Maple code for the standard algorithm for computing the Cholesky decomposition is shown in Algorithm 1.

Exercise 7. Determine the computational complexity of Algorithm 1. Compare it to the computational complexity of the steps shown in Example 1.48. Use Algorithm 1 to confirm the Cholesky decomposition from Example 1.48.

```
CholeskyDecomp := proc (M::Matrix, n::integer)::list;
  local MM, H, ret, i, j, k;
  MM := Matrix(M);
  H := Matrix(n, n);
  ret := true;
  for i to n do
    if evalb(not ret) then
      ret := false;
      break:
    end if:
    if 0 <= MM[i, i] then
      H[i, i] := sqrt(MM[i, i])
    else
      ret := false;
      break;
    end if;
    for j from i+1 to n do
      if H[i, i] = 0 then
        ret := false;
        break
      end if;
      H[j, i] := MM[j, i]/H[i, i];
      for k from i+1 to j do
        MM[j, k] := MM[j, k]-H[j, i]*H[k, i]
      end do
    end do
  end do;
  if ret then
    [H, ret]
  else
    [M, ret]
  end if
end proc:
```

Algorithm 1. Cholesky Decomposition

CHAPTER 2

Fundamentals of Unconstrained Optimization

1. Mean Value and Taylor's Theorems

Remark 2.1. We state, without proof, some classic theorems from multi-variable calculus. The proofs of these theorems are not terribly hard, but they build on several other results from single variable calculus and reviewing all these results is tedious and outside the scope of the course.

Lemma 2.2 (Mean Value Theorem). *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. (That is $f \in C^1$.) Let $\mathbf{x}_0, \mathbf{h} \in \mathbb{R}^n$. Then there is a $t \in (0,1)$ such that:*

$$f(\mathbf{x}_0 + \mathbf{h}) - f(\mathbf{x}_0) = \nabla f(\mathbf{x}_0 + t\mathbf{h})^T\mathbf{h} \tag{2.1}$$

□

Remark 2.3. This is the natural generalization of the one-variable mean value theorem from calculus (Math 140 at Penn State). The expression $\mathbf{x}_0 + t\mathbf{h}$ is simply the line segment connecting $\mathbf{x}_0$ and $\mathbf{x}_0 + \mathbf{h}$. If we imagine this being the "x-axis" and the corresponding slice of $f(\cdot)$, then we can see that we're just applying the single variable mean value theorem to this slice.

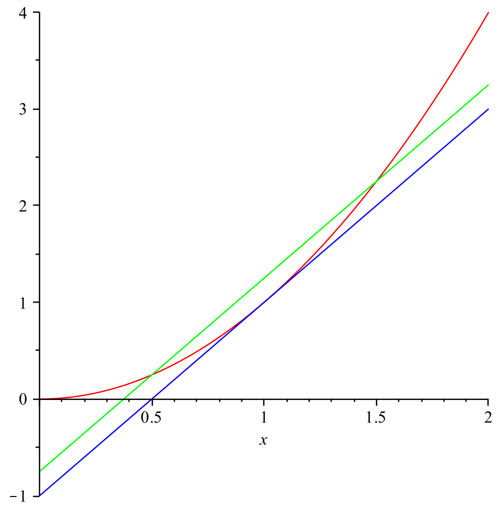


Figure 2.1. An illustration of the mean value theorem in one variable. The multi-variable mean value theorem is simply an application of the single variable mean value theorem applied to a slice of a function.

Lemma 2.4 (Second Order Mean Value Theorem). *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable. (That is $f \in C^2$.) Let $\mathbf{x}_0, \mathbf{h} \in \mathbb{R}^n$. Then there is a $t \in (0,1)$*

such that:

$$f(\mathbf{x}_0 + \mathbf{h}) - f(\mathbf{x}_0) = \nabla f(\mathbf{x}_0)^T\mathbf{h} + \frac{1}{2}\mathbf{h}^T\nabla^2 f(\mathbf{x}_0 + t\mathbf{h})\mathbf{h} \tag{2.2}$$ □

Lemma 2.5 (Mean Value Theorem – Vector Valued Function). *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a differentiable function and let $\mathbf{x}_0, \mathbf{h} \in \mathbb{R}^n$. Then:*

$$f(\mathbf{x}_0 + \mathbf{h}) - f(\mathbf{x}_0) = \int_0^1 \mathbf{D}f(\mathbf{x}_0 + t\mathbf{h})\mathbf{h}dt \tag{2.3}$$

where $\mathbf{D}$ is the Jacobian operator. □

Remark 2.6. It should be noted that there is no exact analog of the mean value theorem for vector valued functions. The previous lemma is the closes thing to such an analog and it is generally referred to as such.

Corollary 2.7. *Let $f : \mathbb{R}^n \to \mathbb{R}$ with $f \in C^2$ and let $\mathbf{x}_0, \mathbf{h} \in \mathbb{R}^n$. Then:*

$$\nabla f(\mathbf{x}_0 + \mathbf{h}) - \nabla f(\mathbf{x}_0) = \int_0^1 \nabla^2 f(\mathbf{x}_0 + t\mathbf{h})\mathbf{h}dt \tag{2.4}$$ □

Remark 2.8. The proof of this lemma rests on the single variable mean value theorem and the mean value theorem for integrals in single variable calculus.

Lemma 2.9 (Taylor's Theorem – Second Order). *Let $f : \mathbb{R}^n \to \mathbb{R}$ with $f \in C^2$ and let $\mathbf{x}_0, \mathbf{h} \in \mathbb{R}^n$. Then:*

$$f(\mathbf{x}_0 + \mathbf{h}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T\mathbf{h} + R_2(\mathbf{x}_0, \mathbf{h}) \tag{2.5}$$

where:

$$R_2(\mathbf{x}_0, \mathbf{h}) = \frac{1}{2}\mathbf{h}^T\nabla^2 f(\mathbf{x}_0 + t\mathbf{h})\mathbf{h} \tag{2.6}$$

for some $t \in (0, 1)$ or:

$$R_2(\mathbf{x}_0, \mathbf{h}) = \int_0^1 (1-t)\nabla^2 f(\mathbf{x}_0 + t\mathbf{h})\mathbf{h}dt \tag{2.7}$$ □

Remark 2.10. Taylor's Theorem in the general case considers functions in C^k. One can convert between the two forms of the remainder using the mean value theorem, though this is not immediately obvious without some concentration. Most of the proofs of the remainder term use the mean value theorems. There is a very nice proof, very readable proof of Taylor's theorem in [**MT03**] (Chapter 3.2).

Remark 2.11. From Taylor's theorem, we obtain first and second order approximations for functions. That is, the first order approximation for $f(\mathbf{x}_0 + \mathbf{h})$ is:

$$f(\mathbf{x}_0 + \mathbf{h}) \sim f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T\mathbf{h} \tag{2.8}$$

while the second order approximation is:

$$f(\mathbf{x}_0 + \mathbf{h}) \sim f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T\mathbf{h} + \frac{1}{2}\mathbf{h}^T\nabla^2 f(\mathbf{x}_0)\mathbf{h} \tag{2.9}$$

We will use these approximations and Taylor's theorem repeatedly throughout this set of notes.

2. Necessary and Sufficient Conditions for Optimality

THEOREM 2.12. *Suppose that $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ is differentiable in an open neighborhood of a local maximizer $\mathbf{x}^* \in D$, then $\nabla f(\mathbf{x}^*) = \mathbf{0}$.*

PROOF. By way of contradiction, suppose that $\nabla f(\mathbf{x}^*) \neq \mathbf{0}$. The differentiability of f implies that $\nabla f(\mathbf{x})$ is continuous in the open neighborhood of $\mathbf{x}^*$. Thus, for all ϵ, there is a δ so that if $||\mathbf{x}^* - \mathbf{x}|| < \delta$, then $||\nabla f(\mathbf{x}^*) - \nabla f(\mathbf{x})|| < \epsilon$. Let $\mathbf{h} = \nabla f(\mathbf{x}^*)$. Trivially, $\mathbf{h}^T\mathbf{h} > 0$ and (by continuity), for some $t \in (0, 1)$ (perhaps very small), we know that:

$$\mathbf{h}^T \nabla f(\mathbf{x}^* + t\mathbf{h}) > 0 \tag{2.10}$$

Let $\mathbf{p} = t\mathbf{h}$. From the mean value theorem, we know there is an $s \in (0, 1)$ such that:

$$f(\mathbf{x}_0 + \mathbf{p}) - f(\mathbf{x}_0) = \nabla f(\mathbf{x}_0 + s\mathbf{p})^T \mathbf{p} > 0$$

by our previous argument. Thus, $\mathbf{x}^*$ is not a (local) maximum. □

EXERCISE 8. In the last step of the previous proof, we assert the existence of a $t \in (0, 1)$ so that Equation 2.10 holds. Explicitly prove such a t must exist. [Hint: Use a component by component argument with the continuity of ∇f.]

EXERCISE 9. Construct an analagous proof for the statement: *Suppose that $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ is differentiable in an open neighborhood of a local minimizer $\mathbf{x}^* \in D$, then $\nabla f(\mathbf{x}^*) = \mathbf{0}$.*

EXERCISE 10. Construct an alternate proof of Theorem 2.12 by studying the single-variable function $g(t) = f(\mathbf{x} + t\mathbf{h})$. You may use the fact from Math 140 that $g'(t^*) = 0$ is a necessary condition for a local maxima in the one dimensional case. (Extra credit if you prove that fact as well.)

THEOREM 2.13. *Suppose that $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ is twice differentiable in an open neighborhood of a local maximizer $\mathbf{x}^* \in D$, then $\nabla f(\mathbf{x}^*) = \mathbf{0}$ and $\mathbf{H}(\mathbf{x}^*) = \nabla^2 f(\mathbf{x}^*)$ is negative-semidefinite.*

PROOF. From Theorem 2.12 $\nabla f(\mathbf{x}^*) = \mathbf{0}$. Our assumption that f is twice differentiable implies that $\mathbf{H}(\mathbf{x})$ is continuous and therefore for all ϵ there exists a δ so that $||\mathbf{x}^* - \mathbf{x}|| < \delta$ then $|\mathbf{H}_{ij}(\mathbf{x}^*) - \mathbf{H}_{ij}(\mathbf{x})| < \epsilon$ for all $i = 1, \dots, n$ and $j = 1, \dots n$. We are just asserting pointwise continuity for the elements of the matrix $\mathbf{H}(\mathbf{x})$.

Suppose we may choose a vector $\mathbf{h}$ so that $\mathbf{h}^T\mathbf{H}(\mathbf{x}^*)\mathbf{h} > 0$ and thus $\mathbf{H}(\mathbf{x}^*)$ is not negative semidefinite. Then by our continuity argument, we can choose an $\mathbf{h}$ with norm small enough, so we can assure that $\mathbf{h}^T\mathbf{H}(\mathbf{x}^* + \mathbf{h})\mathbf{h} > 0$. From Lemma 2.9 we have for some $t \in (0, 1)$:

$$f(\mathbf{x}^* + \mathbf{h}) - f(\mathbf{x}^*) = \frac{1}{2}\mathbf{h}^T\mathbf{H}(\mathbf{x}^* + t\mathbf{h})\mathbf{h} \tag{2.11}$$

since $\nabla f(\mathbf{x}^*) = \mathbf{0}$. Then it follows that $f(\mathbf{x}_0 + \mathbf{h}) - f(\mathbf{x}_0) > 0$ and thus we have found a direction in which the value of f increases and $\mathbf{x}^*$ cannot be a local maximum. □

THEOREM 2.14. *Suppose that $f : D \subset \mathbb{R}^n \to \mathbb{R}$ is twice differentiable. If $\nabla f(\mathbf{x}^*) = 0$ and $\mathbf{H}(\mathbf{x}^*) = \nabla^2 f(\mathbf{x}^*)$ is negative definite, then $\mathbf{x}^*$ is a local maximum.*

PROOF. Applying Lemma 2.9, we know for any $h \in \mathbb{R}^n$, there is a $t \in (0, 1)$ so that:

$$f(\mathbf{x}^* + \mathbf{h}) = f(\mathbf{x}^*) + \frac{1}{2}\mathbf{h}^T\nabla^2 f(\mathbf{x}^* + t\mathbf{h})\mathbf{h} \tag{2.12}$$

By the same argument as in the proof of Theorem 2.13, we know that there is an $\epsilon > 0$ so that if $|\mathbf{h}| < \epsilon$ then for all $t \in (0, 1)$, $\nabla^2 f(\mathbf{x}^* + t\mathbf{h})$ is negative definite if $\nabla^2 f(\mathbf{x}^*)$ is negative definite. Let $B_\epsilon(\mathbf{x}^*)$ the open ball centered at $\mathbf{x}^*$ with radius ϵ.

Thus we can see that for all $\mathbf{x} \in B_\epsilon(\mathbf{x}^*)$:

$$\frac{1}{2}\mathbf{h}^T\nabla^2 f(\mathbf{x})\mathbf{h} < 0$$

where $\mathbf{x} = \mathbf{x}^* + t\mathbf{h}$ for some appropriately chosen $\mathbf{h}$ and $t \in (0, 1)$. Equation 2.12 combined with the previous observation shows that for all $\mathbf{x} \in B_\epsilon(\mathbf{x}^*)$, $f(\mathbf{x}) < f(\mathbf{x}^*)$ and thus $\mathbf{x}^*$ is a local maximum. This completes the proof. □

Exercise 11. Theorem 2.14 provides sufficient conditions for $\mathbf{x}^*$ to be a *strict* local minimum. Give an example showing that the conditions are not necessary.

Example 2.15. The importance of the Cholesky Decomposition is now apparent. To confirm that a point is a local maximum, we must simply check that the Hessian at the point in question is negative definite, which can be done with the Cholesky Decomposition on the negative of the Hessian. Local minima can be verified by checking for minima. Consider the function:

$$(1-x)^2 + 100\,\left(y - x^2\right)^2$$

The gradient of this function is:

$$[-2 + 2\,x - 400\,\left(y - x^2\right)x, 200\,y - 200\,x^2]^T$$

While the Hessian of this function is:

$$\begin{bmatrix} 2 - 400\,y + 1200\,x^2 & -400\,x \\ -400\,x & 200 \end{bmatrix}$$

At point $x = 1, y = 1$, it is easy to see that the gradient is $\mathbf{0}$, while the Hessian is:

$$\begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}$$

This matrix has a Cholesky decomposition:

$$\begin{bmatrix} \sqrt{802} & 0 \\ -\frac{200}{401}\sqrt{802} & \frac{10}{401}\sqrt{802} \end{bmatrix}$$

Since the Hessian is positive definite, we know that $x = 1, y = 1$ must be a local minima for this function. As it happens, this is a global minima for this function.

3. Concave/Convex Functions and Convex Sets

Definition 2.16 (Convex Set). Let $X \subseteq \mathbb{R}^n$. Then the set X is convex if and only if for all pairs $\mathbf{x}_1, \mathbf{x}_2 \in X$ we have $\lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \in X$ for all $\lambda \in [0, 1]$.

Theorem 2.17. *The intersection of a finite number of convex sets in $\mathbb{R}^n$ is convex.*

Proof. Let $C_1, \dots, C_n \subseteq \mathbb{R}^n$ be a finite collection of convex sets. Let

$$C = \bigcap_{i=1}^{n} C_i \tag{2.13}$$

be the set formed from the intersection of these sets. Choose $\mathbf{x}_1, \mathbf{x}_2 \in C$ and $\lambda \in [0,1]$. Consider $\mathbf{x} = \lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2$. We know that $\mathbf{x}_1, \mathbf{x}_2 \in C_1, \dots, C_n$ by definition of C. By convexity, we know that $\mathbf{x} \in C_1, \dots, C_n$ by convexity of each set. Therefore, $\mathbf{x} \in C$. Thus C is a convex set. $\square$

Definition 2.18 (Convex Function). A function $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function if it satisfies:

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2) \tag{2.14}$$

for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$ and for all $\lambda \in [0,1]$. When the inequality is strict, for $\lambda \in (0,1)$, the function is a strictly convex function.

Example 2.19. This definition is illustrated in Figure 2.2. When f is a univariate

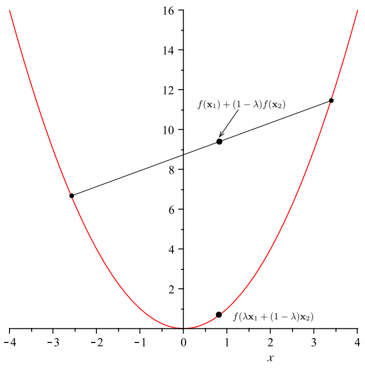


Figure 2.2. A convex function: A convex function satisfies the expression $f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2)$ for all $\mathbf{x}_1$ and $\mathbf{x}_2$ and $\lambda \in [0,1]$.

function, this definition can be shown to be equivalent to the definition you learned in Calculus I (Math 140) using first and second derivatives.

Definition 2.20 (Concave Function). A function $f : \mathbb{R}^n \to \mathbb{R}$ is a concave function if it satisfies:

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \geq \lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2) \tag{2.15}$$

for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$ and for all $\lambda \in [0,1]$. When the inequality is strict, for $\lambda \in (0,1)$, the function is a strictly concave function.

To visualize this definition, simply flip Figure 2.2 upside down. The following theorem is a powerful tool that can be used to show sets are convex. It's proof is outside the scope of the class, but relatively easy.

Theorem 2.21. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then the set $C = \{\mathbf{x} \in \mathbb{R}^n : f(x) \leq c\}$, where $c \in \mathbb{R}$, is a convex set.* $\square$

Exercise 12. Prove the Theorem 2.21.

Definition 2.22 (Linear Function). A function $z : \mathbb{R}^n \to \mathbb{R}$ is *linear* if there are constants $c_1, \dots, c_n \in \mathbb{R}$ so that:

$$z(x_1, \dots, x_n) = c_1x_1 + \dots + c_nx_n \tag{2.16}$$

Definition 2.23 (Affine Function). A function $z : \mathbb{R}^n \rightarrow \mathbb{R}$ is *affine* if $z(\mathbf{x}) = l(\mathbf{x}) + b$ where $l : \mathbb{R}^n \rightarrow \mathbb{R}$ is a linear function and $b \in \mathbb{R}$.

Exercise 13. Prove that every affine function is both convex and concave.

Theorem 2.24. *Suppose that $g_1, \dots, g_m : \mathbb{R}^n \rightarrow \mathbb{R}$ are convex functions and $h_1, \dots, h_l : \mathbb{R}^n \rightarrow \mathbb{R}$ are affine functions. Then the set:*

$$X = \{\mathbf{x} \in \mathbb{R}^n : g_i(\mathbf{x}) \leq 0, (i = 1, \dots, m) \text{ and } h_j(\mathbf{x}) = 0, (j = 1, \dots, l)\} \tag{2.17}$$

is convex.

Exercise 14. Prove Theorem 2.24

Theorem 2.25. *Suppose that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is concave and that $\mathbf{x}^*$ is a local maximizer of f. Then $\mathbf{x}^*$ is a global maximizer of f.*

Proof. Suppose $\mathbf{x}^+ \in \mathbb{R}^n$ has the property that $f(\mathbf{x}^+) > f(\mathbf{x}^*)$. For any $\lambda \in (0, 1)$ we know that:

$$f(\lambda\mathbf{x}^* + (1 - \lambda)\mathbf{x}^+) \geq \lambda f(\mathbf{x}^*) + (1 - \lambda)f(\mathbf{x}^+)$$

Since $\mathbf{x}^*$ is a local maximum there is an $\epsilon > 0$ so that for all $\mathbf{x} \in B_\epsilon(\mathbf{x}^*)$, $f(\mathbf{x}^*) \geq f(\mathbf{x})$. Choose λ so that $\lambda\mathbf{x}^* + (1 - \lambda)\mathbf{x}^+$ is in $B_\epsilon(\mathbf{x}^*)$ and let $\mathbf{x} = \lambda\mathbf{x}^* + (1 - \lambda)\mathbf{x}^+$. Let $r = f(\mathbf{x}^+) - f(\mathbf{x}^*)$. By assumption $r > 0$. Then we have:

$$f(\mathbf{x}) \geq \lambda f(\mathbf{x}^*) + (1 - \lambda)(f(\mathbf{x}^*) + r)$$

But this implies that:

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + (1 - \lambda)r$$

But $\mathbf{x} \in B_\epsilon(\mathbf{x}^*)$ by choice of λ, which contradicts our assumption that $\mathbf{x}^*$ is a local maximum. Thus, $\mathbf{x}^*$ must be a global maximum. □

Theorem 2.26. *Suppose that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is strictly concave and that $\mathbf{x}^*$ is a global maximizer of f. Then $\mathbf{x}^*$ is the unique global maximizer of f.*

Exercise 15. Prove Theorem 2.26. [Hint: Proceed by contradiction as in the proof of Theorem 2.25.]

Remark 2.27. We generally think of a function as either being convex (or concave) or not. However, it is often useful to think of functions being convex (or concave) on a subset of its domain. This can be important for determining the existence of (global) maxima on a constrained region.

Definition 2.28. Let $f : D \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ and suppose that $X \subseteq D$. Then f is convex on X if for all $\mathbf{x}_1, \mathbf{x}_2 \in \text{int}(X)$ and

$$f(\lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1 - \lambda)f(\mathbf{x}_2) \tag{2.18}$$

for $\lambda \in [0, 1]$.

Example 2.29. Consider the function $f(x) = -(x - 4)^4 + 3(x - 4)^3 + 6(x - 4)^2 - 3(x - 4) + 100$. This function is neither convex nor concave. However on a subset of its domain (say $x \in [6, \infty)$) the function is concave.

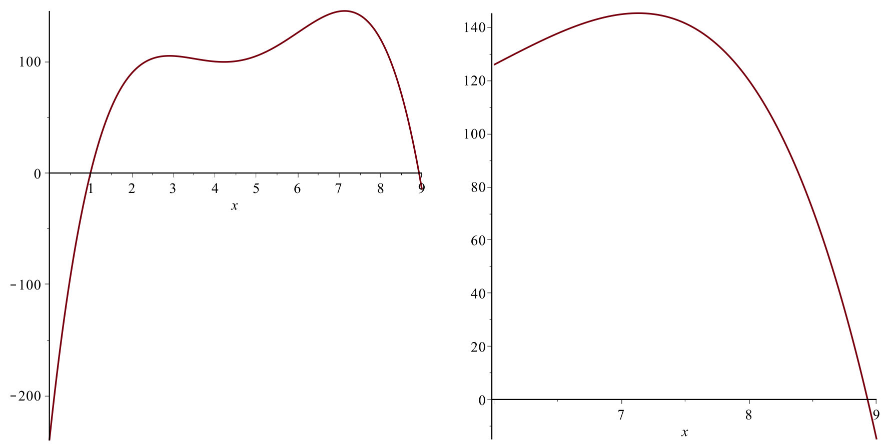


Figure 2.3. (Left) A simple quartic function with two local maxima and one local minima. (Right) A segment of the function that is locally concave.

4. Concave Functions and Differentiability

Remark 2.30. The following theorem is interesting, but its proof is outside the scope of the course. There is a proof for the one-dimensional case in Rudin [**Rud76**]. The proof for the general case can be derived from this. The general proof can also be found in Appendix B of [**Ber99**].

Theorem 2.31. *Every concave (convex) function is continuous on the interior of its domain.* □

Remark 2.32. The proofs of the next two theorems are variations on those found in [**Ber99**], with some details added for clarity.

Theorem 2.33. *A function $f : \mathbb{R}^n \to \mathbb{R}$ is concave if and only if for all $\mathbf{x}_0, \mathbf{x} \in \mathbb{R}^n$*

$$f(\mathbf{x}) \leq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) \tag{2.19}$$

Proof. ($\Leftarrow$) Suppose Inequality 2.19 holds. Let $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$ and let $\lambda \in (0,1)$ and let $\mathbf{x}_0 = \lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2$. We may write:

$$f(\mathbf{x}_1) \leq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x}_1 - \mathbf{x}_0)$$
$$f(\mathbf{x}_2) \leq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x}_2 - \mathbf{x}_0)$$

Multiplying the first equation by λ and the second by $1-\lambda$ and adding yields:

$$\lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2) \leq \\ \lambda f(\mathbf{x}_0) + (1-\lambda)f(\mathbf{x}_0) + \lambda\nabla f(\mathbf{x}_0)^T(\mathbf{x}_1 - \mathbf{x}_0) + (1-\lambda)\nabla f(\mathbf{x}_0)^T(\mathbf{x}_2 - \mathbf{x}_0)$$

Simplifying the inequality, we have:

$$\lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2) \leq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T\left(\lambda(\mathbf{x}_1 - \mathbf{x}_0) + (1-\lambda)(\mathbf{x}_2 - \mathbf{x}_0)\right)$$

Which simplifies further to:

$$\lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2) \leq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T\left(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 - \lambda\mathbf{x}_0 - (1-\lambda)\mathbf{x}_0\right)$$

Or:

$$\lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2) \leq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T\left(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 - \mathbf{x}_0\right) = f(\mathbf{x}_0)$$

because we assumed that:$\mathbf{x}_0 = \lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2$.

($\Rightarrow$) Now let $\mathbf{x}, \mathbf{x}_0 \in \mathbb{R}^n$ and let $\lambda \in (0,1)$. Define $\mathbf{h} = \mathbf{x} - \mathbf{x}_0$. Let:

$$g(\lambda) = \frac{f(\mathbf{x}_0 + \lambda\mathbf{h}) - f(\mathbf{x}_0)}{\lambda} \tag{2.20}$$

Clearly, as λ approaches 0 from the right, $g(\alpha)$ approaches the directional derivative of $f(\mathbf{x})$ at $\mathbf{x}_0$ in the direction $\mathbf{h}$.

Claim 1. *The function $g(\lambda)$ is monotonically decreasing.*

Proof. Consider $\lambda_1, \lambda_2 \in (0,1)$ with $\lambda_1 < \lambda_2$ and let $\alpha = \lambda_1/\lambda_2$. Define $\mathbf{z} = \mathbf{x}_0 + \lambda_2\mathbf{h}$. Note:

$$\alpha\mathbf{z} + (1-\alpha)\mathbf{x}_0 = \mathbf{x}_0 + \alpha(\mathbf{z}-\mathbf{x}_0) = \mathbf{x}_0 + \frac{\lambda_1}{\lambda_2}\left(\mathbf{x}_0 + \lambda_2(\mathbf{z}-\mathbf{x}_0) - \mathbf{x}_0\right) = \mathbf{x}_0 + \lambda_1(\mathbf{z}-\mathbf{x}_0)$$

Thus:

$$f(\mathbf{x}_0 + \alpha(\mathbf{z}-\mathbf{x}_0)) \geq \alpha f(\mathbf{z}) + (1-\alpha)f(\mathbf{x}_0) = f(\mathbf{x}_0) + \alpha f(\mathbf{z}) - \alpha f(\mathbf{x}_0) \tag{2.21}$$

Simplifying, we obtain:

$$\frac{f(\mathbf{x}_0 + \alpha(\mathbf{z}-\mathbf{x}_0)) - f(\mathbf{x}_0)}{\alpha} \geq f(\mathbf{z}) - f(\mathbf{x}_0) \tag{2.22}$$

Since $\mathbf{z} = \mathbf{x}_0 + \lambda_2\mathbf{h}$, we have:

$$\frac{f(\mathbf{x}_0 + \alpha(\mathbf{z}-\mathbf{x}_0)) - f(\mathbf{x}_0)}{\alpha} \geq f(\mathbf{z}) - f(\mathbf{x}_0) \tag{2.23}$$

Recall $\mathbf{z} = \mathbf{x}_0 + \lambda_2\mathbf{h}$, thus $\mathbf{z} - \mathbf{x}_0 = \lambda_2\mathbf{h}$. Thus the left hand side simplifies to:

$$\frac{f(\mathbf{x}_0 + (\lambda_1/\lambda_2)(\lambda_2\mathbf{h})) - f(\mathbf{x}_0)}{\alpha} = \frac{f(\mathbf{x}_0 + \lambda_1\mathbf{h}) - f(\mathbf{x}_0)}{\lambda_1/\lambda_2} \geq f(\mathbf{x}_0 + \lambda_2\mathbf{h}) - f(\mathbf{x}_0) \tag{2.24}$$

Lastly, dividing both sides by λ_2 yields:

$$\frac{f(\mathbf{x}_0 + \lambda_1\mathbf{h}) - f(\mathbf{x}_0)}{\lambda_1} \geq \frac{f(\mathbf{x}_0 + \lambda_2\mathbf{h}) - f(\mathbf{x}_0)}{\lambda_2} \tag{2.25}$$

Thus $g(\lambda)$ is monotonically decreasing. This completes the proof of the claim. □

Since $g(\lambda)$ is monotonically decreasing, we must have:

$$\lim_{\lambda\to 0^+} g(\lambda) \geq g(1) \tag{2.26}$$

But this implies that:

$$\lim_{\lambda\to 0^+} \frac{f(\mathbf{x}_0 + \lambda\mathbf{h}) - f(\mathbf{x}_0)}{\lambda} \geq f(\mathbf{x}_0+\mathbf{h}) - f(\mathbf{x}_0) = f(\mathbf{x}_0+\mathbf{x}-\mathbf{x}_0) - f(\mathbf{x}_0) = f(\mathbf{x}) - f(\mathbf{x}_0) \tag{2.27}$$

since $\mathbf{h} = \mathbf{x} - \mathbf{x}_0$. Applying Theorem 1.22, the inequality becomes:

$$\nabla f(\mathbf{x}_0)^T\mathbf{h} \geq f(\mathbf{x}) - f(\mathbf{x}_0) \tag{2.28}$$

which can be rewritten as:

$$f(\mathbf{x}) \leq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T\mathbf{h} = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x}-\mathbf{x}_0) \tag{2.29}$$

This completes the proof. □

Exercise 16. Argue that for strictly concave functions, Inequality 2.19 is strict and the theorem still holds.

Exercise 17. State and prove a similar theorem for convex functions.

Theorem 2.34. *Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a twice differentiable function. If f is concave, then $\nabla^2 f(\mathbf{x})$ is negative semidefinite.*

Proof. Suppose that there is a point $\mathbf{x}_0 \in \mathbb{R}^n$ and $\mathbf{h} \in \mathbb{R}^n$ such that $\mathbf{h}^T\nabla^2 f(\mathbf{x})\mathbf{h} > 0$. By the same argument as in the proof of Theorem 2.13, we may chose an $\mathbf{h}$ with a small norm so that for every $t \in [0, 1]$, $\mathbf{h}^T\nabla^2 f(\mathbf{x} + t\mathbf{h})\mathbf{h} > 0$ as well. By Lemma 2.9 for any $\mathbf{x} = \mathbf{x}_0 + \mathbf{h}$ we have some $t \in (0, 1)$ so that:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T\mathbf{h} + \frac{1}{2}\mathbf{h}^T\nabla^2 f(\mathbf{x}_0 + t\mathbf{h})\mathbf{h} \tag{2.30}$$

But since $\mathbf{h}^T\nabla^2 f(\mathbf{x} + t\mathbf{h})\mathbf{h} > 0$, we know that

$$f(\mathbf{x}) > f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T\mathbf{h} \tag{2.31}$$

and thus by Theorem 2.19, $f(\mathbf{x})$ cannot be concave, a contradiction. This completes the proof. □

Theorem 2.35. *Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a twice differentiable function. If $\nabla^2 f(\mathbf{x})$ is negative semidefinite, then $f(\mathbf{x})$ is concave.*

Proof. From Lemma 2.9, we know that for every $\mathbf{x}, \mathbf{x}_0 \in \mathbb{R}^n$ there is a $t \in (0, 1)$ such that when $\mathbf{h} = \mathbf{x} - \mathbf{x}_0$:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T\mathbf{h} + \frac{1}{2}\mathbf{h}^T\nabla^2 f(\mathbf{x}_0 + t\mathbf{h})\mathbf{h} \tag{2.32}$$

Since $\nabla^2 f(\mathbf{x})$ is negative semidefinite, it follows that: $\frac{1}{2}\mathbf{h}^T\nabla^2 f(\mathbf{x}_0 + t\mathbf{h})\mathbf{h} \leq 0$ and thus:

$$f(\mathbf{x}) \leq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) \tag{2.33}$$

Thus by Theorem 2.19, $f(\mathbf{x})$ is concave. □

Exercise 18. Suppose that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ with $f(\mathbf{x}) = \mathbf{x}^T\mathbf{H}\mathbf{x}$, where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{H} \in \mathbb{R}^{n\times n}$. Show that $f(\mathbf{x})$ is convex if and only if $\mathbf{H}$ is positive semidefinite.

Exercise 19. State and prove a theorem on the nature of the Hessian matrix when $f(\mathbf{x})$ is strictly concave.

Remark 2.36 (A Last Remark on Concave and Convex Functions). In this chapter, we've focused on primarily on concave functions. It's clear there are similar theorems for convex functions and many books focus on convex functions and minimization. In these notes, we'll focus on maximization, because the geometry of certain optimality conditions makes more sense on first exposure in a maximization problem. If you want to see the minimization side of the story, look at [Ber99] and [BSS06]. In reality, it doesn't matter, any maximization problem can be converted to a minimization problem and vice versa.

CHAPTER 3

Introduction to Gradient Ascent and Line Search Methods

1. Gradient Ascent Algorithm

Remark 3.1. In Theorem 1.23, we proved that the gradient points in the direction of fastest increase for a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. If we are trying to identify a point $\mathbf{x}^* \in \mathbb{R}^n$ that is a local or global maximum for f, then a reasonable approach is to walk along some direction $\mathbf{p}$ so that $\nabla f(\mathbf{x})^T\mathbf{p} > 0$.

Definition 3.2. Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a continuous and differentiable function and let $\mathbf{p} \in \mathbb{R}^n$. If $\nabla f(\mathbf{x})^T\mathbf{p} > 0$, then $\mathbf{p}$ is called an *ascent direction.*

Remark 3.3. Care must be taken, however, since the $\nabla f(\mathbf{x})$ represents only the direction of fastest increase at $\mathbf{x}$. As a result, we only want to take a small step in the direction of $\mathbf{p}$, then re-evaluate the gradient and continue until a stopping condition is reached.

```
Basic Ascent Algorithm
Input: $f : \mathbb{R}^n \rightarrow \mathbb{R}$ a function to maximize, $\mathbf{x}_0 \in \mathbb{R}^n$, a starting position
Initialize: $k = 0$
    (1) do
    (2)     Choose $\mathbf{p}_k \in \mathbb{R}^{n\times 1}$ and $\delta_k \in \mathbb{R}_+$ so that $\nabla f(\mathbf{x})^T\mathbf{p}_k > 0$.
    (3)     $\mathbf{x}_{k+1} := \mathbf{x}_k + \delta_k\mathbf{p}_k$
    (4) while some stopping criteria are not met.
Output: $\mathbf{x}_{k+1}$
```

Algorithm 2. Basic Ascent Algorithm

Remark 3.4. There are some obvious ambiguities with Algorithm 2. We have neither specified how to choose $\mathbf{p}_k$ nor δ_k in Line (2) of Algorithm 2, nor have we defined specific stopping criteria for the while loop. More importantly, we'd like to prove that there is a way of choosing $\mathbf{p}_k$ so that when we use this method at Line (2), the algorithm both converges i.e., at some point we exit the loop in Lines (1)-(4) and when we exit, we have identified a local maximum, or at least a point that satisfies the necessary conditions for a local maximum (see Theorem 2.12).

Remark 3.5. For the remainder of these notes, we will assume that:

$$\mathbf{p}_k = \mathbf{B}_k^{-1}\nabla f(\mathbf{x}_k)$$

where $\mathbf{B}_k \in \mathbb{R}^{n\times n}$ is some appropriately chosen symmetric and non-singular matrix.

Definition 3.6 (Gradient Ascent). When $\mathbf{B}_k = \mathbf{I}_n$, for some n, then Algorithm 2 is called *Gradient Ascent.*

Definition 3.7 (Newton's Method). When $\mathbf{B}_k = -\nabla^2 f(\mathbf{x}_k)$, for some $\delta_k \in \mathbb{R}_+$, then Algorithm 2 is called *Newton's Method*, which we will study in the next chapter.

2. Some Results on the Basic Ascent Algorithm

LEMMA 3.8. *Suppose that $\mathbf{B}_k$ is positive definite. Then: $\mathbf{p}_k = \mathbf{B}_k^{-1}\nabla f(\mathbf{x})$ is an ascent direction. That is, $\nabla f(\mathbf{x})^T\mathbf{p}_k > 0$.*

PROOF. By Corollary 1.37, $\mathbf{B}_k^{-1}$ is positive definite. Thus:

$$\nabla f(\mathbf{x})^T\mathbf{p}_k = \nabla f(\mathbf{x})^T\mathbf{B}_k^{-1}\nabla f(\mathbf{x}) > 0$$

by definition. □

REMARK 3.9. Since $\mathbf{B}_k = \mathbf{I}_n$ is always positive definite and non-singular, we need never worry that $\mathbf{p}_k = \nabla f(\mathbf{x})$ is not a properly defined ascent direction in gradient ascent.

REMARK 3.10. As we've noted, we cannot simply choose δ_k arbitrarily in Line (2) of Algorithm 2, since $\nabla f(\mathbf{x})$ is only the direction of greatest ascent at $\mathbf{x}$ and thus, $\mathbf{p}_k = \mathbf{B}_k^{-1}\nabla f(\mathbf{x}_k)$ is only a ascent direction in a neighborhood about $\mathbf{x}$. If we define:

$$\phi(\delta_k) = f(\mathbf{x}_k + \delta_k\mathbf{p}_k) \tag{3.1}$$

then our problem is to solve:

$$\begin{cases} \max & \phi(\delta_k) \\ s.t. & \delta_k \geq 0 \end{cases} \tag{3.2}$$

This is an optimization problem in a single variable, δ_k and assuming its solution is δ_k^* and we compute $\mathbf{x}_{k+1} := \mathbf{x}_k + \delta_k^*\mathbf{p}_k$, then we will assuredly have increased (or at least not decreased) the value of $\mathbf{f}(\mathbf{x}_{k+1})$ compared to $f(\mathbf{x}_k)$.

REMARK 3.11. As we'll see in Section 2, it's not enough to just increase the value of $f(\mathbf{x}_k)$ at each iteration k, we must increase it by a sufficient amount. One of the easiest, though not necessarily the most computationally efficient, ways to make sure this happens is to ensure that we identify a solution to the problem in Expression 3.2. In the following sections, we discuss several methods for determining a solution.

3. Maximum Bracketing

REMARK 3.12. For the remainder of this section, let us assume that $\phi(x)$ is a one dimensional function that we wish to maximize. One problem with attempting to solve Problem 3.2 is that the half-interval $\delta_k \geq 0$ is unbounded and thus we could have to search a very long way to find a solution. To solve this problem, we introduce the notion of a constrained search on an interval $[a, c]$ where we assume that we have a third value $b \in [0, b]$ such that $\phi(a) < \phi(b)$ and $\phi(b) > \phi(c)$. Such a triple (a, b, c) is called a *bracket*.

PROPOSITION 3.13. *For simplicity of notation, assume that $\phi(x) = \phi_x \in \mathbb{R}$. The coefficients of the quadratic curve $rx^2 + sx + t$ passing through the points (a, ϕ_a), (b, ϕ_b), (c, ϕ_c) are given by the expressions:*

$$r = -\frac{-b\phi_a + b\phi_c - a\phi_c + \phi_a c - \phi_b c + \phi_b a}{b^2c - b^2a - bc^2 + ba^2 - a^2c + ac^2}$$

$$s = \frac{-a^2\phi_c + a^2\phi_b - \phi_a b^2 + \phi_a c^2 - c^2\phi_b + \phi_c b^2}{(-c + a)(ab - ac - b^2 + cb)}$$

$$t = \frac{-\phi_b a^2c + a^2b\phi_c + ac^2\phi_b - a\phi_c b^2 - c^2b\phi_a + c\phi_a b^2}{(-c + a)(ab - ac - b^2 + cb)}$$

Exercise 20. Prove proposition 3.13. [Hint: Use a computer algebra system.]

Corollary 3.14. *The maximum (or minimum) of the quadratic curve $rx^2 + sx + t$ passing through the points (a, ϕ_a), (b, ϕ_b), (c, ϕ_c) is given by:*

$$u = \frac{1}{2}\frac{-a^2\phi_c + a^2\phi_b - \phi_a b^2 + \phi_a c^2 - c^2\phi_b + \phi_c b^2}{-b\phi_a + b\phi_c - a\phi_c + \phi_a c - \phi_b c + \phi_b a} \tag{3.3}$$

Remark 3.15. The Maple code for finding the maximum point for the curve passing through (a, ϕ_a), (b, ϕ_b), (c, ϕ_c) is shown below:

```
1 QuadraticTurn :=
2 proc (a::numeric, fa::numeric, b::numeric, fb::numeric, c::numeric, fc::numeric)
    ::list;
3  local qf, r, s, t, SOLN;
4  qf := proc (x) options operator, arrow; r*x^2+s*x+t end proc;
5  SOLN := solve([fa = qf(a), fb = qf(b), fc = qf(c)], [r, s, t]);
6  [eval(-(1/2)*s/r, SOLN[1]), evalb(nops(SOLN) > 0 and eval(r < 0, SOLN[1]))]
7 end proc;
```

Note in Line 6, we actually return a list, whose second element is *true* if and only if there is a solution to the proposed problem and the point is the maximum of a quadratic approximation, rather than the minimum. Note also for simplicity we use `fa`, `fb`, and `fc` instead of ϕ_a, ϕ_b and ϕ_c.

Remark 3.16. In our case, we know that $a = 0$ and we could choose an arbitrarily small value $b = a + \epsilon$ as our second value. The problem is identifying point c. To solve this problem, we introduce a bracketing algorithm (see Algorithm 3). Essentially, the bracketing algorithm keeps moving the points a, b and c further and further to the right using a quadratic approximation of the function when possible. *Note, this algorithm assumes that the local behavior of $\phi(x)$ is concave, which is true in our line search, especially when $f(\mathbf{x})$ is concave.*

Remark 3.17. Note in Algorithm 3, our $\phi(x)$ is written $f(x)$ and we use τ as the Golden ratio, which is used to define our initial value $c = b + \tau \cdot (b - a)$. This is also used when the parabolic approximation of the function fails. In practice a tiny term is usually added to the denominator of the fraction in Corollary 3.14 to prevent numerical instability (see [**PTVF07**], Page 491). Also note, that we compute $\phi(x)$ many times. The number of times that $\phi(x)$ can be reduced by storing these values as discussed in ([**PTVF07**], Page 491).

Proposition 3.18. *If $\phi(x)$ has a maximum value on $\mathbb{R}_+$, then Algorithm 3 will identify a bracket containing a local maximum.*

Proof. The proof is straightforward. The algorithm will continue to move the bracket (a, b, c) rightward ensuring that $\phi(a) < \phi(b)$. At some point, the algorithm will choose c so that $\phi(b) > \phi(c)$ with certainty (since $\phi(x)$ has a maximum and its maximum is contained in $\mathbb{R}_+$). When this occurs, the algorithm terminates. □

Example 3.19. Consider the following function:

$$\phi(x) = -(x-4)^4 + 3(x-4)^3 + 6(x-4)^2 - 3(x-4) + 100$$

The plot of this function is shown in Figure 2.3. When we execute the parabolic bracketing algorithm on this function starting at $a = 0$ and $b = 0.25$, we obtain the output:

```
ParabolicBracket := proc (phi::operator, a::numeric, b::numeric)::list;
  local aa, bb, cc, tau, ret, M, umax, u, bSkip, uu;
  tau := evalf(1/2+(1/2)*sqrt(5));
  M := 10.0;
  aa := a; bb := b;
  if phi(bb) < phi(aa) and aa < bb then
    ret := []
  else
    cc := bb+tau*(bb-aa);
    ret := [aa,bb,cc]:
    while phi(bb) < phi(cc) do
      umax := (cc-bb)*M+cc;
      uu := QuadraticTurn(aa, phi(aa), bb, phi(bb), cc, phi(cc));
      u := uu[1];
      print(sprintf("a=%f\t\tb=%f\t\tc=%f\t\tu=%f", aa, bb, cc, u));
      if uu[2] then
        bSkip := false;
        if aa < u and u < bb then
          if phi(aa) < phi(u) and phi(bb) < phi(u) then
            ret := [aa, u, bb];
            bb := u; cc := bb
          else
            aa := bb; bb := cc; cc := bb+tau*(bb-aa);
            ret := [aa, bb, cc]
          end if:
        elif bb < u and u < cc then
          if phi(bb) < phi(u) and phi(cc) < phi(u) then
            ret := [bb, u, cc];
            aa := bb; bb := u
          else
            aa := bb; bb := u; cc := cc; cc := bb+tau*(cc-bb);
            ret := [aa, bb, cc]
          end if:
        elif cc < u and u < umax then
          if phi(cc) < phi(u) then
            aa := bb; bb := cc; cc := u; cc := bb+tau*(cc-bb);
            ret := [aa, bb, cc]
          else
            aa := bb; bb := cc; cc := bb+tau*(bb-aa);
            ret := [aa, bb, cc]
          end if:
        elif umax < u then
           aa := bb; bb := cc; cc := bb+tau*(bb-aa);
           ret := [aa, bb, cc]
        end if:
      else
        aa := bb; bb := cc; cc := bb+tau*(bb-aa);
        ret := [aa, bb, cc]
      end if:
    end do:
  end if:
  ret
end proc:
```

Algorithm 3. Bracketing Algorithm

```
"a=0.000000 b=0.250000 c=0.654508 u=1.580537"
"a=0.250000 b=0.654508 c=2.152854 u=2.095864"
"a=0.654508 b=2.095864 c=2.188076 u=2.461833"
"a=2.095864 b=2.188076 c=2.631024 u=2.733759"
"a=2.188076 b=2.631024 c=2.797253 u=2.839039"
"a=2.631024 b=2.797253 c=2.864864 u=2.889411"
"a=2.797253 b=2.864864 c=2.904582 u=2.899652"
Output = [2.864863884, 2.899652455, 2.904582162]
```

When we execute the algorithm starting at $a = 4.3$ and $b = 4.5$, we obtain the output:

```
"a=4.300000 b=4.500000 c=4.823607 u=4.234248"
"a=4.500000 b=4.823607 c=5.347214 u=4.235687"
"a=4.823607 b=5.347214 c=6.194427 u=3.781555"
"a=5.347214 b=6.194427 c=7.565248 u=7.317872"
Output = [6.194427192, 7.317871765, 7.565247585]
```

The actual (local) maxima for this function occur at $x \sim 2.900627632$ and $x \sim 4.217851814$, showing our maximum points are bracketed as expected.

Remark 3.20. Parabolic fitting is often overly complicated, especially if the function $\phi(x)$ is locally concave. In this case, we can simply compute $c = b + \alpha(b - a)$ and successively reassign $a = b$ and $b = c$.

Exercise 21. Write the algorithm discussed in Remark 3.20 and prove that if $\phi(x)$ has a maximum $[0, \infty]$, this algorithm will eventually bracket the maximum if you start with $a = 0$ and $b = a + \epsilon$.

Exercise 22. Compare the bracket size obtained from Algorithm 3 vs. the algorithm you wrote in Exercise 21. Is there a substantial difference in the bracket size?

4. Dichotomous Search

Remark 3.21. Assuming we have a bracket around a (local) maximum of $\phi(x)$, it is now our objective to find a value close to the value x^* such that $\phi(x^*)$ yields that (local) maximum value. A dichotomous search is a popular (but not the most efficient) method for finding such a value. The Maple code for Dichotomous Search is shown in Figure 4.

Remark 3.22. Dichotomous Search begins with an interval around which we are certain there is a maximum. The interval is repeatedly reduced until the maximum is localized to a small enough interval. At Lines 11 and 12, two test points are computed. Depending upon the value the function ϕ at these points, the interval is either sub-divided to the left or right. This is illustrated in Figure 3.1. If the value of the function is the same at both test points, then the direction is chosen arbitrarily. In the implementation shown in Algorithm 4, the algorithm always chooses the right half of the interval. Convergence can be sped up by defining $x_1 = u - \delta$ and $x_2 = u + \delta$ from the algorithm for some small $\delta > 0$.

Exercise 23. For some small δ (say 0.01) empirically compare the rate of convergence of the Dichotomous Search Algorithm shown in Algorithm 4 vs. the rate of the algorithm for the given δ.

```
DichotomousSearch := proc (phi::operator, a::numeric, b::numeric, epsilon::
    numeric)::real;
  local u, aa, bb, x;
  if a < b then
    aa := a; bb := b
  else bb := a; aa := b
  end if:
  u := (1/2)*aa+(1/2)*bb;
  print("a\t\t\t\t\tb");
  while epsilon < bb-aa do
    print(sprintf("%f␣\t\t\t␣%f", aa, bb));
    x[1] := (1/2)*aa+(1/2)*u;
    x[2] := (1/2)*bb+(1/2)*u;
    if phi(x[2]) < phi(x[1]) then
      bb := x[2]; u := (1/2)*aa+(1/2)*bb
    else
      aa := x[1]; u := (1/2)*aa+(1/2)*bb
    end if:
  end do;
  (1/2)*aa+(1/2)*bb
end proc
```

Algorithm 4. Dichotomous Search

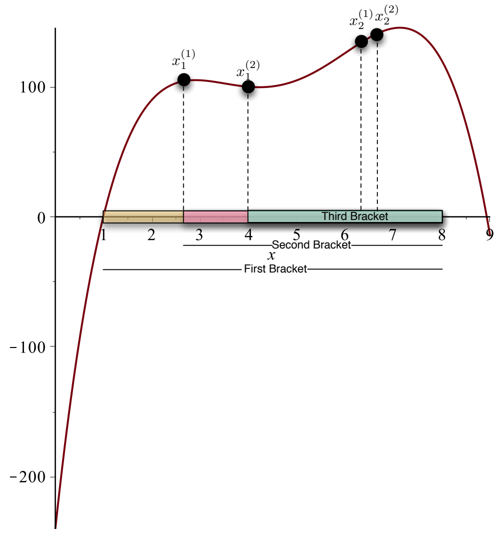


Figure 3.1. Dichotomous Search iteratively refines the size of a bracket containing a maximum of the function ϕ by splitting the bracket in two.

THEOREM 3.23. *Suppose $\phi(x)$ is strictly concave and attains its unique maximum on the (initial) interval $[a, b]$ at x^*. Then Dichotomous Search will converge to a point x^+ such that $|x^+ - x^*| < \epsilon$.*

Proof. We will show by induction on the algorithm iteration that x^* is always in the interval returned by the algorithm. At initialization, this is clearly true. Suppose after n iterations we have the interval $[l, r]$ and $x^* \in [l, r]$. Let let $u = (l + r)/2$ and without loss of generality, suppose $x^* \in [u, r]$. Let x_1 and x_2 be the two test points with $x_1 \in [l, u]$ and $x_2 \in [u, r]$. If $\phi(x_2) > \phi(x_1)$, then at the next iteration the interval is $[x_1, r]$ and clearly $x^* \in [x_1, r]$. Conversely, suppose that $\phi(x_1) > \phi(x_2)$. Since $\phi(x)$ is concave and must always lie above its secant, it follows that $x^* \in [x_1, x_2]$ (because clearly the function has reached its turning point between x_1 and x_2). Thus, $x^* \in [u, x_2]$. The next interval when $\phi(x_1) > \phi(x_2)$ is $[l, x_2]$ and thus $x^* \in [l, x_2]$. It follows by induction that x^* is contained in the interval computed at each iteration of the while loop at Line 9. Suppose that $[l^*, r^*]$ is the final interval upon termination of the while loop at Line 9. We know that $x^* \in [l^*, r^*]$ and $r^* - l^* < \epsilon$, thus $|x^+ - x^*| < \epsilon$. This completes the proof. □

Theorem 3.24. *Suppose $\phi(x)$ is concave and attains a maximum on the (initial) interval $[a, b]$ at x^*. Assume that at each iteration, the interval of uncertainty around x^* is reduced by a factor of $r \in (0, 1)$. Then Dichotomous search converges in:*

$$n = \left\lceil \left(\frac{\log\left(\frac{\epsilon}{|b-a|}\right)}{\log(r)} \right) \right\rceil$$

steps.

Exercise 24. Prove Theorem 3.24. Illustrate the correctness of your proof by implementing Dichotomous Search and maximizing the function $\phi(x) = 10 - (x - 5)^2$ on the interval $[0, 10]$.

Example 3.25. The assumption that $\phi(x)$ is concave (on $[a, b]$) is not necessary to ensure convergence to the maximum within $[a, b]$. Generally speaking it is sufficient to assume $\phi(x)$ is unimodal on $[a, b]$. To see this, consider the function:

$$\phi(x) = \begin{cases} 2x + 1 & x < 2 \\ 5 & 2 \le x \le 8 \\ x - 3 & 8 < x \le 9 \\ -x + 15 & x > 9 \end{cases}$$

The function is illustrated in Figure 3.2. If we execute the Dichotomous Search algorithm

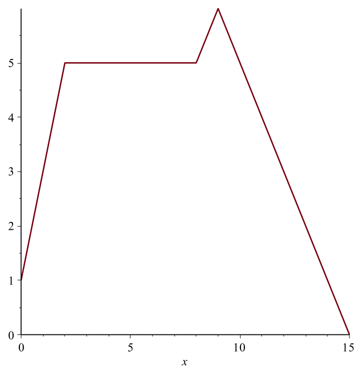


Figure 3.2. A non-concave function with a maximum on the interval $[0, 15]$.

as illustrated in Algorithm 4, it will converge to the bracket $[8.99, 9.00]$ even though this function is not concave on the interval. The output from the algorithm is shown below:

```
        "a b"
"0.000000   15.000000"
"0.000000   11.250000"
"2.812500   11.250000"
"4.921875   11.250000"
"6.503906   11.250000"
"6.503906   10.063477"
"7.393799   10.063477"
"8.061218   10.063477"
"8.061218   9.562912"
"8.436642   9.562912"
"8.718209   9.562912"
"8.718209   9.351736"
"8.718209   9.193355"
"8.836996   9.193355"
"8.836996   9.104265"
"8.903813   9.104265"
"8.903813   9.054152"
"8.941398   9.054152"
"8.969586   9.054152"
"8.969586   9.033010"
"8.969586   9.017154"
"8.981478   9.017154"
"8.990397   9.017154"
"8.990397   9.010465"
"8.990397   9.005448"
"8.994160   9.005448"
Output =  9.001215067
```

5. Golden Section Search

Remark 3.26. In the Dichotomous search each iteration requires two new evaluations of the value of ϕ. If ϕ is very costly to evaluate, it might be more efficient to construct an algorithm that evaluates $\phi(x)$ only once at each iteration by cleverly constructing the intervals. The Golden Section Search does exactly that.

Remark 3.27. At each iteration of the golden section search, there are four points under consideration, $x_1 < x_2 < x_3 < x_4$, where x_1 and x_4 are the left and right endpoints of the current interval. At the next iteration, if $\phi(x_2) > \phi(x_3)$, then x_1 remains the left end point, x_3 becomes the right end point and x_2 plays the role of x_3 in the next iteration. On the other hand, if $\phi(x_2) < \phi(x_3)$, then x_4 remains the right endpoint, x_2 becomes the left endpoint and x_3 plays the role of x_2 in the next iteration. The objective in the Golden Section Search is to ensure the the sizes of the intervals remain proportionally constant throughout algorithm execution.

PROPOSITION 3.28. *Given an initial bracket $[x_1, x_4]$ containing a maximum for the function $\phi(x)$. To ensure that the ratio of $x_2 - x_1$ to $x_4 - x_2$ is kept constant, then $x_2 = (1/\tau)x_1 + (1 - 1/\tau)x_4$ and $x_3 = (1 - 1/\tau)x_1 + (1/\tau)x_4$, where τ is the Golden ratio; i.e., $\tau = \frac{1}{2}(1 + \sqrt{5})$.*

PROOF. Consider Figure 3.3: In the initial interval, the ratio of $x_2 - x_1$ to $x_4 - x_2$ is

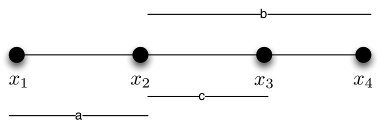


Figure 3.3. The relative sizes of the interval and sub-interval lengths in a Golden Section Search.

a/b. At the next iteration, either $\phi(x_2) < \phi(x_3)$ or not (in the case of a tie, it an arbitrary left/right choice is made). If $\phi(x_2) > \phi(x_3)$, then x_2 becomes the new x_3 point and thus we require:

$$\frac{a}{b} = \frac{c}{a}$$

On the other hand, if $\phi(x_2) < \phi(x_3)$ then x_3 becomes the new x_2 point and we require:

$$\frac{a}{b} = \frac{c}{b - c}$$

From the first equation, we see that $a^2/c = b$ and substituting that into the second equation yields:

$$\left(\frac{a}{c}\right)^2 - \frac{a}{c} - 1 = 0$$

Which implies:

$$\frac{a}{c} = \tau = \frac{b}{a}$$

Without loss of generality, suppose that $x_1 = 0$ and $x_4 = 1$. To compute x_2 we have:

$$\frac{a}{1 - a} = \frac{1}{\tau} \implies a = \frac{1}{1 + \tau}$$

For the Golden Ratio, $1/(1 + \tau) = 1 - 1/\tau$ and thus it follows $x_2 = (1/\tau)x_1 + (1 - 1/\tau)x_4$. To compute x_3, suppose $a + c = r$. Then we have:

$$\frac{r - x_2}{1 - r} = \frac{1}{\tau}$$

When we replace x_2 with $1/(1 + \tau)$ we have:

$$r = \frac{1 + 2\tau}{(1 + \tau)^2}$$

In the case of the Golden Ratio, the previous fraction reduces to $r = 1/\tau$. Thus we have $x_3 = (1 - 1/\tau)x_1 + (1/\tau)x_4$. This completes the proof. □

REMARK 3.29. The Golden Section Search is shown illustrated in Algorithm 5. This is, essentially, the Golden Section search as it is implemented in [PTVF07] (Page 495).

```
GoldenSectionSearch := proc (phi::operator, a::numeric, b::numeric,
  c::numeric, epsilon::numeric)::real;
  local aa, bb, cc, dd, tau, ret;
  tau := evalf(1/2+(1/2)*sqrt(5));
  aa := a; dd := c;
  if b-a < c-b then
    bb := b; cc := b+(1-1/tau)*(c-b)
  else
    bb := b-(1-1/tau)*(b-a); cc := b
  end if;
  while epsilon < dd-aa do
    print(sprintf("aa=%f\t\tbb=%f\t\tcc=%f\t\tdd=%f", aa, bb, cc, dd));
    if phi(cc) < phi(bb) then
      dd := cc; cc := bb; bb := cc/tau+(1-1/tau)*aa;
    else
      aa := bb; bb := cc; cc := bb/tau+(1-1/tau)*dd;
    end if
  end do;
  if phi(cc) < phi(bb) then
    ret := bb;
  else
    ret := cc;
  end if;
  ret;
end proc:
```

Algorithm 5. Golden Section Search

THEOREM 3.30. *Suppose $\phi(x)$ is strictly concave and attains its unique maximum on the (initial) interval $[a, b]$ at x^*. Then Golden Section Search will converge to a point x^+ such that $|x^+ - x^*| < \epsilon$.*

EXERCISE 25. Prove Theorem 3.30.

EXAMPLE 3.31. In Example 3.25, we saw an example of a non-concave function for which Dichotomous Search converged. In this example, we show an instance of a function for which the Golden Section Search does not converge to the global maximum of the function. Let:

$$\phi(x) = \begin{cases} 10x & x < \frac{1}{2} \\ -10x + 10 & x \geq \frac{1}{2} \text{ and } x < \frac{3}{4} \\ \frac{5}{2} & x \geq \frac{3}{4} \text{ and } x < 9 \\ -x + \frac{23}{2} & x \geq 9 \end{cases}$$

The function in question is shown in Figure 3.4. If we use the bracket $[0, 11]$, note that the global functional maximum occurs in a position very close to the left end of the bracket. As a result, the interval search misses the global maximum. This illustrates the importance of tight bracketing in line search algorithms. The output of the search is shown below.

```
"aa=0.000000 bb=4.201626 cc=6.798374 dd=11.000000"
```

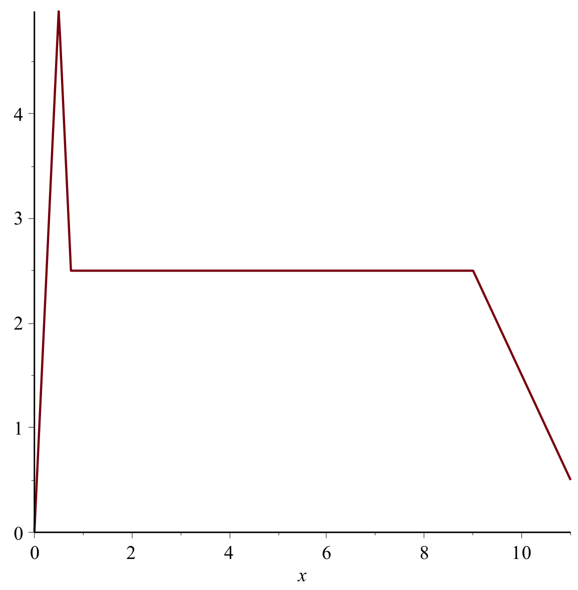


Figure 3.4. A function for which Golden Section Search (and Dichotoous Search) might fail to find a global solution.

```
"aa=4.201626 bb=6.798374 cc=8.403252 dd=11.000000"
"aa=6.798374 bb=8.403252 cc=9.395122 dd=11.000000"
"aa=6.798374 bb=7.790243 cc=8.403252 dd=9.395122"
"aa=7.790243 bb=8.403252 cc=8.782113 dd=9.395122"
"aa=8.403252 bb=8.782113 cc=9.016261 dd=9.395122"
"aa=8.403252 bb=8.637401 cc=8.782113 dd=9.016261"
"aa=8.637401 bb=8.782113 cc=8.871549 dd=9.016261"
"aa=8.782113 bb=8.871549 cc=8.926824 dd=9.016261"
"aa=8.871549 bb=8.926824 cc=8.960986 dd=9.016261"
"aa=8.926824 bb=8.960986 cc=8.982099 dd=9.016261"
"aa=8.960986 bb=8.982099 cc=8.995148 dd=9.016261"
"aa=8.982099 bb=8.995148 cc=9.003213 dd=9.016261"
"aa=8.982099 bb=8.990164 cc=8.995148 dd=9.003213"
"aa=8.990164 bb=8.995148 cc=8.998228 dd=9.003213"
Output = 8.998228439
```

Remark 3.32. There are derivative other free line search methods. One of the simplest applies the parabolic approximation we used in the bracketing method in Algorithm 3 (see [BSS06] or [Ber99]). Another method, generally called Brent's method, combines parabolic approximation with the Golden Section method (see [PTVF07], Chapter 10 for an implementation of Brent's Method). These techniques extend and combine derivative free methods presented in this section, but do not break any substantially new ground beyond their efficiency.

6. Bisection Search

Remark 3.33. We are now going to assume that we can compute the first and possibly second derivative of $\phi(x)$. In doing so, we will construct new methods for finding points to maximize the function.

Remark 3.34. Before proceeding any further, we warn the reader that methods requiring function derivatives should use the actual derivative of the function rather than an approximation. While approximate methods are necessary in multi-dimensional optimization and often used, line searches can be sensitive to incorrect derivatives. Thus, it is often safer to use a derivative free method even though it may not be as efficient.

Remark 3.35. The bisection search assumes that for a given univariate function $\phi(x)$, we have a bracket $[a, b] \subset \mathbb{R}$ containing a (local) maximum and furthermore that $\phi(x)$ is differentiable on this closed interval and $\phi'(a) > 0$ and $\phi'(b) < 0$. As we will see, convergence can be ensured when $\phi(x)$ is concave.

Remark 3.36. The bisection search algorithm is illustrated in Algorithm 6. Given the assumption that $\phi'(a) > 0$ and $\phi'(b) < 0$, we choose a test point $u \in [a, b]$. In the implementation, we choose $u = (a + b)/2$. If $\phi'(u) > 0$, then this point is presumed to lie on the left side of the maximum and we repeat on the new interval $[u, b]$. If $\phi'(u) < 0$, then u is presumed to lie on the right side of the maximum and we repeat on the new interval $[a, u]$. If $\phi'(u) = 0$, then we terminate our search and return u as x^* the maximizing value for $\phi(x)$. Because of round off, we will never reach a condition in which $\phi'(u) = 0$, so instead we test to see if $|\phi'(u)| < \epsilon$, where ϵ is a small constant provided by the user.

Theorem 3.37. *If $\phi(x)$ is concave and continuously differentiable on $[a, b]$ with a maximum $x^* \in (a, b)$, then for all ϵ (an input parameter) there is a $\delta > 0$ so that Bisection Search converges to u with the property that $|x^* - u| \leq \delta$.*

Proof. The fact that $\phi(x)$ is concave on $[a, b]$ combined with the fact that $\phi'(a) > 0$ and $\phi'(b) < 0$ implies that $\phi'(x)$ is a monotonically decreasing continuous function on the interval $[a, b]$ and therefore if there are two values x^* and x^+ maximizing $\phi(x)$ on $[a, b]$ then for all $\lambda \in [0, 1]$, $x^\circ = \lambda x^* + (1 - \lambda)x^+$ also maximizes $\phi(x)$ and by necessity $\phi'(x^\circ) = \phi'(x^*) = \phi'(x^+) = 0$. Thus, for any iteration if we have the bracket $[l, r]$ if $\phi'(l) > 0$ and $\phi'(r) < 0$ we can be sure that $x^* \in [l, r]$. Suppose after n iterations we have interval $[l, r]$ with the property that $\phi'(l) > 0$ and $\phi'(r) < 0$. Lines 15 and 17 of Bisection search ensure that at iteration $n + 1$ we continue to have a new interval $[l', r'] \subset [l, r]$ with the property that $\phi'(l') > 0$ and $\phi'(r') < 0$, assuming that $|\phi'(u)| > \epsilon$. Thus, at each iteration x^* is in the bracket. Furthermore, since $\phi'(x)$ is monotonically decreasing we can be sure that as each iteration shrinks the length of the interval $[l, r]$, $\phi'(u)$ will not increase. Thus at some point either $\phi'(u) < \epsilon$ or $|r - l| < \epsilon$. If the latter occurs, then we know that $|x^* - u| < \epsilon$ since $u \in [l, r]$ at the last iteration and $\delta = \epsilon$. On the other hand, if $\phi'(u) = 0$, then u is a maximizer of $\phi(x)$ and we may assume that $x^* = u$, so $\delta = 0$. Finally, suppose $0 < \phi'(u) < \epsilon$. Then there is some non-zero $\delta < |r - l|$ so that $u + \delta = x^*$. Thus $|x^* - u| \leq \delta$. The case when $\phi' < 0$ and $|\phi'(u)| < \epsilon$ is analogous. □

Remark 3.38. Note, we do not put a detailed bound on the δ in the proof of the previous theorem. As a rule of thumb, we'd like, $\phi'(x)$ to have a somewhat nice property like having a continuous inverse function, in which case we could make an exact ϵ-δ style argument to

```
BisectionSearch := proc (phi::operator, a::numeric, b::numeric, epsilon::numeric)
    ::real;
  local u, aa, bb, x, y;
  if a < b then
    aa := a; bb := b
  else
    bb := a; aa := b
  end if;
  u := (1/2)*aa+(1/2)*bb;
  print("a\t\t\t\tb");
  while epsilon < bb-aa do
    print(sprintf("%f␣\t\t\t␣%f", aa, bb));
    y := eval(diff(phi(z), z), z = u);
    if epsilon < abs(y) then
      if y < 0 then
        bb := u; u := (1/2)*aa+(1/2)*bb
      else
        aa := u; u := (1/2)*aa+(1/2)*bb
      end if
    else
      break
    end if
  end do;
  u
end proc:
```

Algorithm 6. Bisection Search

assert that if $|\phi'(u)-\phi'(x^*)| < \epsilon$, then there better be some small δ so that $|u-x^*| < \delta$, for an appropriately chosen maximizer x^* (if the maximizer is not unique). Unfortunately, plain old continuity does not work for this argument. We could require $\phi'(x)$ to be bilipschitz, which would also achieve this effect. See Exercise 27 for an alternative way to bound $|u - x^*| < \delta$ in terms of ϵ.

EXERCISE 26. A function $f : \mathbb{R} \to \mathbb{R}$ is called Lipschitz continuous if there is a $K > 0$ so that for all x_1 and x_2 in $\mathbb{R}$ we have $|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$. A function is bilipschitz if there is a $K \geq 1$ so that:

$$\frac{1}{K}|x_1 - x_2| \leq |f(x_1) - f(x_2)| \leq K|x_1 - x_2| \tag{3.4}$$

Suppose $\phi'(x)$ is bilipschitz. Find an exact expression for δ.

EXERCISE 27. This exercise will tell you something about the convergence proof for bisection search under a stronger assumption on $\phi(x)$.

Suppose that $\phi : \mathbb{R} \to \mathbb{R}$ is *twice* continuously differentiable and strictly concave on $[a, b]$ with its unique maximum $x^* \in [a, b]$ (so $\phi'(x^*) = 0$). Assume that $|\phi'(u)| < \epsilon$. Use the Mean Value Theorem to find a bound on $|u - x^*|$.

Remark 3.39. If $\phi(x)$ is twice differentiable then there is a t between u and x^* so that

$$\phi'(u) - \phi'(x^*) = \phi''(t)\,(u - x^*) \tag{3.5}$$

This implies that:

$$|\phi'(u) - \phi'(x^*)| = |\phi''(t)|\,|u - x^*| \tag{3.6}$$

From this we know that

$$|\phi'(u) - \phi'(x^*)| = |\phi''(t)|\,|u - x^*| < \epsilon \tag{3.7}$$

Therefore:

$$|u - x^*| < \frac{\epsilon}{|\phi''(t)|} \tag{3.8}$$

Exercise 28. Modify the bisection search to detect minima. Test your algorithm on the function $f(x) = (x - 5)^2$.

Example 3.40. Below we show example output from Bisection Search when executed on the function $\phi(x) = 10 - (x - 5)^2$ when we search on the interval $[0, 9]$ and set $\epsilon = 0.01$.

```
        "a b"
"0.000000   9.000000"
"4.500000   9.000000"
"4.500000   6.750000"
"4.500000   5.625000"
"4.500000   5.062500"
"4.781250   5.062500"
"4.921875   5.062500"
"4.992188   5.062500"
"4.992188   5.027344"
"4.992188   5.009766"
Output = 5.000976562
```

7. Newton's Method

Remark 3.41. Newton's method is a popular method for root finding that can also be applied to the problem of univariate functional maximization (or minimization) provided the second derivative of the function can be computed. Since we will cover Newton's Method for multidimensional functions in the next chapter, we will not go into great detail on the theory here. Note in theory, we do not require a bracket for Newton's Method, just a starting point. A bracket is useful though for finding a starting point.

Remark 3.42. The algorithm assumes that the quadratic Taylor approximation is reasonable for the function $\phi(x)$ to be optimized about the current iterated value x_k. That is:

$$\phi(x) \approx \phi(x_k) + \phi'(x_0)(x - x_k) + \frac{1}{2}\phi''(x_0)(x - x_k)^2 \tag{3.9}$$

If we assume that $\phi''(x_k) < 0$, then differentiating and setting equal to zero will yield a maximum for the function approximation:

$$x_{k+1} = -\frac{\phi'(x_k) - \phi''(x_k)x_k}{\phi''(x_k)} = x_k - \frac{\phi'(x_k)}{\phi''(x_k)} \tag{3.10}$$

Iteration continues until some termination criterion is achieved. An implementation of the algorithm is shown in Algorithm 7

```
NewtonsMethod := proc (phi::operator, a::numeric, epsilon::numeric)::list;
  local u, fp, fpp;
  u := a;
  fp := infinity;
  while epsilon < abs(fp) do
    print(sprintf("xk = %f", u));
    fp := eval(diff(phi(z), z), z = u); #Take a first derivative at u.
    fpp := eval(diff(phi(z), `$`(z, 2)), z = u); #Take a second derivative at u.
    u := evalf(-(fp-fpp*u)/fpp)
  end do;
  [u, evalb(fpp < 0)]:
end proc:
```

Algorithm 7. Newton's Method in One Dimension

Remark 3.43. Note that our implementation of Newton's Algorithm checks to see if the final value of $\phi''(x_k) < 0$. This allows us to make sure we identify a maximum rather than a minimum.

Example 3.44. Suppose we execute our bracketing algorithm on the function:

$$\phi(x) = -(x-4)^4 + 3(x-4)^3 + 6(x-4)^2 - 3(x-4) + 100$$

starting at 0 and 0.25. We obtain the bracket $(2.864863884, 2.899652455, 2.904582162)$. If we begin Newton's method with the second point of the bracket 2.899652455 we obtain the output:

```
"xk = 2.899652"
"xk = 2.900627"
Output = [2.900627632, true]
```

Exercise 29. Find a simple necessary condition for which Newton's Method converges in one iteration.

Exercise 30. Implement Newton's Method. Using

$$\phi(x) = -(x-4)^4 + 3(x-4)^3 + 6(x-4)^2 - 3(x-4) + 100$$

find a starting point for which Newton's Method converges to a maximum and another starting point for which Newton's Method converges to a minimum.

8. Convergence of Newton's Method

Remark 3.45. Most of this discussion is taken from Chapter 8.1 of [Avr03]. A slightly more general discussion can be found in [BSS06].

Definition 3.46. Let $S = [a, b] \subset \mathbb{R}$. A *contractor* or *contraction mapping* on S is a continuous function $f : S \to S$ with the property that there is a $q \in (0, 1)$ so that:

$$|f(x_1) - f(x_2)| \leq q|x_1 - x_2| \tag{3.11}$$

Remark 3.47. Definition 3.46 is a special form of Lipschitz continuity in which the Lipschitz constant is required to be less than 1.

Lemma 3.48. *Let $f : S \to S$ be a contraction mapping on $S = [a, b] \subset \mathbb{R}$. Define $x^{(k+1)} = f(x^{(k)})$ and suppose $x^{(0)} \in S$. Then there is a unique fixed point x^* to which the sequence $\{x^{(k)}\}$ converges and:*

$$|x^{(k+1)} - x^*| \leq q^{k+1}|x^{(0)} - x^*| \quad k = 0, 1, \dots \tag{3.12}$$

Proof. The fact that $f(x)$ has a fixed point is immediate from a variation on Brower's Fixed Point Theorem (see [Mun00], Page 351 - 353). Denote this point by x^*. The fact that f is a contraction mapping implies that:

$$|f(x^{(0)}) - x^*| \leq q|x^{(0)} - x^*| \tag{3.13}$$

which implies that:

$$|x^{(1)} - x^*| \leq q|x^{(0)} - x^*| \tag{3.14}$$

Now proceed by induction and assume for

$$|x^{(k)} - x^*| \leq q^k|x^{(0)} - x^*| \quad k = 0, 1, \dots, K \tag{3.15}$$

From Expression 3.15, we know that:

$$|f(x^{(K)}) - x^*| \leq q|x^{(K)} - x^*| \leq q^{K+1}|x^{(0)} - x^*| \tag{3.16}$$

Thus:

$$|x^{(K+1)} - x^*| \leq q^{K+1}|x^{(0)} - x^*| \tag{3.17}$$

Thus Expression 3.12, follows by induction. Finally, since $q < 1$, we see at once that $\{x^{(k)}\}$ converges to x^*.

To prove the uniqueness of x^*, suppose there is a second, distinct, fixed point x^+. Then:

$$0 < |x^* - x^+| = |f(x^*) - f(x^+)| \leq q|x^* - x^+| \tag{3.18}$$

As $0 < q < 1$, this is a contradiction unless $x^* = x^+$. Thus x^* is unique. This completes the proof. □

Lemma 3.49. *Suppose that f is continuously differentiable on $S = [a, b] \subset \mathbb{R}$ and maps S into itself. If $|f'(x)| < 1$ for every $x \in S$, then f is a contraction mapping.*

Proof. Let x_1 and x_2 be two points in S. Then by the Mean Value Theorem (Theorem 2.2) we know that there is some $x^* \in (x_1, x_2)$ so that:

$$f(x_1) = f(x_2) + f'(x^*)(x_1 - x_2) \tag{3.19}$$

Thus:

$$|f(x_1) - f(x_2)| = |f'(x^*)||(x_1 - x_2)| \tag{3.20}$$

By our assumption that $|f'(x^*)| < 1$ for all $x \in S$, we can set q equal to the maximal value of $|f'(x^*)|$ on S. The fact that f is a contraction mapping follows from the definition. □

THEOREM 3.50. *Let h and γ be continuously differentiable functions on $S = [a, b] \subset \mathbb{R}$. Suppose further that:*

$$h(a)h(b) < 0 \tag{3.21}$$

and for all $x \in S$:

$$\gamma(x) > 0 \tag{3.22}$$

$$h'(x) > 0 \tag{3.23}$$

$$0 \leq 1 - (\gamma(x)h(x))' \leq q < 1 \tag{3.24}$$

Let:

$$x^{(k+1)} = x^{(k)} - \gamma(x^{(k)})h(x^{(k)}) \quad k = 0, 1, \dots \tag{3.25}$$

with $x^{(0)} \in S$. Then the sequence $\{x^{(k)}\}$ converges to a solution x^* with $h(x^*) = 0$.

PROOF. Define:

$$\rho(x) = x - \gamma(x)h(x) \tag{3.26}$$

$$\rho'(x) = 1 - (\gamma(x)h(x))' \tag{3.27}$$

From Inequality 3.24, we see that $0 < \rho'(x) \leq q < 1$ for all $x \in S$ and ρ is monotonically increasing (non-decreasing) on S. Inequality 3.23 shows that h is monotonically increasing on S and therefore since $h(a)h(b) < 0$, it follows that $h(a) < 0$ and $h(b) > 0$. From this, and Equation 3.26 we conclude that $\rho(a) > a$ and $\rho(b) < b$, since $\gamma(x) > 0$ for all $x \in S$.By the monotonicity of ρ, we can conclude then that $a < \rho(x) < b$ for all $x \in S$. Thus, ρ maps S into itself. Moreover, since we assume that $|\rho'(x)| < 1$ in Inequality 3.24. From Lemma 3.49, it follows that $\rho(x)$ is a contraction mapping. □

COROLLARY 3.51. *Suppose $\phi : \mathbb{R} \to \mathbb{R}$ is a univariate function to be maximized that is three-times continuously differentiable. Let $h(x) = \phi'(x)$ and let $\gamma(x) = 1/\phi''(x)$. Then $\rho(x) = x - \gamma(x)h(x)$ is Newton's Method. If the hypotheses on $h(x)$ and $g(x)$ from Theorem 3.50 hold, then Newton's method converges to a stationary point ($\phi'(x) = 0$) on $S = [a, b] \subset \mathbb{R}$.* □

DEFINITION 3.52 (Rate of Convergence). If a sequence $\{\mathbf{x}_k\} \subseteq \mathbb{R}^n$ converges to $\curvearrowleft^* \in \mathbb{R}^n$, but the sequence does not attain $\mathbf{x}^*$ for any finite k. If there is a $p \in \mathbb{R}$ and an $\alpha \in \mathbb{R}$ with $\alpha \neq 0$ such that:

$$\lim_{k\to\infty} \frac{||\mathbf{x}_{k+1} - \mathbf{x}^*||}{||\mathbf{x}_k - \mathbf{x}^*||^p} = \alpha \tag{3.28}$$

then p is the *order of convergence* of the sequence $\{\mathbf{x}_k\}$. If $p = 1$, then convergence is *linear*. If $p > 1$, then convergence is *superlinear* and if $p = 2$, then convergence is *quadratic*.

THEOREM 3.53. *Suppose that the hypotheses of Theorem 3.50 and Corollary 3.51 hold and the sequence $\{x_k\} \subset \mathbb{R}$ is generated by Newton's method and this sequence converges to $x^* \in \mathbb{R}$. Then the rate of convergence is quadratic.*

PROOF. From Theorem 3.50 and Corollary 3.51, we are maximizing $\phi : S = [a, b] \subset \mathbb{R} \to \mathbb{R}$ with $h(x) = \phi'(x)$ and $\gamma(x) = 1/\phi''(x)$ and ϕ is three-times continuously differentiable.

From Theorem 3.50, we know that $\phi'(x^*) = h(x^*) = 0$. Furthermore, we know that x^* is a fixed point of the equation:

$$f(x_k) = x_k - \frac{h(x_k)}{h'(x_k)} \tag{3.29}$$

if and only if $h(x^*) = 0$. To see this, note that if $h(x^*) = 0$, then $\phi(x^*) = x^*$ and so x^* is a fixed point. If x^* is a fixed point, then:

$$0 = -\frac{h(x^*)}{h'(x^*)} \tag{3.30}$$

and thus, $h(x^*) = 0$. By the Mean Value Theorem (Theorem 2.2), for any iterate x_k, we know there is a $t \in \mathbb{R}$ between x_k and x^* so that there is a

$$x_{k+1} - x^* = f(x_k) - f(x^*) = f'(t)(x_k - x^*) \tag{3.31}$$

From this, we conclude that:

$$|x_{k+1} - x^*| = |f(x_k) - f(x^*)| = |f'(t)||x_k - x^*| \tag{3.32}$$

We note that:

$$f'(t) = 1 - \frac{h'(t)^2 - h''(t)h(t)}{h'(t)^2} = \frac{h''(t)h(t)}{h'(t)^2} \tag{3.33}$$

We can rewrite Equation 3.32 as:

$$|x_{k+1} - x^*| = |f(x_k) - f(x^*)| = \frac{|h''(t)||h(t)|}{h'(t)^2}|x_k - x^*| \tag{3.34}$$

Since $h(x^*) = 0$, we know that there is some value s between t and x^* so that:

$$|h(t)| = |h(t) - h(x^*)| = |h'(s)||t - x^*| \tag{3.35}$$

again by he Mean Value Theorem (Theorem 2.2). But since s is between t and x^* and t is between x_k and x^*, we conclude that:

$$|h(t)| = |h(t) - h(x^*)| = |h'(s)||t - x^*| \leq |h'(s)||x_k - x^*| \tag{3.36}$$

Combining Equations 3.34 and 3.36, yields:

$$|x_{k+1} - x^*| = \frac{|h''(t)||h(t)|}{h'(t)^2}|x_k - x^*| \leq \frac{|h''(t)||h'(s)|}{h'(t)^2}|x_k - x^*|^2 \tag{3.37}$$

If:

$$\beta = \sup_k \frac{|h''(t)||h'(s)|}{h'(t)^2}$$

then

$$|x_{k+1} - x^*| \leq \beta|x_k - x^*|^2 \tag{3.38}$$

The fact that $f'(x)$ is bounded is sufficient to ensure that the supremum β exists. Thus the convergence rate of Newton's Method is quadratic. □

Exercise 31. Using your code for Newton's Method, illustrate empirically that convergence is quadratic.

CHAPTER 4

Approximate Line Search and Convergence of Gradient Ascent

1. Approximate Search: Armijo Rule and Curvature Condition

Remark 4.1. It is sometimes the case that optimizing the function $\phi(\delta_k)$ is expensive. In this case, it would be nice to find a δ_k by that is sufficient to guarantee convergence of a gradient ascent (or related algorithm). We can then stop our line search before convergence to an optimal solution and speed up our overall optimization algorithm. The following conditions, usually called the *Wolfe Conditions* can be used to ensure convergence of gradient descent and related algorithms.

Definition 4.2 (Armijo Rule). Given a function $f : \mathbb{R}^n \to \mathbb{R}$ and an ascent direction $\mathbf{p}_k$ with constant $\sigma_1 \in (0, 1)$, the Armijo rule is satisfied if:

$$f(\mathbf{x}_k + \delta_k\mathbf{p}_k) - f(\mathbf{x}_k) \geq \sigma_1\delta_k\nabla f(\mathbf{x}_k)^T\mathbf{p}_k \tag{4.1}$$

Remark 4.3. Recall, $\phi(\delta_k) = f(\mathbf{x}_k + \delta_k\mathbf{p}_k)$ consequently, Equation 4.1 simply states that:

$$\phi(\delta_k) - \phi(0) \geq \sigma_1\delta_k\nabla f(\mathbf{x}_k)^T\mathbf{p}_k = \sigma_1\delta_k\phi'(0) \tag{4.2}$$

which simply means there is a sufficient increase in the value of the function.

Definition 4.4 (Curvature Rule). Given a function $f : \mathbb{R}^n \to \mathbb{R}$ and an ascent direction $\mathbf{p}_k$ with constant $\sigma_2 \in (\sigma_1, 1)$, the curvature condition is statisfied if:

$$\sigma_2\nabla f(\mathbf{x}_k)^T\mathbf{p}_k \geq \nabla f(\mathbf{x}_k + \delta_k\mathbf{p}_k)^T\mathbf{p}_k \tag{4.3}$$

Remark 4.5. Recall from Proposition 1.20 and Theorem 1.22:

$$\phi'(\delta_k) = \nabla f(\mathbf{x}_k)^T\mathbf{p}_k$$

Thus we can write the curvature condition as:

$$\sigma_2\phi'(0) \geq \phi'(\delta_k) \tag{4.4}$$

Remember, since $\nabla f(\mathbf{x}_k)^T\mathbf{p}_k > 0$ (since $\mathbf{p}_k$ is an ascent direction), $\phi'(0) > 0$. Thus, all the curvature condition is saying is that the slope of the univariate function ϕ at the chosen point δ_k is not so steep as it was at 0.

Example 4.6. Consider the function:

$$f(x, y) = \exp\left(-\frac{1}{10}\left(x^2 + y^2\right)\right)\cos\left(x^2 + y^2\right)$$

The function is illustrated in Figure 4.1. Suppose we start at point $x_0 = y_0 = 1$, then the function $\phi(t)$ is given by: $f(x_0 + t\nabla f(x_0, y_0), y_0 + t\nabla f(x_0, y_0))$. A plot of $\phi(t)$ is shown in Figure 4.2. Armijo's Rule simply says:

$$\phi(t) \geq \phi(0) + \sigma_1\phi'(0)t \tag{4.5}$$

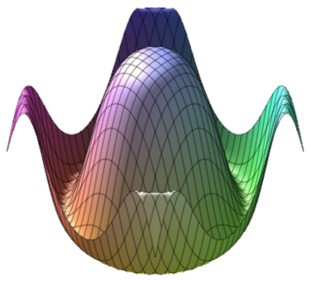


Figure 4.1. The function $f(x, y)$ has a maximum at $x = y = 0$, where the function attains a value of 1.

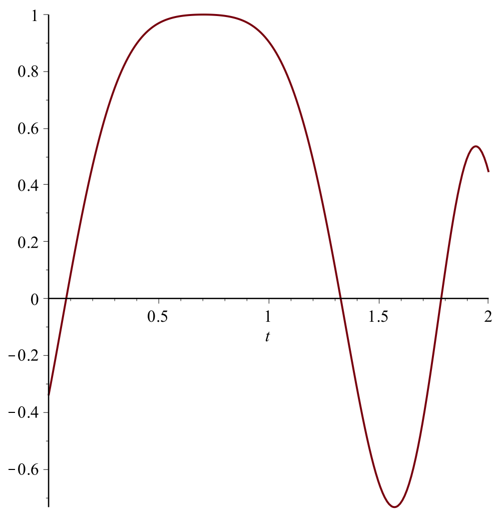


Figure 4.2. A plot of $\phi(t)$ illustrates the function increases as we approach the global maximum of the function and then decreases.

Here $\sigma_1 \in (0, 1)$ is a constant we choose and $\phi'(0)$ and $\phi(0)$ are computed from $\phi(t)$. This rule can be associated to the set of t so that $\phi(t)$ lies above the line $\phi(0) + \sigma_1\phi'(0)t$. This is illustrated in Figure 4.3. Likewise, the Curvature Condition asserts that:

$$\sigma_2\phi'(0) \geq \phi'(t) \tag{4.6}$$

This rule can be associated to the set of t so that $\phi'(t)$ is greater than some constant, where σ_2 is chosen between σ_1 and 1 and $\phi'(0)$ is computed. This is illustrated in Figure 4.3. When these two conditions are combined, it is clear that we have a good bracket of the maximum of $\phi(t)$.

Lemma 4.7. *Suppose* $\mathbf{p}_k$ *is an ascent direction at* $\mathbf{x}_k$ *and that* $f(\mathbf{x})$ *is bounded on the ray* $\mathbf{x}_k + t\mathbf{p}_k$*, where* $t \geq 0$*. Then if* $0 < \sigma_1 < \sigma_2 < 1$*, then there exists an interval of values for* t *satisfying the Wolfe Conditions.*

Proof. Define $\phi(t) = f(\mathbf{x}_k + t\mathbf{p}_k)$. The fact that $\mathbf{p}_k$ is an ascent direction, means:

$$\phi'(0) = \nabla f(\mathbf{x}_k)^T\mathbf{p}_k > 0$$

by Definition 3.2. Thus, since $0 < \sigma_1 < 1$, the line $l(t) = \phi(0) + \sigma_1\phi'(0)t$ is unbounded above and must intersection the graph of $\phi(t)$ at least once because $\phi(t)$ is bounded above by

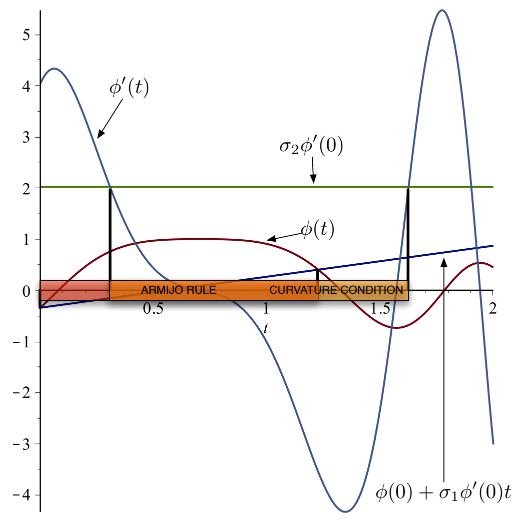


Figure 4.3. The Wolfe Conditions are illustrated. Note the region accepted by the Armijo rule intersects with the region accepted by the curvature condition to bracket the (closest local) maximum for δ_k. Here $\sigma_1 = 0.15$ and $\sigma_2 = 0.5$

assumption. (Note, $\phi(t)$ must lie above $l(t)$ initially since we choose $\sigma_1 < 1$ and at $\phi(0)$ the tangent line has slope $\phi'(0)$.) Let $t' > 0$ be the least such value of t at which $l(t)$ intersects $\phi(t)$. That is:

$$\phi(t') = \phi(0) + \sigma_1\phi'(0)t'$$

It follows that the interval $(0, t')$ satisfies the Armijo rule.

Applying the univariate Mean Value Theorem (Lemma 2.2), we see that there is a $t'' \in (0, t')$ so that:

$$\phi(t') - \phi(0) = \phi'(t'')(t' - 0)$$

Combining the two previous equations yields:

$$\sigma_1\phi'(0)t' = \phi'(t'')t'$$

Since $\sigma_2 > \sigma_1$, we conclude that:

$$\phi'(t'')t' = \sigma_1\phi'(0)t' < \sigma_2\phi'(0)t'$$

Thus $\phi'(t'') < \sigma_2\phi'(0)$ and t'' satisfies the Curvature Condition. Since we assumed that f was continuously differentiable, it follows that there is a neighborhood around t'' satisfying the Curvature Condition as well. This completes the proof. □

Remark 4.8. It turns out that the curvature condition is not necessary to ensure that gradient descent converges to a point $\mathbf{x}^*$ at which $\nabla f(\mathbf{x}^*) = 0$. From this observation, we can derive an algorithm for identifying a value for δ_k that satisfies the Armijo rule.

Remark 4.9. Algorithm 8 codifies our observation about the Armijo rule. Here, we choose a σ_1 (called `sigma` in the algorithm) and a value $\beta \in (0, 1)$ and an initial value t_0. We set $t = \beta^k t_0$ and iteratively test to see if

$$\phi(t) \geq \phi(0) + \sigma_1 t\phi'(0) \qquad (4.7)$$

When this holds, we terminate execution and return $t = \beta^k t_0$ for some k. The proof of Lemma 4.7 is sufficient to ensure this algorithm terminates with a point satisfying the Armijo rule.

```
BackTrace := proc (phi::operator, beta::numeric, t0::numeric, sigma)::numeric;
  local t, k, dphi;
  if beta < 0 or 1 < beta then
    t := -1
  else
    k := 0;
    t := beta^k*t0;
    dphi := evalf(eval(diff(phi(x), x), x = 0));

    while evalf(phi(t)) < evalf(phi(0))+sigma*dphi*t do
      k := k+1;
      t := beta^k*t0
    end do
  end if;
  t #return t.
end proc
```

Algorithm 8. The back-tracing algorithm

Remark 4.10. In practice, σ_1 is chosen small (between 10^{-5} and 0.1). The value β is generally between 0.1 and 0.5. Often $t_0 = 1$, though this varies based on problem at hand [Ber99].

2. Algorithmic Convergence

Definition 4.11 (Gradient Related). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function. A sequence of ascent directions $\{\mathbf{p}_k\}$ is *gradient related* if for any subsequence $K = \{k_1, k_2, \dots\}$ of $\{\mathbf{x}_k\}$ that converges to a non-stationary point of f the corresponding subsequence $\{\mathbf{p}_k\}_{k\in K}$ is bounded and has the property that:

$$\limsup_{k\to\infty, k\in K} \nabla f(\mathbf{x}_k)^T \mathbf{p}_k > 0 \tag{4.8}$$

Theorem 4.12. *Assume that δ_k is chosen to ensure the Armijo rule holds at each iteration and the ascent directions $\mathbf{p}_k$ are chosen so they are gradient related. Then if Algorithm 2 converges, it converges to a point $\mathbf{x}^*$ so that $\nabla f(\mathbf{x}^*) = \mathbf{0}$ and the stopping criteria:*

$$||\nabla f(\mathbf{x}_k)|| < \epsilon \tag{4.9}$$

for some small $\epsilon > 0$ may be used.

Proof. If the algorithm converges to a point $\mathbf{x}^*$ at which $\nabla f(\mathbf{x}^*) = \mathbf{0}$, then the stopping criteria $||\nabla f(\mathbf{x}_k)|| < \epsilon$ can clearly be used.

We proceed by contradiction: Suppose that the sequence of points $\{\mathbf{x}_k\}$ converges to a point $\mathbf{x}^+$ with the property that $\nabla f(\mathbf{x}^+) \neq \mathbf{0}$. We construct $\{\mathbf{x}_k\}$ so that $\{f(\mathbf{x}_k)\}$ is a monotonically nondecreasing sequence and therefore, either $\{f(\mathbf{x}_k)\}$ converges to some finite value or diverges to $+\infty$. By assumption, $f(\mathbf{x})$ is continuous and since $\mathbf{x}^+$ is a limit point of $\{\mathbf{x}_k\}$, it follows $f(\mathbf{x}^+)$ is a limit point of $\{f(\mathbf{x}_k)\}$ and thus the sequence $\{f(\mathbf{x}_k)\}$ must converge to this value. That is:

$$\lim_{k\to\infty} f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) = 0 \tag{4.10}$$

From the Armijo rule we have:

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) \geq \sigma_1 \delta_k \nabla f(\mathbf{x}_k)^T \mathbf{p}_k \tag{4.11}$$

since $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k \mathbf{p}_k$. Note the left hand side of this inequality goes to zero, while the right hand side is always positive ($\mathbf{p}_k$ are ascent directions and δ_k and σ_1 are positive). Hence:

$$\lim_{k\to\infty} \delta_k \nabla f(\mathbf{x}_k)^T \mathbf{p}_k = 0 \tag{4.12}$$

Let $K = \{k_1, k_2, \dots\}$ be any subsequence of $\{\mathbf{x}_k\}$ that converges to $\mathbf{x}^+$. Since $\mathbf{p}_k$ is gradient related, we know that:

$$\limsup_{k\to\infty, k\in K} \nabla f(\mathbf{x}_k)^T \mathbf{p}_k > 0 \tag{4.13}$$

Therefore, $\lim_{k\in K} \delta_k = 0$ (otherwise, Equation 4.12 can't hold). Consider the backtrace algorithm. Since $\lim_{k\in K} \delta_k = 0$, we know that for some $k^+ \in K$ so that if $k \geq k^+$:

$$f(\mathbf{x}_k + \delta_k/\beta \mathbf{p}_k) - f(\mathbf{x}_k) < \sigma_1 \delta_k/\beta \nabla f(\mathbf{x}_k)^T \mathbf{p}_k \tag{4.14}$$

That is, if $k \geq k^+$, the while-loop in the Algorithm 8 executes at least once and when it executes, it only stops once Inequality 4.14 *doesn't hold*. Therefore, at the iteration before Inequality 4.14 *must hold*.

Since $\{\mathbf{p}_k\}$ is gradient related, the sequence $\{\mathbf{p}_k\}$, for $k \in K$, is bounded and thus by the Bolzano-Weierstrass theorem there is a subsequence $K^+ \subseteq K$ so that $\{\mathbf{p}_k\}_{k\in K^+}$ converges to a vector $\mathbf{p}^+$. From Equation 4.14:

$$\frac{f(\mathbf{x}_k + \delta_k^+ \mathbf{p}_k) - f(\mathbf{x}_k)}{\delta_k^+} < \sigma_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k \tag{4.15}$$

where $\delta_k^+ = \delta_k/\beta$. Let $\phi(\delta_k) = f(\mathbf{x}_k + \delta_k \mathbf{p}_k)$. Remember, that $\phi'(\delta_k) = \nabla f(\mathbf{x}_k + \delta_k \mathbf{p}_k)^T \mathbf{p}_k$. We can rewrite Expression 4.15 as:

$$\frac{\phi(\delta_k^+) - \phi(0)}{\delta_k^+} < \sigma_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k \tag{4.16}$$

Applying the mean value theorem: means there is a $\bar{\delta}_k \in (0, \delta_k^+)$ so that:

$$\frac{\phi'(\bar{\delta}_k)(\delta_k^+ - 0)}{\delta_k^+} = \phi'(\bar{\delta}_k) < \sigma_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k \tag{4.17}$$

If we write $\phi'(\bar{\delta}_k)$ in terms of the gradient of f and $\mathbf{p}_k$ we obtain:

$$\nabla f(\mathbf{x}_k + \bar{\delta}_k \mathbf{p}_k)^T \mathbf{p}_k < \sigma_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k \tag{4.18}$$

Taking the limit as $k \to \infty$ (and $k \in K$) we see that:

$$\nabla f(\mathbf{x}^+)^T \mathbf{p}^+ < \sigma_1 \nabla f(\mathbf{x}^+)^T \mathbf{p}^+ \tag{4.19}$$

But since $\sigma_1 \in (0, 1)$, this cannot be true, contradicting our assumption that $\nabla f(\mathbf{x}^+) \neq \mathbf{0}$. $\square$

Corollary 4.13. *Assume that δ_k is chosen by maximizing $\phi(\delta_k)$ at each iteration and the ascent directions $\mathbf{p}_k$ are chosen so they are gradient related. Then Algorithm 2 converges to a point $\mathbf{x}^*$ so that $\nabla f(\mathbf{x}^*) = \mathbf{0}$.*

Corollary 4.14. *Suppose that $\mathbf{p}_k = \mathbf{B}_k^{-1}\nabla f(\mathbf{x}_k)$ in each iteration of Algorithm 2 and every matrix $\mathbf{B}_k$ is symmetric, positive definite. Then, Algorithm 2 converges to a stationary point of f.*

Exercise 32. Prove Corollary 4.13 by arguing that if $\mathbf{x}_{k+1}$ is generated from $\mathbf{x}_k$ by maximizing $\phi(\delta_k)$, then:

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) \geq f(\hat{\mathbf{x}}_{k+1}) - f(\mathbf{x}_k) \geq \sigma\hat{\delta}_k \nabla f(\mathbf{x}_k)^T \mathbf{p}_k \tag{4.20}$$

where $\hat{\mathbf{x}}_{k+1}$ and $\hat{\delta}_k$ are generated by the Armijo rule.

Exercise 33. Prove Corollary 4.14. [Hint: Use Lemma 3.8.]

Example 4.15 (Convergence Failure). We illustrate a simple case in which we do not use minimization or the Armijo rule to chose a step length. Consider the function:

$$f(x) = \begin{cases} -\frac{4}{5}\left(1-x\right)^2 + 2 - 2x & 1 < x \\ -\frac{4}{5}\left(1+x\right)^2 + 2 + 2x & x < -1 \\ -x^2 + 1 & \text{otherwise} \end{cases}$$

Suppose we start at $x_0 = -2$ and fix a decreasing step length $\delta_k = 1$. If we follow a gradient ascent, then we obtain a sequence of values $\{x_k\}$ $(k = 1, 2, \dots)$ shown in Figure 4.4. At each

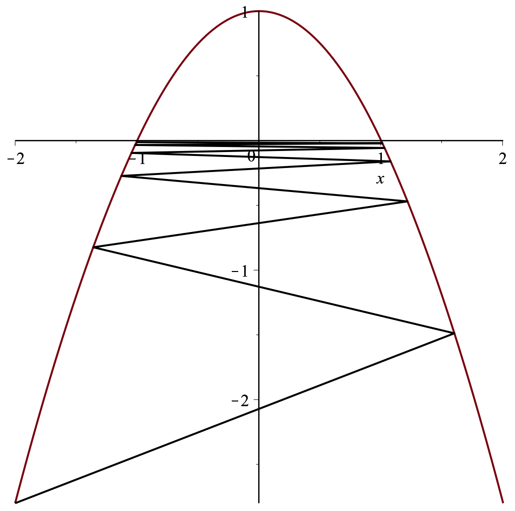


Figure 4.4. We illustrate the failure of the gradient ascent method to converge to a stationary point when we do not use the Armijo rule or minimization.

algorithmic step we have:

$$x_{k+1} = x_k + \left.\frac{df}{dx}\right|_{x=x_k} \tag{4.21}$$

Evaluation of Equation 4.21 yields:

$$x_{k+1} = (-1)^n \cdot \left(\frac{3}{5}\right)^n \cdot x_0 + (-1)^{n+1} \cdot \left(1 - \left(\frac{3}{5}\right)^n\right) \tag{4.22}$$

As $n \to \infty$, we see this sequence oscillates between -1 and 1 and does not converge.

Remark 4.16. We state, but do not prove, the capture theorem, a result that quantifies, in some sense, why gradient methods tend to converge to unique limit points. The proof can be found in [Ber99] (50 - 51).

Theorem 4.17 (Capture Theorem). *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable and let $\{\mathbf{x}_k\}$ be a sequence satisfying $f(\mathbf{x}_{k+1}) \geq f(\mathbf{x}_k)$ for all k generated as $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k \mathbf{p}_k$ that is convergent in the sense that every limit point of sequences that it generates is a stationary point of f. Assume there exists scalars $s > 0$ and $c > 0$ so that for all k we have:*

$$\delta_k \leq s \quad ||\mathbf{p}_k|| \leq c||\nabla f(\mathbf{x}_k)|| \tag{4.23}$$

Let $\mathbf{x}^*$ be a local maximum of f be the only stationary point in an open neighborhood of $\mathbf{x}^*$. Then there is an open set S containing $\mathbf{x}^*$ so that if $\mathbf{x}_K \in S$ for some $K \geq 0$, then $\mathbf{x}_k \in S$ for all $k \geq K$ and $\{\mathbf{x}_k\}$ $(k \geq K)$ converges to $\mathbf{x}^*$. Furthermore, given any scalar $\epsilon > 0$, the set S can be chosen so that $||\mathbf{x} - \mathbf{x}^*|| < \epsilon$ for all $\mathbf{x} \in S$. □

3. Rate of Convergence for Pure Gradient Ascent

Remark 4.18. We state, but do not prove Kantorovich's Lemma. A sketh of the proof can be found on Page 76 of [Ber99]. A variation of the proof provided for Theorem 4.20 can be found in [Ber99] for *gradient descent*.

Lemma 4.19 (Kantorovich's Inequality). *Let $\mathbf{Q} \in \mathbb{R}^{n\times n}$ be a symmetric positive definite matrix furthermore, suppose the eigenvalues of $\mathbf{Q}$ are $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. For any $\mathbf{x} \in \mathbb{R}^{n\times 1}$ with $\mathbf{x} \neq \mathbf{0}$ we have:*

$$\frac{\left(\mathbf{x}^T\mathbf{x}\right)^2}{\left(\mathbf{x}^T\mathbf{Q}\mathbf{x}\right)\left(\mathbf{x}^T\mathbf{Q}^{-1}\mathbf{x}\right)} \geq \frac{4\lambda_1\lambda_n}{\left(\lambda_1 + \lambda_n\right)^2} \tag{4.24}$$

Exercise 34. Prove that Kantorovich's Inequality also holds for negative definite matrices.

Theorem 4.20. *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a strictly concave function defined by:*

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} \tag{4.25}$$

where $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is symmetric (and negative definite). Suppose the sequence $\{\mathbf{x}_k\}$ is generated using an exact line search method and $\mathbf{x}^*$ is the (unique) global maximizer of f. Finally, suppose the eigenvalues of $\mathbf{Q}$ are $0 > \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Then:

$$f(\mathbf{x}_{k+1}) \geq \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^2 f(\mathbf{x}_k) \tag{4.26}$$

Remark 4.21. The value:

$$\left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)$$

is called the *condition number* of the optimization problem. Condition numbers close to 1 lead to slow convergence of gradient ascent (see Example 4.24).

Proof. Denote by $\mathbf{g}_k = \nabla f(\mathbf{x}_k) = \mathbf{Q}\mathbf{x}_k$. If $\mathbf{g}_k = \mathbf{0}$, then $f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k)$ and the theorem is proved. Suppose $\mathbf{g}_k \neq \mathbf{0}$. Consider the function:

$$\phi(\delta) = f(\mathbf{x}_k + \delta\mathbf{g}_k) = \frac{1}{2}\left(\mathbf{x}_k + \delta\mathbf{g}_k\right)^T \mathbf{Q}\left(\mathbf{x}_k + \delta\mathbf{g}_k\right) \tag{4.27}$$

We see at once that:

$$\phi(\delta) = f(\mathbf{x}_k) + \frac{1}{2}\delta\mathbf{x}_k^T\mathbf{Q}\mathbf{g}_k + \frac{1}{2}\delta\mathbf{g}_k^T\mathbf{Q}\mathbf{x}_k + \delta^2 f(\mathbf{g}_k) \tag{4.28}$$

Note that since $\mathbf{Q} = \mathbf{Q}^T$ and $\mathbf{Q}\mathbf{x}_k = \mathbf{g}_k$ and $\mathbf{x}_k^T\mathbf{Q}^T = \mathbf{x}_k^T\mathbf{Q} = \mathbf{g}_k^T$ we have:

$$\phi(\delta) = f(\mathbf{x}_k) + \frac{1}{2}\delta\mathbf{g}_k^T\mathbf{g}_k + \frac{1}{2}\delta\mathbf{g}_k^T\mathbf{g}_k + \delta^2 f(\mathbf{g}_k) = f(\mathbf{x}_k) + \delta\mathbf{g}_k^T\mathbf{g}_k + \delta^2 f(\mathbf{g}_k) \tag{4.29}$$

Differentiating and setting $\phi'(\delta) = 0$ yields:

$$\mathbf{g}_k^T\mathbf{g}_k + 2\delta f(\mathbf{g}_k) = 0 \implies \delta = \frac{-\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{g}_k^T\mathbf{Q}\mathbf{g}_k} \tag{4.30}$$

Notice that $\delta > 0$ because $\mathbf{Q}$ is negative definite. Substituting this value into $f(\mathbf{x}_{k+1})$ yields:

$$\begin{aligned} f(\mathbf{x}_{k+1}) &= f(\mathbf{x}_k + \delta\mathbf{g}_k) = \frac{1}{2}\left(\mathbf{x}_k + \delta\mathbf{g}_k\right)^T \mathbf{Q}\left(\mathbf{x}_k + \delta\mathbf{g}_k\right) = \\ & f(\mathbf{x}_k) + \delta\mathbf{g}_k^T\mathbf{g}_k + \delta^2 f(\mathbf{g}_k) = f(\mathbf{x}_k) + \left(\frac{-\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{g}_k^T\mathbf{Q}\mathbf{g}_k}\right)\mathbf{g}_k^T\mathbf{g}_k + \left(\frac{-\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{g}_k^T\mathbf{Q}\mathbf{g}_k}\right)^2 f(\mathbf{g}_k) = \\ & \frac{1}{2}\mathbf{x}_k^T\mathbf{Q}\mathbf{x}_k + \frac{-\left(\mathbf{g}_k^T\mathbf{g}_k\right)^2}{\mathbf{g}_k^T\mathbf{Q}\mathbf{g}_k} + \frac{1}{2}\frac{\left(\mathbf{g}_k^T\mathbf{g}_k\right)^2}{\mathbf{g}_k^T\mathbf{Q}\mathbf{g}_k} = \frac{1}{2}\left(\mathbf{x}_k^T\mathbf{Q}\mathbf{x}_k - \frac{\left(\mathbf{g}_k^T\mathbf{g}_k\right)^2}{\mathbf{g}_k^T\mathbf{Q}\mathbf{g}_k}\right) \end{aligned} \tag{4.31}$$

Note that since $\mathbf{Q}$ is symmetric:

$$\mathbf{x}_k^T\mathbf{Q}\mathbf{x}_k = \mathbf{x}_k^T\mathbf{Q}^T\mathbf{Q}^{-1}\mathbf{Q}\mathbf{x}_k = \mathbf{g}_k^T\mathbf{Q}^{-1}\mathbf{g}_k \tag{4.32}$$

Therefore, we can write:

$$\begin{aligned} f(\mathbf{x}_{k+1}) &= \frac{1}{2}\left(\mathbf{x}_k^T\mathbf{Q}\mathbf{x}_k - \frac{\left(\mathbf{g}_k^T\mathbf{g}_k\right)^2}{\mathbf{g}_k^T\mathbf{Q}\mathbf{g}_k}\right) = \\ & \frac{1}{2}\left(1 - \frac{\left(\mathbf{g}_k^T\mathbf{g}_k\right)^2}{\left(\mathbf{g}_k^T\mathbf{Q}\mathbf{g}_k\right)\left(\mathbf{g}_k^T\mathbf{Q}^{-1}\mathbf{g}_k\right)}\right)\mathbf{x}_k^T\mathbf{Q}\mathbf{x}_k = \left(1 - \frac{\left(\mathbf{g}_k^T\mathbf{g}_k\right)^2}{\left(\mathbf{g}_k^T\mathbf{Q}\mathbf{g}_k\right)\left(\mathbf{g}_k^T\mathbf{Q}^{-1}\mathbf{g}_k\right)}\right) f(\mathbf{x}_k) \end{aligned} \tag{4.33}$$

Since $\mathbf{Q}$ is negative definite (and thus $-\mathbf{Q}$ is positive definite), we know three things:

(1) $f(\mathbf{x}_k), f(\mathbf{x}_{k+1}) \leq 0$
(2) We know:

$$\frac{\left(\mathbf{g}_k^T\mathbf{g}_k\right)^2}{\left(\mathbf{g}_k^T\mathbf{Q}\mathbf{g}_k\right)\left(\mathbf{g}_k^T\mathbf{Q}^{-1}\mathbf{g}_k\right)} > 0$$

(3) Thus, in order for the equality in Expression 4.33 to be true it follows that:

$$0 < 1 - \frac{\left(\mathbf{g}_k^T\mathbf{g}_k\right)^2}{\left(\mathbf{g}_k^T\mathbf{Q}\mathbf{g}_k\right)\left(\mathbf{g}_k^T\mathbf{Q}^{-1}\mathbf{g}_k\right)} < 1$$

Lemma 4.19 and our previous observations imply that:

$$f(\mathbf{x}_{k+1}) \geq \left(1 - \frac{4\lambda_1\lambda_n}{(\lambda_1+\lambda_n)^2}\right) f(\mathbf{x}_k) = \left(\frac{(\lambda_1+\lambda_n)^2}{(\lambda_1+\lambda_n)^2} - \frac{4\lambda_1\lambda_n}{(\lambda_1+\lambda_n)^2}\right) f(\mathbf{x}_k) = \left(\frac{\lambda_1^2 - 2\lambda_1\lambda_2 + \lambda_n^2}{(\lambda_1+\lambda_n)^2}\right) f(\mathbf{x}_k) = \left(\frac{(\lambda_1-\lambda_n)^2}{(\lambda_1+\lambda_n)^2}\right) f(\mathbf{x}_k) = \left(\frac{\lambda_1-\lambda_n}{\lambda_1+\lambda_n}\right)^2 f(\mathbf{x}_k) \tag{4.34}$$

This completes the proof. □

Remark 4.22. It's a little easier to see the proof of the previous theorem when we are minimizing, since everything is positive instead of negative and we do not have to keep track of sign changes. Nevertheless, such exercises are good to ensure understanding of the proof.

Remark 4.23. A reference implementation for Gradient Ascent is shown in Algorithm 9. The `LineSearch` method called at Line 49 is a combination of the parabolic bracketing algorithm (Algorithm 3 and Golden Section Search (Algorithm 5) along with a simple backtrace (Algorithm 8) to find the second parameter that is passed into the parabolic bracketing algorithm (starting with $a = 0$ and $\Delta a = 1$ we check to see if $\phi(a + \Delta a) > \phi(a)$, if not, we set $\Delta a = \beta\Delta a$).

Example 4.24. Consider the function $F(x, y) = -2\,x^2 - 10\,y^2$. If we initialize our gradient ascent at $x = 15$, $y = 5$ and set $\epsilon = 0.001$, then we obtain the following output, which converges near the optimal point $x^* = 0$, $y^* = 0$. The zig-zagging motion is typical of the gradient ascent algorithm in cases where the largest and smallest eigenvalues for the matrix $\mathbf{Q}$ (in a pure quadratic function) are very different (see Theorem 4.20). In this case,

$$\mathbf{Q} = \begin{bmatrix} -2 & 0 \\ 0 & -10 \end{bmatrix}$$

Thus, we can see that the condition number is 2/3, which is close enough to 1 to cause slow convergence, leading to the zig-zagging.

Exercise 35. Implement Gradient Ascent with inexact line search. Using $F(x, y)$ from Example 4.24, draw a picture like Figure 4.5 to illustrate your algorithm's steps.

4. Rate of Convergence for Basic Ascent Algorithm

Theorem 4.25. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable and suppose that $\{\mathbf{x}_k\}$ is generated by Algorithm 2 so that $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k\mathbf{p}_k$ and suppose that $\{\mathbf{x}_k\}$ converges to $\mathbf{x}^*$, $\nabla f(\mathbf{x}^*) = \mathbf{0}$ and $\mathbf{H}(\mathbf{x}^*) = \nabla^2 f(\mathbf{x}^*)$ is negative definite. Finally, assume that $\nabla f(\mathbf{x}_k) \neq \mathbf{0}$ for any $k = 1, 2, \dots$ and:*

$$\lim_{k\to\infty} \frac{||\mathbf{p}_k + \mathbf{H}^{-1}(\mathbf{x}^*)\nabla f(\mathbf{x}_k)||}{||\nabla f(\mathbf{x}_k)||} = 0 \tag{4.35}$$

Then if δ_k is chosen by means of the backtrace algorithm with $t_0 = 1$ and $\sigma_1 < \frac{1}{2}$, we have:

$$\lim_{k\to\infty} \frac{||\mathbf{x}_{k+1} - \mathbf{x}^*||}{||\mathbf{x}_k - \mathbf{x}^*||} = 0 \tag{4.36}$$

Furthermore, there is an integer $\bar{k} \geq 0$ so that $\delta_k = 1$ for all $k \geq \bar{k}$ (that is, eventually, the backtrace algorithm will converge with no iterations of the while-loop).

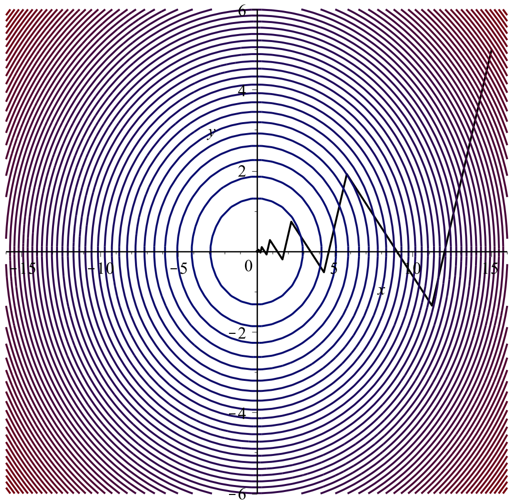


Figure 4.5. Gradient ascent is illustrated on the function $F(x, y) = -2\,x^2 - 10\,y^2$ starting at $x = 15$, $y = 5$. The zig-zagging motion is typical of the gradient ascent algorithm in cases where λ_n and λ_1 are very different (see Theorem 4.20).

Remark 4.26. This theorem asserts that if we have a method for choosing our ascent direction $\mathbf{p}_k$ so that at large iteration numbers, the ascent direction approaches a Newton step, then the algorithm will converge super-linearly (since Equation 4.36 goes to zero.)

Remark 4.27. We note that in Newton's Method we have $\mathbf{p}_k = -\mathbf{H}^{-1}(\mathbf{x}^*)$. Thus, Equation 4.35 tells us that in the limit as k approaches ∞, we are taking pure Newton steps.

Proof. By the Second Order Mean Value Theorem (Lemma 2.4), we know that there is some $t \in (0, 1)$ so that:

$$f(\mathbf{x}_k + \mathbf{p}_k) - f(\mathbf{x}_k) = \nabla f(\mathbf{x}_k)^T \mathbf{p}_k + \frac{1}{2}\mathbf{p}_k^T \mathbf{H}(\mathbf{x}_k + t\mathbf{p}_k)\mathbf{p}_k \tag{4.37}$$

If we can show that for k sufficiently large we have:

$$\nabla f(\mathbf{x}_k)^T \mathbf{p}_k + \frac{1}{2}\mathbf{p}_k^T \mathbf{H}(\mathbf{x}_k + t\mathbf{p}_k)\mathbf{p}_k \geq \sigma_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k \tag{4.38}$$

then we will have established that there is a $\bar{k}$ so that for any $k \geq \bar{k}$, the backtrace algorithm converges with no executions of the while loop. Equation 4.38 may equivalently be written as:

$$(1 - \sigma_1)\nabla f(\mathbf{x}_k)^T \mathbf{p}_k + \frac{1}{2}\mathbf{p}_k^T \mathbf{H}(\mathbf{x}_k + t\mathbf{p}_k)\mathbf{p}_k \geq 0 \tag{4.39}$$

If we let:

$$\mathbf{q}_k = \frac{\nabla f(\mathbf{x}_k)}{||\nabla f(\mathbf{x}_k)||}$$
$$\mathbf{r}_k = \frac{\mathbf{p}_k}{||\nabla f(\mathbf{x}_k)||}$$

Dividing Inequality 4.39 by $||\nabla f(\mathbf{x}_k)||^2$, we obtain an equivalent inequality:

$$(1 - \sigma_1)\mathbf{q}_k^T \mathbf{r}_k + \frac{1}{2}\mathbf{r}_k^T \mathbf{H}(\mathbf{x}_k + t\mathbf{p}_k)\mathbf{r}_k \geq 0 \tag{4.40}$$

From Equation 4.35, we know that:

$$\lim_{k\to\infty} ||\mathbf{r}_k + \mathbf{H}^{-1}(\mathbf{x}^*)\mathbf{q}_k|| = 0 \tag{4.41}$$

The fact that $\mathbf{H}(\mathbf{x}^*)$ is negative definite, and $||\mathbf{q}_k|| = 1$, it follows that $\mathbf{r}_k$ must be a bounded sequence and since $\nabla f(\mathbf{x}_k)$ converges to $\nabla f(\mathbf{x}^*) = \mathbf{0}$, it's clear that $\mathbf{p}_k$ must also converge to $\mathbf{0}$ and hence $\mathbf{x}_k + \mathbf{p}_k$ converges to $\mathbf{x}^*$. From this, it follows that $\mathbf{x}_k + t\mathbf{p}_k$ also converges to $\mathbf{x}^*$ and $\mathbf{H}(\mathbf{x}_k + t\mathbf{p}_k)$ converges to $\mathbf{H}(\mathbf{x}^*)$. We can re-write Equation 4.35 as:

$$\mathbf{r}_k = -\mathbf{H}^{-1}(\mathbf{x}^*)\mathbf{q}_k + \mathbf{z}_k \tag{4.42}$$

where $\{\mathbf{z}_k\}$ is a sequence of vectors that converges to $\mathbf{0}$. Substituting the previous equation into Equation 4.40 yields:

$$(1-\sigma_1)\mathbf{q}_k^T\left(-\mathbf{H}^{-1}(\mathbf{x}^*)\mathbf{q}_k + \mathbf{z}_k\right) + \\ \frac{1}{2}\left(-\mathbf{H}^{-1}(\mathbf{x}^*)\mathbf{q}_k + \mathbf{z}_k\right)^T \mathbf{H}(\mathbf{x}_k + t\mathbf{p}_k)\left(-\mathbf{H}^{-1}(\mathbf{x}^*)\mathbf{q}_k + \mathbf{z}_k\right) \geq 0 \tag{4.43}$$

The preceding equation can be re-written as:

$$-(1-\sigma_1)\mathbf{q}_k^T\mathbf{H}^{-1}(\mathbf{x}^*)\mathbf{q}_k + \frac{1}{2}\left(\mathbf{H}^{-1}(\mathbf{x}^*)\mathbf{q}_k\right)^T \mathbf{H}(\mathbf{x}_k + t\mathbf{p}_k)\left(\mathbf{H}^{-1}(\mathbf{x}^*)\mathbf{q}_k\right) + g(\mathbf{z}_k) \geq 0 \tag{4.44}$$

where $g(\mathbf{z}_k)$ is a function of $\mathbf{z}_k$ with the property that $g(\mathbf{z}_k) \to 0$ as $k \to \infty$. Since $\mathbf{H}(\mathbf{x})$ is symmetric (and so is its inverse), for large enough k, $\mathbf{H}(\mathbf{x}_k + t\mathbf{p}_k) \sim \mathbf{H}(\mathbf{x}^*)$ and we can re-write the previous expression as:

$$-(1-\sigma_1)\mathbf{q}_k^T\mathbf{H}^{-1}(\mathbf{x}^*)\mathbf{q}_k + \frac{1}{2}\mathbf{q}_k^T\mathbf{H}^{-1}(\mathbf{x}^*)\mathbf{q}_k + \bar{g}(\mathbf{z}_k) \geq 0 \tag{4.45}$$

Where $\bar{g}(\mathbf{z}_k)$ is another function with the same properties as $g(\mathbf{z}_k)$. We can re-write this expression as:

$$-\left(\frac{1}{2} - \sigma_1\right)\mathbf{q}_k^T\mathbf{H}^{-1}(\mathbf{x}^*)\mathbf{q}_k + \bar{g}(\mathbf{z}_k) \geq 0 \tag{4.46}$$

Now, since $\sigma_1 < \frac{1}{2}$ and $\mathbf{H}^{-1}(\mathbf{x}^*)$ is negative definite and $\bar{g}(\mathbf{z}_k) \to 0$, the inequality in Expression 4.46 must hold for very large k. Thus, Expression 4.39 must hold and therefore Expression 4.38 must hold. Therefore, it follows there is some $\bar{k}$ so that for all $k \geq \bar{k}$, the unit step in the Armijo rule is sufficient and no iterations of the while-loop in the back-tracing algorithm are executed.

We can re-write Expression 4.35 as:

$$\mathbf{p}_k + \mathbf{H}^{-1}(\mathbf{x}^*)\nabla f(\mathbf{x}_k) = ||\nabla f(\mathbf{x}_k)||\mathbf{y}_k \tag{4.47}$$

where $\mathbf{y}_k$ is a vector sequence that approaches $\mathbf{0}$ as k approaches infinity. Applying Taylor's Theorem to the many-valued function $\nabla f(\mathbf{x}_k)$, we see that:

$$\nabla f(\mathbf{x}_k) = \nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^*) = \mathbf{H}(\mathbf{x}^*)(\mathbf{x}_k - \mathbf{x}^*) + o\left(||\mathbf{x}_k - \mathbf{x}^*||\right) \tag{4.48}$$

since $\nabla f(\mathbf{x}^*) = \mathbf{0}$, by assumption. This implies:

$$\mathbf{H}^{-1}(\mathbf{x}^*)\nabla f(\mathbf{x}_k) = (\mathbf{x}_k - \mathbf{x}^*) + o\left(||\mathbf{x}_k - \mathbf{x}^*||\right) \tag{4.49}$$

We also see that: $||\nabla f(\mathbf{x}_k)|| = O(||\mathbf{x}_k - \mathbf{x}^*||)$. These two relations show that:

$$\mathbf{p}_k + (\mathbf{x}_k - \mathbf{x}^*) = o\left(||\mathbf{x}_k - \mathbf{x}^*||\right) \tag{4.50}$$

When $k \geq \bar{k}$, we know that:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k \tag{4.51}$$

Combining this with Equation 4.50 yields:

$$\mathbf{x}_{k+1} - \mathbf{x}^* = o\left(||\mathbf{x}_k - \mathbf{x}^*||\right) \tag{4.52}$$

Now, letting k approach infinity yields:

$$\lim_{k\to\infty} \frac{||\mathbf{x}_{k+1} - \mathbf{x}^*||}{||\mathbf{x}_k - \mathbf{x}^*||} = \lim_{k\to\infty} \frac{o\left(||\mathbf{x}_k - \mathbf{x}^*||\right)}{||\mathbf{x}_k - \mathbf{x}^*||} = 0 \tag{4.53}$$

This completes the proof. $\square$

```
GradientAscent := proc (F::operator, initVals::list, epsilon::numeric) :: list;
  # A function to execute Gradient Ascent! Pass in a function
  # and an initial point in the format [x = x0, y=y0, z=z0,...]
  # epsilon is the tolerance on the gradient.

  local vars, xnow, i, G, normG, OUT, passIn, phi, LS, vals, ttemp;

  # The first few lines are housekeeping...
  vars := [];
  xnow := [];
  vals := initVals;
  for i to nops(initVals) do
    vars := [op(vars), lhs(initVals[i])];
    xnow := [op(xnow), rhs(initVals[i])]
  end do;

  # Compute the gradient of the function using the VectorCalculus package
  # This looks nicer in "Maple" Format.
  G := Vector(Gradient](F(op(vars)), vars));

  # Compute the norm of the gradient at the initial point. This uses the Linear
  # Algebra package.
  normG := Norm(evalf(eval(G, initVals)));

  # Output will be the path we take to get to the "optimal point."
  # You can just output the last point for simplicity.
  OUT := [];

  # While we're not at a stationary point...
  while evalb(epsilon < normG) do

    # Update the output.
    OUT := [op(OUT), xnow];

    # Do some house keeping: This is x_next = x_now + s * G(x_now)
    # Essentially, we're going to be solving for the distance to walk "s" below.
    # Most of this line is just getting Maple to treat things as floating point
    # and returning a list of the inputs we can pass to F.
    passIn := convert(Vector(xnow) + s * evalf(eval(G, vals)), list);
    # Create a local procedure to be the line function f(x_k + delta * p_k)
    # Here 't' is playing the role of delta.
    phi := proc (t) options operator, arrow;
        evalf(eval(F(op(passIn)), s = t))
      end proc;

    # Get the output from a line search function. I'm using the
    # ParabolicBracket code and the GoldenSection search
    # found in the notes.
    ttemp := LineSearch(phi);

    # Compute the new value for xnow using the 't' we just found.
    xnow := evalf(eval(passIn, [s = ttemp]));

    # Do some housekeeping.
    vals := [];
    for i to nops(vars) do
      vals := [op(vals), vars[i] = xnow[i]]
    end do;

    # Evaluate the norm of the gradient at the new point.
    normG := Norm(evalf(eval(G, vals)))
  end do;

  # Record the very last point you found (it's the local max).
  OUT := [op(OUT), xnow];

  #Return the list of points.
  OUT
end proc;
```

Algorithm 9. Gradient Ascent Reference Implementation

CHAPTER 5

Newton's Method and Corrections

1. Newton's Method

Remark 5.1. Suppose $f : \mathbb{R}^n \to \mathbb{R}$ and f is twice continuously differentiable. Recall in our general ascent algorithm, when $\mathbf{B}_k = \nabla^2 f(\mathbf{x}_k) = \mathbf{H}(\mathbf{x}_k)$, then we call the general ascent algorithm Newton's method. If we fix $\delta_k = 1$ for $k = 1, 2, \dots,$ then the algorithm is *pure* Newton's method. When δ_k is chosen in each iteration using a line search technique, then the technique is a variable step length Newton's method.

Remark 5.2. The general Newton's Method algorithm (with variable step length) is shown in Algorithm 10.

Example 5.3. Consider the function $F(x, y) = -2x^2 - 10y^4$. This is a concave function as illustrated by its Hessian matrix:

$$\mathbf{H}(x, y) = \begin{bmatrix} -4 & 0 \\ 0 & -120y^2 \end{bmatrix}$$

This matrix is negative definite except when $y = 0$, at which point it is negative semi-definite. Clearly, $x = y = 0$ is the global maximizer for the function. If we begin execution of Newtons Method at $x = 15$, $y = 5$, Newton's method converges in 11 steps. As illustrated in Figure 5.1

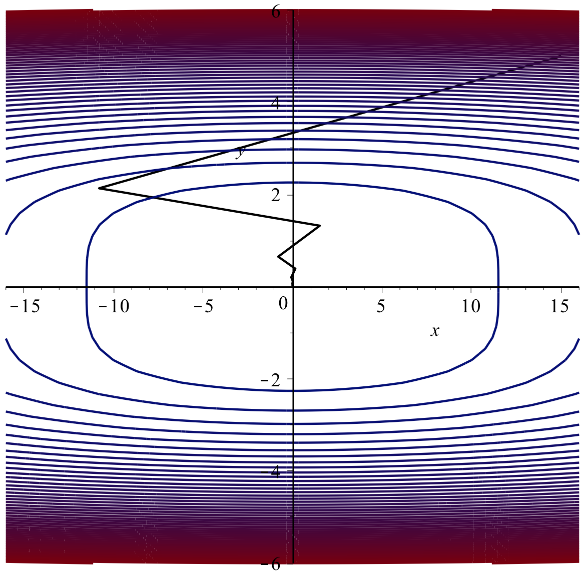


Figure 5.1. Newton's Method converges for the function $F(x, y) = -2x^2 - 10y^4$ in 11 steps, with minimal zigzagging.

```
NewtonsMethod := proc (F::operator, initVals::list, epsilon::numeric)::list;
  local vars, xnow, i, G, normG, OUT, passIn, phi, LS, vals, ttemp, H;
  vars := []; xnow := []; vals := initVals;

  #The first few lines are housekeeping, so we can pass variables around.
  for i to nops(initVals) do
    vars := [op(vars), lhs(initVals[i])];
    xnow := [op(xnow), rhs(initVals[i])];
  end do;

  #Compute the gradient and hessian using the VectorCalculus package.
  #Store the Gradient as a Vector (that's not the default).
  G := Vector(Gradient(F(op(vars)), vars));
  H := Hessian(F(op(vars)), vars);

  #Compute the current gradient norm.
  normG := Norm(evalf(eval(G, initVals)));

  #Output will be a path to the optimal solution.
  #Your output could just be the optimal solution.
  OUT := [];

  #While we're not at a stationary point...
  while evalb(epsilon < normG) do

    #Update the output.
    OUT := [op(OUT), xnow];

    #Do some housekeeping. This is x_next = x_now - s * H^(-1) * G.
    #Essentially, we're going to be solving for the distance to walk "s" below.
    #Most of this line is just getting Maple to treat things as floating point and
    #returning a list of the inputs we can pass to F.
    passIn := convert(Vector(xnow) -
              s*evalf(eval(H,vals))^(-1).evalf(eval(G,vals)), list);

    #Build a univariate function that we can maximize. This is a little weird.
    #We defined our list (above) in terms of s, and we're going to evaluate
    #s at the value t we pass in.
    phi := proc (t)
      options operator, arrow;
      evalf(eval(F(op(passIn)), s = t))
    end proc;

    #Find the optimal step using a line search.
    ttemp := LineSearch(phi);

    #Update xnow using ttemp. This is our next point.
    xnow := evalf(eval(passIn, [s = ttemp]));

    #Store the information in xnow in the vals list.
    vals := [];
    for i to nops(vars) do
      vals := [op(vals), vars[i] = xnow[i]]
    end do;

    #Compute the new norm of the gradient. Notice, we don't have to recompute the
    #gradient, it's symbolic.
    normG := Norm(evalf(eval(G, vals)))
  end do;

  #Pass the last point in (the while loop doesn't get executed the last time
     around.
  OUT := [op(OUT), xnow];

  #Return the optimal solution.
  OUT:
end proc:
```

Algorithm 10. Variable Step Newton's Method

Exercise 36. Implement Newton's Method and test it on a variation of Rosenbrock's Function:

$$-(1-x)^2 - 100\left(y - x^2\right)^2 \tag{5.1}$$

2. Convergence Issues in Newton's Method

Example 5.4. Consider the function:

$$F(x,y) = \left(x^2 + 3\,y^2\right) \mathrm{e}^{1-x^2-y^2} \tag{5.2}$$

A plot of the function is shown in Figure 5.2. It has two global maxima, a local minima between the maxima and saddle points near the peaks. If we execute Newton's Method starting at $x = 1$ and $y = 0.5$, we obtain an interesting phenomenon. The Hessian matrix at this point is:

$$\mathbf{H}(1, 0.5) = \begin{bmatrix} -1.947001959 & -3.504603525 \\ -3.504603525 & -1.362901370 \end{bmatrix} \tag{5.3}$$

This matrix is not positive definite, as illustrated by the fact that its $(1,1)$ element is negative (and therefore, the Cholesky decomposition algorithm will fail). In fact, this matrix is indefinite. The consequence of this is that the direction $-s\mathbf{H}(1,0.5)\nabla F(1,0.5)$ is not an ascent direction for any positive value of s, and thus the line search algorithm converges to

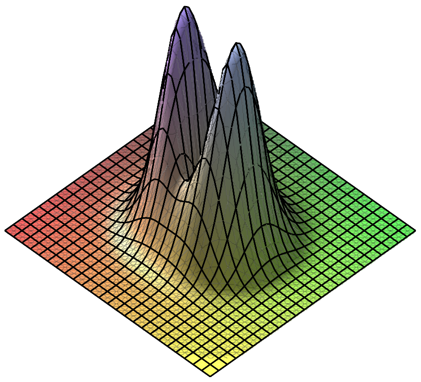


Figure 5.2. A double peaked function with a local minimum between the peaks. This function also has saddle points.

$s = 0$ when maximizing the function:

$$\phi(s) = [1, 0.5]^T - s\mathbf{H}(1, 0.5)\nabla F(1, 0.5)$$

Thus, the algorithm fails to move from the initial point. Unless, one includes a loop counter to Line 24 in Algorithm 10, the algorithm will cycle forever.

Exercise 37. Implement Pure Newton's method and try it on the previous example. (You should not see infinite cycling, but you should not see good convergence.)

Remark 5.5. We are now in a situation were we'd like to know whether Newton's Method converges and if so how fast. Naturally, we've already foreshadowed the results presented below by our discussion of Newton's Method in one dimension (see Theorem 3.53).

Definition 5.6 (Matrix Norm). Let $\mathbf{M} \in \mathbb{R}^{n\times n}$. Then the matrix norm of $\mathbf{M}$ is:

$$||\mathbf{M}|| = \max_{||\mathbf{x}||=1} ||\mathbf{M}\mathbf{x}|| \tag{5.4}$$

Theorem 5.7. *Let $f : \mathbb{R}^n \to \mathbb{R}$ with local maximum at $\mathbf{x}^*$. For $\delta > 0$, let $S_\delta = \{\mathbf{x} : ||\mathbf{x} - \mathbf{x}^*|| \leq \delta\}$ Suppose that $\mathbf{H}(\mathbf{x}^*)$ is invertible and within some sphere S_ϵ, f is twice continuously differentiable. Then:*

(1) *There exists a $\delta > 0$ such that if $\mathbf{x}_0 \in S_\delta$, the sequence $\mathbf{x}_k$ generated by the pure Newton's method:*

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k) \tag{5.5}$$

is defined, belongs to S_δ and converges to $\mathbf{x}^*$. Furthermore, $||\mathbf{x}_k - \mathbf{x}^*||$ converges superlinearly.

(2) *If for some $L > 0$, $M > 0$ and $\delta > 0$ and for all $\mathbf{x}_1, \mathbf{x}_2 \in S_\delta$:*

$$||\mathbf{H}(\mathbf{x}_1) - \mathbf{H}(\mathbf{x}_2)|| \leq L||\mathbf{x}_1 - \mathbf{x}_2|| \quad \text{and} \quad \left|\left|\mathbf{H}^{-1}(\mathbf{x}_1)\right|\right| \leq M \tag{5.6}$$

then, if $\mathbf{x}_0 \in S_\delta$ we have:

$$||\mathbf{x}_{k+1} - \mathbf{x}^*|| \leq \frac{LM}{2}||\mathbf{x}_k - \mathbf{x}^*||^2, \quad \forall k = 0, 1, \dots \tag{5.7}$$

Thus, if $LM\delta/2 < 1$ and $\mathbf{x}_0 \in S_\delta$, $||\mathbf{x}_k - \mathbf{x}^*||$ converges superlinearly with order at least 2.

PROOF. To prove the first statement, choose $\delta > 0$ so that $\mathbf{H}(x)$ exists and is invertible for all $\mathbf{x} \in S_\delta$. Define $M > 0$ so that:

$$\left|\left|\mathbf{H}^{-1}(\mathbf{x})\right|\right| \leq M \quad \forall \mathbf{x} \in S_\delta$$

Let $\mathbf{h} = \mathbf{x}_k - \mathbf{x}^*$. Then from Lemma 2.7 (the Mean Value Theorem for Vector Valued Functions) we have:

$$\nabla f(\mathbf{x}^* + \mathbf{h}) - \nabla f(\mathbf{x}^*) = \int_0^1 \nabla^2 f(\mathbf{x}^* + t\mathbf{h})\mathbf{h}dt \implies \nabla f(\mathbf{x}_k) = \int_0^1 \nabla^2 f(\mathbf{x}^* + t\mathbf{h})\mathbf{h}dt \tag{5.8}$$

We can write:

$$||\mathbf{x}_{k+1} - \mathbf{x}^*|| = ||\mathbf{x}_k - \mathbf{H}^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k) - \mathbf{x}^*|| = ||\mathbf{x}_k - \mathbf{x}^* - \mathbf{H}^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k)|| \tag{5.9}$$

We now factor $\mathbf{H}^{-1}(\mathbf{x}_k)$ out of the above equation to obtain:

$$||\mathbf{x}_{k+1} - \mathbf{x}^*|| = \left|\left|\mathbf{H}^{-1}(\mathbf{x}_k)\left[\mathbf{H}(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}^*) - \nabla f(\mathbf{x}_k)\right]\right|\right| \tag{5.10}$$

Using Equation 5.8, we can write the previous inequality as:

$$\begin{aligned} ||\mathbf{x}_{k+1} - \mathbf{x}^*|| &= \left|\left|\mathbf{H}^{-1}(\mathbf{x}_k)\left[\mathbf{H}(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}^*) - \int_0^1 \nabla^2 f(\mathbf{x}^* + t\mathbf{h})\mathbf{h}dt\right]\right|\right| \\ &= \left|\left|\mathbf{H}^{-1}(\mathbf{x}_k)\left[\mathbf{H}(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}^*) - \left(\int_0^1 \nabla^2 f(\mathbf{x}^* + t\mathbf{h})dt\right)(\mathbf{x}_k - \mathbf{x}^*)\right]\right|\right| \end{aligned} \tag{5.11}$$

because $\mathbf{h} = \mathbf{x}_k - \mathbf{x}^*$ and is not dependent on t. Factoring $(\mathbf{x}_k - \mathbf{x}^*)$ from both terms inside the square brackets on the right hand side yields:

$$\begin{aligned} ||\mathbf{x}_{k+1} - \mathbf{x}^*|| &= \left|\left|\mathbf{H}^{-1}(\mathbf{x}_k)\left[\mathbf{H}(\mathbf{x}_k) - \left(\int_0^1 \nabla^2 f(\mathbf{x}^* + t\mathbf{h})dt\right)\right](\mathbf{x}_k - \mathbf{x}^*)\right|\right| \\ &= \left|\left|\mathbf{H}^{-1}(\mathbf{x}_k)\left[\mathbf{H}(\mathbf{x}_k) - \left(\int_0^1 \mathbf{H}(t\mathbf{x}_k + (1-t)\mathbf{x}^*)dt\right)\right](\mathbf{x}_k - \mathbf{x}^*)\right|\right| \end{aligned} \tag{5.12}$$

Thus we conclude:

$$\begin{aligned} ||\mathbf{x}_{k+1} - \mathbf{x}^*|| &= ||\mathbf{H}^{-1}(\mathbf{x}_k)|| \left|\left|\left[\mathbf{H}(\mathbf{x}_k) - \left(\int_0^1 \mathbf{H}(t\mathbf{x}_k + (1-t)\mathbf{x}^*)dt\right)\right]\right|\right| ||\mathbf{x}_k - \mathbf{x}^*|| \\ = ||\mathbf{x}_{k+1} - \mathbf{x}^*|| &= ||\mathbf{H}^{-1}(\mathbf{x}_k)|| \left|\left|\left(\int_0^1 \mathbf{H}(\mathbf{x}_k) - \mathbf{H}(t\mathbf{x}_k + (1-t)\mathbf{x}^*)dt\right)\right|\right| ||\mathbf{x}_k - \mathbf{x}^*|| \\ &\leq M\left(\int_0^1 ||\mathbf{H}(\mathbf{x}_k) - \mathbf{H}(t\mathbf{x}_k + (1-t)\mathbf{x}^*)||dt\right) ||\mathbf{x}_k - \mathbf{x}^*|| \end{aligned} \tag{5.13}$$

By our assumption on $\mathbf{H}(\mathbf{x})$ for $\mathbf{x} \in S_\delta$. Note, in the penultimate step, we've moved $\mathbf{H}(\mathbf{x}_k)$ under the integral sign since:

$$\int_0^1 \mathbf{H}(\mathbf{x}_k)dt = \mathbf{H}(\mathbf{x}_k)$$

Our assumption on $f(\mathbf{x})$ (namely that it is twice continuously differentiable in S_δ) tells us we can take δ small enough to ensure that the norm of the integral is sufficiently small. This establishes the superlinear convergence of the algorithm inside S_δ.

To prove the second assertion, suppose that $\mathbf{H}(\mathbf{x})$ is Lipschitz inside S_δ. Then Inequality 5.13 becomes:

$$
\begin{aligned}
(5.14)\quad ||\mathbf{x}_{k+1} - \mathbf{x}^*|| &\leq M \left(\int_0^1 L||\mathbf{x}_k - (t\mathbf{x}_k + (1-t)\mathbf{x}^*)||dt \right) ||\mathbf{x}_k - \mathbf{x}^*|| \\
&= M \left(\int_0^1 L(1-t)||\mathbf{x}_k - \mathbf{x}^*)||dt \right) ||\mathbf{x}_k - \mathbf{x}^*|| = \frac{LM}{2}||\mathbf{x}_k - \mathbf{x}^*||^2
\end{aligned}
$$

This completes the proof. □

Remark 5.8. Notice that at no time did we state that Newton's Method would be attracted to a local maximum. In fact, Newton's method tends to be attracted to stationary points of any kind.

Remark 5.9. What we've shown is that, when Newton's method gets close to a solution, it works extremely well. Unfortunately, there's no way to ensure that Newton's method – in its raw form – will get to a point where it works well. For this reason, a number of corrections have been devised.

3. Newton Method Corrections

Remark 5.10. The original "Newton's Method" correction was a simple one developed by Cauchy (and was the motivation for developing the method of steepest ascent). In essence, one executes a gradient ascent algorithm until the Hessian matrix becomes negative definite. At this point, we are close to a maximum and we use Newton's method.

Example 5.11. Consider, again the function:

$$F(x, y) = \left(x^2 + 3\,y^2\right) \mathrm{e}^{1-x^2-y^2}$$

Suppose instead of using variable step Newton's method algorithm, we take gradient ascent steps until $\mathbf{H}(\mathbf{x})$ becomes negative definite, at which point we switch to variable length Newton steps. This can be accomplished by adding a test on $\mathbf{H}$ before line 29 of Algorithm 10. When we do this and start at $x_0 = 1$, $y_0 = 0.1$, we obtain convergence to $x^* = 1$, $y^* = 1$ in five steps, as illustrated in Figure 5.3.

Exercise 38. Implement the modification to Newton's method discussed above. And test the the rate of convergence on Rosenbrock's function, compared to pure gradient ascent when starting from $x_0 = 0$ and $y_0 = 0$. Prove that the resulting sequence $\{\mathbf{x}_k\}$ generated by this algorithm is gradient related and therefore the algorithm converges to a stationary point.

Exercise 39. Prove that Cauchy's modification to Newton's algorithm converges superlinearly.

Remark 5.12. The most common correction to Newton's Method is the *Modified Cholesky Decomposition* approach. In this approach, we will be passing $-\mathbf{H}(\mathbf{x})$ into a modification of the Cholesky decomposition. The goal is to correct the Hessian matrix so that it is negative definite (and thus its negative is positive definite). In this way, we will always take an ascent step. The problem is, we'd like to maintain as much of the Hessian structure as possible and also ensure that when the Hessian is naturally negative definite, we do not change it at all. A simple way to do this is to choose $\mu_1, \mu_2 \in \mathbb{R}_+$ with $\mu_1 < \mu_2$. In the Cholesky

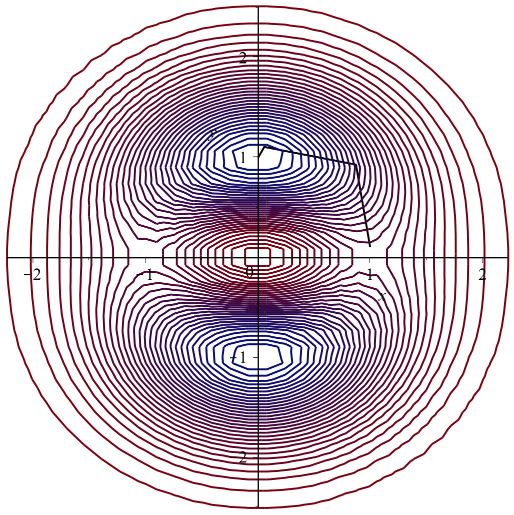


Figure 5.3. A simple modification to Newton's method first used by Gauss. While $\mathbf{H}(\mathbf{x}_k)$ is not negative definite, we use a gradient ascent to converge to the neighborhood of a stationary point (ideally a local maximum). We then switch to a Newton step.

decomposition at Line 12, we check to see if $\mathbf{H}_{ii} > \mu_1$. If not, we replace $\mathbf{H}_{ii}$ with μ_2 and continue. In this way, the modified Cholesky decomposition will always return a factorization for a positive definite matrix that has some resemblance to the Hessian matrix. A reference implementation is shown in Algorithm 11.

Remark 5.13. Suppose that we have constructed $\mathbf{L} \in \mathbb{R}^{n\times n}$ from the negative of the Hessian matrix for the function $\mathbf{f}(\mathbf{x}_k)$. That is:

$$-\mathbf{H}(\mathbf{x}_k) \sim \mathbf{L}\mathbf{L}^T \tag{5.15}$$

where $\sim$ indicates that $-\mathbf{H}(\mathbf{x}_k)$ may be positive definite, or may just be loosely related to $\mathbf{L}\mathbf{L}^T$, depending on the steps executed in the modified Cholesky decomposition. We must solve the problem:

$$\left[\mathbf{L}\mathbf{L}^T\right]^{-1} \nabla f(\mathbf{x}_k) = \mathbf{p} \tag{5.16}$$

where $\mathbf{p}$ is our direction. This can be accomplished efficiently by solving:

$$\nabla f(\mathbf{x}_k) = \mathbf{L}\mathbf{z} \tag{5.17}$$

by forward substitution, and then solving:

$$\mathbf{L}^T\mathbf{p} = \mathbf{z} \tag{5.18}$$

by backwards substitution. Since $\mathbf{L}$ is a lower triangular matrix and $\mathbf{L}^T$ is an upper triangular matrix.

Example 5.14. Consider, again the function:

$$F(x,y) = \left(x^2 + 3\,y^2\right) \mathrm{e}^{1-x^2-y^2}$$

```
ModifiedCholeskyDecomp := proc (M::Matrix, n::integer,
  mu1::numeric, mu2::numeric)::list;
  local MM, H, ret, i, j, k;
  MM := Matrix(M);
  H := Matrix(n, n);
  ret := true;
  for i to n do
    if mu1 <= MM[i, i] then
      H[i, i] := sqrt(MM[i, i])
    else
      H[i, i] := mu2
    end if;

    for j from i+1 to n do
      H[j, i] := MM[j, i]/H[i, i];

      for k from i+1 to j do
        MM[j, k] := MM[j, k]-H[j, i]*H[k, i]
      end do:
    end do:
  end do:
  if ret then [H, ret]
    else [M, ret]
  end if:
end proc:
```

Algorithm 11. The modified Cholesky algorithm always returns a factored matrix close to an input matrix, but that is positive definite.

and suppose we are starting at position $x = 1$ and $y = 1/2$. Then the Hessian matrix is:

$$\mathbf{H} = \begin{bmatrix} -5/2\,e^{-1/4} & -9/2\,e^{-1/4} \\ -9/2\,e^{-1/4} & -7/4\,e^{-1/4} \end{bmatrix}$$

As we noted, this matrix is not positive definite. When we perform the modified Cholesky decomposition, we obtain the matrix:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 \\ -9/2\,e^{-1/4} & 1 \end{bmatrix}$$

The gradient at this point is:

$$\nabla f\left(1, \tfrac{1}{2}\right) = \begin{bmatrix} -3/2\,e^{-1/4} \\ 5/4\,e^{-1/4} \end{bmatrix}$$

This leads to the first problem:

$$\begin{bmatrix} 1 & 0 \\ -9/2\,e^{-1/4} & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} -3/2\,e^{-1/4} \\ 5/4\,e^{-1/4} \end{bmatrix} \tag{5.19}$$

Thus we see immediately that:

$$z_1 = -3/2\,\mathrm{e}^{-1/4}$$
$$z_2 = 5/4\,\mathrm{e}^{-1/4} - \left(9/2\,\mathrm{e}^{-1/4}\right)\left(3/2\,\mathrm{e}^{-1/4}\right) = -\frac{27}{4}\left(\mathrm{e}^{-1/4}\right)^2 + 5/4\,\mathrm{e}^{-1/4}$$

We can now solve the problem:

$$\begin{bmatrix} 1 & -9/2\,\mathrm{e}^{-1/4} \\ 0 & 1 \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} -3/2\,\mathrm{e}^{-1/4} \\ -\frac{27}{4}\left(\mathrm{e}^{-1/4}\right)^2 + 5/4\,\mathrm{e}^{-1/4} \end{bmatrix} \tag{5.20}$$

Thus:

$$p_2 = -\frac{27}{4}\left(\mathrm{e}^{-1/4}\right)^2 + 5/4\,\mathrm{e}^{-1/4}$$
$$p_1 = -3/2\,\mathrm{e}^{-1/4} + \left(9/2\,\mathrm{e}^{-1/4}\right)p_2 = -\frac{243}{8}\left(\mathrm{e}^{-1/4}\right)^3 + \frac{45}{8}\left(\mathrm{e}^{-1/4}\right)^2 - 3/2\,\mathrm{e}^{-1/4}$$

If we compute $\nabla f\left(1, \frac{1}{2}\right)^T \mathbf{p}$ we see:

$$\nabla f\left(1, \tfrac{1}{2}\right)^T \mathbf{p} = -3/2\,\mathrm{e}^{-1/4}\left(-\frac{243}{8}\left(\mathrm{e}^{-1/4}\right)^3 + \frac{45}{8}\left(\mathrm{e}^{-1/4}\right)^2 - 3/2\,\mathrm{e}^{-1/4}\right) + 5/4\,\mathrm{e}^{-1/4}\left(-\frac{27}{4}\left(\mathrm{e}^{-1/4}\right)^2 + 5/4\,\mathrm{e}^{-1/4}\right) \sim 11.103 > 0 \tag{5.21}$$

Thus $\mathbf{p}$ is an ascent direction.

REMARK 5.15. A reference implementation of the modified Cholesky correction to Newton's Method is shown in Algorithm 12.

THEOREM 5.16. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable. Suppose $0 < \mu_1 < \mu_2$. Let $\{\mathbf{x}_k\}$ be the sequence of points generated by the modified Newton's method. Then this sequence converges to a stationary point $\mathbf{x}^*$ (i.e., $\nabla f(\mathbf{x}^*)$).*

PROOF. By construction $\mathbf{p}_k = \mathbf{B}_k^{-1}\nabla f(\mathbf{x}_k)$ at each stage, where $\mathbf{B}_k^{-1}$ is positive definite and non-singular for each k. Thus, $\nabla f(\mathbf{x}_k)^T\mathbf{p}_k > 0$ just in case, $\nabla f(\mathbf{x}_k) \neq \mathbf{0}$. Therefore $\{\mathbf{p}_k\}$ is gradient related and the necessary conditions of Theorem 4.12 are satisfied and $\{\mathbf{x}_k\}$ converges to a stationary point. □

THEOREM 5.17. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable. Suppose that $\mathbf{x}^*$ is the stationary point to which the sequence $\{\mathbf{x}_k\}$ converges, when $\{\mathbf{x}_k\}$ is the sequence of points generated by the modified Newton's method. If $\mathbf{H}(\mathbf{x}^*)$ is negative definite then there is a $\mu_1 > 0$ so that the algorithm converges superlinearly when using the Armijo rule with $\sigma_1 < \frac{1}{2}$ and $t_0 = 1$ in the backtrace algorithm.*

PROOF. Let $0 < \mu_1$ be any value that is less than the minimal diagonal value M_{ii} at Line 8 of Algorithm 11 when executed on $\mathbf{H}(\mathbf{x}^*)$. By the continuity of f, there is some neighborhood S of $\mathbf{x}^*$ so that the modified Cholesky decomposition of $\mathbf{H}(\mathbf{x})$ is identical to the Cholesky decomposition of $\mathbf{H}(\mathbf{x})$ for all $\mathbf{x} \in S$. Then, for all $\mathbf{x}_k$ in S, the modified Newton's algorithm becomes Newton's method and thus:

$$\lim_{k\to\infty} \frac{||\mathbf{p}_k + \mathbf{H}^{-1}(\mathbf{x}^*)\nabla f(\mathbf{x}_k)||}{||\nabla f(\mathbf{x}_k)||} = 0$$

```
ModifiedNewtonsMethod := proc (F::operator, initVals::list, epsilon::numeric,
               maxIter::integer)::list;
  local vars, xnow, i, G, normG, OUT, passIn, phi, LS, vals, ttemp, H;
  vars := []; xnow := []; vals := initVals;

  #The first few lines are housekeeping, so we can pass variables around.
  for i to nops(initVals) do
    vars := [op(vars), lhs(initVals[i])];
    xnow := [op(xnow), rhs(initVals[i])];
  end do;

  #Compute the gradient and hessian using the VectorCalculus package.
  #Store the Gradient as a Vector (that's not the default).
  G := Vector(Gradient(F(op(vars)), vars));
  H := Hessian(F(op(vars)), vars);

  #Compute the current gradient norm.
  normG := Norm(evalf(eval(G, initVals)));

  #Output will be a path to the optimal solution.
  #Your output could just be the optimal solution.
  OUT := [];

  #While we're not at a stationary point...
  while evalb(epsilon < normG and count < maxIter) do

    #Update the output.
    OUT := [op(OUT), xnow];

    #Update count
    count := count + 1;

    #Compute the modified Cholesky decomposition for the Hessian.
    L := ModifiedCholeskyDecomp(evalf(eval(-H, vals)), nops(vals), .1, 1)[1];

    #Now solve for the ascent direction P.
    X := LinearSolve(L, evalf(eval(G, vals)),method='subs');
    P := LinearSolve(Transpose(L), X,method='subs')

    #Do some housekeeping. This is x_next = x_now + s * P.
    #Essentially, we're going to be solving for the distance to walk "s" below.
    passIn := convert(Vector(xnow)+s*P, list)
    #Build a univariate function that we can maximize. This is a little weird.
    #We defined our list (above) in terms of s, and we're going to evaluate
    #s at the value t we pass in.
    phi := proc (t)
      options operator, arrow;
      evalf(eval(F(op(passIn)), s = t))
    end proc;

    #Find the optimal step using a line search.
    ttemp := LineSearch(phi);

    #Update xnow using ttemp. This is our next point.
    xnow := evalf(eval(passIn, [s = ttemp]));

    #Store the information in xnow in the vals list.
    vals := [];
    for i to nops(vars) do
      vals := [op(vals), vars[i] = xnow[i]]
    end do;

    #Compute the new norm of the gradient. Notice, we don't have to recompute the
    #gradient, it's symbolic.
    normG := Norm(evalf(eval(G, vals)))
  end do;

  #Pass the last point in (the while loop doesn't get executed the last time
      around.
  OUT := [op(OUT), xnow];

  #Return the optimal solution.
  OUT:
end proc:
```

Algorithm 12. Variable Step Newton's Method using the Modified Cholesky Decomposition

In this case, the necessary conditions of Theorem 4.25 are satisfied and we see that:

$$\lim_{k\to\infty} \frac{||\mathbf{x}_{k+1} - \mathbf{x}^*||}{||\mathbf{x}_k - \mathbf{x}^*||} = 0$$

This completes the proof. □

Remark 5.18. The previous theorem can also be modified so that $\mu_1 = c||\nabla f(\mathbf{x}_k)||$, where is a fixed constant with $c > 0$.

Exercise 40. Prove the superlinear convergence of the modified Newton's method when $\mu_1 = c||\nabla f(\mathbf{x}_k)||$ and $c > 0$.

Remark 5.19. There is no *scientific* method for choosing μ_1 and μ_2 in the modified Newton's method. Various authors suggest different approaches. Most recommend beginning with a very small μ_1. If μ_2 is too small, the matrix $\mathbf{LL}^T$ may be near singular. On the other hand, if μ_2 is too large, then convergence will be slow, since the matrix will be diagonally dominated. Generally, it is up to the user to find adequate values of μ_1 and μ_2, though [**Ber99**] suggests varying the size of μ_1 and μ_2 based on the largest value of the diagonal of the computed Hessian matrix.

Example 5.20. We illustrate the convergence of the modified Newton's method for the function:

$$F(x,y) = \left(x^2 + 3\,y^2\right) \mathrm{e}^{1-x^2-y^2}$$

starting at position $x = 1$ and $y = 1/2$. We set $\mu_1 = 0.1$ and $\mu_2 = 1$. Algorithmic steps are shown in Figure 5.4. Notice in this case, we obtain convergence to a global maximum in 5 iterations of the algorithm.

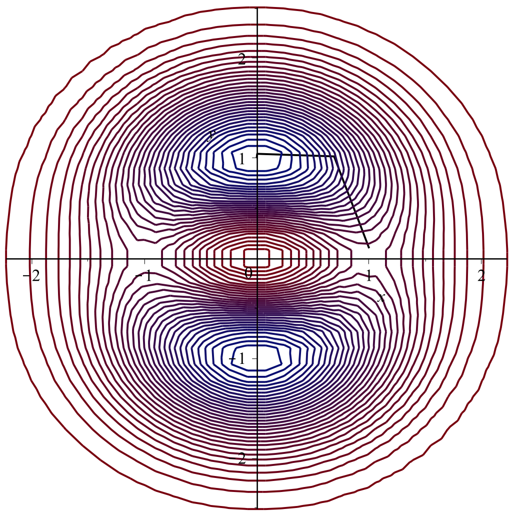


Figure 5.4. Modified Newton's method uses the modified Cholesky decomposition and efficient linear solution methods to find an ascent direction in the case when the Hessian matrix is not negative definite. This algorithm converges superlinearly, as illustrated in this case.

CHAPTER 6

Conjugate Direction Methods

Remark 6.1. We begin this chapter with the study of conjugate gradient methods, which have the property that they converge to an optimal solution for a concave quadratic objective function $f : \mathbb{R}^n \to \mathbb{R}$ in n iterations, as we will see. These methods can be applied to non-quadratic functions (with some efficiency loss), but they are remarkably good for problems of very large dimensionality.

1. Conjugate Directions

Definition 6.2 (Conjugate Directions). Let $\mathbf{Q} \in \mathbb{R}^{n\times n}$ be a negative (or positive) definite matrix. Directions $\mathbf{p}_0, \dots \mathbf{p}_{n-1} \in \mathbb{R}^n$ are $\mathbf{Q}$ conjugate if for all $i \neq j$ we have:

$$\mathbf{p}_i^T\mathbf{Q}\mathbf{p}_j = 0 \tag{6.1}$$

Remark 6.3. For the remainder of this section when we say $\mathbf{Q}$ is *definite* we mean it is either positive or negative definite and our results will apply in each case.

Lemma 6.4. *If $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is definite and $\mathbf{p}_0, \dots \mathbf{p}_{n-1} \in \mathbb{R}^n$ are $\mathbf{Q}$ conjugate, then $\mathbf{p}_0, \dots \mathbf{p}_{n-1}$ are linearly independent.*

Proof. By way of contradiction, suppose that:

$$\mathbf{p}_{n-1} = \alpha_0\mathbf{p}_0 + \cdots + \alpha_{n-2}\mathbf{p}_{n-2} \tag{6.2}$$

Multiplying on the left by $\mathbf{p}_{n-1}^T\mathbf{Q}$ yields:

$$\mathbf{p}_{n-1}^T\mathbf{Q}\mathbf{p}_{n-1} = \alpha_0\mathbf{p}_{n-1}^T\mathbf{Q}\mathbf{p}_0 + \cdots + \alpha_{n-2}\mathbf{p}_{n-1}^T\mathbf{Q}\mathbf{p}_{n-2} = 0 \tag{6.3}$$

by the definition of $\mathbf{Q}$ conjugacy. However, since $\mathbf{Q}$ is definite, we know that $\mathbf{p}_{n-1}^T\mathbf{Q}\mathbf{p}_{n-1} \neq 0$. □

Definition 6.5 (Conjugate Direction Method). Suppose $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is a symmetric negative definite matrix with conjugate directions $\mathbf{p}_0, \dots \mathbf{p}_{n-1} \in \mathbb{R}^n$ and that $f : \mathbb{R}^n \to \mathbb{R}$ is give by:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{b}^T\mathbf{x} \tag{6.4}$$

with $\mathbf{b} \in \mathbf{R}^n$. If $\mathbf{x}_0 \in \mathbb{R}^n$, then *the sequence generated by the conjugate direction method*, $\mathbf{x}_k$ $(k = 0, \dots, n-1)$, is given by:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k\mathbf{p}_k \tag{6.5}$$

where δ_k solves the problem:

$$\max_{\delta_k} f(\mathbf{x}_k + \delta_k\mathbf{p}_k) \tag{6.6}$$

That is, $\delta_k = \arg\max_{\delta_k}\ f(\mathbf{x}_k + \delta_k\mathbf{p}_k)$

Lemma 6.6. *For any k, in the conjugate direction method, δ_k is given by:*

$$\delta_k = -\frac{\mathbf{p}_k^T\left(\mathbf{Q}\mathbf{x}_k + \mathbf{b}\right)}{\mathbf{p}_k^T\mathbf{Q}\mathbf{p}_k} = -\frac{\mathbf{p}_k^T\nabla f(\mathbf{x}_k)}{\mathbf{p}_k^T\mathbf{Q}\mathbf{p}_k} \tag{6.7}$$

Exercise 41. Prove Lemma 6.6. Argue that when $\mathbf{p}_k$ is an ascent direction, $\delta_k > 0$.

Theorem 6.7. *Let $\mathcal{M}^k \subseteq \mathbb{R}^n$ be the subspace spanned by $\mathbf{p}_0, \dots, \mathbf{p}_{k-1}$. Then $f(\mathbf{x})$ is maximized on $\mathbf{M}^k$ by the conjugate direction method after k iterations and therefore after n iterations, the conjugate direction method converges to $\mathbf{x}_n = \mathbf{x}^*$, the global maximum of $f(\mathbf{x})$.*

Proof. Since $\mathbf{p}_0, \dots, \mathbf{p}_{k-1}$ are linearly independent and:

$$\mathbf{x}_k = \mathbf{x}_0 + \delta_0\mathbf{p}_0 + \cdots + \delta_{k-1}\mathbf{p}_{k-1} \tag{6.8}$$

it suffices to show that[1]:

$$\frac{\partial f(\mathbf{x}_k)}{\partial \delta_i} = 0 \quad \forall i = 0, \dots, k-1 \tag{6.9}$$

Observe first that:

$$\frac{\partial f(\mathbf{x}_k)}{\partial \delta_{k-1}} = 0 \tag{6.10}$$

as a necessary condition for the conjugate direction step. That is, since $\delta_{k-1} = \arg\max_{\delta_{k-1}} f(\mathbf{x}_{k-1} + \delta_{k-1}\mathbf{p}_{k-1})$, Equation 6.10 follows at once. We also note for $i = 0, \dots, k-1$:

$$\frac{\partial f(\mathbf{x}_k)}{\partial \delta_i} = \nabla f(\mathbf{x}_k)^T\mathbf{p}_i \tag{6.11}$$

since $\partial f(\mathbf{x}_k)/\partial\delta_i$ is a directional derivative. Thus we already know that:

$$\nabla f(\mathbf{x}_k)^T\mathbf{p}_{k-1} = 0 \tag{6.12}$$

Now, observe that:

$$\mathbf{x}_k = \mathbf{x}_i + \delta_i\mathbf{p}_i + \delta_{i+1}\mathbf{p}_{i+1} + \cdots + \delta_{k-1}\mathbf{p}_{k-1} = \mathbf{x}_i + \sum_{j=i}^{k-1}\delta_j\mathbf{p}_j \tag{6.13}$$

For $i = 0, \dots, k-2$

$$\frac{\partial f(\mathbf{x}_k)}{\partial \delta_i} = \nabla f(\mathbf{x}_k)^T\mathbf{p}_i = \left(\mathbf{Q}\mathbf{x}_k + \mathbf{b}\right)^T\mathbf{p}_i = \tag{6.14}$$

$$\left(\mathbf{Q}\mathbf{x}_k\right)^T\mathbf{p}_i + \mathbf{b}^T\mathbf{p}_i = \mathbf{x}_k^T\mathbf{Q}\mathbf{p}_i + \mathbf{b}^T\mathbf{p}_i = \left(\mathbf{x}_i + \sum_{j=i}^{k-1}\delta_j\mathbf{p}_j\right)\mathbf{Q}\mathbf{p}_i + \mathbf{b}^T\mathbf{p}_i$$

Since $\mathbf{p}_j^T\mathbf{Q}\mathbf{p}_i = 0$ if $i \neq j$, the previous equation becomes:

$$\frac{\partial f(\mathbf{x}_k)}{\partial \delta_i} = \left(\mathbf{x}_i + \delta_i\mathbf{p}_i\right)\mathbf{Q} + \mathbf{b}^T\mathbf{p}_i = \left(\mathbf{x}_{i+1}^T\mathbf{Q} + \mathbf{b}^T\right)\mathbf{p}_i \tag{6.15}$$

[1]There is something a little subtle going on here. We are actually arguing that a strictly concave function achieves a unique global maximum on a constraining set $\mathcal{M}^k$. We have not shown that the sufficient condition given below is a sufficient condition outside of $\mathbb{R}^n$. This can be corrected, but for now, we assume it is clear to the reader.

since $\mathbf{x}_{i+1} = \mathbf{x}_i + \delta_i \mathbf{p}_i$. But:

$$\nabla f(\mathbf{x}_{i+1})^T = \mathbf{x}_{i+1}^T \mathbf{Q} + \mathbf{b}^T \tag{6.16}$$

Thus, we have proved that:

$$\frac{\partial f(\mathbf{x}_k)}{\partial \delta_i} = \nabla f(\mathbf{x}_{i+1})^T \mathbf{p}_i = 0 \tag{6.17}$$

when we apply Equation 6.12 to i (instead of k). Thus we have shown Equation 6.9 holds for all k and it follows that $f(\mathbf{x})$ is maximized on $\mathbf{M}^k$ by the conjugate direction method after k iterations and therefore after n iterations, the conjugate direction method converges to $\mathbf{x}_n = \mathbf{x}^*$, the global maximum of $f(\mathbf{x})$ since $\mathbf{p}_0, \dots, \mathbf{p}_{n-1}$ must be a basis for $\mathbb{R}^n$. □

Example 6.8. Consider the case when:

$$\mathbf{Q} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

and suppose we are given the Q conjugate directions:

$$\mathbf{p}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \mathbf{p}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

and begin at $x_1 = x_2 = 1$. To compute δ_0, we use Equation 6.7:

$$\delta_0 = -\frac{\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}}{\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}} = -\frac{-1}{-1} = -1 \tag{6.18}$$

Notice that $\delta_0 < 0$ because $\mathbf{p}_0$ is not an ascent direction. Thus:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + (-1) \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{6.19}$$

Repeating this logic with $\mathbf{p}_1$, we see that $\delta_1 = -1$ as well and:

$$\mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} + (-1) \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{6.20}$$

which is clearly the global maximum of the function.

Remark 6.9. All conjugate direction methods can be thought of as scaled versions of the previous example. Essentially, when transformed into an appropriate basis, we are simply minimizing along the various basis elements. The challenge is to determine a basis that is Q conjugate.

2. Generating Conjugate Directions

Remark 6.10. The principle problem we have yet to overcome is the generation of Q conjugate directions. We overcome this problem by applying the Graham-Schmidt procedure, which we define in the following theorem.

THEOREM 6.11. *Suppose that $\mathbf{d}_0, \dots, \mathbf{d}_{n-1} \in \mathbb{R}^n$ are arbitrarily chosen linearly independent directions (that span $\mathbb{R}^n$). Then the directions:*

$$\mathbf{p}_0 = \mathbf{d}_0 \tag{6.21}$$

$$\mathbf{p}_{i+1} = \mathbf{d}_{i+1} + \sum_{j=0}^{i} c_j^{(i+1)} \mathbf{p}_j \tag{6.22}$$

where:

$$c_m^{(i+1)} = \frac{-\mathbf{d}_{i+1}^T \mathbf{Q} \mathbf{p}_m}{\mathbf{p}_m^T \mathbf{Q} \mathbf{p}_m} \quad m = 0, \dots, i, \forall i = 0, \dots, n-2 \tag{6.23}$$

are $\mathbf{Q}$ conjugate.

PROOF. Assume that Equations 6.21 and 6.22 hold. At construction stage $i+1$, to ensure $\mathbf{Q}$ conjugacy we require: $\mathbf{p}_{i+1}^T \mathbf{Q} \mathbf{p}_m = 0$ for $m = 0, \dots, i$. Using Equation 6.22 yields:

$$\mathbf{p}_{i+1}^T \mathbf{Q} \mathbf{p}_m = \left(\mathbf{d}_{i+1} + \sum_{j=0}^{i} c_j^{(i+1)} \mathbf{p}_j \right)^T \mathbf{Q} \mathbf{p}_m = \mathbf{d}_{i+1} \mathbf{Q} \mathbf{p}_m + \left(\sum_{j=0}^{i} c_j^{(i+1)} \mathbf{p}_j \right)^T \mathbf{Q} \mathbf{p}_m \tag{6.24}$$

If we apply induction, we know that only when $j = m$ is $c_j^{(i+1)} \mathbf{p}_j^T \mathbf{Q} \mathbf{p}_m \neq 0$. Thus, if $\mathbf{p}_{i+1}^T \mathbf{Q} \mathbf{p}_m = 0$, then:

$$\mathbf{d}_{i+1} \mathbf{Q} \mathbf{p}_m + c_m^{(i+1)} \mathbf{p}_m^T \mathbf{Q} \mathbf{p}_m = 0 \quad m = 0, \dots, i \tag{6.25}$$

That is, we have $i+1$ equations with $i+1$ unknowns and one unknown in each equation. Solving Equation 6.25 yields:

$$c_m^{(i+1)} = \frac{-\mathbf{d}_{i+1}^T \mathbf{Q} \mathbf{p}_m}{\mathbf{p}_m^T \mathbf{Q} \mathbf{p}_m} \quad m = 0, \dots, i, \forall i = 0, \dots, n-2 \tag{6.26}$$

This completes the proof. □

COROLLARY 6.12. *The space spanned by $\mathbf{d}_0, \dots, \mathbf{d}_k$ is identical to the space spanned by $\mathbf{p}_0, \dots, \mathbf{p}_k$.*

EXERCISE 42. Prove Corollary 6.12. [Hint: Note that Equations 6.21 and 6.22 show that $\mathbf{d}_{i+1}$ is a linear combination of $\mathbf{p}_{i+1}, \dots, \mathbf{p}_0$.]

REMARK 6.13. The approach described in the preceding theorem is known as the Graham-Schmidt method.

EXERCISE 43. In Example 6.8, show that the directions generated by the Graham-Schmidt method are identical to the provided directions.

3. The Conjugate Gradient Method

REMARK 6.14. One major problem to overcome with a conjugate direction method is obtaining a set of $\mathbf{Q}$ conjugate directions. This can be accomplished by using the gradients of $f(\mathbf{x})$ at $\mathbf{x}_k$. Naturally, we must show that these vectors are linearly independent (and thus will form a basis of $\mathbb{R}^n$ after n iterations).

LEMMA 6.15. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be defined as in Definition 6.5 and let $\mathbf{x}_k$ be the sequence generated by the conjugate direction method using the Graham-Schmidt generated directions with $\mathbf{d}_k = \mathbf{g}_k = \nabla f(\mathbf{x}_k)$. Then $\mathbf{g}_k$ is orthogonal to $\mathbf{p}_0, \dots, \mathbf{p}_{k-1}$ (and by extension $\mathbf{g}_0, \dots, \mathbf{g}_{k-1}$).*

PROOF. We proceed by induction on k. When $k = 0$, the set of vectors is just $\mathbf{g}_0$ and this is clearly a linearly independent set and the space spanned by $\mathbf{g}_0$ is identical to the space spanned by $\mathbf{d}_0$ from Corollary 6.12. Assume the statement is true up to stage $k-1$. To see it is true for stage k, consider the following cases:

(1) Suppose $\mathbf{g}_k = \mathbf{0}$. Then $\mathbf{x}_k = \mathbf{x}_{k-1}$ and we must be at a global maximum, and the theorem is true by assumption.
(2) Suppose $\mathbf{g}_k \neq \mathbf{0}$. Then from Equations 6.10 and 6.11 and , we know that:

$$\nabla f(\mathbf{x}_k)^T \mathbf{p}_i = 0$$

for $i = 0, \dots, k-1$.

Thus, $\mathbf{g}_k$ is orthogonal to $\mathbf{p}_0, \dots, \mathbf{p}_{k-1}$ and since (by Corollary 6.12) the space spanned by $\mathbf{p}_0, \dots, \mathbf{p}_{k-1}$ is identical to the space spanned by $\mathbf{g}_0, \dots, \mathbf{g}_{k-1}$, we see that $\mathbf{g}_k$ must be orthogonal to $\mathbf{g}_0, \dots, \mathbf{g}_{k-1}$. □

THEOREM 6.16. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be as in Definition 6.5 and let $\mathbf{x}_k$ be the sequence generated by the conjugate direction method using the Graham-Schmidt generated directions with $\mathbf{d}_k = \mathbf{g}_k = \nabla f(\mathbf{x}_k)$. If $\mathbf{x}^*$ is the global maximum for $f(\mathbf{x})$, then the sequence converges to $\mathbf{x}^*$ after at most n steps and furthermore, we can write:*

$$\mathbf{p}_k = \mathbf{g}_k + \beta_k \mathbf{p}_{k-1} \tag{6.27}$$

where:

$$\beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}} \tag{6.28}$$

PROOF. From Equations 6.21 and 6.22, we know that:

$$\mathbf{p}_0 = \mathbf{g}_0 \tag{6.29}$$

$$\mathbf{p}_{k+1} = \mathbf{g}_{k+1} + \sum_{j=0}^{k} c_j^{(k+1)} \mathbf{p}_j \tag{6.30}$$

where:

$$c_m^{(k+1)} = \frac{-\mathbf{g}_{k+1}^T \mathbf{Q} \mathbf{p}_m}{\mathbf{p}_m^T \mathbf{Q} \mathbf{p}_m} \quad m = 0, \dots, k, \forall k = 0, \dots, n-2 \tag{6.31}$$

Note that:

$$\mathbf{g}_{m+1} - \mathbf{g}_m = \mathbf{Q}(\mathbf{x}_{m+1} - \mathbf{x}_m) = \alpha_m \mathbf{Q} \mathbf{p}_m \tag{6.32}$$

and $\alpha_m \neq 0$ (for otherwise $\mathbf{x}_{m+1} = \mathbf{x}_m$ and the algorithm will have converged). This is because $\mathbf{x}_{m+1} = \mathbf{x}_m + \alpha_m \mathbf{p}_m$, where α_m is the optimal δ_m found by line search. Left multiplying by $-\mathbf{g}_{k+1}^T$ yields:

$$-\alpha_k \mathbf{g}_{k+1}^T \mathbf{Q} \mathbf{p}_m = -\mathbf{g}_{k+1}^T (\mathbf{g}_{m+1} - \mathbf{g}_m) = \begin{cases} -\mathbf{g}_{k+1}^T \mathbf{g}_{k+1} & m = k \\ 0 & \text{otherwise} \end{cases} \tag{6.33}$$

by Lemma 6.15. Thus we have:

$$c_m^{(k+1)} = \frac{-\mathbf{g}_{k+1}^T\mathbf{Q}\mathbf{p}_m}{\mathbf{p}_m^T\mathbf{Q}\mathbf{p}_m} = \begin{cases} -\frac{1}{\alpha_k}\frac{\mathbf{g}_{k+1}^T\mathbf{g}_{k+1}}{\mathbf{p}_k^T\mathbf{Q}\mathbf{p}_k} & m = k \\ 0 & \text{otherwise}\end{cases} \tag{6.34}$$

By a similar argument, we note that:

$$\mathbf{p}_k^T\mathbf{Q}\mathbf{p}_k = \frac{1}{\alpha_k}\mathbf{p}_k^T\left(\mathbf{g}_{k+1} - \mathbf{g}_k\right) \tag{6.35}$$

Then substituting this into the previous Equation 6.35 into Equation 6.34 yields:

$$\beta_{k+1} = c_k^{(k+1)} = \frac{-\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{p}_k^T\left(\mathbf{g}_{k+1} - \mathbf{g}_k\right)} \tag{6.36}$$

Substituting this into the Graham-Schmidt procedure yields:

$$\mathbf{p}_{k+1} = \mathbf{g}_{k+1} - \frac{\mathbf{g}_{k+1}^T\mathbf{g}_{k+1}}{\mathbf{p}_k^T\left(\mathbf{g}_{k+1} - \mathbf{g}_k\right)}\mathbf{p}_k \tag{6.37}$$

Written for k, rather than $k+1$, we have:

$$\mathbf{p}_k = \mathbf{g}_k - \frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{p}_{k-1}^T\left(\mathbf{g}_k - \mathbf{g}_{k-1}\right)}\mathbf{p}_{k-1} \tag{6.38}$$

Finally, observe that $\mathbf{p}_{k-1}^T\mathbf{g}_k = 0$ and:

$$\mathbf{p}_{k-1} = \mathbf{g}_{k-1} + \beta_{k-1}\mathbf{p}_{k-2} \tag{6.39}$$

Thus:

$$-\mathbf{p}_{k-1}^T\mathbf{g}_{k-1} = -\mathbf{g}_{k-1}^T\mathbf{g}_{k-1} - \beta_{k-1}\mathbf{p}_{k-2}^T\mathbf{g}_{k-1} = -\mathbf{g}_{k-1}^T\mathbf{g}_{k-1} \tag{6.40}$$

because $\mathbf{p}_{k-2}^T\mathbf{g}_{k-1} = 0$. Finally we see that:

$$\mathbf{p}_k = \mathbf{g}_k + \frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{g}_{k-1}^T\mathbf{g}_{k-1}}\mathbf{p}_{k-1} \tag{6.41}$$

Thus:

$$\beta_k = \frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{g}_{k-1}^T\mathbf{g}_{k-1}} \tag{6.42}$$

The fact that this algorithm converges to $\mathbf{x}^*$ in at most n iterations is a result of Theorem 6.7. This completes the proof. □

Exercise 44. Decide whether the following is true and if so, prove it; if not, provide a counter-example: In the conjugate gradient method, $\mathbf{p}_k$ is always an ascent direction and therefore $\beta_k > 0$.

Definition 6.17. Let $f : \mathbb{R}^n \to \mathbb{R}$ be as in Definition 6.5 and suppose that $\mathbf{x} = \mathbf{M}\mathbf{y}$ where $\mathbf{M} \in \mathbb{R}^{n\times n}$ is symmetric and invertible. Then when the conjugate gradient method is applied to:

$$h(\mathbf{y}) = f(\mathbf{M}\mathbf{y}) = \frac{1}{2}\mathbf{y}^T\mathbf{M}\mathbf{Q}\mathbf{M}\mathbf{y} - \mathbf{b}^T\mathbf{M}\mathbf{y} \tag{6.43}$$

So that:

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \delta_k\mathbf{d}_k$$

where:

$$\begin{aligned}\mathbf{d}_0 &= \nabla h(\mathbf{y}_0)\\ \mathbf{d}_k &= \nabla h(\mathbf{y}_k) + \beta \mathbf{d}_{k-1}\end{aligned}$$

and

$$\beta_k = \frac{\nabla h(\mathbf{y}_k)^T \nabla h(\mathbf{y}_k)}{\nabla h(\mathbf{y}_{k-1})^T \nabla h(\mathbf{y}_{k-1})}$$

When δ_k is obtained by line maximization, then the method is called the *preconditioned conjugate gradient method.*

THEOREM 6.18. *Consider the preconditioned conjugate gradient method with matrix* $\mathbf{M}$*. Then when* $\mathbf{x}_k = \mathbf{M}\mathbf{y}_k$*, then the preconditioned conjugate gradient method is equivalent to the conjugate gradient method where:*

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k \mathbf{p}_k \tag{6.44}$$

$$\mathbf{p}_0 = \mathbf{M}\mathbf{g}_0 \tag{6.45}$$

$$\mathbf{p}_k = \mathbf{M}\mathbf{g}_k + \beta \mathbf{p}_{k-1}, \quad k = 1, \dots, n-1 \tag{6.46}$$

$$\beta_k = \frac{\mathbf{g}_k^T \mathbf{M}^2 \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{M}^2 \mathbf{g}_{k-1}} \tag{6.47}$$

and δ_k *is obtained by line maximization. Furthermore,* $\mathbf{p}_k$ *are* $\mathbf{Q}$ *conjugate.*

EXERCISE 45. Prove Theorem 6.18.

4. Application to Non-Quadratic Problems

REMARK 6.19. The conjugate gradient method can be applied to non-quadratic functions, usually in an attempt to exploit locally quadratic behavior. Assuming $f : \mathbb{R}^n \to \mathbb{R}$ is a differentiable function, the update rule is given by:

$$\beta_k = \frac{\nabla f(\mathbf{x}_k)^T \left(\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{k-1})\right)}{\nabla f(\mathbf{x}_{k-1})^T \nabla f(\mathbf{x}_{k-1})} \tag{6.48}$$

Alternatively we can also use:

$$\beta_k = \frac{\nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_k)}{\nabla f(\mathbf{x}_{k-1})^T \nabla f(\mathbf{x}_{k-1})} \tag{6.49}$$

where the first formulation is called the Polak-Ribiéra formulation and is derived from Equation 6.33, while the second is the Fletcher-Reeves formulation and adapted from the straightforward transformation of the conjugate gradient method for quadratic functions. We note that the Polak-Ribiéra formulation is generally preferred.

EXERCISE 46. Prove that the Polak-Ribiéra formulation and the Fletcher-Reeves formulation for β_k are identical in the case when $f(\mathbf{x})$ is a strictly concave quadratic function.

REMARK 6.20. Sample code for the general conjugate gradient method is shown in Algorithm 13.

REMARK 6.21. Notice that we have a while-loop encasing the steps of the conjugate gradient method. There are three ways to handle the general conjugate gradient method:

```
ConjugateGradient := proc (F::operator, initVals::list, epsilon::numeric,
  maxIter::integer)::list;
  uses VectorCalculus,LinearAlgebra,Optimization:
  local vars, xnow, i, j, G, normG, OUT, passIn, phi, vals, ttemp, count,
    X, p, gnext, gnow, beta, pnext;

  #The first few lines are house keeping.
  vars := []; xnow := []; vals := initVals;
  for i to nops(initVals) do
    vars := [op(vars),
    lhs(initVals[i])];
    xnow := [op(xnow), rhs(initVals[i])]:
  end do;

  #Compute the gradient and its norm, use the VectorCalculus package.
  G := Gradient(F(op(vars)), vars));
  normG := Norm(evalf(eval(G, initVals)));

  #Output will be the path we take to the "optimal" solution.
  #Your output could just be the optimal solution.
  OUT := [];

  #An iteration counter.
  count := 0;

  #While we're not at a stationary point:
  while evalb(epsilon < normG and count < maxIter) do
    count := count+1;

    #Compute the initial direction. This is p[0].
    p := evalf(eval(G, vals));

    #Here's the actual conjugate gradient search.
    for j to nops(initVals) do
      #Append the current position.
      OUT := [op(OUT), xnow];

      #Compute the gradient (wherever we are). This is g[k-1].
      gnow := evalf(eval(G, vals));

      #Compute the next x-position, this will be x[k]
      #Do housekeeping first.
      passIn := convert(Vector(xnow)+s*p, list):
```

```
      #Define the line function f(x+delta*p)
      phi := proc (t) options operator, arrow;
        evalf(eval(F(op(passIn)), s = t))
      end proc;

      #Run line search to find delta
      ttemp := LineSearch(phi);

      #Compute the new position x + delta * p (This is x[k].)
      xnow := evalf(eval(passIn, [s = ttemp]));

      #Do some house keeping.
      vals := [];
      for i to nops(vars) do
        vals := [op(vals), vars[i] = xnow[i]]
      end do;

      #Compute the next gradient (this is g[k]).
      gnext := evalf(eval(G, vals));

      #Compute beta[k]
      beta := (Transpose(gnext).(gnext - gnow))/(Transpose(gnow).gnow);

      #Compute p[k].
      pnext := gnext + beta * p:
      p:=pnext:


    end do;

    #Append any remaining x positions.
    OUT := [op(OUT), xnow];

    #Compute the new gradient norm.
    normG := Norm(evalf(eval(G, vals)));

    #Print out the gradient norm.
    print(sprintf("normg=%f", normG))
  end do;
  #Return the output.
  OUT:
end proc:
```

Algorithm 13. Conjugate Gradient Method for non-quadratic functions with a simple loop.

(1) The conjugate gradient sub-steps can be executed n times (when maximizing a function $f : \mathbb{R}^n \to \mathbb{R}$).
(2) The conjugate gradient sub-steps can be executed $k < n$ times.
(3) The conjugate gradient method can be executed n times or as long as:

$$||\nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_{k-1})|| \leq \delta ||\nabla f(\mathbf{x}_{k-1})||^2 \tag{6.50}$$

for $\delta \in (0,1)$. Otherwise, restart with a gradient step. The previous equation acts as a test for conjugacy. F

The idea behind these conjugate gradient methods is to execute a gradient ascent step every "few" iterations to ensure global convergence to a stationary point, while simultaneously trying to speed up convergence by using conjugate gradient steps in between.

Example 6.22. Consider the execution of the conjugate gradient method to the function:

$$F(x,y) = \left(x^2 + 3\,y^2\right) \mathrm{e}^{1-x^2-y^2}$$

We obtain convergence in four iterations (8 steps of the conjugate gradient method) when we choose $\epsilon = 0.001$ and begin at $x = 1$, $y = 0.1$. This is illustrated in Figure 6.1.

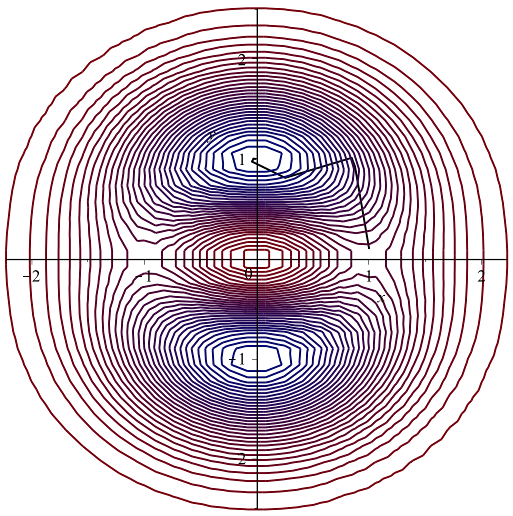


Figure 6.1. The steps of the conjugate gradient algorithm applied to $F(x,y)$.

We can compare this behavior with the case when:

$$F(x,y) = -2x^2 - 10y^4$$

and we start from $x = 15$, $y = 5$. This is illustrated in Figure 6.2. In this example, the conjugate gradient method converges in two iterations (four steps), with much less *zig-zagging* than the gradient descent method or even Newton's method. Notice also how the steps are highly normal to each other as we expect from Lemma 6.15.

Example 6.23. Consider the function:

$$F(x,y) = -2x^2 - 10y^2 \tag{6.51}$$

The conjugate gradient method *should* converge in two steps for this function, but in practice (depending on the value of ϵ) it may not because of floating point error. In fact, if we run the conjugate gradient algorithm on this function, starting from $x = 15$, $y = 5$, we obtain

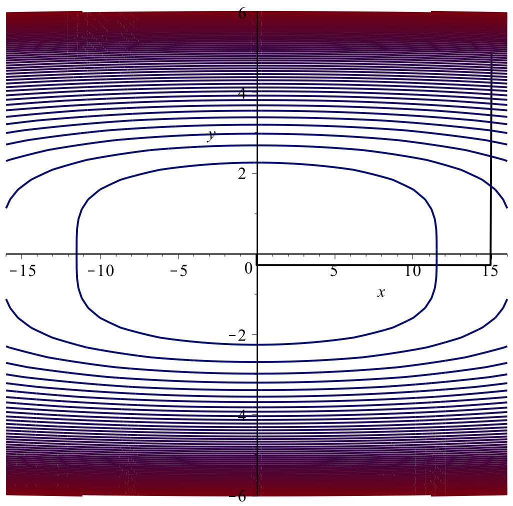


Figure 6.2. In this example, the conjugate gradient method also converges in four total steps, with much less *zig-zagging* than the gradient descent method or even Newton's method.

convergence after a surprising 4 steps when $\epsilon = 0.001$ because of floating point error. This will not present itself if we use exact arithmetic. We can solve this problem in floating point by using the preconditioned conjugate gradient method. First, let us write:

$$F(x, y) = \frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} -4 & 0 \\ 0 & -20 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} \tag{6.52}$$

Consider the matrix:

$$\mathbf{M} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/10\sqrt{5} \end{bmatrix} \tag{6.53}$$

If we let:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \mathbf{M} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/10\sqrt{5} \end{bmatrix}\begin{bmatrix} z \\ w \end{bmatrix} \tag{6.54}$$

Then:

$$h(z, w) = \frac{1}{2}\begin{bmatrix} z & w \end{bmatrix}\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} z \\ w \end{bmatrix} \tag{6.55}$$

because:

$$\mathbf{M}^T\mathbf{Q}\mathbf{M} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \tag{6.56}$$

where:

$$\mathbf{Q} = \begin{bmatrix} -4 & 0 \\ 0 & -20 \end{bmatrix} \tag{6.57}$$

If we execute the preconditioned conjugate gradient algorithmon $h(z,w)$ starting from the position $z = 30$ and $w = 10\sqrt{5}$, where is:

$$\begin{bmatrix} 30 \\ 10\sqrt{5} \end{bmatrix} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/10\sqrt{5} \end{bmatrix}^{-1} \begin{bmatrix} 15 \\ 5 \end{bmatrix}$$

we obtain convergence to $z^* = w^* = 0$ in two steps (as expected) and further we can see at once that $x^* = y^* = 0$. Thus, $\mathbf{M}$ has *stretched* (and it could have twisted) our problem into the problem from Example 6.8.

Exercise 47. Implement the conjugate gradient method. Try is on $F(x,y) = -2x^2 - 10y^2$.

Remark 6.24. We state but do not prove one final theorem regarding the conjugate gradient method, which helps explain why the conjugate gradient method might only be executed $k < n$ times before a restart. The proof is available in [**Ber99**].

Theorem 6.25. *Assume* $\mathbf{Q}$ *has* $n-k$ *eigenvalues on the interval* $[a,b]$ *with* $b < 0$ *and the remaining eigenvalues are less than* a*. Then for every* $\mathbf{x}_0$*, the vector* $\mathbf{x}_{k+1}$ *produced after* $k+1$ *steps of the conjugate gradient method satisfies:*

$$f(\mathbf{x}_{k+1}) \geq \left(\frac{b-a}{b+a}\right)^2 f(\mathbf{x}_0) \tag{6.58}$$

□

Remark 6.26. A similar condition as in Theorem 6.25 can be proved for pre-conditioned conjugate gradient method with appropriate transformations to the matrix. Thus, Theorem 6.25 helps explain conditions in which we may only be interested in executing the conjugate gradient loop $k < n$ times.

CHAPTER 7

Quasi-Newton Methods

Remark 7.1. We noted in Chapter 5 that Newton's method only works in a region around local maxima (or minima) and otherwise, the direction $-\mathbf{H}(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)$ may not be an ascent direction. This is solved in modified Newton's method approaches by adjusting the hessian matrix to ensure negative (or positive definiteness). Quasi-Newton methods, by contrast construct positive definite matrices $\mathbf{B}_k$ and use the update:

$$\mathbf{r}_k = \mathbf{B}_k \nabla f(\mathbf{x}_k) \tag{7.1}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k \mathbf{r}_k \tag{7.2}$$

These methods have the property that $\mathbf{B}_k$ approaches $-\mathbf{H}(\mathbf{x}_k)^{-1}$ as $\mathbf{x}_k$ approaches a stationary point $\mathbf{x}^*$. This property can be showed exactly for quadratic functions.

Exercise 48. Show that the Conjugate Gradient method with the Fletcher-Reeves rule can be thought of as like a quasi-Newton method when:

$$\mathbf{B}_k = \mathbf{I}_n + \frac{(\mathbf{x}_k - \mathbf{x}_{k-1})\nabla f(\mathbf{x}_k)^T}{\nabla f(\mathbf{x}_{k-1})^T \nabla f(\mathbf{x}_{k-1})} \tag{7.3}$$

while the Conjugate Gradient method with the Polak-Ribiéra rule can be thought of as like a quasi-Newton method when:

$$\mathbf{B}_k = \mathbf{I}_n + \frac{(\mathbf{x}_k - \mathbf{x}_{k-1})(\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{k-1}))^T}{\nabla f(\mathbf{x}_{k-1})^T \nabla f(\mathbf{x}_{k-1})} \tag{7.4}$$

You do not have to prove $\mathbf{B}_k$ converges to $\mathbf{H}^{-1}(\mathbf{x}^*)$.

1. Davidon-Fletcher-Powell (DFP) Quasi-Newton Method

Definition 7.2 (Davidon-Fletcher-Powell Approximation). Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function. Let:

$$\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k = \delta_k \mathbf{r}_k \tag{7.5}$$

$$\mathbf{q}_k = \nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{k+1}) \tag{7.6}$$

Then the *Davidon-Fletcher-Powell* (DFP) inverse-Hessian approximation is:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{p}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{q}_k} - \frac{\mathbf{B}_k \mathbf{q}_k \mathbf{q}_k^T \mathbf{B}_k}{\mathbf{q}_k^T \mathbf{B}_k \mathbf{q}_k} \tag{7.7}$$

where $\mathbf{B}_0 \in \mathbb{R}^{n\times n}$ is a symmetric positive definite matrix (usually $\mathbf{I}_n$).

Remark 7.3. This formulation is useful for **maximization problems**. The DFP minimization formulation set $\mathbf{q}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ and $\mathbf{r}_k = -\mathbf{B}_k \nabla f(\mathbf{x}_k)$.

Lemma 7.4. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{x}_0 \in \mathbb{R}^n$ and $\mathbf{B}_0 \in \mathbb{R}^{n\times n}$ is a symmetric and positive definite matrix and $\mathbf{B}_1, \dots, \mathbf{B}_n$ are generated using Equation 7.7 and $\mathbf{x}_1, \dots, \mathbf{x}_n$ are generated by:*

$$\mathbf{r}_k = \mathbf{B}_k \nabla f(\mathbf{x}_k) \tag{7.8}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k \mathbf{r}_k \tag{7.9}$$

with $\delta_k = \arg\max f(\mathbf{x}_k + \delta_k \mathbf{r}_k)$. If $\nabla f(\mathbf{x}_k) \neq \mathbf{0}$ for $k = 0, \dots, n$ then $\mathbf{B}_1, \dots \mathbf{B}_n$ are symmetric and positive definite. Thus, $\mathbf{r}_0, \dots, \mathbf{r}_{n-1}$ are ascent directions.

Proof. We proceed by induction. Clear $\mathbf{B}_0$ is symmetric and positive definite and thus $\mathbf{p}_0$ is an ascent direction since $\nabla f(\mathbf{x}_0)^T \mathbf{p}_0 = \nabla f(\mathbf{x}_0)^T \mathbf{B}_0 \nabla f(\mathbf{x}_0) > 0$. Symmetry is ensured by the nature of DFP formula. Assume that statement is true for all $k < n$. We show the statement is true for $\mathbf{B}_{k+1}$.

Consider $\mathbf{y}^T \mathbf{B}_{k+1} \mathbf{y}$ for any $\mathbf{y} \in \mathbb{R}^n$. We have:

$$\mathbf{y}^T \mathbf{B}_{k+1} \mathbf{y} = \mathbf{y}^T \mathbf{B}_k \mathbf{y} + \frac{\mathbf{y}^T \mathbf{p}_k \mathbf{p}_k^T \mathbf{y}}{\mathbf{p}_k^T \mathbf{q}_k} - \frac{\mathbf{y}^T \mathbf{B}_k \mathbf{q}_k \mathbf{q}_k^T \mathbf{B}_k \mathbf{y}}{\mathbf{q}_k^T \mathbf{B}_k \mathbf{q}_k} \tag{7.10}$$

Note $\mathbf{y}^T \mathbf{p}_k = \mathbf{p}_k^T \mathbf{y}$ and $\mathbf{y}^T \mathbf{B}_k \mathbf{q}_k = \mathbf{q}_k^T \mathbf{B}_k \mathbf{y}$ because $\mathbf{B}_k$ is symmetric by the induction hypothesis. Furthermore, $\mathbf{B}_k$ has a Cholesky decomposition so that $\mathbf{B}_k = \mathbf{L}_k \mathbf{L}_k^T$ for some lower triangular matrix $\mathbf{L}_k$. Let:

$$\mathbf{a}^T = \mathbf{y}^T \mathbf{L}_k \tag{7.11}$$

$$\mathbf{b}^T = \mathbf{q}_k^T \mathbf{L}_k \tag{7.12}$$

Then we can write:

$$\mathbf{y}^T \mathbf{B}_{k+1} \mathbf{y} = \mathbf{a}^T \mathbf{a} - \frac{\left(\mathbf{a}^T \mathbf{b}\right)\left(\mathbf{b}^T \mathbf{a}\right)}{\mathbf{b}^T \mathbf{b}} + \frac{\left(\mathbf{y}^T \mathbf{p}_k\right)^2}{\mathbf{p}_k^T \mathbf{q}_k} = \frac{\left(\mathbf{a}^T \mathbf{a}\right)\left(\mathbf{b}^T \mathbf{b}\right) - \left(\mathbf{a}^T \mathbf{b}\right)\left(\mathbf{b}^T \mathbf{a}\right)}{\mathbf{b}^T \mathbf{b}} + \frac{\left(\mathbf{y}^T \mathbf{p}_k\right)^2}{\mathbf{p}_k^T \mathbf{q}_k} \tag{7.13}$$

Recall the Schwartz Inequality (Lemma 1.13):

$$(\mathbf{a}^T \mathbf{b})(\mathbf{b}^T \mathbf{a}) = (\mathbf{a}^T \mathbf{b})^2 \leq (\mathbf{a}^T \mathbf{a}) \cdot (\mathbf{b}^T \mathbf{b}) \tag{7.14}$$

Thus,

$$\frac{\left(\mathbf{a}^T \mathbf{a}\right)\left(\mathbf{b}^T \mathbf{b}\right) - \left(\mathbf{a}^T \mathbf{b}\right)\left(\mathbf{b}^T \mathbf{a}\right)}{\mathbf{b}^T \mathbf{b}} > 0 \tag{7.15}$$

since $\mathbf{b}^T \mathbf{b} \geq 0$. As in the proof of Theorem 6.7, we know that:

$$\frac{\partial f(\mathbf{x}_{k+1})}{\partial \delta_k} = \nabla f(\mathbf{x}_{k+1})^T \mathbf{r}_k = \nabla f(\mathbf{x}_{k+1})^T \mathbf{p}_k = 0 \tag{7.16}$$

because $\delta_k = \arg\max f(\mathbf{x}_k + \delta_k \mathbf{r}_k)$. This means that:

$$\mathbf{p}_k^T \mathbf{q}_k = \mathbf{p}_k^T (\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{k+1})) = \mathbf{p}_k^T \nabla f(\mathbf{x}_k) = \nabla f(\mathbf{x}_k)^T \mathbf{B}_k^T \nabla f(\mathbf{x}_k) > 0 \tag{7.17}$$

Thus:

$$\frac{\left(\mathbf{y}^T \mathbf{p}_k\right)^2}{\mathbf{p}_k^T \mathbf{q}_k} > 0 \tag{7.18}$$

and $\mathbf{y}^T\mathbf{B}_{k+1}\mathbf{y} > 0$. Thus, $\mathbf{B}_{k+1}$ is positive definite and since $\mathbf{r}_{k+1} = \mathbf{B}_{k+1}\nabla f(\mathbf{x}_{k+1})$ is an ascent direction. □

Exercise 49. Prove: Suppose $f : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{x}_0 \in \mathbb{R}^n$ and $\mathbf{B}_0 \in \mathbb{R}^{n\times n}$ is a symmetric and positive definite matrix and $\mathbf{B}_1, \dots, \mathbf{B}_n$ are generated using the DFP formula for minimization and $\mathbf{x}_1, \dots, \mathbf{x}_n$ are generated by:

$$\mathbf{r}_k = -\mathbf{B}_k\nabla f(\mathbf{x}_k) \tag{7.19}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k\mathbf{r}_k \tag{7.20}$$

with $\delta_k = \arg\min f(\mathbf{x}_k + \delta_k\mathbf{r}_k)$. If $\nabla f(\mathbf{x}_k) \neq \mathbf{0}$ for $k = 0, \dots, n$ then $\mathbf{B}_1, \dots \mathbf{B}_n$ are symmetric and positive definite. Thus, $\mathbf{r}_0, \dots, \mathbf{r}_{n-1}$ are descent directions.

Theorem 7.5. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ with:*

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{b}^T\mathbf{x} \tag{7.21}$$

where $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is symmetric and negative definite. Suppose that $\mathbf{B}_1, \dots, \mathbf{B}_{n-1}$ are generated using the DFP formulation (Equation 7.7) with $\mathbf{x}_0 \in \mathbb{R}^n$ and $\mathbf{B}_0 \in \mathbb{R}^{n\times n}$, symmetric and positive definite. If $\mathbf{x}_1, \dots, \mathbf{x}_n$ and $\mathbf{r}_0, \dots, \mathbf{r}_{n-1}$ are generated by:

$$\mathbf{r}_k = \mathbf{B}_k\nabla f(\mathbf{x}_k) \tag{7.22}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k\mathbf{r}_k \tag{7.23}$$

with $\delta_k = \arg\max f(\mathbf{x}_k + \delta_k\mathbf{r}_k)$ and

$$\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k = \delta_k\mathbf{r}_k \tag{7.24}$$

$$\mathbf{q}_k = \nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{k+1}) \tag{7.25}$$

then:

1. *$\mathbf{p}_0, \dots, \mathbf{p}_{n-1}$ are $\mathbf{Q}$ conjugate and*
2. *$\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_j = -\mathbf{p}_j$ for $j = 0, \dots, k$*
3. *$\mathbf{B}_n = -\mathbf{Q}^{-1}$ and*
4. *$\mathbf{x}_n = \mathbf{x}^*$ is the global maximum of $f(\mathbf{x})$.*

Proof. It suffices to prove (1) and (2) above. To see this note that if (1) holds, then the DFP quasi-Newton method is a conjugate direction method and therefore after n iterations, it must converge to the maximum $\mathbf{x}^*$ of $f(\mathbf{x})$. Furthermore, since $\mathbf{p}_0, \dots, \mathbf{p}_{n-1}$ must be linearly independent by Lemma 6.4, they are a basis for $\mathbb{R}^n$. Therefore, for each standard basis vector $\mathbf{e}_i$ ($i = 1, \dots, n$) we see that there are $\alpha_0, \dots, \alpha_{n-1}$ so that:

$$\mathbf{e}_i = \alpha_0\mathbf{p}_0 + \cdots + \alpha_{n-1}\mathbf{p}_{n-1}$$

and thus: $\mathbf{B}_{k+1}\mathbf{Q}\mathbf{e}_i = -\mathbf{e}_i$. The previous result holds for $i = 1, \dots, n$ and therefore $\mathbf{B}_{k+1}\mathbf{Q} = -\mathbf{I}_n$, which means that $\mathbf{B}_n = -\mathbf{Q}^{-1}$.

Recall that $\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and therefore:

$$\mathbf{Q}\mathbf{p}_k = \mathbf{Q}(\mathbf{x}_{k+1} - \mathbf{x}_k) = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) = -\mathbf{q}_k \tag{7.26}$$

Multiplying by $\mathbf{B}_{k+1}$ we have:

$$\begin{aligned}(7.27)\quad & \mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_k = -\mathbf{B}_{k+1}\mathbf{q}_k = \\ & -\left(\mathbf{B}_k + \frac{\mathbf{p}_k\mathbf{p}_k^T}{\mathbf{p}_k^T\mathbf{q}_k} - \frac{\mathbf{B}_k\mathbf{q}_k\mathbf{q}_k^T\mathbf{B}_k}{\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k}\right)\mathbf{q}_k = -\mathbf{B}_k\mathbf{q}_k - \frac{\mathbf{p}_k\mathbf{p}_k^T\mathbf{q}_k}{\mathbf{p}_k^T\mathbf{q}_k} + \frac{\mathbf{B}_k\mathbf{q}_k\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k}{\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k} = \\ & -\mathbf{B}_k\mathbf{q}_k - \mathbf{p}_k + \mathbf{B}_k\mathbf{q}_k = -\mathbf{p}_k\end{aligned}$$

We now proceed by induction to show that $\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_j = -\mathbf{p}_j$ for $j = 0, \dots, k$. We will also prove $\mathbf{Q}$ conjugacy. We have just proved the based case when $k = 0$ and therefore, we assume this is true for all iterations up to some k. Note that:

$$\begin{aligned}(7.28)\quad & \nabla f(\mathbf{x}_{k+1}) = \nabla f(\mathbf{x}_{i+1}) + \\ & (\nabla f(\mathbf{x}_{i+2}) - \nabla f(\mathbf{x}_{i+1})) + (\nabla f(\mathbf{x}_{i+3}) - \nabla f(\mathbf{x}_{i+2})) + \cdots + (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)) \\ & = \nabla f(\mathbf{x}_{i+1}) + \mathbf{Q}\mathbf{p}_{i+1} + \mathbf{Q}\mathbf{p}_{i+2} + \cdots + \mathbf{Q}\mathbf{p}_k = \nabla f(\mathbf{x}_{i+1}) + \mathbf{Q}\left(\mathbf{p}_{i+1} + \mathbf{p}_{i+2} + \cdots + \mathbf{p}_k\right)\end{aligned}$$

By the induction hypothesis, we know that $\mathbf{p}_i$ is orthogonal to $\mathbf{p}_j$ where $j = i+1 \dots k$ and we know a fortiori that $\mathbf{p}_i$ is orthogonal to $\nabla f(\mathbf{x}_{i+1})$ (see Equation 7.16, as in the proof of Theorem 6.7). We conclude that: $\mathbf{p}_i^T\nabla f(\mathbf{x}_{k+1}) = 0$ for $0 \leq i \leq k$. Applying the induction hypothesis again, we know that: $\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_j = -\mathbf{p}_j$ for $j = 0, \dots, k$ and therefore:

$$(7.29)\quad \nabla f(\mathbf{x}_{k+1})^T\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_j = -\nabla f(\mathbf{x}_{k+1})^T\mathbf{p}_j = 0$$

for $j = 0, \dots, k$. But:

$$(7.30)\quad \delta_{k+1}\mathbf{r}_{k+1}^T = \delta_{k+1}\nabla f(\mathbf{x}_{k+1})^T\mathbf{B}_{k+1} = \mathbf{p}_{k+1}$$

Therefore:

$$(7.31)\quad \mathbf{p}_{k+1}^T\mathbf{Q}\mathbf{p}_j = \delta_{k+1}\mathbf{r}_{k+1}^T\mathbf{Q}\mathbf{p}_j = \delta_{k+1}\nabla f(\mathbf{x}_{k+1})^T\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_j = 0$$

and thus, $\mathbf{p}_0, \dots, \mathbf{p}_{k+1}$ are $\mathbf{Q}$ conjugate. We now need only prove $\mathbf{B}_{k+2}\mathbf{Q}\mathbf{p}_j = -\mathbf{p}_j$ for $j = 0, \dots, k+1$. We first note by the induction hypothesis that:

$$(7.32)\quad \mathbf{q}_{k+1}^T\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_i = -\mathbf{q}_{k+1}^T\mathbf{p}_i = \left(\mathbf{Q}\mathbf{p}_{k+1}\right)^T\mathbf{p}_i = \mathbf{p}_{k+1}^T\mathbf{Q}\mathbf{p}_i = 0$$

because $\mathbf{Q}$ is symmetric and we have established that $\mathbf{p}_0, \dots, \mathbf{p}_{k+1}$ are $\mathbf{Q}$ conjugate. Then we write:

$$\begin{aligned}(7.33)\quad & \mathbf{B}_{k+2}\mathbf{q}_i = \left(\mathbf{B}_{k+1} + \frac{\mathbf{p}_{k+1}\mathbf{p}_{k+1}^T}{\mathbf{p}_{k+1}^T\mathbf{q}_{k+1}} - \frac{\mathbf{B}_{k+1}\mathbf{q}_{k+1}\mathbf{q}_{k+1}^T\mathbf{B}_{k+1}}{\mathbf{q}_{k+1}^T\mathbf{B}_{k+1}\mathbf{q}_{k+1}}\right)\mathbf{q}_i = \\ & \mathbf{B}_{k+1}\mathbf{q}_i + \frac{\mathbf{p}_{k+1}\mathbf{p}_{k+1}^T\mathbf{q}_i}{\mathbf{p}_{k+1}^T\mathbf{q}_{k+1}} - \frac{\mathbf{B}_{k+1}\mathbf{q}_{k+1}\mathbf{q}_{k+1}^T\mathbf{B}_{k+1}\mathbf{q}_i}{\mathbf{q}_{k+1}^T\mathbf{B}_{k+1}\mathbf{q}_{k+1}}\end{aligned}$$

Note that: $\mathbf{Q}\mathbf{p}_i = -\mathbf{q}_i$ and since $\mathbf{p}_{k+1}^T\mathbf{Q}\mathbf{p}_i = 0$, we know $-\mathbf{p}_{k+1}^T\mathbf{q}_i = -\mathbf{p}_{k+1}\mathbf{Q}\mathbf{p}_i = 0$. Which implies $\mathbf{p}_{k+1}^T\mathbf{q}_i = 0$. Therefore,

$$(7.34)\quad \frac{\mathbf{p}_{k+1}\mathbf{p}_{k+1}^T\mathbf{q}_i}{\mathbf{p}_{k+1}^T\mathbf{q}_{k+1}} = 0$$

Finally since $\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_i = -\mathbf{p}_i$, we know:

$$(7.35)\quad \mathbf{q}_{k+1}^T\mathbf{B}_{k+1}\mathbf{q}_i = -\mathbf{q}_{k+1}^T\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_i = \mathbf{q}_{k+1}^T\mathbf{p}_i = -\mathbf{p}_{k+1}^T\mathbf{Q}\mathbf{p}_i = 0$$

Thus:

$$\frac{\mathbf{B}_{k+1}\mathbf{q}_{k+1}\mathbf{q}_{k+1}^T\mathbf{B}_{k+1}\mathbf{q}_i}{\mathbf{q}_{k+1}^T\mathbf{B}_{k+1}\mathbf{q}_{k+1}} = 0 \tag{7.36}$$

Thus we conclude that:

$$\mathbf{B}_{k+2}\mathbf{q}_i = \mathbf{B}_{k+1}\mathbf{q}_i = -\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_i = \mathbf{p}_i \tag{7.37}$$

and therefore:

$$\mathbf{B}_{k+2}\mathbf{Q}\mathbf{p}_i = -\mathbf{p}_i \tag{7.38}$$

So for $i = 0, \dots, k$ we have $\mathbf{B}_{k+2}\mathbf{Q}\mathbf{p}_i = -\mathbf{p}_i$. To prove that this is also true when $i = k + 1$ recall that Equation 7.27, handles this case. Thus we have shown that $\mathbf{B}_{k+2}\mathbf{Q}\mathbf{p}_i = -\mathbf{p}_i$ for $i = 0, \dots, k + 1$. This completes the proof. □

2. Implementation and Example of DFP

Example 7.6. Consider $F(x, y) = -2x^2 - 10y^2$. We know with exact arithmetic, the DFP method should converge in two steps (1 iteration). To see this, suppose $\mathbf{B}_0 = \mathbf{I}_2$. If we start at $x = 15$ and $y = 5$. Then our first gradient and ascent direction are:

$$\mathbf{r}_0 = \mathbf{g}_0 = \begin{bmatrix} -60 \\ -100 \end{bmatrix} \tag{7.39}$$

Our first line function, then is:

$$\phi_0(t) = -2\ (15 - 60\,t)^2 - 10\ (5 - 100\,t)^2 \tag{7.40}$$

Solving $\phi'(t) = 0$ for t yields:

$$t^* = \frac{17}{268} \tag{7.41}$$

This leads to our next point $x_1 = 15 + \frac{17}{268}(-60)$ and $y_1 = 5 + \frac{17}{268}(-100)$ or $x_1 = \frac{750}{67}$ and $y_1 = \frac{-90}{67}$. We compute the gradient at this new point to obtain:

$$\mathbf{g}_1 = \begin{bmatrix} \frac{-3000}{67} \\ \frac{1800}{67} \end{bmatrix} \tag{7.42}$$

This leads to the two expressions:

$$\mathbf{p}_0 = \begin{bmatrix} \frac{-255}{67} \\ \frac{-425}{67} \end{bmatrix} \tag{7.43}$$

$$\mathbf{q}_0 = \begin{bmatrix} \frac{-1020}{67} \\ \frac{-8500}{67} \end{bmatrix} \tag{7.44}$$

If we compute $\mathbf{B}_1$ we obtain:

$$\mathbf{B}_1 = \begin{bmatrix} \frac{170353}{169912} & -\frac{15345}{169912} \\ -\frac{15345}{169912} & \frac{10337}{169912} \end{bmatrix} \tag{7.45}$$

We now compute $\mathbf{r}_1$:

$$\mathbf{r}_1 = \begin{bmatrix} -\frac{15000}{317} \\ \frac{1800}{317} \end{bmatrix} \tag{7.46}$$

From $\mathbf{r}_1$, we can compute our new line search function:

$$\phi_1(t) = -2\left(\frac{750}{67} - \frac{15000}{317}t\right)^2 - 10\left(-\frac{90}{67} + \frac{1800}{317}t\right)^2 \tag{7.47}$$

We can find $t^* = \frac{317}{1340}$ as before and compute our new position:

$$x_2 = \frac{750}{67} + \left(\frac{317}{1340}\right)\left(\frac{-15000}{317}\right) = 0 \tag{7.48}$$

$$y_2 = \frac{-90}{67} + \left(\frac{317}{1340}\right)\left(\frac{1800}{317}\right) = 0 \tag{7.49}$$

Thus showing that the DFP method converges in one iteration or two steps.

Remark 7.7. An implementation of the DFP quasi-Newton method is shown in Algorithm 14[1].

Example 7.8. We can apply the DFP method to:

$$F(x, y) = \left(x^2 + 3\,y^2\right) e^{1-x^2-y^2}$$

We obtain convergence in four iterations (8 steps of the conjugate gradient method) when we choose $\epsilon = 0.001$ and begin at $x = 1$, $y = 0.1$. This is illustrated in Figure 7.1. Notice

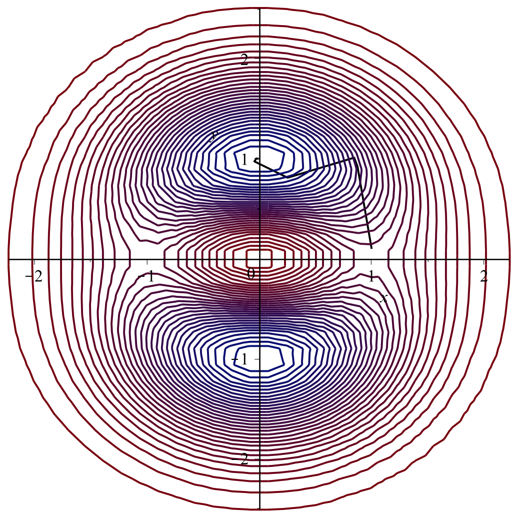


Figure 7.1. The steps of the DFP algorithm applied to $F(x, y)$.

that the steps are identical to those shown in the conjugate gradient method (see Figure 6.1). This is because the DFP method is a conjugate direction method.

[1]Thanks to Simon Miller who noticed that line 75 was missing in versions before 0.8.7. That's a big mistake!

```
DFPQuasiNewton := proc (F::operator, initVals::list,
  epsilon::numeric, maxIter::integer)::list;

  local vars, xnow, i, j, G, normG, OUT, passIn, phi, vals, ttemp, count, X, p,
      gnext,
    gnow, beta, pnext, r, B, xnext, q;

  #Do some house keeping.
  vars := [];
  xnow := [];
  vals := initVals;
  for i to nops(initVals) do
    vars := [op(vars), lhs(initVals[i])];
    xnow := [op(xnow), rhs(initVals[i])]
  end do;

  #Define the gradient vector and the current norm of the gradient.
  G := Vector(Gradient(F(op(vars)), vars));
  normG := Norm(evalf(eval(G, initVals)));

  #Output will be the path we take to an optimal point. We also define an
  #iteration counter.
  OUT := [];
  count := 0;

  #While we're not at a stationary point...
  while evalb(epsilon < normG and count < maxIter) do
    count := count + 1;

    #Define B[0] and r[0], the first direction to walk. (It's a gradient step.)
    B := IdentityMatrix(nops(vars), nops(vars));
    r := B.evalf(eval(G, vals)));

    #Now go into our conjugate direction method.
    for j to nops(initVals) do
      print(j);

      #Append some output.
      OUT := [op(OUT), xnow];

      #Evaluate the gradient at this point.
      gnow := evalf(eval(G, vals));

      #Do some housekeeping and define the line function.
      passIn := convert(Vector(xnow) + s * r), list);
      phi := proc (t) options operator, arrow;
        evalf(eval(F(op(passIn)), s = t))
      end proc;

      #Compute the optimal step length using parabolic bracketing and
      #Golden section search.
      ttemp := LineSearch(phi);

      #Define the next x position and the p-vector in DFP.
      xnext := evalf(eval(passIn, [s = ttemp]));
      p := Vector(xnext - xnow);

      #Do some housekeeping.
      xnow := xnext;
      vals := [];
      for i to nops(vars) do
        vals := [op(vals), vars[i] = xnow[i]]
      end do;

      #Evaluate the gradient at the next position
      gnext := evalf(eval(G, vals));

      #Compute the q vector.
      q := Vector(gnow-gnext);

      #Update the B matrix B[k]
      B := B + (p.Transpose(p))/(Transpose(p).q) -
        (B.q.Transpose(q).B)/(Transpose(q).B.q);

      #Compute the next direction.
      r := B.gnext
    end do;

    #Compute the norm of the gradient
    normG := Norm(evalf(eval(G, vals)));
   end do;
   OUT
  end proc
```

Algorithm 14. Davidon-Fletcher-Powell quasi-Newton method.

3. Derivation of the DFP Method

Suppose we wish to derive the DFP update from first principles. Recall we have defined:

(7.50) $$\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k = \delta_k \mathbf{r}_k$$

(7.51) $$\mathbf{q}_k = \nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{k+1}) = -\mathbf{Q}\mathbf{p}_k$$

Proceeding inductively, suppose that at Step k, we know that $\mathbf{p}_0, \dots, \mathbf{p}_k$ are all:

1. $\mathbf{Q}$ conjugate and
2. $\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_j = -\mathbf{p}_j$ for $j = 0, \dots, k$

The second statement asserts that $\mathbf{p}_0, \dots, \mathbf{p}_k$ are to be *eigenvectors* of $\mathbf{B}_{k+1}\mathbf{Q}$ with eigenvalue 1. If this holds all the way through to $k = n-1$, then clearly $\mathbf{p}_0, \dots, \mathbf{p}_{n-1}$ are eigenvalues of $\mathbf{B}_n\mathbf{Q}$, each with eigenvalue -1, yielding precisely the fact that $\mathbf{B}_n\mathbf{Q} = -\mathbf{I}_n$.

If we want to ensure that this pattern, continues to hold, suppose that we want to write:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \mathbf{C}_k \tag{7.52}$$

where $\mathbf{C}_k$ is a *correction matrix* that will ensure that the nice properties above are also true at $k+1$. That is, it ensures that: $\mathbf{p}_0, \dots, \mathbf{p}_k$ are eigenvectors of $\mathbf{B}_{k+1}\mathbf{Q}$ with eigenvalue -1. This means we require:

$$\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_j = -\mathbf{p}_j \implies \mathbf{B}_{k+1}\mathbf{q}_j = \mathbf{p}_j \quad j = 0, \dots, k \tag{7.53}$$

Combining this with Equation 7.52 yields:

$$(\mathbf{B}_k + \mathbf{C}_k)\mathbf{q}_j = \mathbf{B}_k\mathbf{q}_j + \mathbf{C}_k\mathbf{q}_j = \mathbf{p}_j \quad j = 0, \dots, k \tag{7.54}$$

But:

$$\mathbf{B}_k\mathbf{q}_j = -\mathbf{B}_k\mathbf{Q}_k\mathbf{p}_j = \mathbf{p}_j \quad j = 0, \dots, k-1 \tag{7.55}$$

Thus:

$$\mathbf{B}_k\mathbf{q}_j + \mathbf{C}_k\mathbf{q}_j = \mathbf{p}_j \implies \mathbf{p}_j + \mathbf{C}_k\mathbf{q}_j = \mathbf{p}_j \implies \mathbf{C}_k\mathbf{q}_j = \mathbf{0} \quad j = 0, \dots, k-1 \tag{7.56}$$

When $k = j$, we require:

$$\mathbf{C}_k\mathbf{q}_k = \mathbf{p}_k - \mathbf{B}_k\mathbf{q}_k \tag{7.57}$$

from Equation 7.54. We are now free to use our imagination about the structure of $\mathbf{C}_k$. Suppose that $\mathbf{C}_k$ had the rank-one term:

$$T_1 = \frac{\mathbf{p}_k\mathbf{p}_k^T}{\mathbf{p}_k^T\mathbf{q}_k} \tag{7.58}$$

Then we see that $T_1\mathbf{q}_k = \mathbf{p}_k$ and we could satisfy part of our requirement from Equation 7.57. On the other hand, if $\mathbf{C}_k$ had a second rank-one term:

$$T_2 = -\frac{\mathbf{B}_k\mathbf{q}_k\mathbf{q}_k^T\mathbf{B}_k}{\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k} \tag{7.59}$$

then $T_2\mathbf{q}_k = -\mathbf{B}_k\mathbf{q}_k$. Combining these we see:

$$\mathbf{C}_k = T_1 + T_2 = \frac{\mathbf{p}_k\mathbf{p}_k^T}{\mathbf{p}_k^T\mathbf{q}_k} - \frac{\mathbf{B}_k\mathbf{q}_k\mathbf{q}_k^T\mathbf{B}_k}{\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k} \tag{7.60}$$

as expected. Now, note that for any $\mathbf{p}_j$ ($j \neq k$) we see that:

$$\mathbf{C}_k\mathbf{q}_j = \frac{\mathbf{p}_k\mathbf{p}_k^T\mathbf{q}_j}{\mathbf{p}_k^T\mathbf{q}_k} - \frac{\mathbf{B}_k\mathbf{q}_k\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_j}{\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k} = -\frac{\mathbf{p}_k\mathbf{p}_k^T\mathbf{Q}\mathbf{p}_j}{\mathbf{p}_k^T\mathbf{p}_k} + \frac{\mathbf{B}_k\mathbf{q}_k\mathbf{q}_k^T\mathbf{B}_k\mathbf{Q}\mathbf{p}_j}{\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k} \tag{7.61}$$

We know that:

$$\frac{\mathbf{p}_k\mathbf{p}_k^T\mathbf{Q}\mathbf{p}_j}{\mathbf{p}_k^T\mathbf{p}_k} = \mathbf{0} \tag{7.62}$$

by $\mathbf{Q}$ conjugacy. While:

$$\frac{\mathbf{B}_k\mathbf{q}_k\mathbf{q}_k^T\mathbf{B}_k\mathbf{Q}\mathbf{p}_j}{\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k} = \frac{\mathbf{B}_k\mathbf{q}_k\mathbf{p}_k^T\mathbf{Q}\mathbf{p}_j}{\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k} = \mathbf{0} \tag{7.63}$$

again by conjugacy. Note in the previous equation, we transformed $\mathbf{q}_k^T$ to $-\mathbf{p}_k^T\mathbf{Q}$ and $\mathbf{B}_k\mathbf{Q}\mathbf{p}_j$ to $-\mathbf{p}_j$. Thus, $\mathbf{C}_k\mathbf{q}_j = \mathbf{0}$ are required.

Remark 7.9. Since we are adding two rank-one corrections together to obtain $\mathbf{C}_k$, the matrix $\mathbf{C}_k$ is sometimes called a rank-two correction.

4. Broyden-Fletcher-Goldfarb-Shanno (BFGS) Quasi-Newton Method

Remark 7.10. Recall from our construction of the DFP correction that we really only require two things of the correction matrix $\mathbf{C}_k$:

$$\mathbf{C}_k\mathbf{q}_j = \mathbf{0} \quad j = 0, \dots, k-1$$

and Equation 7.57:

$$\mathbf{C}_k\mathbf{q}_k = \mathbf{p}_k - \mathbf{B}_k\mathbf{q}_k$$

This we could add any extra term $\mathbf{T}_k$ we like as long as $\mathbf{T}_k\mathbf{q}_j = 0$ for $j = 0, \dots, k$.

Definition 7.11 (Broyden Family of Updates). Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function. Let:

$$\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k = \delta_k\mathbf{r}_k \tag{7.64}$$

$$\mathbf{q}_k = \nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{k+1}) \tag{7.65}$$

Then the *Brodyen Family of Updates* for the inverse-Hessian approximation is:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{p}_k\mathbf{p}_k^T}{\mathbf{p}_k^T\mathbf{q}_k} - \frac{\mathbf{B}_k\mathbf{q}_k\mathbf{q}_k^T\mathbf{B}_k}{\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k} + \phi_k\tau_k\mathbf{v}_k\mathbf{v}_k^T \tag{7.66}$$

where:

$$\mathbf{v}_k = \frac{\mathbf{p}_k}{\mathbf{p}_k^T\mathbf{q}_k} - \frac{\mathbf{B}_k\mathbf{q}_k}{\tau_k} \tag{7.67}$$

$$\tau_k = \mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k \tag{7.68}$$

$$0 \leq \phi_k \leq 1 \tag{7.69}$$

and $\mathbf{B}_0 \in \mathbb{R}^{n\times n}$ is a symmetric positive definite matrix (usually $\mathbf{I}_n$).

Exercise 50. Verify that when:

$$\mathbf{T}_k = \phi_k\tau_k\mathbf{v}_k\mathbf{v}_k^T \tag{7.70}$$

then $\mathbf{T}_k\mathbf{q}_j = 0$ for $j = 0, \dots, k$.

Definition 7.12 (BFGS Update). If $\phi_k = 1$ for all k, then Equation 7.66 is called the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update.

Remark 7.13. We state, but do not prove the following proposition, which can be verified by (extensive) matrix arithmetic.

PROPOSITION 7.14. *The BFGS Update is given by:*

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \left(1 + \frac{\mathbf{q}_k^T\mathbf{B}_k\mathbf{q}_k}{\mathbf{p}_k^T\mathbf{q}_k}\right)\frac{\mathbf{p}_k\mathbf{p}_k^T}{\mathbf{p}_k^T\mathbf{q}_k} - \frac{\mathbf{B}_k\mathbf{q}_k\mathbf{p}_k^T + \mathbf{p}_k\mathbf{q}_k^T\mathbf{B}_k}{\mathbf{p}_k^T\mathbf{q}_k} \tag{7.71}$$

Thus if $\mathbf{C}_B(\phi_k)$ is the Broyden family of updates correction matrix, $\mathbf{C}_{DFP}$ is the DFP correction matrix and $\mathbf{C}_{BFGS}$ is the BFGS correction matrix, then:

$$\mathbf{C}_B = \phi_k\mathbf{C}_{DFP} + (1-\phi_k)\mathbf{C}_{BFGS} \tag{7.72}$$

□

EXERCISE 51. Prove Proposition 7.14.

REMARK 7.15. The following lemma and theorem are proved in precisely the same way as the corresponding results for the DFP method, with appropriate allowances made for the extra term $\mathbf{T}_k = \phi_k\tau_k\mathbf{v}_k\mathbf{v}_k^T$.

LEMMA 7.16. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{x}_0 \in \mathbb{R}^n$ and $\mathbf{B}_0 \in \mathbb{R}^{n\times n}$ is a symmetric and positive definite matrix and $\mathbf{B}_1, \dots, \mathbf{B}_n$ are generated using Equation 7.66 and $\mathbf{x}_1, \dots, \mathbf{x}_n$ are generated by:*

$$\mathbf{r}_k = \mathbf{B}_k\nabla f(\mathbf{x}_k) \tag{7.73}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k\mathbf{r}_k \tag{7.74}$$

with $\delta_k = \arg\max f(\mathbf{x}_k + \delta_k\mathbf{r}_k)$. If $\nabla f(\mathbf{x}_k) \neq \mathbf{0}$ for $k = 0, \dots, n$ then $\mathbf{B}_1, \dots \mathbf{B}_n$ are symmetric and positive definite. Thus, $\mathbf{r}_0, \dots, \mathbf{r}_{n-1}$ are ascent directions. □

THEOREM 7.17. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ with:*

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{b}^T\mathbf{x} \tag{7.75}$$

where $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is symmetric and negative definite. Suppose that $\mathbf{B}_1, \dots, \mathbf{B}_{n-1}$ are generated using the Broyden family of updates formulation (Equation 7.66) with $\mathbf{x}_0 \in \mathbb{R}^n$ and $\mathbf{B}_0 \in \mathbb{R}^{n\times n}$, symmetric and positive definite. If $\mathbf{x}_1, \dots, \mathbf{x}_n$ and $\mathbf{r}_0, \dots, \mathbf{r}_{n-1}$ are generated by:

$$\mathbf{r}_k = \mathbf{B}_k\nabla f(\mathbf{x}_k) \tag{7.76}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k\mathbf{r}_k \tag{7.77}$$

with $\delta_k = \arg\max f(\mathbf{x}_k + \delta_k\mathbf{r}_k)$ and

$$\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k = \delta_k\mathbf{r}_k \tag{7.78}$$

$$\mathbf{q}_k = \nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{k+1}) \tag{7.79}$$

then:

1. $\mathbf{p}_0, \dots, \mathbf{p}_{n-1}$ *are $\mathbf{Q}$ conjugate and*
2. $\mathbf{B}_{k+1}\mathbf{Q}\mathbf{p}_j = -\mathbf{p}_j$ *for $j = 0, \dots, k$*
3. $\mathbf{B}_n = -\mathbf{Q}^{-1}$ *and*
4. $\mathbf{x}_n = \mathbf{x}^*$ *is the global maximum of $f(\mathbf{x})$.*

□

Theorem 7.18. *The sequence of iterates $\{\mathbf{x}_k\}$ generated by any quasi-Newton algorithm in the Broyden family when applied to a maximization problem for a quadratic function:*

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{b}^T\mathbf{x}$$

where $\mathbf{Q}$ is negative definite is identical to the sequence generated by the preconditioned conjugate gradient method with scaling matrix $\mathbf{B}_0$.

Sketch of Proof. It is sufficient to show that $\mathbf{x}_{k+1}$ maximizes $f(\mathbf{x})$ for the subspace:

$$\mathcal{M}^k = \{\mathbf{x} : \mathbf{x} = \mathbf{x}_0 + \alpha_0\mathbf{B}_0\nabla f(\mathbf{x}_0) + \cdots + \alpha_k\mathbf{B}_0\nabla f(\mathbf{x}_k), \delta_0, \dots, \alpha_0, \dots, \alpha_k \in \mathbb{R}\} \tag{7.80}$$

This can be proved when $\mathbf{B}_0 = \mathbf{I}_n$ by applying induction to that that for all k there are scalars β_{ij}^k such that:

$$\mathbf{B}_k = \mathbf{I}_n + \sum_{i=0}^{k}\sum_{j=0}^{k}\beta_{ij}^k\nabla f(\mathbf{x}_i)\nabla f(\mathbf{x}_j)^T \tag{7.81}$$

Therefore we can write:

$$\mathbf{p}_k = \mathbf{B}_k\nabla f(\mathbf{x}_k) = \sum_{i=0}^{k} b_i^k\nabla f(\mathbf{x}_i) \tag{7.82}$$

for some scalars $b_i^k \in \mathbb{R}$. Thus for all i, $\mathbf{x}_{i+1}$ lies in $\mathcal{M}^i$. By Theorem 7.17, we know that any quasi-Newton method in the Broyden family is a conjugate gradient method and therefore, $\mathbf{x}_{k+1}$ maximize $f(\mathbf{x})$ over $\mathcal{M}^k$ and by the nature of $f(\mathbf{x})$ this maximum must be unique. To see this for the case when $\mathbf{B}_0 \neq \mathbf{I}_n$, note that we can simply transform $\mathbf{Q}$ to a space in which it is $-\mathbf{I}_n$ as in the preconditioned conjugate gradient method. (See Example 6.23). □

Remark 7.19. This explains why Figure 7.1 is identical to Figure 6.1 and why we will see the same figure again when we apply the BFGS method to $F(x, y) = (x^2 + 3\,y^2)\,e^{1-x^2-y^2}$.

5. Implementation of the BFGS Method

Remark 7.20. The only reason that users prefer the BFGS method to the DFP method is that the BFGS method has better convergence properties in practice than the DFP method, which tends to produce singular matrices in a wider range of optimization problems. The BFGS method seems to be more stable.

Remark 7.21. We note also that the ϕ_k can be varied from iteration to iteration. This is not recommended (in fact, we usually either set it to be 0 or 1) since it can be shown that for each iteration there is a specific ϕ_k that will generate a singular matrix at iteration $k + 1$ and it is better to not accidentally cause this to occur.

Remark 7.22. The BFGS method is implemented by replacing Line 70 - 71 of Algorithm 14 with the line:

```
#Update the B matrix B[k]
B := B +
(1 + (Transpose(q).B.q)/(Transpose(p).q)) * (p.Transpose(p))/(Transpose(p).q) -
(B.q.Transpose(p) + p.Transpose(q).B)/(Transpose(p).q)
```

Exercise 52. Show that the steps taken in Example 7.6 are identical for the BFGS method.

Example 7.23. We can apply the BFGS method to:

$$F(x, y) = \left(x^2 + 3\,y^2\right) \mathrm{e}^{1-x^2-y^2}$$

We obtain convergence in four iterations when we choose $\epsilon = 0.001$ and begin at $x = 1$, $y = 0.1$ (as with the DFP method). This is illustrated in Figure 7.2. Notice that the steps

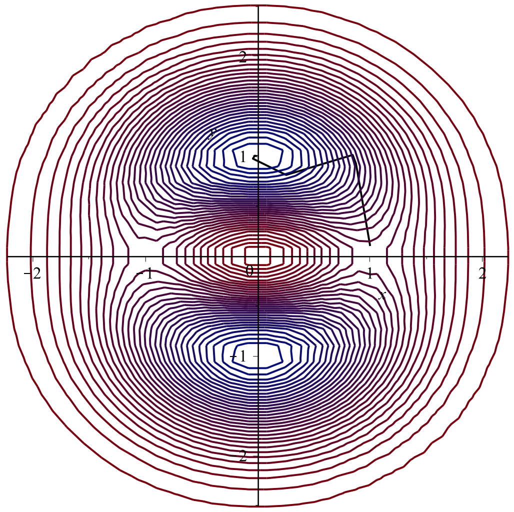


Figure 7.2. The steps of the DFP algorithm applied to $F(x, y)$.

are identical to those shown in the conjugate gradient method (see Figure 6.1) and the DFP method.

Remark 7.24. There are several extensions of the BFGS method that are popular. The simplest is the limited memory BFGS (LBFGS) method [NL89], which does not require as much storage as the BFGS method since $\mathbf{B}_k$ is never stored explicitly, but approximated from the previous m values of $\mathbf{p}_k$ and $\mathbf{q}_k$, where $m < n$. This makes it possible to apply the BFGS method to problems with many variables. The LBFGS method can also be extended for simple boundary constraints, in the LBGS with bounding (LBFGSB) method [ZBLN97]. The code for these algorithms is available freely and often incorporated into popular commercial software. For example, Matlab uses a variation on BFGS for its `fmin` function. BFGS is also implement in the GNU Scientific Library, which is available for free.

Remark 7.25. It is worth noting that **implementations** of the DFP and BFGS methods may produce singular or even negative definite matrices as a result of round-off errors. A check can be placed in the code (using the Cholesky decomposition) to determine this. When this happens, one can re-initialize $\mathbf{B}_k$ to $\mathbf{I}_n$ (e.g.) and continue with execution.

Exercise 53. Implement the BFGS algorithm with a check for singularity. Use it on Rosenbrock's Function:

$$-(1 - x)^2 + 100(y - x^2)^2 \tag{7.83}$$

start at point $(0, 0)$.

CHAPTER 8

Numerical Differentiation and Derivative Free Optimization

The topics contained in this chapter are actually important and each one could easily deserve a chapter on its own. Unfortunately, in order to get on to constrained problems, we must cut somewhere and this is as reasonable a place to do it as any. If this were a real book, each of these sections would have its own chapter. Therefore, the concerned reader should see [**NW06**] for details.

1. Numerical Differentiation

Remark 8.1. Occasionally, it is non-trivial to compute a derivative of a function. In such a case, a numerical derivative can be used to compute an approximation of the gradient of the function at that point.

Definition 8.2 (Forward Difference Approximation). Let $f : \mathbb{R}^n \to \mathbb{R}$. The *forward difference approximation* is given by the formula:

$$\frac{\partial f(\mathbf{x}_0)}{\partial x_i} \approx \frac{\delta f(\mathbf{x}_0)}{\delta \mathbf{x}_i} = \frac{f(\mathbf{x}_0 + \epsilon \mathbf{e}_i) - f(\mathbf{x}_0)}{\epsilon} \tag{8.1}$$

here $\epsilon > 0$ is a tiny value and $\mathbf{e}_i$ is the i^{th} standard basis vector for $\mathbb{R}^n$.

Remark 8.3. Note that by Taylor's Theorem (in one dimension):

$$f(\mathbf{x}_0 + \epsilon \mathbf{e}_i) = f(\mathbf{x}_0) + \epsilon \frac{\partial f(\mathbf{x}_0)}{\partial x_i} + O(\epsilon^2) \tag{8.2}$$

Thus:

$$\frac{\delta f(\mathbf{x}_0)}{\delta \mathbf{x}_i} = \frac{\partial f(\mathbf{x}_0)}{\partial x_i} + O(\epsilon) \tag{8.3}$$

We know that $\delta \mathbf{x}_i = \epsilon$. Assuming a floating point error of h when evaluating $\delta f(\mathbf{x}_0)$ we see that:

$$\frac{h}{\epsilon} = O(\epsilon) \tag{8.4}$$

or $h = O(\epsilon^2)$. If the machine precision for a given system is $h_{\min}$, then setting ϵ any *smaller* than $\sqrt{h_{\min}}$ could lead to numerical instability. Thus, a useful estimate is $\epsilon \geq \sqrt{h_{\min}}$. Note that, to a point, the smaller ϵ, the better the estimate of the derivative.

Definition 8.4 (Central Difference Approximation). Let $f : \mathbb{R}^n \to \mathbb{R}$. The *central difference approximation* is given by the formula:

$$\frac{\partial f(\mathbf{x}_0)}{\partial x_i} \approx \frac{\Delta f(\mathbf{x}_0)}{\Delta \mathbf{x}_i} = \frac{f(\mathbf{x}_0 + \epsilon \mathbf{e}_i) - f(\mathbf{x}_0 - \epsilon \mathbf{e}_i)}{2\epsilon} \tag{8.5}$$

here $\epsilon > 0$ is a tiny value and $\mathbf{e}_i$ is the i^{th} standard basis vector for $\mathbb{R}^n$.

Remark 8.5. Note that by Taylor's Theorem (in one dimension):

$$f(\mathbf{x}_0 - \epsilon\mathbf{e}_i) = f(\mathbf{x}_0) - \epsilon\frac{\partial f(\mathbf{x}_0)}{\partial x_i} + \frac{1}{2}\epsilon^2\frac{\partial^2 f(\mathbf{x}_0)}{\partial x_i^2} + O(\epsilon^3) \tag{8.6}$$

A similar expression holds for $f(\mathbf{x}_0 + \epsilon\mathbf{e}_i)$. Thus, we can see that:

$$f(\mathbf{x}_0 + \epsilon\mathbf{e}_i) - f(\mathbf{x}_0 - \epsilon\mathbf{e}_i) = 2\epsilon\frac{\partial f(\mathbf{x}_0)}{\partial x_i} + O(\epsilon^3) \tag{8.7}$$

because that quadratic terms (of identical sign) will cancel out. The result is:

$$\frac{f(\mathbf{x}_0 + \epsilon\mathbf{e}_i) - f(\mathbf{x}_0 - \epsilon\mathbf{e}_i)}{2\epsilon} = \frac{\partial f(\mathbf{x}_0)}{\partial x_i} + O(\epsilon^2) \tag{8.8}$$

Using a similar argument to the one above we have: $h = O(\epsilon^3)$. This yields a rule for choosing ϵ when using the central difference approximation: $\epsilon \geq \sqrt[3]{h_{\min}}$. We also note that the central difference formula is substantially more accurate (an order of magnitude in ϵ) than the forward difference formula, but requires two functional evaluations rather than one.

Remark 8.6. From the previous observations, one might wonder when it is best to use the forward difference formula and when it is best to use the central difference formula. Bertsekas [Ber99] provides the practical advice that when the approximated derivative(s) are not very close to zero, then the forward difference is fine. Once the derivative begins to approach zero (and the error may swamp the computation), then a transition to the central difference is called for.

Remark 8.7. Code for computing a numerical gradient is shown in Algorithm 17. It uses the Forward Differencing Method shown in Algorithm 15 and the Central Differencing Method shown in Algorithm 16.

Exercise 54. Re-implement the BFGS algorithm using a finite difference approximation of the gradient. Compare your results to your original implementation on the function:

$$F(x,y) = \left(x^2 + 3\,y^2\right)\mathrm{e}^{1-x^2-y^2}$$

Remark 8.8. In a similar fashion, we can compute the numerical hessian approximation. If $\mathbf{H}(\mathbf{x}_0) = \nabla^2 f(\mathbf{x}_0)$, then:

$$\mathbf{H}_{i,j} = \frac{\nabla f(\mathbf{x}_0 + \epsilon\mathbf{e}_j)_i - \nabla f(\mathbf{x}_0)_i}{\epsilon} \tag{8.9}$$

Here: $\nabla f(\mathbf{x}_0)_i$ is the i^{th} component of the gradient vector.

Exercise 55. Implement a numerical Hessian computation algorithm and illustrate its effect on Newton's method.

Example 8.9. Numerical gradients have little impact on convergence of quasi-Newton, conjugate gradient (or even modified Newton's method) for well behaved functions. This is illustrated in Figure 8.1. In this figure, we compare a BFGS run using numerical gradients with a BFGS run using exact gradients. The dashed black line uses symbolic gradients, while the solid red line uses numerical gradients. As you can see there is very little difference detectable by inspection.

You may face a time penalty for functions that are difficult to evaluate, (because you are introducing additional functional evaluations). This is a practical issue that you must deal with.

```
NumericalGradientF := proc (f::operator, xnow::list, epsilon::numeric)::Vector;
  local G, fnow, i, x;

  #Initialize a list to store output.
  G := [];

  #Passed in variables are non-modifiable in Maple.
  x := xnow;

  #We'll use this over and over, don't keep computing it.
  fnow := f(op(xnow));

  #For each index...
  for i to nops(xnow) do

    #Perturb x[i].
    x[i] := x[i]+epsilon;

    #Store the forward difference in G.
    G := [op(G), (f(op(x))-fnow)/epsilon];

    #Unperturb x[i].
    x[i] := x[i]-epsilon
  end do;

  #Return a gradient vector.
  Vector(G)
end proc:
```

Algorithm 15. Forward Differencing Method for Numerical Gradients

2. Derivative Free Methods: Powell's Method

REMARK 8.10. We will discuss two derivative free methods for finding optimal points of functions. The first, Powell's method, is a conjugate direction method, while the second, the Hooke-Jeeves algorithm, is a type of optimization method called a pattern search. We will not prove convergence properties of either method, though as a conjugate direction method, Powell's method will have all the properties one should expect when applying it to a concave quadratic function.

REMARK 8.11. Given a function $f : \mathbb{R}^n \to \mathbb{R}$, Powell's *basic* method works in the following way:

(1) Choose a starting position $\mathbf{x}_0$ and n directions $\mathbf{p}_0, \dots, \mathbf{p}_{n-1}$.
(2) For each $k = 0, \dots, n-1$, let $\delta_k = \arg\max f(\mathbf{x}_k + \delta_k \mathbf{p}_n)$ and let $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k \mathbf{p}_k$.
(3) For each $k = 0, \dots, n-2$, set $\mathbf{p}_k = \mathbf{p}_{k+1}$.
(4) Set $\mathbf{p}_{n-1} = \mathbf{x}_n - \mathbf{x}_0$.
(5) If $||\mathbf{x}_n - \mathbf{x}_0|| < \epsilon$, stop.

```
NumericalGradientF := proc (f::operator, xnow::list, epsilon::numeric)::Vector;
  local G, fnow, i, x;

  #Initialize a list to store output.
  G := [];

  #Passed in variables are non-modifiable in Maple.
  x := xnow;

  #For each index...
  for i to nops(xnow) do

    #Perturn x[i] forward and store the function value.
    x[i] := x[i]+epsilon;
    f1 := f(op(x));

    #Perturb x[i] backward and store the function value
    x[i] := x[i]-2*epsilon;
    f2 := f(op(x));

    #Store the central difference in G.
    G := [op(G), (f1-f2)/(2*epsilon)];

    #Unperturn x[i]
    x[i] := x[i]+epsilon
  end do;

  #Return a gradient vector.
  Vector(G)
end proc:
```

Algorithm 16. Central Differencing Method for Numerical Gradients

(6) Solve $\delta_n = \arg\max f(\mathbf{x}_n + \delta_n \mathbf{p}_n)$. Set $\mathbf{x}_0 = \mathbf{x}_n + \delta_n \mathbf{p}_n$. Goto (1).

Usually the directions $\mathbf{p}_0, \dots, \mathbf{p}_{n-1}$ are initialized as the standard basis vectors.

REMARK 8.12. One problem with Powell's method is that the directions may tend to become linearly dependent at the directions tend (more or less) toward a gradient, especially in valleys (like in Rosenbrock's function). There are a few ways to solve this problem.

(1) Reinitialization of the basic directions every n or $n + 1$ iterations of the basic algorithm, (once all the original directions are gone).
(2) You can reset the directions to any set of orthogonal directions, at the expense of loosing information on the directions you've already constructed, or construct a special set of orthogonal directions (say following the Gram-Schmidt procedure), but that assumes that you have some special knowledge about the function (e.g., that it is quadratic, which obviates the need for derivative free methods).

```
NumericalGradient := proc (f::operator, xnow::list, epsilon::numeric)::Vector;
  local G;

  #Compute the forward difference approximation.
  G := NumericalGradientF(f, xnow, epsilon);

  #Test to see how small the norm of G is. (You can supply your own rule.)
  if Norm(G) <= 2*epsilon then
    G := NumericalGradientC(f, xnow, epsilon)
  end if;

  #Return the vector.
  G
end proc:
```

Algorithm 17. A simple routine to compute the numerical gradient using forward and central differencing.

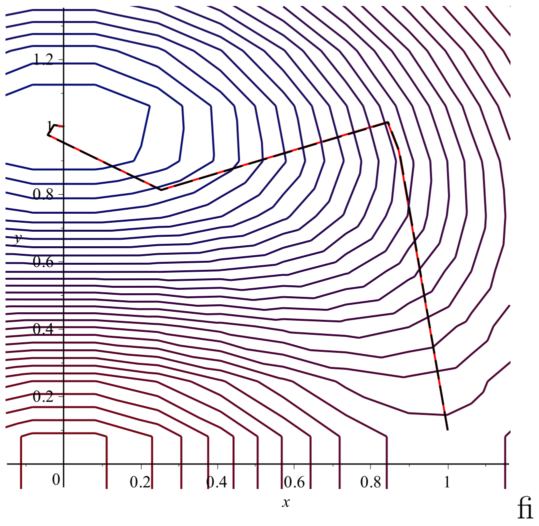


Figure 8.1. A comparison of the BFGS method using numerical gradients vs. exact gradients.

(3) You can use a heuristic, provided by Powell.

Remark 8.13 (Powell's Heuristic). Let:

$$f_0 = f(\mathbf{x}_0) \tag{8.10}$$

$$f_n = f(\mathbf{x}_n) \tag{8.11}$$

$$f_E = f(2\mathbf{x}_n - x_0) = f(x_n + (x_n - x_0)) \tag{8.12}$$

Here f_E is the value of $f(\mathbf{x})$ when move x_n as far as possible following direction $\mathbf{x}_n - \mathbf{x}_0$. Finally, define Δf to be the magnitude of the largest decrease only any direction encountered in Line 2.

Powell then provides the following rules:

(1) $f_E \leq f_0$, then keep the old set of directions for the next procedural iteration. That is, *do not* execute Steps 3,4, or 6, simply check for convergence and return to Step 1 with $\mathbf{x}_0 = \mathbf{x}_n$.
(2) If $2(2f_n + f_E - f_0)((f_n - f_0) - \Delta_f)^2 \geq (f_E - f_0)^2 \Delta f$, then keep the old set of directions for the next procedural iteration. That is, *do not* execute Steps 3,4, or 6, simply check for convergence and return to Step 1 with $\mathbf{x}_0 = \mathbf{x}_n$.
(3) If neither of these situations occur, replace the direction of greatest increase with $\mathbf{x}_n - \mathbf{x}_0$.

A reference implementation for this algorithm is shown in Algorithm 18. We remark the `BasisVector` function called within this algorithm is a custom function that returns the i^{th} basis vector. It is not part of the Maple basline.

Example 8.14. We illustrate two examples of Powell's Method, the first on:

$$F(x, y) = \left(x^2 + 3\,y^2\right) \mathrm{e}^{1-x^2-y^2}$$

and the second on a variation of Rosenbrock's Function:

$$G(x, y) = -(1-x)^2 - 100\left(y - x^2\right)^2$$

In Figure 8.2, we draw the reader's attention to the effect the valley has on the steps in

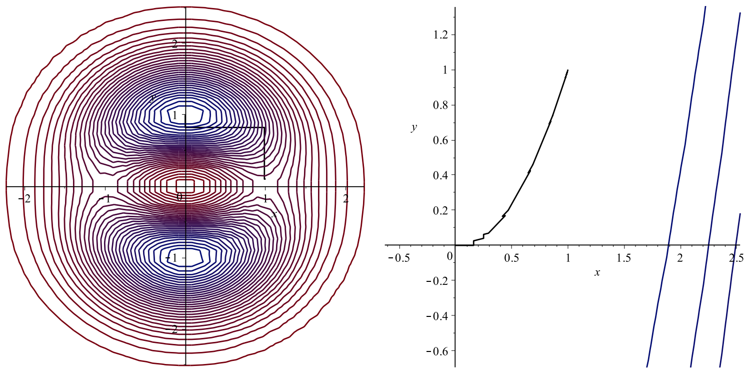


Figure 8.2. Powell's Direction Set Method applied to a bimodal function and a variation of Rosenbrock's function. Notice the impact the valley has on the steps in Rosenbrock's method.

Powell's method. While each step individually is an ascent, the zig-zagging pattern seems almost random.

3. Derivative Free Methods: Hooke-Jeeves Method

Remark 8.15. The method of Hooke and Jeeves is a pattern search method, meaning it creates a search pattern and the replicates that pattern expanding or contracting it when possible. The original discrete form has no proof of convergence [**HJ61**]. Bazarra et al. present a version of Hooke and Jeeves algorithm using line search, which they claim has convergence properties [**BSS06**].

```
PowellsMethod := proc (F::operator, initVals::list, epsilon::numeric,
  maxIter::integer)::list;
  local vars, xnow, i, P, L, vals, xdiff, r, OUT, passIn, phi, ttemp,
    xnext, p, count, Df, x0, Nr, f0, fN, fE;

  vars := [];
  xnow := [];
  vals := initVals;
  for i to nops(initVals) do
    vars := [op(vars), lhs(initVals[i])];
    xnow := [op(xnow), rhs(initVals[i])]
  end do;

  xdiff := 1;
  count := 0;
  OUT := [];
  L := [];
  bestIndex = -1;

  for i to nops(vals) do
    L := [op(L), BasisVector(i, nops(vals))]
  end do;

  while epsilon < xdiff and count < maxIter do
    count := count + 1;
    Df := 0;
    x0 := xnow;

    for i to nops(L) do
      OUT := [op(OUT), xnow];
      p := L[i];
      passIn := convert(Vector(xnow) + s * p), list);

      phi := proc (t) options operator, arrow;
        evalf(VectorCalculus:-eval(F(op(passIn)), s = t))
      end proc;

      ttemp := LineSearch(phi);

      if ttemp < epsilon then
        passIn := convert(Vector(xnow) - s * p), list);

        phi := proc (t) options operator, arrow;
          evalf(VectorCalculus:-eval(F(op(passIn)), s = t))
        end proc;
        ttemp := LineSearch(phi)
      end if;
      xnext := evalf(eval(passIn, [s = ttemp]));
      r := xnext - xnow;
      Nr := Norm(Vector(r));

      if Df < Nr then
        Df := Nr
        bestIndex = i:
      end if;

      xnow := xnext
    end do;
    P := xnow - x0;
    f0 := evalf(F(op(x0)));
    fN := evalf(F(op(xnow)));
    fE := evalf(F(op(2 * xnow - x0)));

    if not (evalb(fE <= f0) or
      evalb(evalf(2 * (2*fN+fE-f0)*((fN-f0)-Df)^2) >= evalf((fE-f0)^2*Df))) then
      L[bestIndex] := Vector(P):
    end if;
    xdiff := Norm(Vector(P)):
  end do;
  OUT
end proc
```

Algorithm 18. Powell's derivative free method of optimization with heuristic.

Remark 8.16. The method of Hooke and Jeeves attempts to find a movement pattern that increases the functional value and then repeats this pattern until there is evidence this is no longer working. Assume $f : \mathbb{R}^n \to \mathbb{R}$. Like Powell's method, we begin with a set of (orthogonal) directions $\mathbf{p}_0, \dots, \mathbf{p}_{n-1}$, and scalars $\epsilon > 0$, $\Delta \geq \epsilon$ and $\alpha \in [0, 1]$. We summarize the method as:

(1) Given a starting point $\mathbf{x}_0 \in \mathbb{R}^n$, set $\mathbf{y}_0 = \mathbf{x}_0$. Set $j = k = 0$.
(2) For each $j = 0, \dots, n-1$,
   (a) if $f(\mathbf{y}_j + \Delta\mathbf{p}_j) > f(\mathbf{y}_j)$, the trial is a success and $\mathbf{y}_{j+1} = \mathbf{y}_j + \Delta\mathbf{p}_j$.
   (b) Otherwise, the trial is a failure and if so, if $f(\mathbf{y}_j - \Delta\mathbf{p}_j) > f(\mathbf{y}_j)$ the new trial is a success and $\mathbf{y}_{j+1} = \mathbf{y}_j - \Delta\mathbf{p}_j$.
   (c) Otherwise $\mathbf{y}_{j+1} = \mathbf{y}_j$.
(3) If $f(\mathbf{y}_n) > f(\mathbf{x}_k)$, then set $\mathbf{x}_{k+1} = \mathbf{y}_n$, set $\mathbf{y}_0 = \mathbf{x}_{k+1} + \alpha(\mathbf{x}_{k+1} - \mathbf{x}_k)$. Goto (2).
(4) Otherwise: If $\Delta < \epsilon$ terminate. If $\Delta \geq \epsilon$, set $\Delta = \Delta/2$. Let $\mathbf{x}_{k+1} = \mathbf{x}_k$. Let $\mathbf{y}_0 = \mathbf{x}_k$. Goto (2).

Remark 8.17. The Hooke-Jeeves pattern search essentially gropes around in each (cardinal) direction attempting to improve the value of $f(\mathbf{x})$ as it goes. When it finds a point where improvement is not possible, it shrinks the distance it looks and tries again, until it reaches a stopping point. When $\mathbf{y}_0 = \mathbf{x}_{k+1} + \alpha(\mathbf{x}_{k+1} - \mathbf{x}_k)$, the algorithm is attempting to use the pattern of improvement just identified to further move $f(\mathbf{x})$ uphill. A reference implementation for Hooke-Jeeves is shown in Algorithm 19.

Example 8.18. We can apply the Hooke-Jeeves algorithm to our bimodal function:

$$F(x,y) = \left(x^2 + 3\,y^2\right) \mathrm{e}^{1-x^2-y^2}$$

When we start at $x_0 = 1$, $y_0 = 0.1$ and $\alpha = 0.5$, $\Delta = 1$, we get good convergence, illustrated in Figure 8.3

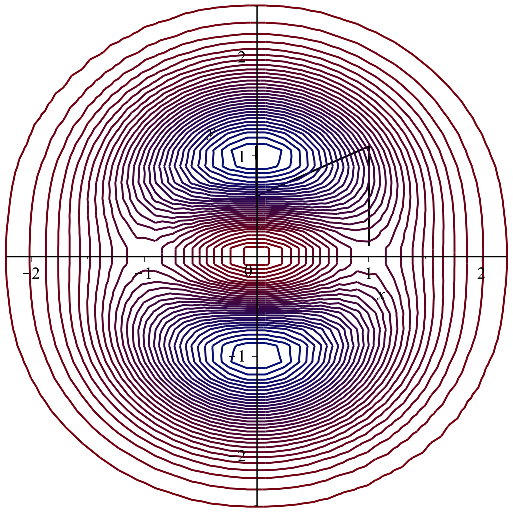


Figure 8.3. Hooke-Jeeves algorithm applied to a bimodal function.

If we apply Hooke-Jeeves to a variation of Rosenbrock's Function:

$$G(x,y) = -(1-x)^2 - 100\left(y - x^2\right)^2$$

we see different behavior. For $\Delta = 1$ and $\alpha = 0.5$, we fail to converge to the optimal point $(1,1)$. If we adjust α to be 1, we do converge in 28 steps. This is shown in Figure 8.4. Thus we can see that the Hooke-Jeeves method may be very sensitive to the parameters supplied and thus it should only be used on functions where evaluation is very easy since we may have to try a few parameter combinations before we identify a true maximum.

```
HookeJeeves := proc (F::operator, initVals::list, epsilon::numeric, DeltaIn::numeric,
  alpha::numeric)::list;

  local i, OUT, vars, xnow, L, vals, Delta, p, xlast, ynow, ytest;
  Delta := DeltaIn;

  vars := [];
  xnow := [];
  vals := initVals;
  for i to nops(initVals) do
    vars := [op(vars), lhs(initVals[i])];
    xnow := [op(xnow), rhs(initVals[i])]
  end do;

  OUT := [xnow];
  L := [];
  for i to nops(vals) do
    L := [op(L), BasisVector(i, nops(vals))]
  end do;

  ynow := xnow;

  while epsilon < Delta do
    for i to nops(L) do
      p := L[i];
      ytest := convert(Vector(ynow)+Delta*p, list);

      if F(op(ynow)) < F(op(ytest)) then
        ynow := ytest
      else ytest := convert(Vector(ynow)-Delta*p, list);
        if F(op(ynow)) < F(op(ytest)) then
          ynow := ytest
        end if
      end if
    end do;

    if F(op(xnow)) < F(op(ynow)) then
      xlast := xnow;
      xnow := ynow;
      ynow := xnow+alpha*(xnow-xlast);
      OUT := [op(OUT), xnow]
    else
      Delta := (1/2)*Delta;
      ynow := xnow
    end if
  end do;
  OUT:
end proc
```

Algorithm 19. Hooke-Jeeves derivative free algorithm.

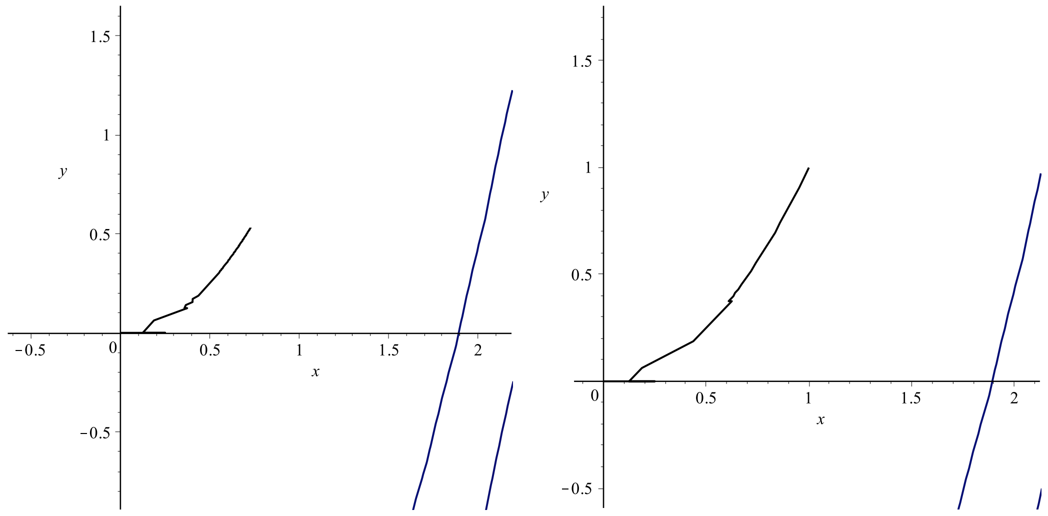


Figure 8.4. Hooke-Jeeves algorithm applied to a bimodal function.

CHAPTER 9

Linear Programming: The Simplex Method

The material covered in this chapter is a summary of my Math 484 lecture notes [Gri11], which go into more detail (and have proofs). If you're interested, I suggest you look there or get a good book on Linear Programming like [BJS04]. Also, under no circumstances do I recommend you implement the simplex method. There are too many "nits" for implementing an efficient algorithm. Instead, you should check out the GNU Linear Programming Kit (http://www.gnu.org/software/glpk/). It's very well implemented and free (in every sense of the word).

1. Linear Programming: Notation

Definition 9.1 (Linear Programming Problem). A *linear programming* problem is an optimization problem of the form:

$$(9.1)\qquad \begin{cases} \max\ z(x_1,\dots,x_n) = c_1x_1 + \cdots + c_nx_n \\ s.t.\ a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1 \\ \quad\vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m \\ h_{11}x_1 + \cdots + h_{n1}x_n = r_1 \\ \quad\vdots \\ h_{l1}x_1 + \cdots + h_{ln}x_n = r_l \end{cases}$$

Remark 9.2. You will recall from your matrices class (Math 220) that matrices can be used as a short hand way to represent linear equations. Consider the following system of equations:

$$(9.2)\qquad \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \qquad\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$

Then we can write this in matrix notation as:

$$(9.3)\qquad \mathbf{Ax} = \mathbf{b}$$

where $\mathbf{A}_{ij} = a_{ij}$ for $i = 1,\dots,m$, $j = 1,\dots,n$ and $\mathbf{x}$ is a column vector in $\mathbb{R}^n$ with entries x_j ($j = 1,\dots,n$) and $\mathbf{b}$ is a column vector in $\mathbb{R}^m$ with entries b_i ($i = 1\dots,m$). Obviously, if we replace the equalities in Expression 9.2 with inequalities, we can also express systems of inequalities in the form:

$$(9.4)\qquad \mathbf{Ax} \leq \mathbf{b}$$

Using this representation, we can write our general linear programming problem using matrix and vector notation. Expression 9.1 can be written as:

$$\left\{\begin{aligned} \max \quad z(\mathbf{x}) =& \mathbf{c}^T\mathbf{x} \\ s.t. \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{H}\mathbf{x} = \mathbf{r} \end{aligned}\right. \tag{9.5}$$

Definition 9.3. In Problem 9.5, if we restrict some of the decision variables (the x_i's) to have only integer (or discrete) values, then the problem becomes a mixed integer linear programming problem. If all of the variables are restricted to integer values, the problem is an integer programming problem and if every variable can only take on the values 0 or 1, the program is called a $0-1$ integer programming problem. [**WN99**] is an excellent reference for Integer Programming.

Definition 9.4 (Canonical Form). A maximization linear programming problem is in *canonical form* if it is written as:

$$\left\{\begin{aligned} \max \quad z(\mathbf{x}) =& \mathbf{c}^T\mathbf{x} \\ s.t. \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}\right. \tag{9.6}$$

A minimization linear programming problem is in *canonical form* if it is written as:

$$\left\{\begin{aligned} \min \quad z(\mathbf{x}) =& \mathbf{c}^T\mathbf{x} \\ s.t. \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}\right. \tag{9.7}$$

Definition 9.5 (Standard Form). A linear programming problem is in *standard form* if it is written as:

$$\left\{\begin{aligned} \max \quad z(\mathbf{x}) =& \mathbf{c}^T\mathbf{x} \\ s.t. \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}\right. \tag{9.8}$$

Remark 9.6. The following theorem is outside the scope of the course. You may cover it in a Math 484 [**Gri11**].

Theorem 9.7. *Every linear programming problem in canonical form can be put into standard form.* □

Exercise 56. Show that a minimization linear programming problem in canonical form can be rephrased as a maximization linear programming problem in canonical form. [Hint: Multiply the objective and constraints -1. Define new matrices.]

Remark 9.8. To illustrate Theorem 9.7, we note that it is relatively easy to convert any inequality constraint into an equality constraint. Consider the inequality constraint:

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i \tag{9.9}$$

We can add a new *slack variable* s_i to this constraint to obtain:

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n + s_i = b_i$$

Obviously this slack variable $s_i \geq 0$. The slack variable then becomes just another variable whose value we must discover as we solve the linear program for which Expression 9.9 is a constraint.

We can deal with constraints of the form:

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i \tag{9.10}$$

in a similar way. In this case we subtract a surplus variable s_i to obtain:

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n - s_i = b_i$$

Again, we must have $s_i \geq 0$.

EXAMPLE 9.9. Consider the linear programming problem:

$$\begin{cases} \max & z(x_1, x_2) = 2x_1 - x_2 \\ s.t. & x_1 - x_2 \leq 1 \\ & 2x_1 + x_2 \geq 6 \\ & x_1, x_2 \geq 0 \end{cases}$$

This linear programming problem can be put into standard form by using both a slack and surplus variable.

$$\begin{cases} \max & z(x_1, x_2) = 2x_1 - x_2 \\ s.t. & x_1 - x_2 + s_1 = 1 \\ & 2x_1 + x_2 - s_2 = 6 \\ & x_1, x_2, s_1, s_2 \geq 0 \end{cases}$$

DEFINITION 9.10 (Row Rank). Let $\mathbf{A} \in \mathbb{R}^{m\times n}$. The *row rank* of $\mathbf{A}$ is the size of the largest set of row (vectors) from $\mathbf{A}$ that are linearly independent.

REMARK 9.11. The column rank of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ is defined analogously on columns rather than rows. The following theorem relates the row and column rank. It's proof is outside the scope of the course.

THEOREM 9.12. *If $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a matrix, then the row rank of $\mathbf{A}$ is equal to the column rank of $\mathbf{A}$. Further,* $\text{rank}(\mathbf{A}) \leq \min\{m, n\}$. □

DEFINITION 9.13. Suppose that $\mathbf{A} \in \mathbb{R}^{m\times n}$ and let $m \leq n$. Then $\mathbf{A}$ has *full row rank* if $\text{rank}(\mathbf{A}) = m$.

REMARK 9.14. We will assume, when dealing with Linear Programming Problems in standard or canonical form that the matrix $\mathbf{A}$ has full row rank and if not, we will adjust it so this is true. The following theorem tells us what can happen in a Linear Programming Problem.

2. Polyhedral Theory and Linear Equations and Inequalities

2.1. Solving Systems with More Variables than Equations.

Suppose now that $\mathbf{A} \in \mathbb{R}^{m\times n}$ where $m \leq n$. Let $\mathbf{b} \in \mathbb{R}^m$. Then the equation:

$$\mathbf{Ax} = \mathbf{b} \tag{9.11}$$

has more variables than equations and is underdetermined and if $\mathbf{A}$ has full row rank then the system will have an infinite number of solutions. We can formulate an expression to describe this infinite set of solutions.

Sine $\mathbf{A}$ has full row rank, we may choose any m linearly independent columns of $\mathbf{A}$ corresponding to a subset of the variables, say $x_{i_1}, \dots, x_{i_m}$. We can use these to form the matrix

$$\mathbf{B} = [\mathbf{A}_{\cdot i_1} \cdots \mathbf{A}_{\cdot i_m}] \tag{9.12}$$

from the columns $\mathbf{A}_{\cdot i_1}, \dots, \mathbf{A}_{\cdot i_m}$ of $\mathbf{A}$, so that B is invertible. It should be clear at this point that B will be invertible precisely because we've chosen m linearly independent column vectors. We can then use elementary column operations to write the matrix $\mathbf{A}$ as:

$$\mathbf{A} = [\mathbf{B}|\mathbf{N}] \tag{9.13}$$

The matrix $\mathbf{N}$ is composed of the $n - m$ other columns of $\mathbf{A}$ not in $\mathbf{B}$. We can similarly sub-divide the column vector $\mathbf{x}$ and write:

$$[\mathbf{B}|\mathbf{N}] \begin{bmatrix} \mathbf{x_B} \\ \mathbf{x_N} \end{bmatrix} = \mathbf{b} \tag{9.14}$$

where the vector $\mathbf{x_B}$ are the variables corresponding to the columns in $\mathbf{B}$ and the vector $\mathbf{x_N}$ are the variables corresponding to the columns of the matrix $\mathbf{N}$.

Definition 9.15 (Basic Variables). For historical reasons, the variables in the vector $\mathbf{x_B}$ are called the *basic variables* and the variables in the vector $\mathbf{x_N}$ are called the *non-basic* variables.

We can use matrix multiplication to expand the left hand side of this expression as:

$$\mathbf{B}\mathbf{x_B} + \mathbf{N}\mathbf{x_N} = \mathbf{b} \tag{9.15}$$

The fact that $\mathbf{B}$ is composed of all linearly independent columns implies that applying Gauss-Jordan elimination to it will yield an $m \times m$ identity and thus that $\mathbf{B}$ is invertible. We can solve for basic variables $\mathbf{x_B}$ in terms of the non-basic variables:

$$\mathbf{x_B} = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x_N} \tag{9.16}$$

We can find an arbitrary solution to the system of linear equations by choosing values for the variables the non-basic variables and solving for the basic variable values using Equation 9.16.

Definition 9.16. (Basic Solution) When we assign $\mathbf{x_N} = 0$, the resulting solution for $\mathbf{x}$ is called a *basic solution* and

$$\mathbf{x_B} = \mathbf{B}^{-1}\mathbf{b} \tag{9.17}$$

Example 9.17. Consider the problem:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 8 \end{bmatrix} \tag{9.18}$$

Then we can let $x_3 = 0$ and:

$$\mathbf{B} = \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \tag{9.19}$$

We then solve[1]:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \mathbf{B}^{-1} \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} \frac{-19}{3} \\ \frac{20}{3} \end{bmatrix} \tag{9.20}$$

Other basic solutions could be formed by creating **B** out of columns 1 and 3 or columns 2 and 3.

Exercise 57. Find the two other basic solutions in Example 9.17 corresponding to

$$\mathbf{B} = \begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$$

and

$$\mathbf{B} = \begin{bmatrix} 1 & 3 \\ 4 & 6 \end{bmatrix}$$

In each case, determine what the matrix **N** is. [Hint: Find the solutions any way you like. Make sure you record exactly which x_i ($i \in \{1, 2, 3\}$) is equal to zero in each case.]

2.2. Polyhedral Sets. Important examples of convex sets are polyhedral sets, the multi-dimensional analogs of polygons in the plane. In order to understand these structures, we must first understand hyperplanes and half-spaces.

Definition 9.18 (Hyperplane). Let $\mathbf{a} \in \mathbb{R}^n$ be a constant vector in n-dimensional space and let $b \in \mathbb{R}$ be a constant scalar. The set of points

$$H = \left\{\mathbf{x} \in \mathbb{R}^n | \mathbf{a}^T\mathbf{x} = b\right\} \tag{9.21}$$

is a *hyperplane* in n-dimensional space. *Note the use of column vectors for* **a** *and* **x** *in this definition.*

Example 9.19. Consider the hyper-plane $2x_1+3x_2+x_3 = 5$. This is shown in Figure 9.1. This hyperplane is composed of the set of points $(x_1, x_2, x_3) \in \mathbb{R}^3$ satisfying $2x_1+3x_2+x_3 = 5$. This can be plotted implicitly or explicitly by solving for one of the variables, say x_3. We can write x_3 as a function of the other two variables as:

$$x_3 = 5 - 2x_1 - 3x_2 \tag{9.22}$$

Definition 9.20 (Half-Space). Let $\mathbf{a} \in \mathbb{R}^n$ be a constant vector in n-dimensional space and let $b \in \mathbb{R}$ be a constant scalar. The sets of points

$$H_l = \left\{\mathbf{x} \in \mathbb{R}^n | \mathbf{a}^T\mathbf{x} \leq b\right\} \tag{9.23}$$

$$H_u = \left\{\mathbf{x} \in \mathbb{R}^n | \mathbf{a}^T\mathbf{x} \geq b\right\} \tag{9.24}$$

are the half-spaces defined by the hyperplane $\mathbf{a}^T\mathbf{x} = b$.

Example 9.21. Consider the two dimensional hyperplane (line) $x_1 + x_2 = 1$. Then the two half-spaces associated with this hyper-plane are shown in Figure 9.2. A half-space is so named because the hyperplane $\mathbf{a}^T\mathbf{x} = b$ literally separates $\mathbb{R}^n$ into two halves: the half above the hyperplane and the half below the hyperplane.

[1]Thanks to Doug Mercer, who found a typo below that was fixed.

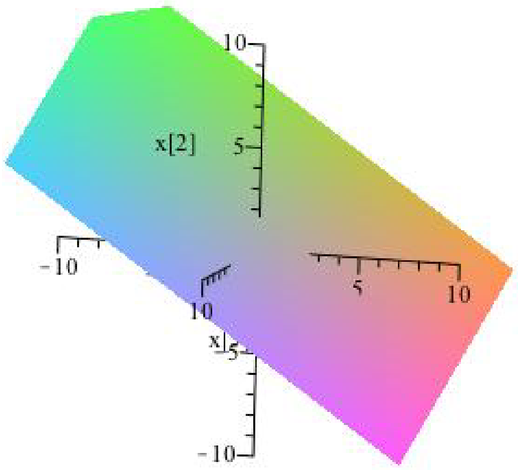
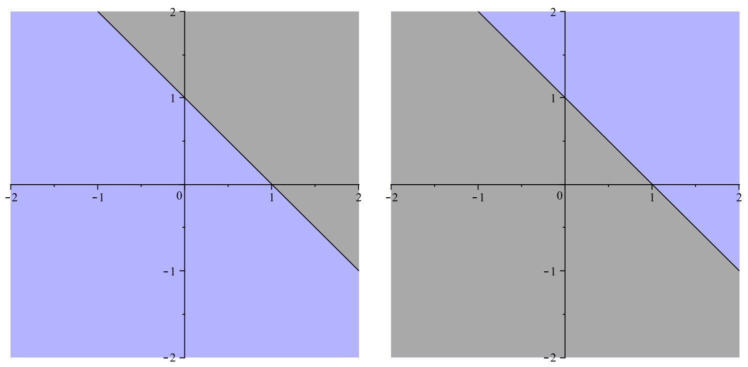


Figure 9.1. A hyperplane in 3 dimensional space: A hyperplane is the set of points satisfying an equation $\mathbf{a}^T\mathbf{x} = b$, where k is a constant in $\mathbb{R}$ and $\mathbf{a}$ is a constant vector in $\mathbb{R}^n$ and $\mathbf{x}$ is a variable vector in $\mathbb{R}^n$. The equation is written as a matrix multiplication using our assumption that all vectors are column vectors.



(a) H_l (b) H_u

Figure 9.2. Two half-spaces defined by a hyper-plane: A half-space is so named because any hyper-plane divides $\mathbb{R}^n$ (the space in which it resides) into two halves, the side "on top" and the side "on the bottom."

Definition 9.22 (Polyhedral Set). If $P \subseteq \mathbb{R}^n$ is the intersection of a finite number of half-spaces, then P is a *polyhedral set*. Formally, let $\mathbf{a}_1, \dots, \mathbf{a}_m \in \mathbb{R}^n$ be a finite set of constant vectors and let $b_1, \dots, b_m \in \mathbb{R}$ be constants. Consider the set of half-spaces:

$$H_i = \{\mathbf{x} | \mathbf{a}_i^T \mathbf{x} \leq b_i\}$$

Then the set:

$$P = \bigcap_{i=1}^{m} H_i \tag{9.25}$$

is a *polyhedral set*.

It should be clear that we can represent any polyhedral set using a matrix inequality. The set P is defined by the set of vectors $\mathbf{x}$ satisfying:

$$\mathbf{A}\mathbf{x} \leq \mathbf{b}, \tag{9.26}$$

where the *rows* of $\mathbf{A} \in \mathbb{R}^{m\times n}$ are made up of the vectors $\mathbf{a}_1, \dots, \mathbf{a}_m$ and $\mathbf{b} \in \mathbb{R}^m$ is a column vector composed of elements $b_1, \dots, b_m$.

THEOREM 9.23. *Every polyhedral set is convex.*

EXERCISE 58. Prove Theorem 9.23. [Hint: You can prove this by brute force, verifying convexity. You can also be clever and use two results that we've proved in the notes.]

2.3. Directions of Polyhedral Sets. Recall the definition of a *line* (Definition 1.17 from Chapter 1). A ray is a one sided line.

DEFINITION 9.24 (Ray). Let $\mathbf{x}_0 \in \mathbb{R}^n$ be a point and and let $\mathbf{d} \in \mathbb{R}^n$ be a vector called the *direction*. Then the *ray* with vertex $\mathbf{x}_0$ and direction $\mathbf{d}$ is the collection of points $\{\mathbf{x}|\mathbf{x} = \mathbf{x}_0 + \lambda\mathbf{d},\ \lambda \geq 0\}$.

DEFINITION 9.25 (Direction of a Convex Set). Let C be a convex set. Then $\mathbf{d} \neq \mathbf{0}$ is a *(recession) direction of the convex set* if for all $\mathbf{x}_0 \in C$ the ray with vertex $\mathbf{x}_0$ and direction $\mathbf{d}$ is contained entirely in C. Formally, for all $\mathbf{x}_0 \in C$ we have:

$$\{\mathbf{x} : \mathbf{x} = \mathbf{x}_0 + \lambda\mathbf{d},\ \lambda \geq 0\} \subseteq C \tag{9.27}$$

REMARK 9.26. There is a unique relationship between the defining matrix $\mathbf{A}$ of a polyhedral set P and a direction of this set that is particularly useful when we assume that P is located in the positive orthant of $\mathbb{R}^n$ (i.e., $\mathbf{x} \geq 0$ are defining constraints of P).

REMARK 9.27. A proof of the next theorem can be found in [Gri11].

THEOREM 9.28. *Suppose that $P \subseteq \mathbb{R}^n$ is a polyhedral set defined by:*

$$P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\} \tag{9.28}$$

If $\mathbf{d}$ is a direction of P, then the following hold:

$$\mathbf{A}\mathbf{d} \leq \mathbf{0},\ \mathbf{d} \geq \mathbf{0},\ \mathbf{d} \neq \mathbf{0}. \tag{9.29}$$

□

COROLLARY 9.29. *If*

$$P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\} \tag{9.30}$$

and $\mathbf{d}$ is a direction of P, then $\mathbf{d}$ must satisfy:

$$\mathbf{A}\mathbf{d} = \mathbf{0},\ \mathbf{d} \geq \mathbf{0},\ \mathbf{d} \neq \mathbf{0}. \tag{9.31}$$

EXERCISE 59. Prove the corollary above.

EXAMPLE 9.30. Consider the polyhedral set defined by the equations:

$$\begin{aligned}
&x_1 - x_2 \leq 1\\
&2x_1 + x_2 \geq 6\\
&x_1 \geq 0\\
&x_2 \geq 0
\end{aligned}$$

This set is clearly unbounded as we showed in class and it has at least one direction. The direction $\mathbf{d} = [0, 1]^T$ pointing directly up is a direction of this set. This is illustrated in Figure 9.3. In this example, we have:

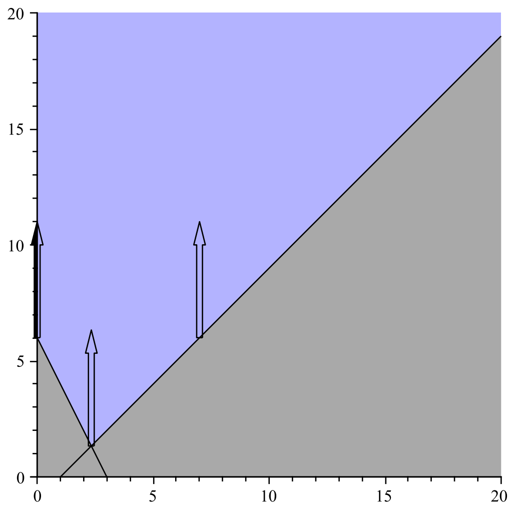


Figure 9.3. An Unbounded Polyhedral Set: This unbounded polyhedral set has many directions. One direction is $[0, 1]^T$.

$$\mathbf{A} = \begin{bmatrix} 1 & -1 \\ -2 & -1 \end{bmatrix} \tag{9.32}$$

Note, the second inequality constraint was a greater-than constraint. We reversed it to a less-than inequality constraint $-2x_1 - x_2 \leq -6$ by multiplying by -1. For our chosen direction $\mathbf{d} = [0, 1]^T$, we can see that:

$$\mathbf{Ad} = \begin{bmatrix} 1 & -1 \\ -2 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \leq \mathbf{0} \tag{9.33}$$

Clearly $\mathbf{d} \geq \mathbf{0}$ and $\mathbf{d} \neq \mathbf{0}$.

2.4. Extreme Points.

Definition 9.31 (Extreme Point of a Convex Set). Let C be a convex set. A point $\mathbf{x}_0 \in C$ is a *extreme point* of C if there are *no points* $\mathbf{x}_1$ and $\mathbf{x}_2$ ($\mathbf{x}_1 \neq \mathbf{x}_0$ or $\mathbf{x}_2 \neq \mathbf{x}_0$) so that $\mathbf{x} = \lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2$ for some $\lambda \in (0, 1)$.[2]

An extreme point is simply a point in a convex set C that cannot be expressed as a strict convex combination of any other pair of points in C. We will see that extreme points must be located in specific locations in convex sets.

Definition 9.32 (Boundary of a set). Let $C \subseteq \mathbb{R}^n$ be (convex) set. A point $\mathbf{x}_0 \in C$ is on the *boundary* of C if for all $\epsilon > 0$,

$$B_\epsilon(\mathbf{x}_0) \cap C \neq \emptyset \text{ and}$$
$$B_\epsilon(\mathbf{x}_0) \cap \mathbb{R}^n \setminus C \neq \emptyset$$

Example 9.33. A convex set, its boundary and a boundary point are illustrated in Figure 9.4.

[2]Thanks to Bob Pakzad-Hurson who fixed a typo in this definition in Version $\leq$ 1.4.

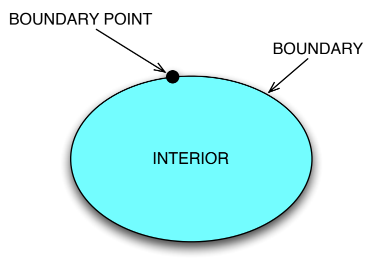


Figure 9.4. Boundary Point: A boundary point of a (convex) set C is a point in the set so that for *every* ball of any radius centered at the point contains some points inside C and some points outside C.

Lemma 9.34. *Suppose C is a convex set. If $\mathbf{x}$ is an extreme point of C, then $\mathbf{x}$ is on the boundary of C.*

Exercise 60. Prove the previous lemma.

Most important in our discussion of linear programming will be the extreme points of polyhedral sets that appear in linear programming problems. The following theorem establishes the relationship between extreme points in a polyhedral set and the intersection of hyperplanes in such a set.

Theorem 9.35. *Let $P \subseteq \mathbb{R}^n$ be a polyhedral set and suppose P is defined as:*

$$P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b}\} \tag{9.34}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. A point $\mathbf{x}_0 \in P$ is an extreme point of P if and only if $\mathbf{x}_0$ is the intersection of n linearly independent hyperplanes from the set defining P.

Remark 9.36. The easiest way to see this as relevant to linear programming is to assume that

$$P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\} \tag{9.35}$$

In this case, we could have $m < n$. In that case, P is composed of the intersection of $n + m$ half-spaces. The first m are for the rows of $\mathbf{A}$ and the second n are for the non-negativity constraints. An extreme point comes from the intersection of n of the hyperplanes defining these half-spaces. We might have m come from the constraints $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ and the other $n - m$ from $\mathbf{x} \geq \mathbf{0}$.

Remark 9.37. A complete proof of the previous theorem can be found in [Gri11].

Definition 9.38. Let P be the polyhedral set from Theorem 9.35. If $\mathbf{x}_0$ is an extreme point of P and *more* than n hyperplanes are binding at $\mathbf{x}_0$, then $\mathbf{x}_0$ is called a *degenerate* extreme point.

Definition 9.39 (Face). Let P be a polyhedral set defined by

$$P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. If $X \subseteq P$ is defined by a non-empty set of binding linearly independent hyperplanes, then X is a *face* of P.

That is, there is some set of linearly independent rows $A_{i_1\cdot}, \dots A_{i_l\cdot}$ with $i_l < m$ so that when $\mathbf{G}$ is the matrix made of these rows and $\mathbf{g}$ is the vector of $b_{i_1}, \dots, b_{i_l}$ then:

$$X = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Gx} = \mathbf{g} \text{ and } \mathbf{Ax} \leq \mathbf{b}\} \tag{9.36}$$

In this case we say that X has *dimension* $n - l$.

Remark 9.40. Based on this definition, we can easily see that an extreme point, which is the intersection n linearly independent hyperplanes is a face of dimension zero.

Definition 9.41 (Edge and Adjacent Extreme Point). An edge of a polyhedral set P is any face of dimension 1. Two extreme points are called *adjacent* if they share $n-1$ binding constraints. That is, they are connected by an edge of P.

Example 9.42. Consider the polyhedral set defined by the system of inequalities:

$$\begin{aligned}
&3x_1 + x_2 \leq 120\\
&x_1 + 2x_2 \leq 160\\
&\frac{28}{16}x_1 + x_2 \leq 100\\
&x_1 \leq 35\\
&x_1 \geq 0\\
&x_2 \geq 0
\end{aligned}$$

The polyhedral set is shown in Figure 9.5. The extreme points of the polyhedral set are shown

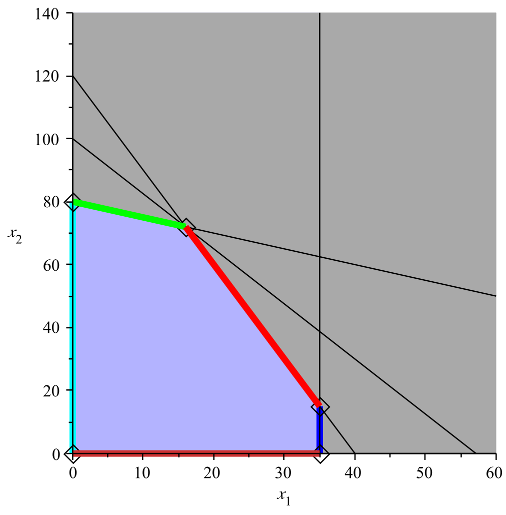


Figure 9.5. A Polyhedral Set: This polyhedral set is defined by five half-spaces and has a single degenerate extreme point located at the intersection of the binding constraints $3x_1 + x_2 \leq 120$, $x_1 + 2x_2 \leq 160$ and $\frac{28}{16}x_1 + x_2 <= 100$. All faces are shown in bold.

as large diamonds and correspond to intersections of binding constraints. Note the extreme point $(16, 72)$ is *degenerate* since it occurs at the intersection of three binding constraints $3x_1 + x_2 \leq 120$, $x_1 + 2x_2 \leq 160$ and $\frac{28}{16}x_1 + x_2 <= 100$. All the faces of the polyhedral set are shown in bold. They are locations where one constraint (or half-space) is binding. An example of a pair of adjacent extreme points is $(16, 72)$ and $(35, 15)$, as they are connected by the edge defined by the binding constraint $3x_1 + x_2 \leq 120$.

Exercise 61. Consider the polyhedral set defined by the system of inequalities:

$$\begin{aligned}
&4x_1 + x_2 \leq 120\\
&x_1 + 8x_2 \leq 160\\
&x_1 + x_2 \leq 30\\
&x_1 \geq 0\\
&x_2 \geq 0
\end{aligned}$$

Identify all extreme points and edges in this polyhedral set and their binding constraints. Are any extreme points degenerate? List all pairs of adjacent extreme points.

2.5. Extreme Directions.

Definition 9.43 (Extreme Direction). Let $C \subseteq \mathbb{R}^n$ be a convex set. Then a direction $\mathbf{d}$ of C is an *extreme direction* if there are no two other directions $\mathbf{d}_1$ and $\mathbf{d}_2$ of C ($\mathbf{d}_1 \neq \mathbf{d}$ and $\mathbf{d}_2 \neq \mathbf{d}$) and scalars $\lambda_1, \lambda_2 > 0$ so that $\mathbf{d} = \lambda_1\mathbf{d}_1 + \lambda_2\mathbf{d}_2$.

We have already seen by Theorem 9.28 that is P is a polyhedral set in the positive orthant of $\mathbb{R}^n$ with form:

$$P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\}$$

then a direction $\mathbf{d}$ of P is characterized by the set of inequalities and equations

$$\mathbf{A}\mathbf{d} \leq \mathbf{0},\ \mathbf{d} \geq \mathbf{0},\ \mathbf{d} \neq \mathbf{0}.$$

Clearly two directions $\mathbf{d}_1$ and $\mathbf{d}_2$ with $\mathbf{d_1} = \lambda\mathbf{d}_2$ for some $\lambda \geq 0$ may both satisfy this system. To isolate a unique set of directions, we can normalize and construct the set:

$$D = \{\mathbf{d} \in \mathbb{R}^n : \mathbf{A}\mathbf{d} \leq \mathbf{0},\ \mathbf{d} \geq \mathbf{0}, \mathbf{e}^T\mathbf{d} = 1\} \tag{9.37}$$

here we are interested only in directions satisfying $\mathbf{e}^T\mathbf{d} = 1$. This is a normalizing constraint that will chose only vectors whose components sum to 1.

Theorem 9.44. *A direction $\mathbf{d} \in D$ is an extreme direction of P if and only if $\mathbf{d}$ is an extreme point of D when D is taken as a polyhedral set.*

Remark 9.45. A complete proof of the previous theorem can be found in [Gri11].

Example 9.46. Let's consider Example 9.30 again. The polyhedral set in this example was defined by the $\mathbf{A}$ matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & -1 \\ -2 & -1 \end{bmatrix}$$

and the $\mathbf{b}$ vector:

$$\mathbf{b} = \begin{bmatrix} 1 \\ -6 \end{bmatrix}$$

If we assume that $P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq 0\}$, then the set of extreme directions of P is the same as the set of extreme points of the set

$$D = \{\mathbf{d} \in \mathbb{R}^n : \mathbf{A}\mathbf{d} \leq \mathbf{0},\ \mathbf{d} \geq \mathbf{0}, \mathbf{e}^T\mathbf{d} = 1\}$$

Then we have the set of directions $\mathbf{d} = [d_1, d_2]^T$ so that:

$$\begin{aligned}
&d_1 - d_2 \leq 0 \\
&-2d_1 - d_2 \leq 0 \\
&d_1 + d_2 = 1 \\
&d_1 \geq 0 \\
&d_2 \geq 0
\end{aligned}$$

The feasible region (which is really only the line $d_1 + d_2 = 1$) is shown in red in Figure 9.6. The critical part of this figure is the red line. It is the true set D. As a line, it has two

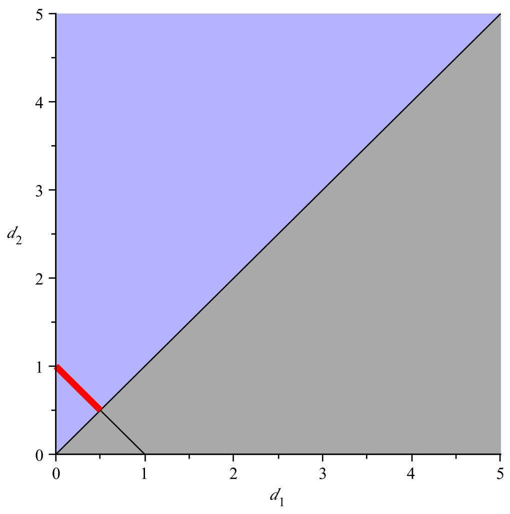


Figure 9.6. Visualization of the set D: This set really consists of the set of points on the red line. This is the line where $d_1 + d_2 = 1$ and all other constraints hold. This line has two extreme points $(0, 1)$ and $(1/2, 1/2)$.

extreme points: $(0, 1)$ and $(1/2, 1/2)$. Note that $(0, 1)$ as an extreme point is one of the direction $[0, 1]^T$ we illustrated in Example 9.30.

Exercise 62. Show that $\mathbf{d} = [1/2, 1/2]^T$ is a direction of the polyhedral set P from Example 9.30. Now find a non-extreme direction (whose components sum to 1) using the feasible region illustrated in the previous example. Show that the direction you found is a direction of the polyhedral set. Create a figure like Figure 9.3 to illustrate *both* these directions.

2.6. Caratheodory Characterization Theorem.

Remark 9.47. The theorem stated in this sub-section is critical to understanding the fundamental theorems of linear programming. Proofs can be found in [Gri11].

Lemma 9.48. *The polyhedral set defined by:*

$$P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\}$$

has a finite, non-zero number of extreme points (assuming that $\mathbf{A}$ *is not an empty matrix)*[3].

[3]Thanks to Bob Pakzah-Hurson for the suggestion to improve the statement of this lemma.

LEMMA 9.49. *Let P be a non-empty polyhedral set. Then the set of directions of P is empty if and only if P is bounded.*

LEMMA 9.50. *Let P be a non-empty unbounded polyhedral set. Then the number extreme directions of P is finite and non-zero.*

THEOREM 9.51. *Let P be a non-empty, unbounded polyhedral set defined by:*

$$P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\}$$

(where we assume $\mathbf{A}$ is not an empty matrix). Suppose that P has extreme points $\mathbf{x}_1, \dots, \mathbf{x}_k$ and extreme directions $\mathbf{d_1}, \dots, \mathbf{d}_l$. If $\mathbf{x} \in P$, then there exists constants $\lambda_1, \dots, \lambda_k$ and $\mu_1, \dots, \mu_l$ such that:

$$(9.38)\qquad \begin{aligned} &\mathbf{x} = \sum_{i=1}^{k} \lambda_i \mathbf{x}_i + \sum_{j=1}^{l} \mu_j \mathbf{d}_j \\ &\sum_{i=1}^{k} \lambda_i = 1 \\ &\lambda_i \geq 0 \quad i = 1, \dots, k \\ &\mu_j \geq 0 \quad 1, \dots, l \end{aligned}$$

EXAMPLE 9.52. The Cartheodory Characterization Theorem is illustrated for a bounded and unbounded polyhedral set in Figure 9.7. This example illustrates simply how one could

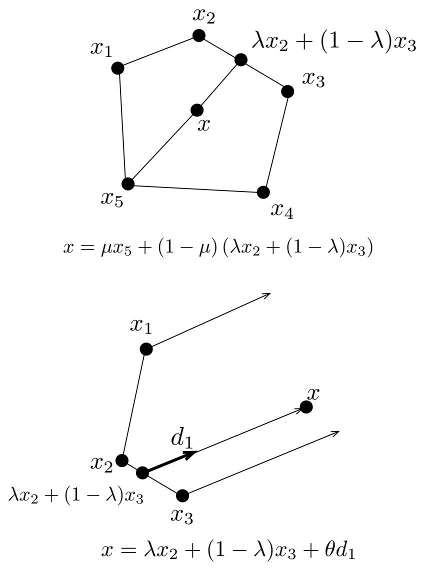

x_1
x_2
$\lambda x_2 + (1-\lambda)x_3$
x_3
x
x_5
x_4
$x = \mu x_5 + (1-\mu)(\lambda x_2 + (1-\lambda)x_3)$
x_1
x
d_1
x_2
$\lambda x_2 + (1-\lambda)x_3$
x_3
$x = \lambda x_2 + (1-\lambda)x_3 + \theta d_1$

Figure 9.7. The Cartheodory Characterization Theorem: Extreme points and extreme directions are used to express points in a bounded and unbounded set.

construct an expression for an arbitrary point $\mathbf{x}$ inside a polyhedral set in terms of extreme points and extreme directions.

3. The Simplex Method

For the remainder of this chapter, assume that $A \in \mathbb{R}^{m \times n}$ with full row rank and $b \in \mathbb{R}^m$ let

$$X = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b}, \ \ \mathbf{x} \geq \mathbf{0}\} \tag{9.39}$$

be a polyhedral set over which we will maximize the objective function $z(x_1, \dots, x_n) = \mathbf{c}^T\mathbf{x}$, where $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$. That is, we will focus on the linear programming problem:

$$P \begin{cases} \max & \mathbf{c}^T\mathbf{x} \\ s.t. & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{cases} \tag{9.40}$$

THEOREM 9.53. *If Problem P has an optimal solution, then Problem P has an optimal extreme point solution.*

PROOF. Applying the Cartheodory Characterization theorem, we know that any point $\mathbf{x} \in X$ can be written as:

$$\mathbf{x} = \sum_{i=1}^{k} \lambda_i \mathbf{x}_i + \sum_{i=1}^{l} \mu_i \mathbf{d}_i \tag{9.41}$$

where $\mathbf{x}_1, \dots \mathbf{x}_k$ are the extreme points of X and $\mathbf{d}_1, \dots, \mathbf{d}_l$ are the extreme directions of X and we know that

$$\begin{aligned} & \sum_{i=1}^{k} \lambda_i = 1 \\ & \lambda_i, \mu_i \geq 0 \ \ \forall i \end{aligned} \tag{9.42}$$

We can rewrite problem P using this characterization as:

$$\begin{aligned} \max \ \ & \sum_{i=1}^{k} \lambda_i \mathbf{c}^T \mathbf{x}_i + \sum_{i=1}^{l} \mu_i \mathbf{c}^T \mathbf{d}_i \\ s.t. \ \ & \sum_{i=1}^{k} \lambda_i = 1 \\ & \lambda_i, \mu_i \geq 0 \ \ \forall i \end{aligned} \tag{9.43}$$

If there is some i such that $\mathbf{c}^T\mathbf{d}_i > 0$, then we can simply choose μ_i as large as we like, making the objective as large as we like, the problem will have no finite solution.

Therefore, assume that $\mathbf{c}^T\mathbf{d}_i \leq 0$ for all $i = 1, \dots, l$ (in which case, we may simply choose $\mu_i = 0$, for all i). Since the set of extreme points $\mathbf{x}_1, \dots \mathbf{x}_k$ is finite, we can simply set $\lambda_p = 1$ if $\mathbf{c}^T\mathbf{x}_p$ has the largest value among all possible values of $\mathbf{c}^T\mathbf{x}_i$, $i = 1, \dots, k$. This is clearly the solution to the linear programming problem. Since $\mathbf{x}_p$ is an extreme point, we have shown that if P has a solution, it must have an extreme point solution. □

COROLLARY 9.54. *Problem P has a finite solution if and only if $\mathbf{c}^T\mathbf{d}_i \leq 0$ for all $i = 1, \dots l$ when $\mathbf{d}_1, \dots, \mathbf{d}_l$ are the extreme directions of X.*

PROOF. This is implicit in the proof of the theorem. □

Corollary 9.55. *Problem P has alternative optimal solutions if there are at least two extreme points $\mathbf{x}_p$ and $\mathbf{x}_q$ so that $\mathbf{c}^T\mathbf{x}_p = \mathbf{c}^T\mathbf{x}_q$ and so that $\mathbf{x}_p$ is the extreme point solution to the linear programming problem.*

Proof. Suppose that $\mathbf{x}_p$ is the extreme point solution to P identified in the proof of the theorem. Suppose $\mathbf{x}_q$ is another extreme point solution with $\mathbf{c}^T\mathbf{x}_p = \mathbf{c}^T\mathbf{x}_q$. Then every convex combination of $\mathbf{x}_p$ and $\mathbf{x}_q$ is contained in X (since X is convex). Thus every $\mathbf{x}$ with form $\lambda\mathbf{x}_p + (1-\lambda)\mathbf{x}_q$ and $\lambda \in [0,1]$ has objective function value:

$$\lambda\mathbf{c}^T\mathbf{x}_p + (1-\lambda)\mathbf{c}^T\mathbf{x}_q = \lambda\mathbf{c}^T\mathbf{x}_p + (1-\lambda)\mathbf{c}^T\mathbf{x}_p = \mathbf{c}^T\mathbf{x}_p$$

which is the optimal objective function value, by assumption. □

Exercise 63. Let $X = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \ \mathbf{x} \geq \mathbf{0}\}$ and suppose that $\mathbf{d}_1, \dots \mathbf{d}_l$ are the extreme directions of X (assuming it has any). Show that the problem:

$$(9.44)\qquad \begin{aligned} \min\ & \mathbf{c}^T\mathbf{x} \\ s.t.\ & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

has a finite optimal solution if (and only if) $\mathbf{c}^T\mathbf{d}_j \geq 0$ for $k = 1, \dots, l$. [Hint: Modify the proof above using the Cartheodory characterization theorem.]

3.1. Algorithmic Characterization of Extreme Points. In the previous sections we showed that if a linear programming problem has a solution, then it must have an extreme point solution. The challenge now is to identify some simple way of identifying extreme points. To accomplish this, let us now assume that we write X as:

$$(9.45)\qquad X = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} = \mathbf{b},\ \ \mathbf{x} \geq \mathbf{0}\}$$

Our work in the previous sections shows that this is possible. Recall we can separate $\mathbf{A}$ into an $m \times m$ matrix B and an $m \times (n-m)$ matrix N and we have the result:

$$(9.46)\qquad \mathbf{x_B} = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x_N}$$

We know that $\mathbf{B}$ is invertible since we assumed that $\mathbf{A}$ had full row rank. If we assume that $\mathbf{x_N} = \mathbf{0}$, then the solution

$$(9.47)\qquad \mathbf{x_B} = \mathbf{B}^{-1}\mathbf{b}$$

was called a *basic solution* (See Definition 9.16.) Clearly any basic solution satisfies the constraints $\mathbf{A}\mathbf{x} = \mathbf{b}$ but it may not satisfy the constraints $\mathbf{x} \geq \mathbf{0}$.

Definition 9.56 (Basic Feasible Solution). If $\mathbf{x_B} = \mathbf{B}^{-1}\mathbf{b}$ and $\mathbf{x_N} = \mathbf{0}$ is a basic solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$ and $\mathbf{x_B} \geq 0$, then the solution $(\mathbf{x_B}, \mathbf{x_N})$ is called *basic feasible solution.*

Theorem 9.57. *Every basic feasible solution is an extreme point of X. Likewise, every extreme point is characterized by a basic feasible solution of $\mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}$.*

Proof. Since $\mathbf{A}\mathbf{x} = \mathbf{B}\mathbf{x_B} + \mathbf{N}\mathbf{x_N} = \mathbf{b}$ this represents the intersection of m linearly independent hyperplanes (since the rank of $\mathbf{A}$ is m). The fact that $\mathbf{x_N} = \mathbf{0}$ and $\mathbf{x_N}$ contains $n-m$ variables, then we have $n-m$ binding, linearly independent hyperplanes in $\mathbf{x_N} \geq 0$. Thus the point $(\mathbf{x_B}, \mathbf{x_N})$ is the intersection of $m + (n-m) = n$ linearly independent hyperplanes. By Theorem 9.35 we know that $(\mathbf{x_B}, \mathbf{x_N})$ must be an extreme point of X.

Conversely, let $\mathbf{x}$ be an extreme point of X. Clearly $\mathbf{x}$ is feasible and by Theorem 9.35 it must represent the intersection of n hyperplanes. The fact that $\mathbf{x}$ is feasible implies that $\mathbf{Ax} = \mathbf{b}$. This accounts for m of the intersecting linearly independent hyperplanes. The remaining $n-m$ hyperplanes must come from $\mathbf{x} \geq \mathbf{0}$. That is, $n-m$ variables are zero. Let $\mathbf{x_N} = \mathbf{0}$ be the variables for which $\mathbf{x} \geq \mathbf{0}$ are binding. Denote the remaining variables $\mathbf{x_B}$. We can see that $\mathbf{A} = [\mathbf{B}|\mathbf{N}]$ and that $\mathbf{Ax} = \mathbf{Bx_B} + \mathbf{Nx_N} = \mathbf{b}$. Clearly, $\mathbf{x_B}$ is the unique solution to $\mathbf{Bx_B} = \mathbf{b}$ and thus $(\mathbf{x_B}, \mathbf{x_N})$ is a basic feasible solution. $\square$

3.2. The Simplex Algorithm–Algebraic Form. In this section, we will develop the simplex algorithm algebraically. The idea behind the simplex algorithm is as follows:

(1) Convert the linear program to standard form.
(2) Obtain an initial basic feasible solution (if possible).
(3) Determine whether the basic feasible solution is optimal. If yes, stop.
(4) If the current basic feasible solution is not optimal, then determine which non-basic variable (zero valued variable) should become basic (become non-zero) and which basic variable (non-zero valued variable) should become non-basic (go to zero) to make the objective function value better.
(5) Determine whether the problem is unbounded. If yes, stop.
(6) If the problem doesn't seem to be unbounded at this stage, find a new basic feasible solution from the old basic feasible solution. Go back to Step 3.

Suppose we have a basic feasible solution $\mathbf{x} = (\mathbf{x_B}, \mathbf{x_N})$. We can divide the cost vector $\mathbf{c}$ into its basic and non-basic parts, so we have $\mathbf{c} = [\mathbf{c_B}|\mathbf{c_N}]^T$. Then the objective function becomes:

$$\mathbf{c}^T\mathbf{x} = \mathbf{c_B}^T\mathbf{x_B} + \mathbf{c_N}^T\mathbf{x_N} \tag{9.48}$$

We can substitute Equation 9.46 into Equation 9.48 to obtain:

$$\mathbf{c}^T\mathbf{x} = \mathbf{c_B}^T\left(\mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x_N}\right) + \mathbf{c_N}\mathbf{x_N} = \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{b} + \left(\mathbf{c_N}^T - \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{N}\right)\mathbf{x}_N \tag{9.49}$$

Let $\mathcal{J}$ be the set of indices of non-basic variables. Then we can write Equation 9.49 as:

$$z(x_1, \dots, x_n) = \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{b} + \sum_{j \in \mathcal{J}} \left(\mathbf{c}_j - \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{A}_{\cdot j}\right) x_j \tag{9.50}$$

Consider now the fact $x_j = 0$ for all $j \in \mathcal{J}$. Further, we can see that:

$$\frac{\partial z}{\partial x_j} = \mathbf{c}_j - \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{A}_{\cdot j} \tag{9.51}$$

This means that if $\mathbf{c}_j - \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{A}_{\cdot j} > 0$ and we *increase* x_j from zero to some new value, then we will *increase* the value of the objective function. For historic reasons, we actually consider the value $\mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{A}_{\cdot j} - \mathbf{c}_j$, called the *reduced cost* and denote it as:

$$-\frac{\partial z}{\partial x_j} = z_j - c_j = \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{A}_{\cdot j} - \mathbf{c}_j \tag{9.52}$$

In a maximization problem, we chose non-basic variables x_j with negative reduced cost to become basic because, in this case, $\partial z/\partial x_j$ is *positive.*

Assume we chose x_j, a non-basic variable to become non-zero (because $z_j - c_j < 0$). We wish to know which of the basic variables will become zero as we *increase* x_j away from zero. We must also be very careful that *none* of the variables become negative as we do this.

By Equation 9.46 we know that the only current basic variables will be affected by increasing x_j. Let us focus explicitly on Equation 9.46 where we include only variable x_j (since all other non-basic variables are kept zero). Then we have:

$$\mathbf{x_B} = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{A}_{\cdot j}x_j \tag{9.53}$$

Let $\overline{\mathbf{b}} = \mathbf{B}^{-1}\mathbf{b}$ be an $m \times 1$ column vector and let that $\overline{\mathbf{a}_j} = \mathbf{B}^{-1}\mathbf{A}_{\cdot j}$ be another $m \times 1$ column. Then we can write:

$$\mathbf{x_B} = \overline{\mathbf{b}} - \overline{\mathbf{a}}_j x_j \tag{9.54}$$

Let $\overline{\mathbf{b}} = [\overline{b}_1, \dots \overline{b}_m]^T$ and $\overline{\mathbf{a}}_j = [\overline{a}_{j_1}, \dots, \overline{a}_{j_m}]$, then we have:

$$\begin{bmatrix} x_{B_1} \\ x_{B_2} \\ \vdots \\ x_{B_m} \end{bmatrix} = \begin{bmatrix} \overline{b}_1 \\ \overline{b}_2 \\ \vdots \\ \overline{b}_m \end{bmatrix} - \begin{bmatrix} \overline{a}_{j_1} \\ \overline{b}_{j_2} \\ \vdots \\ \overline{b}_{j_m} \end{bmatrix} x_j = \begin{bmatrix} \overline{b}_1 - \overline{a}_{j_1}x_j \\ \overline{b}_2 - \overline{a}_{j_2}x_j \\ \vdots \\ \overline{b}_m - \overline{a}_{j_m}x_j \end{bmatrix} \tag{9.55}$$

We know (a priori) that $\overline{b}_i \geq 0$ for $i = 1, \dots, m$. If $\overline{a}_{j_i} \leq 0$, then as we increase x_j, $\overline{b}_i - \overline{a}_{j_i} \geq 0$ no matter how large we make x_j. On the other hand, if $\overline{a}_{j_i} > 0$, then as we increase x_j we know that $\overline{b}_i - \overline{a}_{j_i}x_j$ will get smaller and eventually hit zero. In order to ensure that *all* variables remain non-negative, we cannot increase x_j beyond a certain point.

For each i ($i = 1, \dots, m$) such that $\overline{a}_{j_i} > 0$, the value of x_j that will make x_{B_i} goto 0 can be found by observing that:

$$x_{B_i} = \overline{b}_i - \overline{a}_{j_i}x_j \tag{9.56}$$

and if $x_{B_i} = 0$, then we can solve:

$$0 = \overline{b}_i - \overline{a}_{j_i}x_j \implies x_j = \frac{\overline{b}_i}{\overline{a}_{j_i}} \tag{9.57}$$

Thus, the *largest possible value* we can assign x_j and ensure that all variables remain positive is:

$$\min\left\{\frac{\overline{b}_i}{\overline{a}_{j_i}} : i = 1, \dots, m \text{ and } a_{j_i} > 0\right\} \tag{9.58}$$

Expression 9.58 is called the *minimum ratio test.* We are interested in which index i is the minimum ratio.

Suppose that in executing the minimum ratio test, we find that $x_j = \overline{b}_k/\overline{a}_{j_k}$. The variable x_j (which was non-basic) becomes basic and the variable $x_{\mathbf{B}_k}$ becomes non-basic. All other basic variables remain basic (and positive). In executing this procedure (of exchanging one basic variable and one non-basic variable) we have moved from one extreme point of X to another.

Theorem 9.58. *If $z_j - c_j \geq 0$ for all $j \in \mathcal{J}$, then the current basic feasible solution is (locally) optimal*[4].

[4] We show later that is globally optimal, however we cannot do any better than a local argument for now.

PROOF. We have already shown in Theorem 9.53 that if a linear programming problem has an optimal solution, then it occurs at an extreme point and we've shown in Theorem 9.57 that there is a one-to-one correspondence between extreme points and basic feasible solutions. If $z_j - c_j \geq 0$ for all $j \in \mathcal{J}$, then $\partial z/\partial x_j \leq 0$ for all non-basic variables x_j. That is, we cannot increase the value of the objective function by increasing the value of any non-basic variable. Thus, since moving to another basic feasible solution (extreme point) will not improve the objective function, it follows we must be at a (locally) optimal solution. □

THEOREM 9.59. *In a maximization problem, if* $\overline{a}_{j_i} \leq 0$ *for all* $i = 1, \dots, m$*, and* $z_j - c_j < 0$*, then the linear programming problem is unbounded.*

PROOF. The fact that $z_j - c_j < 0$ implies that increasing x_j will improve the value of the objective function. Since $\overline{a}_{j_i} < 0$ for all $i = 1, \dots, m$, we can increase x_j indefinitely without violating feasibility (no basic variable will ever go to zero). Thus the objective function can be made as large as we like. □

REMARK 9.60. We should note that in executing the exchange of one basic variable and one non-basic variable, we must be *very* careful to ensure that the resulting basis consist of m linearly independent columns of the original matrix $\mathbf{A}$. Specifically, we must be able to write the column corresponding to x_j, the entering variable, as a linear combination of the columns of $\mathbf{B}$ so that:

$$\alpha_1 \mathbf{b}_1 + \dots \alpha_m \mathbf{b}_m = \mathbf{A}_{\cdot j} \tag{9.59}$$

and further if we are exchanging x_j for $\mathbf{x}_{\mathbf{B}_i}$ $(i = 1, \dots, m)$, then $\alpha_i \neq 0$.

We can see this from the fact that $\overline{\mathbf{a}}_j = \mathbf{B}^{-1}\mathbf{A}_{\cdot j}$ and therefore:

$$\mathbf{B}\overline{\mathbf{a}}_j = \mathbf{A}_{\cdot j}$$

and therefore we have:

$$\mathbf{A}_{\cdot j} = \mathbf{B}_{\cdot 1}\overline{\mathbf{a}}_{j_1} + \dots + \mathbf{B}_{\cdot m}\overline{\mathbf{a}}_{j_m}$$

which shows how to write the column $\mathbf{A}_{\cdot j}$ as a linear combination of the columns of $\mathbf{B}$.

EXERCISE 64. Consider the linear programming problem given in Exercise 63. Under what conditions should a non-basic variable enter the basis? State and prove an analogous theorem to Theorem 9.58 using your observation. [Hint: Use the definition of reduced cost. Remember that it is $-\partial z/\partial x_j$.]

EXAMPLE 9.61. Consider a simple Linear Programming Problem:

$$\begin{cases} \max & z(x_1, x_2) = 7x_1 + 6x_2 \\ s.t. & 3x_1 + x_2 \leq 120 \\ & x_1 + 2x_2 \leq 160 \\ & x_1 \leq 35 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{cases}$$

We can convert this problem to standard form by introducing the slack variables s_1, s_2 and s_3:

$$\begin{cases} \max \quad z(x_1, x_2) = 7x_1 + 6x_2 \\ s.t. \;\; 3x_1 + x_2 + s_1 = 120 \\ x_1 + 2x_2 + s_2 = 160 \\ x_1 + s_3 = 35 \\ x_1, x_2, s_1, s_2, s_3 \geq 0 \end{cases}$$

which yields the matrices

$$\mathbf{c} = \begin{bmatrix} 7 \\ 6 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} 3 & 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 120 \\ 160 \\ 35 \end{bmatrix}$$

We can begin with the matrices:

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{N} = \begin{bmatrix} 3 & 1 \\ 1 & 2 \\ 1 & 0 \end{bmatrix}$$

In this case we have:

$$\mathbf{x_B} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} \quad \mathbf{x_N} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \mathbf{c_B} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{c_N} = \begin{bmatrix} 7 \\ 6 \end{bmatrix}$$

and

$$\mathbf{B}^{-1}\mathbf{b} = \begin{bmatrix} 120 \\ 160 \\ 35 \end{bmatrix} \quad \mathbf{B}^{-1}\mathbf{N} = \begin{bmatrix} 3 & 1 \\ 1 & 2 \\ 1 & 0 \end{bmatrix}$$

Therefore:

$$\mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{b} = 0 \quad \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{N} = \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{N} - \mathbf{c_N} = \begin{bmatrix} -7 & -6 \end{bmatrix}$$

Using this information, we can compute:

$$\mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{A}_{\cdot 1} - \mathbf{c}_1 = -7$$
$$\mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{A}_{\cdot 2} - \mathbf{c}_2 = -6$$

and therefore:

$$\frac{\partial z}{\partial x_1} = 7 \text{ and } \frac{\partial z}{\partial x_2} = 6$$

Based on this information, we could chose either x_1 or x_2 to enter the basis and the value of the objective function would increase. If we chose x_1 to enter the basis, then we must determine which variable will leave the basis. To do this, we must investigate the elements of $\mathbf{B}^{-1}\mathbf{A}_{\cdot 1}$ and the current basic feasible solution $\mathbf{B}^{-1}\mathbf{b}$. Since each element of $\mathbf{B}^{-1}\mathbf{A}_{\cdot 1}$ is positive, we must perform the minimum ratio test on each element of $\mathbf{B}^{-1}\mathbf{A}_{\cdot 1}$. We know that $\mathbf{B}^{-1}\mathbf{A}_{\cdot 1}$ is just the first column of $\mathbf{B}^{-1}\mathbf{N}$ which is:

$$\mathbf{B}^{-1}\mathbf{A}_{\cdot 1} = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix}$$

Performing the minimum ratio test, we see have:

$$\min\left\{\frac{120}{3}, \frac{160}{1}, \frac{35}{1}\right\}$$

In this case, we see that index 3 (35/1) is the minimum ratio. Therefore, variable x_1 will enter the basis and variable s_3 will leave the basis. The new basic and non-basic variables will be:

$$\mathbf{x_B} = \begin{bmatrix} s_1 \\ s_2 \\ x_1 \end{bmatrix} \quad \mathbf{x_N} = \begin{bmatrix} s_3 \\ x_2 \end{bmatrix} \quad \mathbf{c_B} = \begin{bmatrix} 0 \\ 0 \\ 7 \end{bmatrix} \quad \mathbf{c_N} = \begin{bmatrix} 0 \\ 6 \end{bmatrix}$$

and the matrices become:

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{N} = \begin{bmatrix} 0 & 1 \\ 0 & 2 \\ 1 & 0 \end{bmatrix}$$

Note we have simply swapped the column corresponding to x_1 with the column corresponding to s_3 in the basis matrix $\mathbf{B}$ and the non-basic matrix $\mathbf{N}$. We will do this repeatedly in the example and we recommend the reader keep track of which variables are being exchanged and why certain columns in $\mathbf{B}$ are being swapped with those in $\mathbf{N}$.

Using the new $\mathbf{B}$ and $\mathbf{N}$ matrices, the derived matrices are then:

$$\mathbf{B}^{-1}\mathbf{b} = \begin{bmatrix} 15 \\ 125 \\ 35 \end{bmatrix} \quad \mathbf{B}^{-1}\mathbf{N} = \begin{bmatrix} -3 & 1 \\ -1 & 2 \\ 1 & 0 \end{bmatrix}$$

The cost information becomes:

$$\mathbf{c}_\mathbf{B}^T\mathbf{B}^{-1}\mathbf{b} = 245 \quad \mathbf{c}_\mathbf{B}^T\mathbf{B}^{-1}\mathbf{N} = \begin{bmatrix} 7 & 0 \end{bmatrix} \quad \mathbf{c}_\mathbf{B}^T\mathbf{B}^{-1}\mathbf{N} - \mathbf{c_N} = \begin{bmatrix} 7 & -6 \end{bmatrix}$$

using this information, we can compute:

$$\mathbf{c}_\mathbf{B}^T\mathbf{B}^{-1}\mathbf{A}_{\cdot 5} - \mathbf{c}_5 = 7$$
$$\mathbf{c}_\mathbf{B}^T\mathbf{B}^{-1}\mathbf{A}_{\cdot 2} - \mathbf{c}_2 = -6$$

Based on this information, we can only choose x_2 to enter the basis to ensure that the value of the objective function increases. We can perform the minimum ration test to figure out which basic variable will leave the basis. We know that $\mathbf{B}^{-1}\mathbf{A}_{\cdot 2}$ is just the second column of $\mathbf{B}^{-1}\mathbf{N}$ which is:

$$\mathbf{B}^{-1}\mathbf{A}_{\cdot 2} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$$

Performing the minimum ratio test, we see have:

$$\min\left\{\frac{15}{1}, \frac{125}{2}\right\}$$

In this case, we see that index 1 (15/1) is the minimum ratio. Therefore, variable x_2 will enter the basis and variable s_1 will leave the basis. The new basic and non-basic variables will be: The new basic and non-basic variables will be:

$$\mathbf{x_B} = \begin{bmatrix} x_2 \\ s_2 \\ x_1 \end{bmatrix} \quad \mathbf{x_N} = \begin{bmatrix} s_3 \\ s_1 \end{bmatrix} \quad \mathbf{c_B} = \begin{bmatrix} 6 \\ 0 \\ 7 \end{bmatrix} \quad \mathbf{c_N} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and the matrices become:

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 3 \\ 2 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{N} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

The derived matrices are then:

$$\mathbf{B}^{-1}\mathbf{b} = \begin{bmatrix} 15 \\ 95 \\ 35 \end{bmatrix} \quad \mathbf{B}^{-1}\mathbf{N} = \begin{bmatrix} -3 & 1 \\ 5 & -2 \\ 1 & 0 \end{bmatrix}$$

The cost information becomes:

$$\mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{b} = 335 \quad \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{N} = \begin{bmatrix} -11 & 6 \end{bmatrix} \quad \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{N} - \mathbf{c_N} = \begin{bmatrix} -11 & 6 \end{bmatrix}$$

Based on this information, we can only choose s_3 to (re-enter) the basis to ensure that the value of the objective function increases. We can perform the minimum ration test to figure out which basic variable will leave the basis. We know that $\mathbf{B}^{-1}\mathbf{A}_{\cdot 5}$ is just the fifth column of $\mathbf{B}^{-1}\mathbf{N}$ which is:

$$\mathbf{B}^{-1}\mathbf{A}_{\cdot 5} = \begin{bmatrix} -3 \\ 5 \\ 1 \end{bmatrix}$$

Performing the minimum ratio test, we see have:

$$\min\left\{\frac{95}{5}, \frac{35}{1}\right\}$$

In this case, we see that index 2 (95/5) is the minimum ratio. Therefore, variable s_3 will enter the basis and variable s_2 will leave the basis. The new basic and non-basic variables will be:

$$\mathbf{x_B} = \begin{bmatrix} x_2 \\ s_3 \\ x_1 \end{bmatrix} \quad \mathbf{x_N} = \begin{bmatrix} s_2 \\ s_1 \end{bmatrix} \quad \mathbf{c_B} = \begin{bmatrix} 6 \\ 0 \\ 7 \end{bmatrix} \quad \mathbf{c_N} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and the matrices become:

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 3 \\ 2 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad \mathbf{N} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}$$

The derived matrices are then:

$$\mathbf{B}^{-1}\mathbf{b} = \begin{bmatrix} 72 \\ 19 \\ 16 \end{bmatrix} \quad \mathbf{B}^{-1}\mathbf{N} = \begin{bmatrix} 6/10 & -1/5 \\ 1/5 & -2/5 \\ -1/5 & 2/5 \end{bmatrix}$$

The cost information becomes:

$$\mathbf{c}_{\mathbf{B}}^T\mathbf{B}^{-1}\mathbf{b} = 544 \quad \mathbf{c}_{\mathbf{B}}^T\mathbf{B}^{-1}\mathbf{N} = \begin{bmatrix} 11/5 & 8/5 \end{bmatrix} \quad \mathbf{c}_{\mathbf{B}}^T\mathbf{B}^{-1}\mathbf{N} - \mathbf{c}_{\mathbf{N}} = \begin{bmatrix} 11/5 & 8/5 \end{bmatrix}$$

Since the reduced costs are now positive, we can conclude that we've obtained an optimal solution because no improvement is possible. The final solution then is:

$$\mathbf{x_B}^* = \begin{bmatrix} x_2 \\ s_3 \\ x_1 \end{bmatrix} = \mathbf{B}^{-1}\mathbf{b} = \begin{bmatrix} 72 \\ 19 \\ 16 \end{bmatrix}$$

Simply, we have $x_1 = 16$ and $x_2 = 72$. The path of extreme points we actually took in traversing the boundary of the polyhedral feasible region is shown in Figure 9.8.

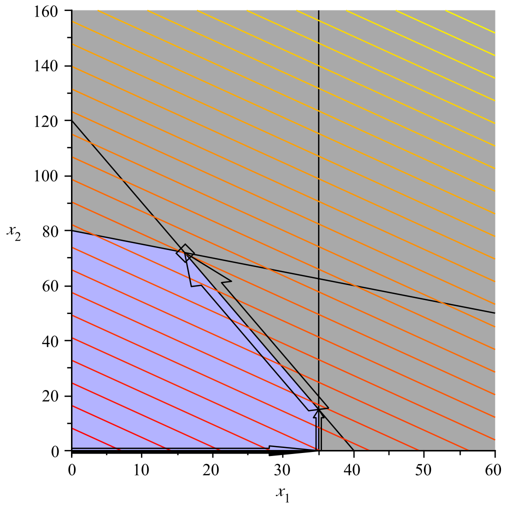


Figure 9.8. The Simplex Algorithm: The path around the feasible region is shown in the figure. Each exchange of a basic and non-basic variable moves us along an edge of the polygon in a direction that increases the value of the objective function.

Exercise 65. Assume that a leather company manufactures two types of belts: regular and deluxe. Each belt requires 1 square yard of leather. A regular belt requires 1 hour of skilled labor to produce, while a deluxe belt requires 2 hours of labor. The leather company receives 40 square yards of leather each week and a total of 60 hours of skilled labor is available. Each regular belt nets \$3 in profit, while each deluxe belt nets \$5 in profit. The company wishes to maximize profit.

(1) Ignoring the divisibility issues, construct a linear programming problem whose solution will determine the number of each type of belt the company should produce.
(2) Use the simplex algorithm to solve the problem you stated above remembering to convert the problem to *standard form* before you begin.

(3) Draw the feasible region and the level curves of the objective function. Verify that the optimal solution you obtained through the simplex method is the point at which the level curves no longer intersect the feasible region in the direction following the gradient of the objective function.

There are a few ways to chose the entering variable:

(1) Using Dantzig's Rule, we choose the variable with the greatest (absolute value) reduced cost.
(2) We can compute the next objective function value for each possible entering variable and chose the one that results in the largest objective function increase, in a greedy search.
(3) We can choose the first variable we find that is an acceptable entering variable (i.e., has a negative reduced cost).
(4) We can use a combination of these approaches.

Most Simplex codes have a complex recipe for choosing entering variables. We will deal with the question of breaking ties among leaving variables in our section on degeneracy.

4. Karush-Kuhn-Tucker (KKT) Conditions

REMARK 9.62. The single most important thing to learn about Linear Programming (or optimization in general) is the Karush-Kuhn-Tucker optimality conditions. These conditions provide necessary and sufficient conditions for a point $\mathbf{x} \in \mathbb{R}^n$ to be an optimal solution to a Linear Programming problem. We state the Karush-Kuhn-Tucker theorem, but do not prove it. A proof can be found in Chapter 8 of [Gri11].

THEOREM 9.63. *Consider the linear programming problem:*

$$P\begin{cases} \max & \mathbf{cx} \\ s.t. & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{cases} \tag{9.60}$$

with $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$ *and (row vector)* $\mathbf{c} \in \mathbb{R}^n$. *Then* $\mathbf{x}^* \in \mathbb{R}^n$ *if and only if there exists (row) vectors* $\mathbf{w}^* \in \mathbb{R}^m$ *and* $\mathbf{v}^* \in \mathbb{R}^n$ *and a slack variable vector* $\mathbf{s}^* \in \mathbf{R}^m$ *so that:*

$$\textit{Primal Feasibility}\begin{cases} \mathbf{Ax}^* + \mathbf{s}^* = \mathbf{b} \\ \mathbf{x}^* \geq \mathbf{0} \end{cases} \tag{9.61}$$

$$\textit{Dual Feasibility}\begin{cases} \mathbf{w}^*\mathbf{A} - \mathbf{v}^* = \mathbf{c} \\ \mathbf{w}^* \geq \mathbf{0} \\ \mathbf{v}^* \geq \mathbf{0} \end{cases} \tag{9.62}$$

$$\textit{Complementary Slackness}\begin{cases} \mathbf{w}^*\left(\mathbf{Ax}^* - \mathbf{b}\right) = 0 \\ \mathbf{v}^*\mathbf{x}^* = 0 \end{cases} \tag{9.63}$$

REMARK 9.64. The vectors $\mathbf{w}^*$ and $\mathbf{v}^*$ are sometimes called *dual variables* for reasons that will be clear in the next section. They are also sometimes called *Lagrange Multipliers*. You may have encountered Lagrange Multipliers in your Math 230 or Math 231 class. These are the same kind of variables except applied to linear optimization problems. There is one element in the dual variable vector $\mathbf{w}^*$ for each constraint of the form $\mathbf{Ax} \leq \mathbf{b}$ and one element in the dual variable vector $\mathbf{v}^*$ for each constraint of the form $\mathbf{x} \geq \mathbf{0}$.

Example 9.65. Consider a simple Linear Programming Problem with Dual Variables (Lagrange Multipliers) listed next to their corresponding constraints:

$$\begin{cases} \max \ z(x_1,x_2) = 7x_1 + 6x_2 & \textbf{Dual Variable} \\ s.t. \ 3x_1 + x_2 \leq 120 & (w_1) \\ \quad x_1 + 2x_2 \leq 160 & (w_1) \\ \quad x_1 \leq 35 & (w_3) \\ \quad x_1 \geq 0 & (v_1) \\ \quad x_2 \geq 0 & (v_2) \end{cases}$$

In this problem we have:

$$\mathbf{A} = \begin{bmatrix} 3 & 1 \\ 1 & 2 \\ 1 & 0 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 120 \\ 160 \\ 35 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 7 & 6 \end{bmatrix}$$

Then the KKT conditions can be written as:

$$\text{Primal Feasibility} \begin{cases} \begin{bmatrix} 3 & 1 \\ 1 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 120 \\ 160 \\ 35 \end{bmatrix} \\ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{cases}$$

$$\text{Dual Feasibility} \begin{cases} \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 2 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} v_1 & v_2 \end{bmatrix} = \begin{bmatrix} 7 & 6 \end{bmatrix} \\ \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \geq \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} v_1 & v_2 \end{bmatrix} \geq \begin{bmatrix} 0 & 0 \end{bmatrix} \end{cases}$$

$$\text{Complementary Slackness} \begin{cases} \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \left(\begin{bmatrix} 3 & 1 \\ 1 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 120 \\ 160 \\ 35 \end{bmatrix} \right) = 0 \\ \begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} x_1 & x_2 \end{bmatrix} = 0 \end{cases}$$

Note, we are suppressing the slack variables $\mathbf{s}$ in the primal feasibility expression. Recall that at optimality, we had $x_1 = 16$ and $x_2 = 72$. The binding constraints in this case where

$$3x_1 + x_2 \leq 120$$
$$x_1 + 2x_2 \leq 160$$

To see this note that if $3(16) + 72 = 120$ and $16 + 2(72) = 160$. Then we should be able to express $\mathbf{c} = [7 \quad 6]$ (the vector of coefficients of the objective function) as a positive combination of the gradients of the binding constraints:

$$\begin{aligned}
\nabla(7x_1 + 6x_2) &= \begin{bmatrix} 7 & 6 \end{bmatrix} \\
\nabla(3x_1 + x_2) &= \begin{bmatrix} 3 & 1 \end{bmatrix} \\
\nabla(x_1 + 2x_2) &= \begin{bmatrix} 1 & 2 \end{bmatrix}
\end{aligned}$$

That is, we wish to solve the linear equation:

$$\begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 7 & 6 \end{bmatrix} \tag{9.64}$$

The result is the system of equations:

$$\begin{aligned}
3w_1 + w_2 &= 7 \\
w_1 + 2w_2 &= 6
\end{aligned}$$

A solution to this system is $w_1 = \frac{8}{5}$ and $w_2 = \frac{11}{5}$. This fact is illustrated in Figure 9.9.

Figure 9.9 shows the gradient cone formed by the binding constraints at the optimal point for the toy maker problem. Since $x_1, x_2 > 0$, we must have $v_1 = v_2 = 0$. Moreover,

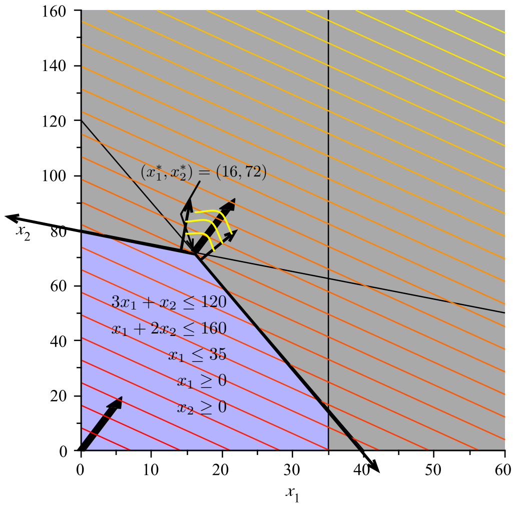


Figure 9.9. The Gradient Cone: At optimality, the cost vector **c** is obtuse with respect to the directions formed by the binding constraints. It is also contained inside the cone of the gradients of the binding constraints, which we will discuss at length later.

since $x_1 < 35$, we know that $x_1 \leq 35$ is not a binding constraint and thus its dual variable w_3 is also zero. This leads to the conclusion:

$$\begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} = \begin{bmatrix} 16 \\ 72 \end{bmatrix} \quad \begin{bmatrix} w_1^* & w_2^* & w_3^* \end{bmatrix} = \begin{bmatrix} 8/5 & 11/5 & 0 \end{bmatrix} \quad \begin{bmatrix} v_1^* & v_2^* \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

and the KKT conditions are satisfied.

EXERCISE 66. Consider the problem:

$$\begin{aligned} \max\ & x_1 + x_2 \\ s.t.\ & 2x_1 + x_2 \leq 4 \\ & x_1 + 2x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Write the KKT conditions for an optimal point for this problem. (You will have a vector $\mathbf{w} = [w_1 \quad w_2]$ and a vector $\mathbf{v} = [v_1 \quad v_2]$).

Draw the feasible region of the problem and use Matlab to solve the problem. At the point of optimality, identify the binding constraints and draw their gradients. Show that the KKT conditions hold. (Specifically find $\mathbf{w}$ and $\mathbf{v}$.)

EXERCISE 67. Find the KKT conditions for the problem:

$$(9.65) \quad \begin{cases} \min\ \mathbf{cx} \\ s.t.\ \mathbf{Ax} \geq \mathbf{b} \\ \quad\ \mathbf{x} \geq \mathbf{0} \end{cases}$$

[Hint: Remember, every minimization problem can be converted to a maximization problem by multiplying the objective function by -1 and the constraints $\mathbf{Ax} \geq \mathbf{b}$ are equivalent to the constraints $-\mathbf{Ax} \leq -\mathbf{b}$.

5. Simplex Initialization

So far we have investigated linear programming problems that had form:

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ s.t.\ & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

In this case, we use slack variables to convert the problem to:

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ s.t.\ & \mathbf{Ax} + \mathbf{I}_m\mathbf{x_s} = \mathbf{b} \\ & \mathbf{x}, \mathbf{x_s} \geq \mathbf{0} \end{aligned}$$

where $\mathbf{x_s}$ are slack variables, one for each constraint. If $\mathbf{b} \geq \mathbf{0}$, then our initial basic feasible solution can be $\mathbf{x} = \mathbf{0}$ and $\mathbf{x_s} = \mathbf{b}$ (that is, our initial basis matrix is $\mathbf{B} = \mathbf{I}_m$).

Suppose now we wish to investigate problems in which we do not have a problem structure that lends itself to easily identifying an initial basic feasible solution. The simplex algorithm requires an initial BFS to begin execution and so we must develop a method for finding such a BFS.

For the remainder of this chapter we will assume, unless told otherwise, that we are interested in solving a linear programming problem provided in Standard Form. That is:

$$(9.66) \quad P \begin{cases} \max\ \mathbf{c}^T\mathbf{x} \\ s.t.\ \mathbf{Ax} = \mathbf{b} \\ \quad\ \mathbf{x} \geq \mathbf{0} \end{cases}$$

and that $\mathbf{b} \geq \mathbf{0}$. Clearly our work in Chapter 3 shows that any linear programming problem can be put in this form.

Suppose to each constraint $\mathbf{A}_{i\cdot}\mathbf{x} = \mathbf{b}_i$ we associate an *artificial variable* x_{a_i}. We can replace constraint i with:

$$\mathbf{A}_{i\cdot}\mathbf{x} + x_{a_i} = \mathbf{b}_i \tag{9.67}$$

Since $\mathbf{b}_i \geq 0$, we will require $x_{a_i} \geq 0$. If $x_{a_i} = 0$, then this is simply the original constraint. Thus if we can find values for the ordinary decision variables $\mathbf{x}$ so that $x_{a_i} = 0$, then constraint i is satisfied. If we can identify values for $\mathbf{x}$ so that all the artificial variables are zero and m variables of $\mathbf{x}$ are non-zero, then the modified constraints described by Equation 9.67 are satisfied and we have identified an initial basic feasible solution.

Obviously, we would like to penalize non-zero artificial variables. This can be done by writing a new linear programming problem:

$$P_1 \begin{cases} \min & \mathbf{e}^T\mathbf{x_a} \\ s.t. & \mathbf{A}\mathbf{x} + \mathbf{I}_m\mathbf{x_a} = \mathbf{b} \\ & \mathbf{x}, \mathbf{x_a} \geq \mathbf{0} \end{cases} \tag{9.68}$$

Remark 9.66. We can see that the artificial variables are similar to slack variables, but they should have zero value because they have no true meaning in the original problem P. They are introduced *artificially* to help identify an initial basic feasible solution to Problem P.

Theorem 9.67. *The optimal objective function value in Problem P_1 is bounded below by 0. Furthermore, if the optimal solution to problem P_1 has $\mathbf{x_a} = \mathbf{0}$, then the values of $\mathbf{x}$ form a feasible solution to Problem P.*

6. Revised Simplex Method

Consider an arbitrary linear programming problem, which we will assume is written in standard form:

$$P \begin{cases} \max & \mathbf{c}^T\mathbf{x} \\ s.t. & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{cases} \tag{9.69}$$

Consider the data we need at each iteration of the Simplex algorithm:

(1) Reduced costs: $\mathbf{c}_\mathbf{B}^T\mathbf{B}^{-1}\mathbf{A}_{\cdot j} - c_j$ for each variable x_j where $j \in \mathcal{J}$ and $\mathcal{J}$ is the set of indices of non-basic variables.
(2) Right-hand-side values: $\overline{\mathbf{b}} = \mathbf{B}^{-1}\mathbf{b}$ for use in the minimum ratio test.
(3) $\overline{\mathbf{a}}_j = \mathbf{B}^{-1}\mathbf{A}_{\cdot j}$ for use in the minimum ratio test.
(4) $\overline{z} = \mathbf{c}_\mathbf{B}^T\mathbf{B}^{-1}\mathbf{b}$, the current objective function value.

The one value that is clearly critical to the computation is $\mathbf{B}^{-1}$ as it appears in each and every computation. It would be far more effective to keep only the values: $\mathbf{B}^{-1}$, $\mathbf{c}_\mathbf{B}^T\mathbf{B}^{-1}$, $\overline{\mathbf{b}}$ and $\overline{z}$ and compute the reduced cost values and vectors $\overline{\mathbf{a}}_j$ as we need them.

Let $\mathbf{w} = \mathbf{c}_\mathbf{B}^T\mathbf{B}^{-1}$, then the pertinent information may be stored in a new *revised simplex tableau* with form:

$$\mathbf{x_B} \left[\begin{array}{c|c} \mathbf{w} & \overline{z} \\ \hline \mathbf{B}^{-1} & \overline{\mathbf{b}} \end{array}\right] \tag{9.70}$$

The revised simplex algorithm is detailed in Algorithm 20. In essence, the revised

Revised Simplex Algorithm

(1) Identify an initial basis matrix $\mathbf{B}$ and compute $\mathbf{B}^{-1}$, $\mathbf{w}$, $\overline{\mathbf{b}}$ and $\overline{z}$ and place these into a revised simplex tableau:

$$\mathbf{x_B}\left[\begin{array}{c|c}\mathbf{w} & \overline{z}\\ \hline \mathbf{B}^{-1} & \overline{\mathbf{b}}\end{array}\right]$$

(2) For each $j \in \mathcal{J}$ use $\mathbf{w}$ to compute: $z_j - c_j = \mathbf{w}\mathbf{A}_{\cdot j} - c_j$.

(3) Choose an entering variable x_j (for a maximization problem, we choose a variable with negative reduced cost, for a minimization problem we choose a variable with positive reduced cost):
   (a) If there is no entering variable, STOP, you are at an optimal solution.
   (b) Otherwise, continue to Step 4.

(4) Append the column $\overline{\mathbf{a}}_j = \mathbf{B}^{-1}\mathbf{A}_{\cdot j}$ to the revised simplex tableau:

$$\mathbf{x_B}\left[\begin{array}{c|c}\mathbf{w} & \overline{z}\\ \hline \mathbf{B}^{-1} & \overline{\mathbf{b}}\end{array}\right]\left[\begin{array}{c}z_j - c_j\\ \hline \overline{\mathbf{a}}_j\end{array}\right]$$

(5) Perform the minimum ratio test and determine a leaving variable (using any leaving variable rule you prefer).
   (a) If $\overline{\mathbf{a}}_j \leq \mathbf{0}$, STOP, the problem is unbounded.
   (b) Otherwise, assume that the leaving variable is x_{B_r} which appears in row r of the revised simplex tableau.

(6) Use *row operations* and pivot on the leaving variable row of the column:

$$\left[\begin{array}{c}z_j - c_j\\ \hline \overline{\mathbf{a}}_j\end{array}\right]$$

transforming the revised simplex tableau into:

$$\mathbf{x}'_{\mathbf{B}}\left[\begin{array}{c|c}\mathbf{w}' & \overline{z}'\\ \hline \mathbf{B}'^{-1} & \overline{\mathbf{b}}'\end{array}\right]\left[\begin{array}{c}0\\ \hline \mathbf{e}_r\end{array}\right]$$

where $\mathbf{e}_r$ is an identity column with a 1 in row r (the row that left). The variable x_j is now the r^{th} element of $\mathbf{x_B}$.

(7) Goto Step 2.

Algorithm 20. Revised Simplex Algorithm

simplex algorithm allows us to avoid computing $\overline{\mathbf{a}}_j$ until we absolutely need to do so. In fact, if we do not apply Dantzig's entering variable rule and simply select the first acceptable entering variable, then we may be able to avoid computing a substantial number of columns in the tableau.

Example 9.68. Consider a software company who is developing a new program. The company has identified two types of bugs that remain in this software: *non-critical* and *critical*. The company's actuarial firm predicts that the risk associated with these bugs are uniform random variables with mean \$100 per non-critical bug and mean \$1000 per critical bug. The software currently has 50 non-critical bugs and 5 critical bugs.

Assume that it requires 3 hours to fix a non-critical bug and 12 hours to fix a critical bug. For each day (8 hour period) beyond two business weeks (80 hours) that the company fails to ship its product, the actuarial firm estimates it will loose \$500 per day.

We can find the optimal number of bugs of each type the software company should fix assuming it wishes to minimize its exposure to risk using a linear programming formulation.

Let x_1 be the number of non-critical bugs corrected and x_2 be the number of critical software bugs corrected. Define:

$$y_1 = 50 - x_1 \tag{9.71}$$

$$y_2 = 5 - x_2 \tag{9.72}$$

Here y_1 is the number of non-critical bugs that are not fixed while y_2 is the number of critical bugs that are not fixed.

The time (in hours) it takes to fix these bugs is:

$$3x_1 + 12x_2 \tag{9.73}$$

Let:

$$y_3 = \frac{1}{8}\left(80 - 3x_1 - 12x_2\right) \tag{9.74}$$

Then y_3 is a variable that is unrestricted in sign and determines the amount of time (in days) either over or under the two-week period that is required to ship the software. As an unrestricted variable, we can break it into two components:

$$y_3 = z_1 - z_2 \tag{9.75}$$

We will assume that $z_1, z_2 \geq 0$. If $y_3 > 0$, then $z_1 > 0$ and $z_2 = 0$. In this case, the software is completed ahead of the two-week deadline. If $y_3 < 0$, then $z_1 = 0$ and $z_2 > 0$. In this case the software is finished after the two-week deadline. Finally, if $y_3 = 0$, then $z_1 = z_2 = 0$ and the software is finished precisely on time.

We can form the objective function as:

$$z = 100y_1 + 1000y_2 + 500z_2 \tag{9.76}$$

The linear programming problem is then:

$$\begin{aligned}
\min\ \ z = & y_1 + 10y_2 + 5z_2\\
s.t.\ \ & x_1 + y_1 = 50\\
& x_2 + y_2 = 5\\
& \frac{3}{8}x_1 + \frac{3}{2}x_2 + z_1 - z_2 = 10\\
& x_1, x_2, y_1, y_2, z_1, z_2 \geq 0
\end{aligned} \tag{9.77}$$

Notice we have modified the objective function by dividing by 100. This will make the arithmetic of the simplex algorithm easier. The matrix of coefficients for this problem is:

$$\left[\begin{array}{cccccc}
x_1 & x_2 & y_1 & y_2 & z_1 & z_2\\
\hline
1 & 0 & 1 & 0 & 0 & 0\\
0 & 1 & 0 & 1 & 0 & 0\\
\frac{3}{8} & \frac{3}{2} & 0 & 0 & 1 & -1
\end{array}\right] \tag{9.78}$$

Notice there is an identity matrix embedded inside the matrix of coefficients. Thus a good initial basic feasible solution is $\{y_1, y_2, z_1\}$. The initial basis matrix is $\mathbf{I}_3$ and naturally, $\mathbf{B}^{-1} = \mathbf{I}_3$ as a result. We can see that $\mathbf{c_B} = [1\ 10\ 0]^T$. It follows that $\mathbf{c_B}^T\mathbf{B}^{-1} = \mathbf{w} = [1\ 10\ 0]$.

Our initial revised simplex tableau is thus:

$$\begin{array}{c} z \\ y_1 \\ y_2 \\ z_1 \end{array} \left[\begin{array}{ccc|c} 1 & 10 & 0 & 100 \\ \hline 1 & 0 & 0 & 50 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 10 \end{array}\right] \tag{9.79}$$

There are three variables that might enter at this point, x_1, x_2 and z_1. We can compute the reduced costs for each of these variables using the columns of the $\mathbf{A}$ matrix, the coefficients of these variables in the objective function and the current $\mathbf{w}$ vector (in row 0 of the revised simplex tableau). We obtain:

$$z_1 - c_1 = \mathbf{w}\mathbf{A}_{\cdot 1} - c_1 = \begin{bmatrix} 1 & 10 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 3/8 \end{bmatrix} - 0 = 1$$

$$z_2 - c_2 = \mathbf{w}\mathbf{A}_{\cdot 2} - c_2 = \begin{bmatrix} 1 & 10 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 3/2 \end{bmatrix} - 0 = 10$$

$$z_6 - c_6 = \mathbf{w}\mathbf{A}_{\cdot 6} - c_6 = \begin{bmatrix} 1 & 10 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} - 5 = -5$$

By Dantzig's rule, we enter variable x_2. We append $\mathbf{B}^{-1}\mathbf{A}_{\cdot 2}$ and the reduced cost to the revised simplex tableau to obtain:

$$\begin{array}{c} z \\ y_1 \\ y_2 \\ z_1 \end{array} \left[\begin{array}{ccc|c} 1 & 10 & 0 & 100 \\ \hline 1 & 0 & 0 & 50 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 10 \end{array}\right] \left[\begin{array}{c} 10 \\ \hline 0 \\ \boxed{1} \\ 3/2 \end{array}\right] \begin{array}{c} MRT \\ - \\ 5 \\ 20/3 \end{array} \tag{9.80}$$

After pivoting on the indicated element, we obtain the new tableau:

$$\begin{array}{c} z \\ y_1 \\ x_2 \\ z_1 \end{array} \left[\begin{array}{ccc|c} 1 & 0 & 0 & 50 \\ \hline 1 & 0 & 0 & 50 \\ 0 & 1 & 0 & 5 \\ 0 & -3/2 & 1 & 5/2 \end{array}\right] \tag{9.81}$$

We can compute reduced costs for the non-basic variables (except for y_2, which we know will **not** re-enter the basis on this iteration) to obtain:

$$z_1 - c_1 = \mathbf{w}\mathbf{A}_{\cdot 1} - c_1 = 1$$
$$z_6 - c_6 = \mathbf{w}\mathbf{A}_{\cdot 6} - c_6 = -5$$

In this case, x_1 will enter the basis and we augment our revised simplex tableau to obtain:

$$\begin{array}{c} z \\ y_1 \\ x_2 \\ z_1 \end{array} \left[\begin{array}{ccc|c} 1 & 0 & 0 & 50 \\ \hline 1 & 0 & 0 & 50 \\ 0 & 1 & 0 & 5 \\ 0 & -3/2 & 1 & 5/2 \end{array}\right] \left[\begin{array}{c} 1 \\ \hline 1 \\ 0 \\ 3/8 \end{array}\right] \begin{array}{c} MRT \\ 50 \\ - \\ \boxed{20/3} \end{array} \tag{9.82}$$

Note that:

$$\mathbf{B}^{-1}\mathbf{A}_{\cdot 1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3/2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 3/8 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 3/8 \end{bmatrix}$$

This is the $\bar{\mathbf{a}}_1$ column that is appended to the right hand side of the tableau along with $z_1 - c_1 = 1$. After pivoting, the tableau becomes:

$$\begin{array}{c} z \\ y_1 \\ x_2 \\ x_1 \end{array} \left[\begin{array}{ccc|c} 1 & 4 & -8/3 & 130/3 \\ \hline 1 & 4 & -8/3 & 130/3 \\ 0 & 1 & 0 & 5 \\ 0 & -4 & 8/3 & 20/3 \end{array}\right] \tag{9.83}$$

We can now check our reduced costs. Clearly, z_1 will not re-enter the basis. Therefore, we need only examine the reduced costs for the variables y_2 and z_2.

$$z_4 - c_4 = \mathbf{w}\mathbf{A}_{\cdot 4} - c_4 = -6$$
$$z_6 - c_6 = \mathbf{w}\mathbf{A}_{\cdot 6} - c_6 = -7/3$$

Since all reduced costs are now negative, no further minimization is possible and we conclude we have arrived at an optimal solution.

Two things are interesting to note: first, the solution for the number of non-critical software bugs to fix is non-integer. Thus, in reality the company must fix either 6 or 7 of the non-critical software bugs. The second thing to note is that this economic model helps to explain why some companies are content to release software that contains known bugs. In making a choice between releasing a flawless product or making a quicker (larger) profit, a selfish, profit maximizer will always choose to fix only those bugs it must fix and release sooner rather than later.

Exercise 68. Solve the following problem using the revised simplex algorithm.

$$\begin{aligned} \max\ & x_1 + x_2 \\ s.t.\ & 2x_1 + x_2 \leq 4 \\ & x_1 + 2x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

7. Cycling Prevention

Theorem 9.69. *Consider Problem P (our linear programming problem). Let $\mathbf{B} \in \mathbb{R}^{m\times m}$ be a basis matrix corresponding to some set of basic variables $\mathbf{x_B}$. Let $\overline{\mathbf{b}} = \mathbf{B}^{-1}\mathbf{b}$. If $\overline{\mathbf{b}}_j = \mathbf{0}$ for some $j = 1, \dots, m$, then $\mathbf{x}_B = \overline{\mathbf{b}}$ and $\mathbf{x_N} = \mathbf{0}$ is a degenerate extreme point of the feasible region of Problem P.*

Degeneracy can cause us to take extra steps on our way from an initial basic feasible solution to an optimal solution. When the simplex algorithm takes extra steps while remaining at the same degenerate extreme point, this is called *stalling*. The problem can become much worse; for certain entering variable rules, the simplex algorithm can become locked in a cycle of pivots each one moving from one characterization of a degenerate extreme point to the next. The following example from Beale and illustrated in Chapter 4 of [BJS04] demonstrates the point.

Remark 9.70. In this section, for the sake of simplicity, we illustrate cycling with full tableau. That is, we show the entire table of reduced costs as the top row and the entire matrix $\mathbf{B}^{-1}\mathbf{A}$. The right-hand-side and $\overline{z}$ values are identical to the revised simplex method.

Example 9.71. Consider the following linear programming problem:

$$(9.84)\quad \begin{aligned} \min\quad & -\frac{3}{4}x_4 + 20x_5 - \frac{1}{2}x_6 + 6x_7 \\ s.t\quad & x_1 + \frac{1}{4}x_4 - 8x_5 - x_6 + 9x_7 = 0 \\ & x_2 + \frac{1}{2}x_4 - 12x_5 - \frac{1}{2}x_6 + 3x_7 = 0 \\ & x_3 + x_6 = 1 \\ & x_i \geq 0 \;\; i = 1, \dots, 7 \end{aligned}$$

It is conducive to analyze the $\mathbf{A}$ matrix of the constraints of this problem. We have:

$$(9.85)\quad \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 1/4 & -8 & -1 & 9 \\ 0 & 1 & 0 & 1/2 & -12 & -1/2 & 3 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The fact that the $\mathbf{A}$ matrix contains an identity matrix embedded within it suggests that an initial basic feasible solution with basic variables x_1, x_2 and x_3 would be a good choice. This leads to a vector of reduced costs given by:

$$(9.86)\quad \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{N} - \mathbf{c_N}^T = \begin{bmatrix} 3/4 & -20 & 1/2 & -6 \end{bmatrix}$$

These yield an initial tableau with structure:

$$\begin{array}{c} \\ z \\ x_1 \\ x_2 \\ x_3 \end{array} \left[\begin{array}{c|ccccccc|c} z & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & RHS \\ \hline 1 & 0 & 0 & 0 & 3/4 & -20 & 1/2 & -6 & 0 \\ \hline 0 & 1 & 0 & 0 & 1/4 & -8 & -1 & 9 & 0 \\ 0 & 0 & 1 & 0 & 1/2 & -12 & -1/2 & 3 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right]$$

If we apply an entering variable rule where we always chose the non-basic variable to enter with the *most positive* reduced cost (since this is a minimization problem), and we choose the leaving variable to be the first row that is in a tie, then we will obtain the following sequence of tableaux:

Tableau I:

$$\begin{array}{c} \\ z \\ x_1 \\ x_2 \\ x_3 \end{array} \left[\begin{array}{c|ccccccc|c} z & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & RHS \\ \hline 1 & 0 & 0 & 0 & 3/4 & -20 & 1/2 & -6 & 0 \\ \hline 0 & 1 & 0 & 0 & \boxed{1/4} & -8 & -1 & 9 & 0 \\ 0 & 0 & 1 & 0 & 1/2 & -12 & -1/2 & 3 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right]$$

Tableau II:

$$\begin{array}{c} \\ z \\ x_4 \\ x_2 \\ x_3 \end{array} \left[\begin{array}{c|ccccccc|c} z & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & RHS \\ \hline 1 & -3 & 0 & 0 & 0 & 4 & 7/2 & -33 & 0 \\ \hline 0 & 4 & 0 & 0 & 1 & -32 & -4 & 36 & 0 \\ 0 & -2 & 1 & 0 & 0 & \boxed{4} & 3/2 & -15 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right]$$

Tableau III:

$$\begin{array}{c|c|ccccccc|c}
 & z & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & RHS \\ \hline
z & 1 & -1 & -1 & 0 & 0 & 0 & 2 & -18 & 0 \\ \hline
x_4 & 0 & -12 & 8 & 0 & 1 & 0 & \boxed{8} & -84 & 0 \\
x_5 & 0 & -1/2 & 1/4 & 0 & 0 & 1 & 3/8 & -15/4 & 0 \\
x_3 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1
\end{array}$$

Tableau IV:

$$\begin{array}{c|c|ccccccc|c}
 & z & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & RHS \\ \hline
z & 1 & 2 & -3 & 0 & -1/4 & 0 & 0 & 3 & 0 \\ \hline
x_6 & 0 & -3/2 & 1 & 0 & 1/8 & 0 & 1 & -21/2 & 0 \\
x_5 & 0 & 1/16 & -1/8 & 0 & -3/64 & 1 & 0 & \boxed{3/16} & 0 \\
x_3 & 0 & 3/2 & -1 & 1 & -1/8 & 0 & 0 & 21/2 & 1
\end{array}$$

Tableau V:

$$\begin{array}{c|c|ccccccc|c}
 & z & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & RHS \\ \hline
z & 1 & 1 & -1 & 0 & 1/2 & -16 & 0 & 0 & 0 \\ \hline
x_6 & 0 & \boxed{2} & -6 & 0 & -5/2 & 56 & 1 & 0 & 0 \\
x_7 & 0 & 1/3 & -2/3 & 0 & -1/4 & 16/3 & 0 & 1 & 0 \\
x_3 & 0 & -2 & 6 & 1 & 5/2 & -56 & 0 & 0 & 1
\end{array}$$

Tableau VI:

$$\begin{array}{c|c|ccccccc|c}
 & z & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & RHS \\ \hline
z & 1 & 0 & 2 & 0 & 7/4 & -44 & -1/2 & 0 & 0 \\ \hline
x_1 & 0 & 1 & -3 & 0 & -5/4 & 28 & 1/2 & 0 & 0 \\
x_7 & 0 & 0 & \boxed{1/3} & 0 & 1/6 & -4 & -1/6 & 1 & 0 \\
x_3 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1
\end{array}$$

Tableau VII:

$$\begin{array}{c|c|ccccccc|c}
 & z & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & RHS \\ \hline
z & 1 & 0 & 0 & 0 & 3/4 & -20 & 1/2 & -6 & 0 \\ \hline
x_1 & 0 & 1 & 0 & 0 & 1/4 & -8 & -1 & 9 & 0 \\
x_2 & 0 & 0 & 1 & 0 & 1/2 & -12 & -1/2 & 3 & 0 \\
x_3 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1
\end{array}$$

We see that the last tableau (VII) is the same as the first tableau and thus we have constructed an instance where (using the given entering and leaving variable rules), the Simplex Algorithm will cycle forever at this degenerate extreme point.

7.1. The Lexicographic Minimum Ratio Leaving Variable Rule. Given the example of the previous section, we require a method for breaking ties in the case of degeneracy is required that prevents cycling from occurring. There is a large literature on cycling prevention rules, however the most well known is the *lexicographic rule for selecting the entering variable.*

DEFINITION 9.72 (Lexicographic Order). Let $\mathbf{x} = [x_1, \dots, x_n]^T$ and $\mathbf{y} = [y_1, \dots, y_n]^T$ be vectors in $\mathbb{R}^n$. We say that $\mathbf{x}$ is *lexicographically greater* than $\mathbf{y}$ if: there exists $m < n$ so that $x_i = y_i$ for $i = 1, \dots, m$, and $x_{m+1} > y_{m+1}$.

Clearly, if there is no such $m < n$, then $x_i = y_i$ for $i = 1, \dots, n$ and thus $\mathbf{x} = \mathbf{y}$. We write $\mathbf{x} \succ \mathbf{y}$ to indicate that $\mathbf{x}$ is lexicographically greater than $\mathbf{y}$. Naturally, we can write $\mathbf{x} \succeq \mathbf{y}$ to indicate that $\mathbf{x}$ is lexicographically greater than or equal to $\mathbf{y}$.

Lexicographic ordering is simply the standard order operation $>$ applied to the individual elements of a vector in $\mathbb{R}^n$ with a precedence on the index of the vector.

Definition 9.73. A vector $\mathbf{x} \in \mathbb{R}^n$ is *lexicographically positive* if $\mathbf{x} \succ \mathbf{0}$ where $\mathbf{0}$ is the zero vector in $\mathbb{R}^n$.

Lemma 9.74. *Let $\mathbf{x}$ and $\mathbf{y}$ be two lexicographically positive vectors in $\mathbb{R}^n$. Then $\mathbf{x}+\mathbf{y}$ is lexicographically positive. Let $c > 0$ be a constant in $\mathbb{R}$, then $c\mathbf{x}$ is a lexicographically positive vector.*

Exercise 69. Prove Lemma 9.74.

Suppose we are considering a linear programming problem and we have chosen an entering variable x_j according to a fixed entering variable rule. Assume further, we are given some current basis matrix $\mathbf{B}$ and as usual, the right-hand-side vector of the constraints is denoted $\mathbf{b}$, while the coefficient matrix is denoted $\mathbf{A}$. Then the minimum ratio test asserts that we will chose as the leaving variable the basis variable with the minimum ratio in the minimum ratio test. Consider the following set:

$$I_0 = \left\{ r : \frac{\overline{b}_r}{\overline{a}_{j_r}} = \min \left[\frac{\overline{b}_i}{\overline{a}_{j_i}} : i = 1, \dots, m \text{ and } a_{j_i} > 0 \right] \right\} \tag{9.87}$$

In the absence of degeneracy, I_0 contains a *single* element: the row index that has the smallest ratio of $\overline{b}_i$ to $\overline{a}_{j_i}$, where naturally: $\overline{\mathbf{b}} = \mathbf{B}^{-1}\mathbf{b}$ and $\overline{\mathbf{a}}_j = \mathbf{B}^{-1}\mathbf{A}_{\cdot j}$. In this case, x_j is swapped into the basis in exchange for x_{B_r} (the r^{th} basic variable).

When we have a degenerate basic feasible solution, then I_0 is not a singleton set and contains all the rows that have *tied* in the minimum ratio test. In this case, we can form a new set:

$$I_1 = \left\{ r : \frac{\overline{a}_{1_r}}{\overline{a}_{j_r}} = \min \left[\frac{\overline{a}_{1_i}}{\overline{a}_{j_i}} : i \in I_0 \right] \right\} \tag{9.88}$$

Here, we are taking the elements in column 1 of $\mathbf{B}^{-1}\mathbf{A}_{\cdot 1}$ to obtain $\overline{\mathbf{a}}_1$. The elements of this (column) vector are then being divided by the elements of the (column) vector $\overline{\mathbf{a}}_j$ on a index-by-index basis. If this set is a singleton, then basic variable x_{B_r} leaves the basis. If this set is not a singleton, we may form a new set I_2 with column $\overline{\mathbf{a}}_2$. In general, we will have the set:

$$I_k = \left\{ r : \frac{\overline{a}_{k_r}}{\overline{a}_{j_r}} = \min \left[\frac{\overline{a}_{k_i}}{\overline{a}_{j_i}} : i \in I_{k-1} \right] \right\} \tag{9.89}$$

Lemma 9.75. *For any degenerate basis matrix $\mathbf{B}$ for any linear programming problem, we will ultimately find a k so that I_k is a singleton.*

Exercise 70. Prove Lemma 9.75. [Hint: Assume that the tableau is arranged so that the identity columns are columns 1 through m. (That is $\overline{\mathbf{a}}_j = \mathbf{e}_j$ for $i = 1, \dots, m$.) Show that this configuration will easily lead to a singleton I_k for $k < m$.]

In executing the lexicographic minimum ratio test, we can see that we are essentially comparing the tied rows in a lexicographic manner. If a set of rows ties in the minimum ratio test, then we execute a minimum ratio test on the first column of the tied rows. If there is a tie, then we move on executing a minimum ratio test on the second column of the rows that tied in both previous tests. This continues until the tie is broken and a single row emerges as the leaving row.

EXAMPLE 9.76. Let us consider the example from Beale again using the lexicographic minimum ratio test. Consider the tableau shown below.

Tableau I:

$$\begin{array}{c} \\ z \\ x_1 \\ x_2 \\ x_3 \end{array}\left[\begin{array}{c|ccccccc|c} z & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & RHS \\ \hline 1 & 0 & 0 & 0 & 3/4 & -20 & 1/2 & -6 & 0 \\ \hline 0 & 1 & 0 & 0 & 1/4 & -8 & -1 & 9 & 0 \\ 0 & 0 & 1 & 0 & \boxed{1/2} & -12 & -1/2 & 3 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right]$$

Again, we chose to enter variable x_4 as it has the most positive reduced cost. Variables x_1 and x_2 tie in the minimum ratio test. So we consider a new minimum ratio test on the first column of the tableau:

$$\min\left\{\frac{1}{1/4}, \frac{0}{1/2}\right\} \tag{9.90}$$

From this test, we see that x_2 is the leaving variable and we pivot on element $1/2$ as indicated in the tableau. Note, we *only* need to execute the minimum ratio test on variables x_1 and x_2 since those were the tied variables in the standard minimum ratio test. That is, $I_0 = \{1, 2\}$ and we construct I_1 from these indexes alone. In this case $I_1 = \{2\}$. Pivoting yields the new tableau:

Tableau II:

$$\begin{array}{c} \\ z \\ x_1 \\ x_4 \\ x_3 \end{array}\left[\begin{array}{c|ccccccc|c} z & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & RHS \\ \hline 1 & 0 & -3/2 & 0 & 0 & -2 & 5/4 & -21/2 & 0 \\ \hline 0 & 1 & -1/2 & 0 & 0 & -2 & -3/4 & 15/2 & 0 \\ 0 & 0 & 2 & 0 & 1 & -24 & -1 & 6 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \boxed{1} & 0 & 1 \end{array}\right]$$

There is no question this time of the entering or leaving variable, clearly x_6 must enter and x_3 must leave and we obtain[5]:

Tableau III:

$$\begin{array}{c} \\ z \\ x_1 \\ x_4 \\ x_6 \end{array}\left[\begin{array}{c|ccccccc|c} z & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & RHS \\ \hline 1 & 0 & -3/2 & -5/4 & 0 & -2 & 0 & -21/2 & -5/4 \\ \hline 0 & 1 & -1/2 & 3/4 & 0 & -2 & 0 & 15/2 & 3/4 \\ 0 & 0 & 2 & 1 & 1 & -24 & 0 & 6 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right]$$

Since this is a minimization problem and the reduced costs of the non-basic variables are now all negative, we have arrived at an optimal solution. The lexicographic minimum ratio test successfully prevented cycling.

REMARK 9.77. The following theorem is proved in [Gri11].

[5]Thanks to Ethan Wright for finding a small typo in this example, that is now fixed.

Theorem 9.78. *Consider the problem:*

$$P\begin{cases}\max & \mathbf{c}^T\mathbf{x}\\ s.t. & \mathbf{A}\mathbf{x}=\mathbf{b}\\ & \mathbf{x}\geq\mathbf{0}\end{cases}$$

Suppose the following hold:

(1) $\mathbf{I}_m$ *is embedded in the matrix* $\mathbf{A}$ *and is used as the starting basis,*
(2) *a consistent entering variable rule is applied (e.g., largest reduced cost first), and*
(3) *the lexicographic minimum ratio test is applied as the leaving variable rule.*

Then the simplex algorithm converges in a finite number of steps.

8. Relating the KKT Conditions to the Tableau

Consider a linear programming problem in Standard Form:

$$P\begin{cases}\max & \mathbf{c}\mathbf{x}\\ s.t. & \mathbf{A}\mathbf{x}=\mathbf{b}\\ & \mathbf{x}\geq\mathbf{0}\end{cases}\tag{9.91}$$

with $\mathbf{A}\in\mathbb{R}^{m\times n}$, $\mathbf{b}\in\mathbb{R}^m$ and (row vector) $\mathbf{c}\in\mathbb{R}^n$.

The KKT conditions for a problem of this type assert that

$$\mathbf{w}\mathbf{A}-\mathbf{v}=\mathbf{c}$$
$$\mathbf{v}\mathbf{x}=0$$

at an optimal point $\mathbf{x}$ for some vector $\mathbf{w}$ unrestricted in sign and $\mathbf{v}\geq\mathbf{0}$. (Note, for the sake of notational ease, we have dropped the * notation.)

Suppose at optimality we have a basis matrix $\mathbf{B}$ corresponding to a set of basic variables $\mathbf{x_B}$ and we simultaneously have non-basic variables $\mathbf{x_N}$. We may likewise divide $\mathbf{v}$ into $\mathbf{v_B}$ and $\mathbf{v_N}$.

Then we have:

$$\mathbf{w}\mathbf{A}-\mathbf{v}=\mathbf{c}\implies\mathbf{w}\begin{bmatrix}\mathbf{B} & \mathbf{N}\end{bmatrix}-\begin{bmatrix}\mathbf{v_B} & \mathbf{v_N}\end{bmatrix}=\begin{bmatrix}\mathbf{c_B} & \mathbf{c_N}\end{bmatrix}\tag{9.92}$$

$$\mathbf{v}\mathbf{x}=0\implies\begin{bmatrix}\mathbf{v_B} & \mathbf{v_N}\end{bmatrix}\begin{bmatrix}\mathbf{x_B}\\ \mathbf{x_N}\end{bmatrix}=0\tag{9.93}$$

We can rewrite Expression 9.92 as:

$$\begin{bmatrix}\mathbf{w}\mathbf{B}-\mathbf{v_B} & \mathbf{w}\mathbf{N}-\mathbf{v_N}\end{bmatrix}=\begin{bmatrix}\mathbf{c_B} & \mathbf{c_N}\end{bmatrix}\tag{9.94}$$

This simplifies to:

$$\mathbf{w}\mathbf{B}-\mathbf{v_B}=\mathbf{c_B}$$
$$\mathbf{w}\mathbf{N}-\mathbf{v_N}=\mathbf{c_N}$$

Let $\mathbf{w}=\mathbf{c_B}\mathbf{B}^{-1}$. Then we see that:

$$\mathbf{w}\mathbf{B}-\mathbf{v_B}=\mathbf{c_B}\implies\mathbf{c_B}\mathbf{B}^{-1}\mathbf{B}-\mathbf{v_B}=\mathbf{c_B}\implies\mathbf{c_B}-\mathbf{v_B}=\mathbf{c_B}\implies\mathbf{v_B}=\mathbf{0}\tag{9.95}$$

Since we know that $\mathbf{x_B} \geq \mathbf{0}$, we know that $\mathbf{v_B}$ should be equal to zero to ensure complementary slackness. Thus, this is consistent with the KKT conditions.

We further see that:

$$\mathbf{wN} - \mathbf{v_N} = \mathbf{c_N} \implies \mathbf{c_B}\mathbf{B}^{-1}\mathbf{N} - \mathbf{v_N} = \mathbf{c_N} \implies \mathbf{v_N} = \mathbf{c_B}\mathbf{B}^{-1}\mathbf{N} - \mathbf{c_N} \tag{9.96}$$

Thus, the $\mathbf{v_N}$ are just the reduced costs of the non-basic variables. ($\mathbf{v_B}$ are the reduced costs of the basic variables.) Furthermore, dual feasibility requires that $\mathbf{v} \geq \mathbf{0}$. Thus we see that at optimality we require:

$$\mathbf{c_B}\mathbf{B}^{-1}\mathbf{N} - \mathbf{c_N} \geq \mathbf{0} \tag{9.97}$$

This is *precisely* the condition for optimality in the simplex tableau.

We now can see the following facts are true about the Simplex Method:

(1) At each iteration of the Simplex Method, primal feasibility is satisfied. This is ensured by the minimum ratio test and the fact that we start at a feasible point.
(2) At each iteration of the Simplex Method, complementary slackness is satisfied. After all, the vector $\mathbf{v}$ is just the reduced cost vector (Row 0) of the Simplex tableau. If a variable is basic x_j (and hence non-zero), then the its reduced cost $v_j = 0$. Otherwise, v_j may be non-zero.
(3) At each iteration of the Simplex Algorithm, we may violate dual feasibility because we may not have $\mathbf{v} \geq \mathbf{0}$. It is *only* at optimality that we achieve dual feasibility and satisfy the KKT conditions.

We can now prove the following theorem:

Theorem 9.79. *Assuming an appropriate cycling prevention rule is used, the simplex algorithm converges in a finite number of iterations to an optimal solution to the linear programming problem.*

Proof. Convergence is guaranteed by the proof of Theorem 9.78 in which we show that when the lexicographic minimum ratio test is used, then the simplex algorithm will always converge. Our work above shows that at optimality, the KKT conditions are satisfied because the termination criteria for the simplex algorithm are precisely the same as the criteria in the Karush-Kuhn-Tucker conditions. This completes the proof. □

9. Duality

Remark 9.80. In this section, we show that to each linear programming problem (the primal problem) we may associate another linear programming problem (the dual linear programming problem). These two problems are closely related to each other and an analysis of the dual problem can provide deep insight into the primal problem.

Consider the linear programming problem

$$P \begin{cases} \max & \mathbf{c}^T\mathbf{x} \\ s.t. & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{cases} \tag{9.98}$$

Then the dual problem for Problem P is:

$$
(9.99) \quad D \begin{cases} \min & \mathbf{wb} \\ s.t. & \mathbf{wA} \geq \mathbf{c} \\ & \mathbf{w} \geq \mathbf{0} \end{cases}
$$

Remark 9.81. Let $\mathbf{v}$ be a vector of *surplus* variables. Then we can transform Problem D into standard form as:

$$
(9.100) \quad D_S \begin{cases} \min & \mathbf{wb} \\ s.t. & \mathbf{wA} - \mathbf{v} = \mathbf{c} \\ & \mathbf{w} \geq \mathbf{0} \\ & \mathbf{v} \geq \mathbf{0} \end{cases}
$$

Thus we already see an intimate relationship between duality and the KKT conditions. The feasible region of the dual problem (in standard form) is precisely the the dual feasibility constraints of the KKT conditions for the primal problem.

In this formulation, we see that we have assigned a dual variable w_i $(i = 1, \dots, m)$ to each constraint in the system of equations $\mathbf{Ax} \leq \mathbf{b}$ of the primal problem. Likewise dual variables $\mathbf{v}$ can be thought of as corresponding to the constraints in $\mathbf{x} \geq \mathbf{0}$.

Remark 9.82. The proof of the following lemma is outside the scope of the class, but it establishes an important fact about duality.

Lemma 9.83. *The dual of the dual problem is the primal problem.* □

Remark 9.84. Lemma 9.83 shows that the notion of dual and primal can be exchanged and that it is simply a matter of perspective which problem is the dual problem and which is the primal problem. Likewise, by transforming problems into canonical form, we can develop dual problems for any linear programming problem.

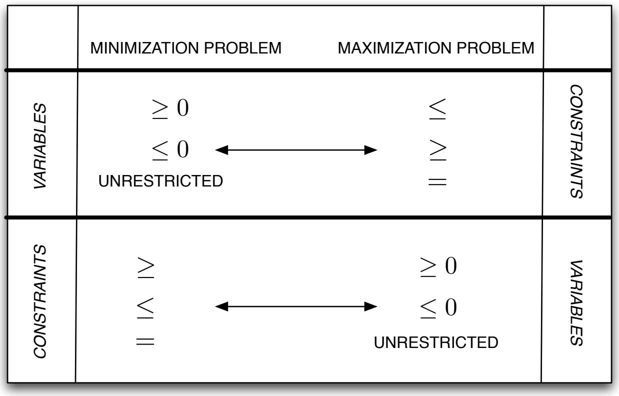
The process of developing these formulations can be exceptionally tedious, as it requires enumeration of all the possible combinations of various linear and variable constraints. The following table summarizes the process of converting an arbitrary primal problem into its dual. This table can be found in Chapter 6 of [BJS04].

Example 9.85. Consider the problem of finding the dual problem when the primal problem is:

$$
\begin{aligned}
\max\ & 7x_1 + 6x_2 \\
s.t.\ & 3x_1 + x_2 + s_1 = 120 && (w_1) \\
& x_1 + 2x_2 + s_2 = 160 && (w_2) \\
& x_1 + s_3 = 35 && (w_3) \\
& x_1, x_2, s_1, s_2, s_3 \geq 0
\end{aligned}
$$

Here we have placed dual variable names (w_1, w_2 and w_3) next to the constraints to which they correspond.

The primal problem variables in this case are all positive, so using Table 1 we know that the constraints of the dual problem will be greater-than-or-equal-to constraints. Likewise, we know that the dual variables will be unrestricted in sign since the primal problem constraints are all equality constraints.



	MINIMIZATION PROBLEM	MAXIMIZATION PROBLEM	
VARIABLES	≥ 0	$\leq$	CONSTRAINTS
	≤ 0	$\geq$	
	UNRESTRICTED	$=$	
CONSTRAINTS	$\geq$	≥ 0	VARIABLES
	$\leq$	≤ 0	
	$=$	UNRESTRICTED	

Table 1. Table of Dual Conversions: To create a dual problem, assign a dual variable to each constraint of the form $\mathbf{A}\mathbf{x} \circ \mathbf{b}$, where $\circ$ represents a binary relation. Then use the table to determine the appropriate sign of the inequality in the dual problem as well as the nature of the dual variables.

The coefficient matrix is:

$$\mathbf{A} = \begin{bmatrix} 3 & 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Clearly we have:

$$\mathbf{c} = \begin{bmatrix} 7 & 6 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 120 \\ 160 \\ 35 \end{bmatrix}$$

Since $\mathbf{w} = [w_1 \;\; w_2 \;\; w_3]$, we know that $\mathbf{wA}$ will be:

$$\mathbf{wA} = \begin{bmatrix} 3w_1 + w_2 + w_3 & w_1 + 2w_2 & w_1 & w_2 & w_3 \end{bmatrix}$$

This vector will be related to $\mathbf{c}$ in the constraints of the dual problem. **Remember, in this case, all variables in the primal problem are greater-than-or-equal-to zero**. Thus we see that the constraints of the dual problem are:

$$\begin{aligned} 3w_1 + w_2 + w_3 &\geq 7 \\ w_1 + 2w_2 &\geq 6 \\ w_1 &\geq 0 \\ w_2 &\geq 0 \\ w_3 &\geq 0 \end{aligned}$$

We also have the redundant set of constraints that tell us $\mathbf{w}$ is unrestricted because the primal problem had equality constraints. This *will always* happen in cases when you've introduced slack variables into a problem to put it in standard form. This should be clear from the definition of the dual problem for a maximization problem in canonical form.

Thus the whole dual problem becomes:

$$
\begin{aligned}
\min\ & 120w_1 + 160w_2 + 35w_3 \\
s.t.\ & 3w_1 + w_2 + w_3 \geq 7 \\
& w_1 + 2w_2 \geq 6 \\
& w_1 \geq 0 \\
& w_2 \geq 0 \\
& w_3 \geq 0 \\
& \mathbf{w} \quad \text{unrestricted}
\end{aligned}
\tag{9.101}
$$

Again, note that in reality, the constraints we derived from the $\mathbf{wA} \geq \mathbf{c}$ part of the dual problem make the constraints "$\mathbf{w}$ unrestricted" redundant, for in fact $\mathbf{w} \geq \mathbf{0}$ just as we would expect it to be if we'd found the dual of the Toy Maker problem given in canonical form.

Exercise 71. Identify the dual problem for:

$$
\begin{aligned}
\max\ & x_1 + x_2 \\
s.t.\ & 2x_1 + x_2 \geq 4 \\
& x_1 + 2x_2 \leq 6 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

Exercise 72. Use the table or the definition of duality to determine the dual for the problem:

$$
\begin{cases}
\min & \mathbf{cx} \\
s.t. & \mathbf{Ax} \geq \mathbf{b} \\
& \mathbf{x} \geq \mathbf{0}
\end{cases}
\tag{9.102}
$$

Remark 9.86. The following theorems are outside the scope of this course, but they can be useful to know and will help cement your understanding of the true nature of duality.

Theorem 9.87 (Strong Duality Theorem). *Consider Problem P and Problem D. Then*

(Weak Duality): $\mathbf{cx}^* \leq \mathbf{w}^*\mathbf{b}$, *thus every feasible solution to the primal problem provides a lower bound for the dual and every feasible solution to the dual problem provides an upper bound to the primal problem.*

Furthermore exactly one of the following statements is true:

(1) *Both Problem P and Problem D possess optimal solutions* $\mathbf{x}^*$ *and* $\mathbf{w}^*$ *respectively and* $\mathbf{cx}^* = \mathbf{w}^*\mathbf{b}$.
(2) *Problem P is unbounded and Problem D is infeasible.*
(3) *Problem D is unbounded and Problem P is infeasible.*
(4) *Both problems are infeasible.*

THEOREM 9.88. *Problem D has an optimal solution* $\mathbf{w}^* \in \mathbb{R}^m$ *if and only if there exists vectors* $\mathbf{x}^* \in \mathbb{R}^n$ *and* $\mathbf{s}^* \in \mathbb{R}^m$ *and a vector of surplus variables* $\mathbf{v}^* \in \mathbb{R}^n$ *such that:*

$$\text{Primal Feasibility}\begin{cases} \mathbf{w}^*\mathbf{A} \geq \mathbf{c} \\ \mathbf{w}^* \geq 0 \end{cases} \tag{9.103}$$

$$\text{Dual Feasibility}\begin{cases} \mathbf{A}\mathbf{x}^* + \mathbf{s}^* = \mathbf{b} \\ \mathbf{x}^* \geq 0 \\ \mathbf{s}^* \geq 0 \end{cases} \tag{9.104}$$

$$\text{Complementary Slackness}\begin{cases} (\mathbf{w}^*\mathbf{A} - \mathbf{c})\,\mathbf{x}^* = 0 \\ \mathbf{w}^*\mathbf{s}^* = 0 \end{cases} \tag{9.105}$$

Furthermore, these KKT conditions are equivalent to the KKT conditions for the primal problem.

CHAPTER 10

Feasible Direction Methods and Quadratic Programming

1. Preliminaries

REMARK 10.1. In this chapter, we return to the problem we originally discussed in Chapter 1, namely: Let $f : \mathbb{R}^n \to \mathbb{R}$; for $i = 1, \dots, m$, $g_i : \mathbb{R}^n \to \mathbb{R}$; and for $j = 1, \dots, l$ $h_j : \mathbb{R}^n \to \mathbb{R}$ be functions. Then the problem we consider is:

$$\begin{cases} \max & f(x_1, \dots, x_n) \\ s.t. & g_1(x_1, \dots, x_n) \leq b_1 \\ & \vdots \\ & g_m(x_1, \dots, x_n) \leq b_m \\ & h_1(x_1, \dots, x_n) = r_1 \\ & \vdots \\ & h_l(x_1, \dots, x_n) = r_l \end{cases}$$

For simplicity, however, we will consider a minor variation of the problem, namely:

$$\begin{cases} \max & f(x_1, \dots, x_n) \\ s.t. & g_1(x_1, \dots, x_n) \leq 0 \\ & \vdots \\ & g_m(x_1, \dots, x_n) \leq 0 \\ & h_1(x_1, \dots, x_n) = 0 \\ & \vdots \\ & h_l(x_1, \dots, x_n) = 0 \end{cases} \tag{10.1}$$

It is easy to see that this is an equivalent formulation, as we have allowed $g_1, \dots, g_m$ and $h_1, \dots, h_l$ to be defined vaguely.

Unlike in other chapters, we may consider the minimization very of this problem, when it is convenient for understanding. When we do so, the reader will be given sufficient warning. In addition, for the remainder of this chapter (unless otherwise stated), we will suppose that:

$$X = \{\mathbf{x} \in \mathbb{R}^n : g_1(\mathbf{x}), \dots, g_m(\mathbf{x}) \leq 0, h_1(\mathbf{x}) = \dots = h_l(\mathbf{x}) = 0\} \tag{10.2}$$

is a *closed, non-empty, convex* set.

REMARK 10.2. We begin our discussion of constrained optimization with a lemma, better known as Weirstrass' Theorem. We will not prove it as the proof requires a bit more analysis than is required for the rest of the notes. The interested reader may consul

LEMMA 10.3 (Weirstrass' Theorem). *Let X be a non-empty closed and bounded set in $\mathbb{R}^n$ and let z be a continuous mapping with $f : X \to \mathbb{R}$. Then the optimization problem:*

$$(10.3) \qquad \begin{cases} \max & f(\mathbf{x}) \\ s.t. & \mathbf{x} \in X \end{cases}$$

has at least one solution $\mathbf{x}^* \in X$.

REMARK 10.4. The proof of the next theorem is a simple modification to the proof of Theorem 2.25.

THEOREM 10.5. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is concave and $\mathbf{x}^*$ is a local maximizer of*

$$(10.4) \qquad P \begin{cases} \max & f(\mathbf{x}) \\ s.t. & \mathbf{x} \in X \end{cases}$$

then $\mathbf{x}^*$ is a global maximizer.

PROOF. Suppose that $\mathbf{x}^+ \in \mathbb{X}$ has the property that $f(\mathbf{x}^+) > f(\mathbf{x}^*)$. For any $\lambda \in (0,1)$ we know that:

$$f(\lambda \mathbf{x}^* + (1-\lambda)\mathbf{x}^+) \geq \lambda f(\mathbf{x}^*) + (1-\lambda) f(\mathbf{x}^+)$$

Moreover, by the convexity of X, $\lambda \mathbf{x}^* + (1-\lambda)\mathbf{x}^+ \in X$. Since $\mathbf{x}^*$ is a local maximum there is an $\epsilon > 0$ so that for all $\mathbf{x} \in B_\epsilon(\mathbf{x}^*) \cap X$, $f(\mathbf{x}^*) \geq f(\mathbf{x})$. Choose λ so that $\lambda \mathbf{x}^* + (1-\lambda)\mathbf{x}^+$ is in $B_\epsilon(\mathbf{x}^*) \cap X$ and let $\mathbf{x} = \lambda \mathbf{x}^* + (1-\lambda)\mathbf{x}^+$. Let $r = f(\mathbf{x}^+) - f(\mathbf{x}^*)$. By assumption $r > 0$. Then we have:

$$f(\mathbf{x}) \geq \lambda f(\mathbf{x}^*) + (1-\lambda)(f(\mathbf{x}^*) + r)$$

But this implies that:

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + (1-\lambda) r$$

But $\mathbf{x} \in B_\epsilon(\mathbf{x}^*) \cap X$ by choice of λ, which contradicts our assumption that $\mathbf{x}^*$ is a local maximum. Thus, $\mathbf{x}^*$ must be a global maximum. □

THEOREM 10.6. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is strictly concave and that $\mathbf{x}^*$ is a global solution of*

$$(10.5) \qquad P \begin{cases} \max & f(\mathbf{x}) \\ s.t. & \mathbf{x} \in X \end{cases}$$

Then $\mathbf{x}^*$ is the unique global maximizer of f.

EXERCISE 73. Prove Theorem 10.6.

THEOREM 10.7. *Suppose $f(x)$ is continuously differentiable on X. If $\mathbf{x}^*$ is a local maximum of $f(x)$, then:*

$$(10.6) \qquad \nabla f(\mathbf{x}^*)^T (\mathbf{x} - \mathbf{x}^*) \leq 0 \quad \forall \mathbf{x} \in X$$

Furthermore, if $f(\mathbf{x})$ is a concave function, then this is a sufficient condition for a maximum.

Proof. Suppose that Inequality 10.6 does not hold, but $\mathbf{x}^*$ is a local maximum. By the mean value theorem, there is some $t \in (0, 1)$ so that:

$$f(\mathbf{x}) = f(\mathbf{x}^*) + \nabla f(\mathbf{x}^* + t(\mathbf{x} - \mathbf{x}^*))^T(\mathbf{x} - \mathbf{x}^*) \tag{10.7}$$

If Inequality 10.6 does not hold for $\mathbf{x} \in X$, then $\nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) > 0$. Choose $\mathbf{y} = \mathbf{x}^* + \epsilon(\mathbf{x} - \mathbf{x}^*)$ where $\epsilon > 0$ is small enough so that by continuity,

$$\nabla f(\mathbf{x}^* + t(\mathbf{y} - \mathbf{x}^*))^T(\mathbf{y} - \mathbf{x}^*) > 0 \tag{10.8}$$

since $\nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) > 0$. Note that $\mathbf{y}$ is in the same direction as $\mathbf{x}$ relative to $\mathbf{x}^*$, ensuring that $\nabla f(\mathbf{x}^*)^T(\mathbf{y} - \mathbf{x}^*) > 0$. Thus:

$$f(\mathbf{y}) - f(\mathbf{x}^*) = +\nabla f(\mathbf{x}^* + t(\mathbf{y} - \mathbf{x}^*))^T(\mathbf{x} - \mathbf{x}^*) > 0 \tag{10.9}$$

and $\mathbf{x}^*$ cannot be a local maximum.

If $f(\mathbf{x})$ is concave, then by Theorem 2.19, we know that:

$$f(\mathbf{x}) \leq f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) \tag{10.10}$$

If Inequality 10.6, then for all $\mathbf{x} \in X$, we know that: $f(\mathbf{x}) \leq f(\mathbf{x}^*)$. Thus, $\mathbf{x}^*$ must be a (local) maximum. When $f(\mathbf{x})$ is strictly convex, then $\mathbf{x}^*$ is a global maximum. □

Definition 10.8 (Stationary Point). Suppose that $\mathbf{x}^* \in X$ satisfies Inequality 10.6. Then $\mathbf{x}^*$ is a stationary point for the problem of maximizing $f(\mathbf{x})$ subject to the constraints that $\mathbf{x} \in \mathbf{X}$.

Remark 10.9. This notion of stationarity takes the place of $\nabla f(\mathbf{x}^*) = 0$ used in unconstrained optimization. We will use it throughout the rest of this chapter. Not only that, but the definition of gradient related remains the same as before. See Definition 4.11.

2. Frank-Wolfe Algorithm

Remark 10.10. For the remainder of this chapter, we assume the following Problem P:

$$P \begin{cases} \max & f(\mathbf{x}) \\ s.t. & \mathbf{x} \in X \end{cases} \tag{10.11}$$

where X is a closed, convex and non-empty set.

Remark 10.11. Our study of Linear Programming in the previous chapter suggests that when $f(\mathbf{x})$ in Problem P is non-linear, but $g_1, \dots, g_m$ and $h_1, \dots, h_l$ are all linear, we might be able to find a simple way of using linear programming to solve this non-linear programming problem. The simplest approach to this is the *Frank-Wolfe* algorithm, or *reduced gradient* algorithm. This method is (almost) the non-linear programming variation of the gradient descent method, in that it works well initially, but slows down as the algorithm approaches an optimal point.

Remark 10.12. The Frank-Wolfe algorithm works by iteratively approximating the objective function as a linear function and then uses linear programming to choose an ascent direction. Univariate maximization is then used to choose a distance along the ascent direction. In essence, given problem P, at step k, with iterate $\mathbf{x}_k$, we solve:

$$L \begin{cases} \max & \nabla f(\mathbf{x}_k)^T \mathbf{x} \\ s.t. & \mathbf{x} \in X \end{cases} \tag{10.12}$$

Note that this is equivalent to maximizing the first order approximation:

$$f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{x} - \mathbf{x}_k) \tag{10.13}$$

If $\mathbf{x}^+$ is the solution to this problem, we then solve the problem:

$$\max_{t\in[0,1]} f(\mathbf{x}_k + t(\mathbf{x}^+ - \mathbf{x}_k)) \tag{10.14}$$

By the convexity of X, we see that restricting t to $[0, 1]$ ensures that $\mathbf{x}_{k+1} = t^*(\mathbf{x}_k+t(\mathbf{x}^+-\mathbf{x}_k))$ is in X, when t^* is the solution to the univariate problem. Clearly when X is a polyhedral set, Problem L is a linear programming problem. Iteration continues until a stopping criterion is reached. A simple one is $||\mathbf{x}_{k+1} - \mathbf{x}_k|| < \epsilon$. A reference implementation for this algorithm is shown in Algorithm 21.

Example 10.13. We can see an example of the Frank-Wolfe algorithm when we attempt to solve the problem:

$$\begin{cases} \max & -(x-2)^2 - (y-2)^2 \\ s.t. & x + y \leq 1 \\ & x, y \geq 0 \end{cases} \tag{10.15}$$

In this case, the sequence of points when starting from $x_0 = 0, y_0 = 0$ is

(1) $x_1 = 0$, $y_1 = 1$
(2) $x_2 = 0.5$, $y_2 = 0.5$

when we set $\epsilon = 0.0001$. This is illustrated in Figure 10.1(a).

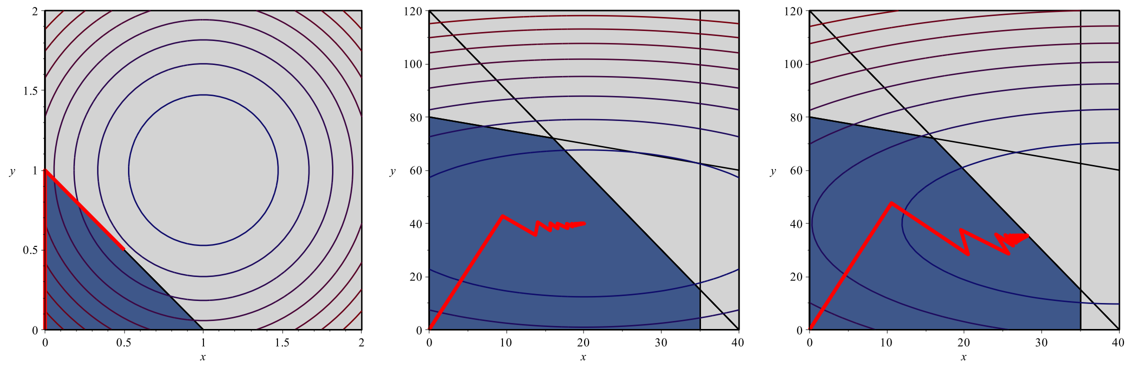


Figure 10.1. (a) The steps of the Frank-Wolfe Algorithm when maximizing $-(x-2)^2-(y-2)^2$ over the set of (x, y) satisfying the constraints $x+y \leq 1$ and $x, y \geq 0$. (b) The steps of the Frank-Wolfe Algorithm when maximizing $-7(x-20)^2-6(y-40)^2$ over the set of (x, y) satisfying the constraints $3x + y \leq 120$, $x + 2y \leq 160$, $x \leq 35$ and $x, y \geq 0$. (c) The steps of the Frank-Wolfe Algorithm when maximizing $-7(x-40)^2-6(y-40)^2$ over the set of (x, y) satisfying the constraints $3x+y \leq 120$, $x + 2y \leq 160$, $x \leq 35$ and $x, y \geq 0$.

```
FrankWolfe := proc (F::operator, LINCON::set, initVals::list, epsilon::numeric,
  maxIter::integer)::list;

  local vars, xnow, xnext, ttemp, FLIN, SOLN, X, vals, i, G, dX, count, p, r,
    passIn, phi, OUT;

  vars := []; xnow := []; vals := initVals;
  for i to nops(initVals) do
    vars := [op(vars), lhs(initVals[i])];
    xnow := [op(xnow), rhs(initVals[i])]
  end do;

  #Compute the gradient.
  G := Gradient(F(op(vars)), vars));
  OUT := [];

  dX := 1;
  count := 0;

  while epsilon < dX and count < maxIter do
    count := count + 1;

    #Store output.
    OUT := [op(OUT), xnow];

    #A vector of variables, used to create the linear objective.
    X := Vector([seq(vars[i], i = 1 .. nops(vars))]);

    #Create the linear objective.
    FLIN := Transpose(evalf(eval(G,vals)).X;

    #Recover the actual solution.
    SOLN := LPSolve(FLIN, LINCON, maximize = true)[2];

    #Compute the direction.
    p := [];
    for i to nops(initVals) do
      p := [op(p), eval(vars[i], SOLN)]
    end do;
    r := Vector(p);
    passIn := convert(Vector(xnow)+s*(r-Vector(xnow)), list);
    #Define the line function
    phi := proc (t) options operator, arrow;
      evalf(eval(F(op(passIn)), s = t))
    end proc;

    #Defined separately.
    ttemp := GoldenSectionSearch(phi, 0, .39, 1, 0.1e-3);

    xnext := evalf(eval(passIn, [s = ttemp]));
    dX := Norm(Vector(xnext - xnow));

    #Move and update the current values.
    xnow := xnext;
    vals := [];
    for i to nops(vars) do
      vals := [op(vals), vars[i] = xnow[i]]
    end do:
  end do;
  OUT
end proc:
```

Algorithm 21. Frank-Wolfe Algorithm for finding the maximum of a (concave) function with linear constraints.

By contrast, we can investigate the behavior of the Frank-Wolfe Algorithm when we attempt to solve the problem:

$$(10.16)\quad \begin{cases} \max & -7(x-20)^2 - 6(y-40)^2 \\ s.t. & 3x + y \leq 120 \\ & x + 2y \leq 160 \\ & x \leq 35 \\ & x, y \geq 0 \end{cases}$$

Here the optimal point is in the interior of the constraint set (as opposed to the exterior) and the Frank-Wolfe algorithm exhibits behavior like Gradient Ascent. Convergence occurs in 73 iterations. This is shown in Figure 10.1(b).

Finally, we can consider the objective function $-7(x-40)^2 - 6(y-40)^2$, with the same constraints as before. In this case, convergence takes over 2000 iterations, but we see we can very close to an approximate solution. (See Figure 10.1(c).) This illustrates the power of the Frank-Wolfe method. It can be used to find a reasonably good solution quickly. If refinement is needed, then an algorithm with better convergence properties can take over.

Lemma 10.14. *Each search direction* $(\mathbf{x}^+ - \mathbf{x}_k)$ *in the Frank-Wolfe algorithm is an ascent direction.*

PROOF. By construction $\mathbf{x}^+$ maximizes $\nabla f(\mathbf{x}_k)^T\mathbf{x}$ subject to the constraint that $\mathbf{x}^+ \in X$. This implies that for all $\mathbf{x} \in X$:

$$\nabla f(\mathbf{x}_k)^T\mathbf{x}^+ \geq \nabla f(\mathbf{x}_k)^T\mathbf{x} \tag{10.17}$$

Thus:

$$\nabla f(\mathbf{x}_k)^T(\mathbf{x}^+ - \mathbf{x}) \geq 0 \quad \forall \mathbf{x} \in X \tag{10.18}$$

Thus, in particular, $\nabla f(\mathbf{x}_k)^T(\mathbf{x}^+ - \mathbf{x}_k) \geq 0$ and thus, $\mathbf{x}^+ - \mathbf{x}_k$ is an ascent direction. □

DEFINITION 10.15. Given $f : \mathbb{R}^n \to \mathbb{R}$ and a convex set X, a *feasible direction method* is one in which at each iteration k a direction $\mathbf{p}_k$ is chosen and $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta\mathbf{p}_k$ is constructed so that $\mathbf{x}_{k+1} \in X$ and for some $\epsilon > 0$ if $0 \leq \delta \leq \epsilon$, $\mathbf{x}_k + \delta\mathbf{p}_k \in X$. Thus, $\mathbf{p}_k$ is a *feasible direction.*

LEMMA 10.16. *Let $\{\mathbf{x}_k\}$ be a sequence generated by a feasible direction method (e.g. the Frank-Wolfe Algorithm). If δ is chosen by limited line search or the Armijo rule, then any point $\mathbf{x}^*$ to which $\{\mathbf{x}_k\}$ converges is a stationary point.*

PROOF. The proof is identical (*mutatis mutandis*) to the proof of Theorem 4.12. □

THEOREM 10.17. *Suppose that the Frank-Wolfe algorithm converges. Then it converges to a stationary point.*

PROOF. Every direction generated by the Frank-Wolfe algorithm is an ascent direction and therefore the sequence of directions must be gradient related. By the previous lemma, the Frank-Wolfe algorithm converges to a stationary point. □

3. Farkas' Lemma and Theorems of the Alternative

LEMMA 10.18 (Farkas' Lemma). *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{c} \in \mathbb{R}^n$ be a row vector. Suppose $\mathbf{x} \in \mathbb{R}^n$ is a column vector and $\mathbf{w} \in \mathbb{R}^m$ is a row vector. Then exactly one of the following systems of inequalities has a solution:*

(1) $\mathbf{A}\mathbf{x} \geq \mathbf{0}$ *and* $\mathbf{c}\mathbf{x} < 0$ *or*
(2) $\mathbf{w}\mathbf{A} = \mathbf{c}$ *and* $\mathbf{w} \geq \mathbf{0}$

REMARK 10.19. Before proceeding to the proof, it is helpful to restate the lemma in the following way:

(1) *If there is a vector $\mathbf{x} \in \mathbb{R}^n$ so that $\mathbf{A}\mathbf{x} \geq \mathbf{0}$ and $\mathbf{c}\mathbf{x} < 0$, then there **is no** vector $\mathbf{w} \in \mathbb{R}^m$ so that $\mathbf{w}\mathbf{A} = \mathbf{c}$ and $\mathbf{w} \geq \mathbf{0}$.*

(2) *Conversely, if there is a vector $\mathbf{w} \in \mathbb{R}^m$ so that $\mathbf{w}\mathbf{A} = \mathbf{c}$ and $\mathbf{w} \geq \mathbf{0}$, then there **is no** vector $\mathbf{x} \in \mathbb{R}^n$ so that $\mathbf{A}\mathbf{x} \geq \mathbf{0}$ and $\mathbf{c}\mathbf{x} < 0$.*

PROOF. We can prove Farkas' Lemma using the fact that a bounded linear programming problem has an extreme point solution. Suppose that System 1 has a solution $\mathbf{x}$. If System 2 also has a solution $\mathbf{w}$, then

$$\mathbf{w}\mathbf{A} = \mathbf{c} \implies \mathbf{w}\mathbf{A}\mathbf{x} = \mathbf{c}\mathbf{x}. \tag{10.19}$$

The fact that System 1 has a solution ensures that $\mathbf{c}\mathbf{x} < 0$ and therefore $\mathbf{w}\mathbf{A}\mathbf{x} < 0$. However, it also ensures that $\mathbf{A}\mathbf{x} \geq \mathbf{0}$. The fact that System 2 has a solution implies that $\mathbf{w} \geq \mathbf{0}$. Therefore we must conclude that:

$$\mathbf{w} \geq \mathbf{0} \text{ and } \mathbf{A}\mathbf{x} \geq \mathbf{0} \implies \mathbf{w}\mathbf{A}\mathbf{x} \geq \mathbf{0}. \tag{10.20}$$

This contradiction implies that if System 1 has a solution, then System 2 cannot have a solution.

Now, suppose that System 1 has no solution. We will construct a solution for System 2. If System 1 has no solution, then there is no vector $\mathbf{x}$ so that $\mathbf{cx} < 0$ and $\mathbf{Ax} \geq \mathbf{0}$. Consider the linear programming problem:

$$(10.21) \quad P_F \begin{cases} \min & \mathbf{cx} \\ s.t. & \mathbf{Ax} \geq \mathbf{0} \end{cases}$$

Clearly $\mathbf{x} = \mathbf{0}$ is a feasible solution to this linear programming problem and furthermore is optimal. To see this, note that the fact that there is no $\mathbf{x}$ so that $\mathbf{cx} < 0$ and $\mathbf{Ax} \geq \mathbf{0}$, it follows that $\mathbf{cx} \geq 0$; i.e., 0 is a lower bound for the linear programming problem P_F. At $\mathbf{x} = \mathbf{0}$, the objective achieves its lower bound and therefore this must be an optimal solution. Therefore P_F is bounded and feasible.

We can covert P_F to standard form through the following steps:

(1) Introduce two new vectors $\mathbf{y}$ and $\mathbf{z}$ with $\mathbf{y}, \mathbf{z} \geq \mathbf{0}$ and write $\mathbf{x} = \mathbf{y} - \mathbf{z}$ (since $\mathbf{x}$ is unrestricted).
(2) Append a vector of surplus variables $\mathbf{s}$ to the constraints.

This yields the new problem:

$$(10.22) \quad P_F' \begin{cases} \min & \mathbf{cy} - \mathbf{cz} \\ s.t. & \mathbf{Ay} - \mathbf{Az} - \mathbf{I}_m\mathbf{s} = \mathbf{0} \\ & \mathbf{y}, \mathbf{z}, \mathbf{s} \geq \mathbf{0} \end{cases}$$

Applying Theorems 9.53 and 9.57, we see we can obtain an optimal basic feasible solution for Problem P_F' in which the reduced costs for the variables are all negative (that is, $z_j - c_j \leq 0$ for $j = 1, \dots, 2n + m$). Here we have n variables in vector $\mathbf{y}$, n variables in vector $\mathbf{z}$ and m variables in vector $\mathbf{s}$. Let $\mathbf{B} \in \mathbb{R}^{m \times m}$ be the basis matrix at this optimal feasible solution with basic cost vector $\mathbf{c_B}$. Let $\mathbf{w} = \mathbf{c_B}\mathbf{B}^{-1}$ (as it was defined for the revised simplex algorithm).

Consider the columns of the simplex tableau corresponding to a variable x_k (in our original $\mathbf{x}$ vector). The variable $x_k = y_k - z_k$. Thus, these two columns are additive inverses. That is, the column for y_k will be $\mathbf{B}^{-1}\mathbf{A}_{\cdot k}$, while the column for z_k will be $\mathbf{B}^{-1}(-\mathbf{A}_{\cdot k}) = -\mathbf{B}^{-1}\mathbf{A}_{\cdot k}$. Furthermore, the objective function coefficient will be precisely opposite as well. Thus the fact that $z_j - c_j \leq 0$ for all variables implies that:

$$\begin{aligned} &\mathbf{wA}_{\cdot k} - c_k \leq 0 \textbf{ and} \\ &-\mathbf{wA}_{\cdot k} + c_k \leq 0 \textbf{ and} \end{aligned}$$

That is, we obtain

$$(10.23) \quad \mathbf{wA} = \mathbf{c}$$

since this holds for all columns of $\mathbf{A}$.

Consider the surplus variable s_k. Surplus variables have zero as their coefficient in the objective function. Further, their simplex tableau column is simply $\mathbf{B}^{-1}(-\mathbf{e}_k) = -\mathbf{B}^{-1}\mathbf{e}_k$. The fact that the reduced cost of this variable is non-positive implies that:

$$(10.24) \quad \mathbf{w}(-\mathbf{e}_k) - 0 = -\mathbf{we}_k \leq 0$$

Since this holds for all surplus variable columns, we see that $-\mathbf{w} \leq \mathbf{0}$ which implies $\mathbf{w} \geq \mathbf{0}$. Thus, the optimal basic feasible solution to Problem P_F' must yield a vector $\mathbf{w}$ that solves System 2.

Lastly, the fact that if System 2 does not have a solution, then System 1 does follows from contrapositive on the previous fact we just proved. □

Exercise 74. Suppose we have two statements A and B so that:

$A \equiv$ System 1 has a solution.

$B \equiv$ System 2 has a solution.

Our proof showed explicitly that NOT $A \implies B$. Recall that contrapositive is the logical rule that asserts that:

$$X \implies Y \equiv \text{NOT } Y \implies \text{NOT } X \tag{10.25}$$

Use contrapositive to prove explicitly that if System 2 has no solution, then System 1 must have a solution. [Hint: NOT NOT $X \equiv X$.]

3.1. Geometry of Farkas' Lemma. Farkas' Lemma has a pleasant geometric interpretation[1]. Consider System 2: namely:

$$\mathbf{wA} = \mathbf{c} \textbf{ and } \mathbf{w} \geq \mathbf{0}$$

Geometrically, this states that $\mathbf{c}$ is inside the positive cone generated by the *rows of* $\mathbf{A}$. That is, let $\mathbf{w} = (w_1, \dots, w_m)$. Then we have:

$$w_1\mathbf{A}_{1\cdot} + \cdots + w_m\mathbf{A}_{m\cdot} \tag{10.26}$$

and $w_i \geq 0$ for $i = 1, \dots, m$. Thus $\mathbf{c}$ is a positive combination of the rows of $\mathbf{A}$. This is illustrated in Figure 10.2. On the other hand, suppose System 1 has a solution. Then let

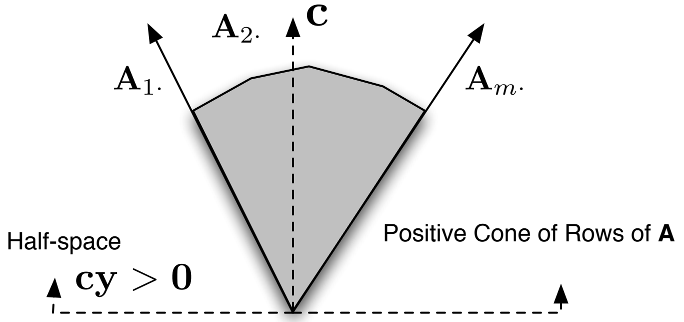


Figure 10.2. System 2 has a solution if (and only if) the vector $\mathbf{c}$ is contained inside the positive cone constructed from the rows of $\mathbf{A}$.

$\mathbf{y} = -\mathbf{x}$. System 1 states that $\mathbf{Ay} \leq \mathbf{0}$ and $\mathbf{cy} > 0$. That means that each row of $\mathbf{A}$ (as a vector) must be at a right angle or obtuse to $\mathbf{y}$. (Since $\mathbf{A}_{i\cdot}\mathbf{x} \geq \mathbf{0}$.) Further, we know that the vector $\mathbf{y}$ must be acute with respect to the vector $\mathbf{c}$. This means that System 1 has a solution only if the vector $\mathbf{c}$ is not in the positive cone of the rows of $\mathbf{A}$ or equivalently the intersection of the open half-space $\{\mathbf{y} : \mathbf{cy} > 0\}$ and the set of vectors $\{\mathbf{y} : \mathbf{A}_{i\cdot}\mathbf{y} \leq \mathbf{0}, i = 1, \dots m\}$ is

[1]Thanks to Akinwale Akinbiyi for pointing out a typo in this discussion.

non-empty. This set is the cone of vectors perpendicular to the rows of $\mathbf{A}$. This is illustrated in Figure 10.3

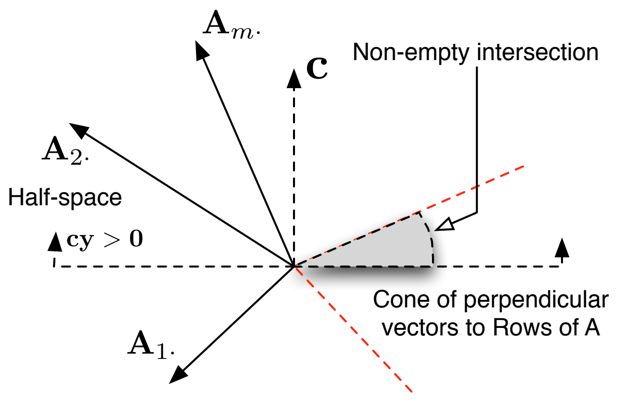


Figure 10.3. System 1 has a solution if (and only if) the vector $\mathbf{c}$ is not contained inside the positive cone constructed from the rows of $\mathbf{A}$.

Example 10.20. Consider the matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and the vector $\mathbf{c} = \begin{bmatrix} 1 & 2 \end{bmatrix}$. Then clearly, we can see that the vector $\mathbf{w} = \begin{bmatrix} 1 & 2 \end{bmatrix}$ will satisfy System 2 of Farkas' Lemma, since $\mathbf{w} \geq \mathbf{0}$ and $\mathbf{wA} = \mathbf{c}$.

Contrast this with $\mathbf{c}' = \begin{bmatrix} 1 & -1 \end{bmatrix}$. In this case, we can choose $\mathbf{x} = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$. Then $\mathbf{Ax} = \begin{bmatrix} 0 & 1 \end{bmatrix}^T \geq \mathbf{0}$ and $\mathbf{c}'\mathbf{x} = -1$. Thus $\mathbf{x}$ satisfies System 1 of Farkas' Lemma.

These two facts are illustrated in Figure 10.4. Here, we see that $\mathbf{c}$ is inside the positive cone formed by the rows of $\mathbf{A}$, while $\mathbf{c}'$ is not.

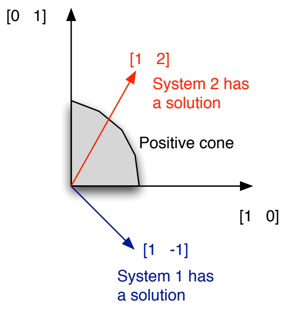


Figure 10.4. An example of Farkas' Lemma: The vector $\mathbf{c}$ is inside the positive cone formed by the rows of $\mathbf{A}$, but $\mathbf{c}'$ is not.

Exercise 75. Consider the following matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

and the vector $\mathbf{c} = \begin{bmatrix} 1 & 2 \end{bmatrix}$. For this matrix and this vector, does System 1 have a solution or does System 2 have a solution? [Hint: Draw a picture illustrating the positive cone formed by the rows of $\mathbf{A}$. Draw in $\mathbf{c}$. Is $\mathbf{c}$ in the cone or not?]

3.2. Theorems of the Alternative. Farkas' lemma can be manipulated in many ways to produce several equivalent statements. The collection of all such theorems are called *Theorems of the Alternative* and are used *extensively* in optimization theory in proving optimality conditions. We state two that will be useful to us.

Corollary 10.21. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ and let $\mathbf{c} \in \mathbb{R}^n$ be a row vector. Then exactly one of the following systems has a solution:*

(1) $\mathbf{Ax} < 0$ *and* $\mathbf{cx} > 0$ *or*
(2) $\mathbf{wA} = w_0\mathbf{c}$ *and* $[w_0, \mathbf{w}] \geq \mathbf{0}$, $[w_0, \mathbf{w}] \neq \mathbf{0}$.

Proof. First define:

$$\mathbf{M} = \begin{bmatrix} -\mathbf{c} \\ - \\ \mathbf{A} \end{bmatrix} \tag{10.27}$$

So that System 1 becomes $\mathbf{Mx} < 0$ for some $\mathbf{x} \in \mathbb{R}^n$. Now, re-write this new System 1 as:

$$\mathbf{Mx} + s\mathbf{1} \leq \mathbf{0} \tag{10.28}$$

$$s > \mathbf{0} \tag{10.29}$$

where $\mathbf{1}$ is a vector of m ones. If we write:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{M} & | & \mathbf{1} \end{bmatrix} \tag{10.30}$$

$$\mathbf{y} = \begin{bmatrix} \mathbf{x} \\ - \\ s \end{bmatrix} \leq \mathbf{0} \tag{10.31}$$

$$\mathbf{r} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{10.32}$$

Then System 1 becomes $\mathbf{My} \leq \mathbf{0}$ and $\mathbf{ry} > 0$. Finally letting $\mathbf{z} = -\mathbf{y}$, we see that System 1 becomes $\mathbf{Qz} \geq \mathbf{0}$ and $\mathbf{rz} < 0$, which is System 1 of Farkas Lemma. Thus, if System 1 has no solution, then there is a solution to:

$$\mathbf{wQ} = \mathbf{r} \qquad \mathbf{w} \geq \mathbf{0} \tag{10.33}$$

This implies that:

$$\mathbf{vM} = \mathbf{0} \tag{10.34}$$

$$\mathbf{v1} = 1 \tag{10.35}$$

$$\mathbf{v} \geq 0 \tag{10.36}$$

Suppose that $\mathbf{v} = [w_0, w_1, \dots, w_m]$ and $\mathbf{w} = [w_1, \dots, w_m]$. Then:

$$\begin{bmatrix} w_0 & \mathbf{w} \end{bmatrix} \begin{bmatrix} -\mathbf{c} \\ - \\ \mathbf{A} \end{bmatrix} = \mathbf{0} \tag{10.37}$$

But this implies that:

$$-w_0\mathbf{c} + \mathbf{wA} = \mathbf{0} \tag{10.38}$$

Therefore: $\mathbf{wA} = w_0\mathbf{c}$ and $\mathbf{v1} = 1$ and $\mathbf{v} \geq 0$ imply that $[w_0, \mathbf{w}] \geq \mathbf{0}$, $[w_0, \mathbf{w}] \neq \mathbf{0}$. □

Exercise 76. Prove the following corollary to Farkas' Lemma:

Corollary 10.22. *Let* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *and* $\mathbf{c} \in \mathbb{R}^n$ *be a row vector. Suppose* $\mathbf{d} \in \mathbb{R}^n$ *is a column vector and* $\mathbf{w} \in \mathbb{R}^m$ *is a row vector and* $\mathbf{v} \in \mathbf{R}^n$ *is a row vector. Then exactly one of the following systems of inequalities has a solution:*

(1) $\mathbf{Ad} \leq \mathbf{0}$*,* $\mathbf{d} \geq \mathbf{0}$ *and* $\mathbf{cd} > 0$ *or*
(2) $\mathbf{wA} - \mathbf{v} = \mathbf{c}$ *and* $\mathbf{w}, \mathbf{v} \geq \mathbf{0}$

[Hint: Write System 2 from this corollary as $\mathbf{wA} - \mathbf{I}_n\mathbf{v} = \mathbf{c}$ and then re-write the system with an augmented vector $[\mathbf{w} \quad \mathbf{v}]$ with an appropriate augmented matrix. Let $\mathbf{M}$ be the augmented matrix you identified. Now write System 1 from Farkas' Lemma using $\mathbf{M}$ and $\mathbf{x}$. Let $\mathbf{d} = -\mathbf{x}$ and expand System 1 until you obtain System 1 for this problem.]

4. Preliminary Results: Feasible Directions, Improving Directions

Remark 10.23. In the previous chapter, we saw the Karush-Kuhn-Tucker (KKT) necessary and sufficient conditions for optimality for linear programming problems. In this chapter, we generalize these conditions to the case of non-linear programming problems.

Remark 10.24. We return to the study of Problem P:

$$P \begin{cases} \max & f(\mathbf{x}) \\ s.t. & \mathbf{x} \in X \end{cases}$$

where X is a set with non-empty interior that is, ideally, closed and convex.

Definition 10.25 (Cone of Feasible Directions). For Problem P, the *cone of feasible directions* at $\mathbf{x} \in X$ is the set:

$$D(\mathbf{x}) = \{\mathbf{p} \in \mathbb{R}^n : \mathbf{p} \neq \mathbf{0} \text{ and for all } \lambda \in (0, \delta)\ , \mathbf{x} + \lambda\mathbf{p} \in X, \text{ for } \delta > 0\} \tag{10.39}$$

Definition 10.26 (Cone of Improving Directions). For Problem P, the *cone of improving directions* at $\mathbf{x} \in X$ is the set:

$$F(\mathbf{x}) = \{\mathbf{p} \in \mathbb{R}^n : \text{for all } \lambda \in (0, \delta)\ , f(\mathbf{x} + \lambda\mathbf{p}) > f(\mathbf{x}), \text{ for } \delta > 0\} \tag{10.40}$$

Proposition 10.27. *Let* $\mathbf{x} \in \mathbb{R}^n$. *If* $\mathbf{p} \in \mathbb{R}^n$ *and* $\nabla f(\mathbf{x})^T\mathbf{p} > 0$*, then* $\mathbf{p} \in F(\mathbf{x})$*. Thus:*

$$F_0(\mathbf{x}) = \{\mathbf{p} \in \mathbb{R}^n : \nabla f(\mathbf{x})^T\mathbf{p} > 0\} \subseteq F(\mathbf{x}) \tag{10.41}$$

Exercise 77. Prove Proposition 10.27.

Lemma 10.28. *For Problem P, suppose that* $\mathbf{x}^* \in X$ *is a local maximum. Then* $F_0(\mathbf{x}^*) \cap D(\mathbf{x}^*) = \emptyset$.

Proof. Suppose this is not the case by contradiction. Choose $\mathbf{p} \in F_0(\mathbf{x}^*) \cap D(\mathbf{x}^*)$. Let $\mathbf{x} = \mathbf{x}^* + \lambda\mathbf{p}$ for some appropriately chosen $\lambda > 0$ so taht $\mathbf{x} \in X$. Such a λ exists because $\mathbf{p} \in D(\mathbf{x}^*)$. Thus, $\mathbf{d} = \mathbf{x} - \mathbf{x}^*$ and

$$\nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) = \lambda\nabla f(\mathbf{x}^*)^T\mathbf{p} > 0 \tag{10.42}$$

because $\mathbf{p} \in F_0(\mathbf{x}^*)$. This contradicts Theorem 10.7 and thus $\mathbf{x}^*$ cannot have been a local maximum. ☐

Proposition 10.29. *If* $f : \mathbb{R}^n \to \mathbb{R}$ *is concave and* $F_0(\mathbf{x}^*) \cap D(\mathbf{x}^*) = \emptyset$*, then* $\mathbf{x}^*$ *is a local maximum of* f *on* X.

EXERCISE 78. Using the sufficiency argument in the proof of Theorem 10.7, prove Proposition 10.29.

REMARK 10.30. It is relatively straight-forward to prove that when f is concave, then $F_0(\mathbf{x}^*) = F(\mathbf{x}^*)$ and when f is strictly concave, then:

$$F(\mathbf{x}^*) = \{\mathbf{p} \in \mathbb{R}^n : \mathbf{p} \neq \mathbf{0}, \nabla f(\mathbf{x}^*)^T\mathbf{p} \geq 0\} \tag{10.43}$$

LEMMA 10.31. *Consider a simplified form of Problem P, called Problem P':*

$$P' \begin{cases} \max & f(\mathbf{x}) \\ s.t. & g_i(\mathbf{x}) \leq 0 \quad i = 1, \dots, m \end{cases} \tag{10.44}$$

Here, $f : \mathbb{R}^n \to \mathbb{R}$ and $g_i(\mathbf{x}) :\to \mathbb{R}^n$ are continuously differentiable at a point $\mathbf{x}_0$. Denote the set $I = \{i \in \{1, \dots, m\} : g_i(\mathbf{x}^*) = 0\}$. Then:

$$G_0(\mathbf{x}_0) = \{\mathbf{p} \in \mathbb{R}^n : \nabla g_i(\mathbf{x}_0)^T\mathbf{p} < 0, \forall i \in I\} \subseteq D(\mathbf{x}_0) \tag{10.45}$$

PROOF. Suppose $\mathbf{p} \in G_0(\mathbf{x}_0)$ and define $\mathbf{x} = \mathbf{x}_0 + \lambda\mathbf{p}$. For all $j \not\in I$, we know that $g_j(\mathbf{x}_0) < 0$. By the continuity of g_j, for all $\epsilon > 0$, there exists a $\delta > 0$ so that if $||\mathbf{x}_0 - \mathbf{x}|| < \delta$, then $|g_j(\mathbf{x}_0) - g_j(\mathbf{x})| < \epsilon$ and thus for some choice of $\lambda > 0$ we can ensure that $g_j(\mathbf{x}) < 0$. Now since $\nabla g_i(\mathbf{x}_0)^T\mathbf{p} < 0$, this is a *descent direction* of g_i for $i \in I$. Thus (by a variation of Proposition 10.27), there is some $\lambda' > 0$ so that $g_i(\mathbf{x}_0 + \lambda'\mathbf{p}) < g_i(\mathbf{x}_0) = 0$. Thus, if we choose $t = \min\{\lambda, \lambda'\}$, then $g_i(\mathbf{x}_0 + t\mathbf{p}) \leq 0$ for $i = 1, \dots, m$ and thus $\mathbf{x}_0 + t\mathbf{p} \in X$. Thus it follows that $\mathbf{p} \in D$. □

LEMMA 10.32. *Consider Problem P' as in Lemma 10.31 and suppose that g_i $(i \in I)$ are strictly convex (as defined in Lemma 10.31). Then $G_0(\mathbf{x}_0) = D(\mathbf{x}_0)$.*

PROOF. Suppose that $\mathbf{p} \in D(\mathbf{x}_0)$ and suppose $\mathbf{p} \not\in G_0(\mathbf{x}_0)$. Then $\nabla g_i(\mathbf{x}_0)^T\mathbf{p} \geq 0$. Thus, $\mathbf{p}$ is not a descent direction. Suppose that $\mathbf{x} = \mathbf{x}_0 + \lambda\mathbf{p}$ and define $\mathbf{h} = \mathbf{x} - \mathbf{x}_0$. Then by strict convexity (and Exercises 16 and 17), for all $\lambda > 0$ we know:

$$g_i(\mathbf{x}_0 + \lambda\mathbf{h}) - g_i(\mathbf{x}_0) > \nabla g_i(\mathbf{x}_0)^T\mathbf{h} = \lambda\nabla g_i(\mathbf{x}_0)^T\mathbf{p} \geq 0 \tag{10.46}$$

Thus, there is no $\lambda > 0$ so that $\mathbf{x} = \mathbf{x}_0 + \lambda\mathbf{p} \in X$ and $\mathbf{p} \not\in D(\mathbf{x}_0)$. □

THEOREM 10.33. *Consider Problem P', where $f : \mathbb{R}^n \to \mathbb{R}$ and $g_i(\mathbf{x}) :\to \mathbb{R}^n$ are continuously differentiable at a local maximum $\mathbf{x}^*$. Denote the set $I = \{i \in \{1, \dots, m\} : g_i(\mathbf{x}^*) = 0\}$. Then $F_0(\mathbf{x}^*) \cap G_0(\mathbf{x}^*) = \emptyset$. Conversely, if g_i $(i = 1, \dots, m)$ are strictly convex and f is concave and $F_0(\mathbf{x}^*) \cap G_0(\mathbf{x}^*) = \emptyset$, then $\mathbf{x}^*$ is a local maximum.*

PROOF. Suppose that $\mathbf{x}^*$ is a local maximum. Then we know from Lemma 10.28 that $F_0(\mathbf{x}^*) \cap D(\mathbf{x}^*) = \emptyset$. Additionally from Lemma 10.31 we know that $G_0(\mathbf{x}^*) \subseteq D(\mathbf{x}^*)$. Thus it follows that $F_0(\mathbf{x}^*) \cap G_0(\mathbf{x}^*) = \emptyset$.

Now suppose that g_i $(i = 1, \dots, m)$ are strictly convex and f is concave and $F_0(\mathbf{x}^*) \cap G_0(\mathbf{x}^*) = \emptyset$. From Lemma 10.32, we know that $D(\mathbf{x}^*) = G_0(\mathbf{x}^*)$ and thus we know at once that $F_0(\mathbf{x}^*) \cap D(\mathbf{x}^*) = \emptyset$. It follows from Proposition 10.29 that $\mathbf{x}^*$ is a local maximum over X. □

REMARK 10.34. All of these theorems can be generalized substantially, to include functions not defined over all $\mathbb{R}^n$, functions that are only locally convex or concave or functions that exhibit generalized concavity/convexity properties. Full details can be found in Chapter 4 of [BSS06], on which most of this section is based.

5. Fritz-John and Karush-Kuhn-Tucker Theorems

THEOREM 10.35 (Fritz-John Theorem). *Consider Problem P', where $f : \mathbb{R}^n \to \mathbb{R}$ and $g_i : \mathbb{R}^n \to \mathbb{R}^n$ are continuously differentiable at a local maximum $\mathbf{x}^*$. Denote the set $I = \{i \in \{1, \dots, m\} : g_i(\mathbf{x}^*) = 0\}$. Then there exists scalars, $u_0, u_1, \dots, u_m$ so that:*

$$(10.47)\quad \begin{cases} u_0 \nabla f(\mathbf{x}^*) - \sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}^*) = \mathbf{0} \\ u_i g_i(\mathbf{x}^*) = 0 \quad i = 1, \dots, m \\ u_0, u_i \geq 0 \quad i = 1, \dots, m \\ [u_0, u_1, \dots, u_m]^T \neq \mathbf{0} \end{cases}$$

PROOF. Let $\mathbf{c} = \nabla f(\mathbf{x}^*)^T$ and let $\mathbf{A}$ be the matrix whose rows are formed from $\nabla g_i(\mathbf{x}^*)^T$ $(i \in I)$. From Theorem 10.33 we know that $F_0(\mathbf{x}^*) \cap G_0(\mathbf{x}^*) = \emptyset$. This implies that there is no $\mathbf{p}$ satisfying $\mathbf{cp} > 0$ and $\mathbf{Ap} < 0$. Then it follows from Corollary 10.21 there is a scalars u_0 and a vector $\mathbf{u}$ so that: $\mathbf{uA} = u_0\mathbf{c}$ and $[u_0, \mathbf{u}] \geq \mathbf{0}$, $[u_0, \mathbf{u}] \neq \mathbf{0}$. Thus we see that:

$$(10.48)\quad \mathbf{uA} = u_0 \mathbf{c} \implies u_0 \nabla f(\mathbf{x}^*) - \sum_{i \in I} u_i \nabla g_i(\mathbf{x}^*) = \mathbf{0}$$

Now let $u_j = 0$ for $j \notin I$. Then we see Equation 10.48 is equivalent to:

$$(10.49)\quad u_0 \nabla f(\mathbf{x}^*) - \sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}^*) = \mathbf{0}$$

The fact that $u_0, u_i \geq 0$ and $[u_0, u_1, \dots, u_m]^T \neq \mathbf{0}$ is also immediate from Corollary 10.21 as well. By definition of the u_i, it is easy to see that if $i \in I$, then $g_i(\mathbf{x}^*) = 0$ and thus $u_i g_i(\mathbf{x}^*) = 0$. On the other hand, if $i \notin I$, then $u_i = 0$ and thus $u_i g_i(\mathbf{x}^*) = 0$. Thus we have shown that Expression 10.47 must hold. □

THEOREM 10.36 (Karush-Kuhn-Tucker Necessary Conditions). *Consider Problem P', where $f : \mathbb{R}^n \to \mathbb{R}$ and $g_i : \mathbb{R}^n \to \mathbb{R}^n$ are continuously differentiable at a local maximum $\mathbf{x}^*$. Denote the set $I = \{i \in \{1, \dots, m\} : g_i(\mathbf{x}^*) = 0\}$. Suppose that the set of vectors $\nabla g_i(\mathbf{x}^*)$ $(i \in I)$ are linearly independent. Then there exist scalars $\lambda_1, \dots, \lambda_m$ not all zero so that:*

$$(10.50)\quad \begin{cases} \nabla f(\mathbf{x}^*) - \sum_{i=1}^{m} \lambda_i \nabla g_i(\mathbf{x}^*) = \mathbf{0} \\ \lambda_i g_i(\mathbf{x}^*) = 0 \quad i = 1, \dots, m \\ \lambda_i \geq 0 \quad i = 1, \dots, m \end{cases}$$

PROOF. Applying the Fritz-John Theorem, we know there are constants $u_0, u_1, \dots, u_m$ so that:

$$(10.51)\quad \begin{cases} u_0 \nabla f(\mathbf{x}^*) - \sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}^*) = \mathbf{0} \\ u_i g_i(\mathbf{x}^*) = 0 \quad i = 1, \dots, m \\ u_0, u_i \geq 0 \quad i = 1, \dots, m \\ [u_0, u_1, \dots, u_m]^T \neq \mathbf{0} \end{cases}$$

Suppose that $u_0 = 0$, then:

$$\sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}^*) = \mathbf{0} \tag{10.52}$$

$$[u_1, \dots, u_m]^T \neq \mathbf{0} \tag{10.53}$$

This is not possible by our assumption that $\nabla g_i(\mathbf{x}^*)$ $(i \in I)$ are linearly independent and that $u_j = 0$ for $j \notin I$. Thus $u_0 > 0$. Therefore, let $\lambda_i = u_i/u_0$. The result follows at once by dividing:

$$u_0 \nabla f(\mathbf{x}^*) - \sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}^*) = \mathbf{0}$$

by u_0. This completes the proof. □

Remark 10.37. The assumption that the set of vectors $\nabla g_i(\mathbf{x}^*)$ $(i \in I)$ are linearly independent is called a *constraint qualification*. There are several alternative weaker (and stronger) constraint qualifications. These are covered in depth in a course on convex optimization or the theory of nonlinear programming.

Theorem 10.38 (Fritz-John Sufficient Theorem). *Consider Problem P', where $f : \mathbb{R}^n \to \mathbb{R}$ is locally concave about $\mathbf{x}^*$ and $g_i : \mathbb{R}^n \to \mathbb{R}^n$ are locally strictly convex about $\mathbf{x}^*$. If there are scalars $u_0, u_1, \dots, u_m$ so that:*

$$\begin{cases} u_0 \nabla f(\mathbf{x}^*) - \displaystyle\sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}^*) = \mathbf{0} \\ u_i g_i(\mathbf{x}^*) = 0 \quad i = 1, \dots, m \\ u_0, u_i \geq 0 \quad i = 1, \dots, m \\ [u_0, u_1, \dots, u_m]^T \neq \mathbf{0} \end{cases} \tag{10.54}$$

then $\mathbf{x}^*$ is a local maximum of Problem P'. Moreover, if the concavity and convexity are global, then x^* is a global maximum.

Proof. Applying the same reasoning as in the proof of the Fritz-John theorem, we have that $F_0(\mathbf{x}^*) \cap G_0(\mathbf{x}^*) = \emptyset$. Suppose we restrict our attention only to the set $B_\epsilon(\mathbf{x}^*)$ in which f is concave and g_i $(i = 1, \dots, m)$ are strictly convex. In doing so, we can (if necessary) redefine f and g_i $(i = 1, \dots, m)$ outside this ball so that they are globally concave and strictly convex as needed. Then we may apply Proposition 10.33 to see that $\mathbf{x}^*$ is a local maximum.

Now suppose that global concavity and strict convexity hold. From Lemma 10.32, we know that $D(\mathbf{x}^*) = G_0(\mathbf{x}^*)$ and thus we know that $F_0(\mathbf{x}^*) \cap D(\mathbf{x}^*) = \emptyset$. Consider a relaxation of the problem in which only the constraints indexed in I are kept. Then we still have $F_0(\mathbf{x}^*) \cap D(\mathbf{x}^*) = \emptyset$ in our relaxed problem. For any feasible solution $\mathbf{x}^*$ in the relaxed feasible set we know that for all $\lambda \in [0, 1]$:

$$g(\mathbf{x}^* + \lambda(\mathbf{x} - \mathbf{x}^*)) < (1 - \lambda)g(\mathbf{x}^*) + \lambda g(\mathbf{x}) = 0 + \lambda g(\mathbf{x}) < 0 \tag{10.55}$$

Thus $\mathbf{p} = \mathbf{x} - \mathbf{x}^* \in D(\mathbf{x}_0)$; that is a $(\mathbf{x} - \mathbf{x}^*)$ is a feasible direction. The fact that $F_0(\mathbf{x}^*) \cap D(\mathbf{x}^*) = \emptyset$ means that $\nabla f(\mathbf{x}^*)^T \mathbf{p} \leq 0$ implies for all $\mathbf{x}$ in the (new) feasible region we have: $\nabla f(\mathbf{x}^*)^T (\mathbf{x} - \mathbf{x}^*) \leq 0$. Thus, by Theorem 10.5, $\mathbf{x}^*$ is a global maximum since the new feasible region must contain the original feasible region X. □

THEOREM 10.39. *Consider Problem P', where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is locally concave about $\mathbf{x}^*$ and $g_i : \mathbb{R}^n \rightarrow \mathbb{R}^n$ are locally strictly convex about $\mathbf{x}^*$. Denote the set $I = \{i \in \{1, \dots, m\} : g_i(\mathbf{x}^*) = 0\}$. Suppose that the set of vectors $\nabla g_i(\mathbf{x}^*)$ ($i \in I$) are linearly independent. If there exist scalars $\lambda_1, \dots, \lambda_m$ not all zero so that:*

$$\left\{\begin{aligned} &\nabla f(\mathbf{x}^*) - \sum_{i=1}^m \lambda_i \nabla g_i(\mathbf{x}^*) = \mathbf{0} \\ &\lambda_i g_i(\mathbf{x}^*) = 0 \quad i = 1, \dots, m \\ &\lambda_i \geq 0 \quad i = 1, \dots, m \end{aligned}\right. \tag{10.56}$$

then $\mathbf{x}^*$ is a local maximum of Problem P'. Moreover, if the concavity and convexity are global, then x^* is a global maximum.

EXERCISE 79. Use the sufficiency condition of the Fritz-John Theorem to prove the previous sufficient condition.

REMARK 10.40. We can extend both these theorems to our general problem P by re-writing the constraints $h_j(\mathbf{x}) = 0$ ($j = 1, \dots, l$) as the pair of constraints $h_j(\mathbf{x}) \leq 0$ and $-h_j(\mathbf{x}) \leq 0$. For the sufficiency to hold, we require $h_j(\mathbf{x})$ to be affine, as otherwise both $h_j(\mathbf{x})$ and $-h_j(\mathbf{x})$ ($j = 1, \dots, l$) could not both be strictly convex. These constraints yield a pair of multipliers $\rho_j \geq 0$ and $\nu_j \geq 0$ ($j = 1, \dots, l$) which satisfy:

$$\nabla f(\mathbf{x}^*) - \sum_{i=1}^m \lambda_i \nabla g_i(\mathbf{x}^*) - \sum_{j=1}^l (\rho_j - \nu_j) \nabla h(\mathbf{x}^*) = \mathbf{0} \tag{10.57}$$

as well as:

$$\begin{aligned} &\rho_j h_j(\mathbf{x}^*) = 0 \quad j = 1, \dots, l \\ &-\nu_j h_j(\mathbf{x}^*) = 0 \quad j = 1, \dots, l \end{aligned}$$

It is now a simple matter to define $\mu_j = (\rho_j - \nu_j)$ and note that it will be a value that is unrestricted in sign (since $\rho_j \geq 0$ and $\nu_j \geq 0$). We also note that $h_j(\mathbf{x}^*) = 0$ ($j = 1, \dots, l$) and thus, $\rho_j h_j(\mathbf{x}^*) = 0$ and $\nu h_j(\mathbf{x}^*) = 0$. The following theorems follow at once.

THEOREM 10.41 (Karush-Kuhn-Tucker Necessary Conditions). *Consider Problem P, where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $g_i : \rightarrow \mathbb{R}^n$ and $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$ are continuously differentiable at a local maximum $\mathbf{x}^*$. Denote the set $I = \{i \in \{1, \dots, m\} : g_i(\mathbf{x}^*) = 0\}$. Suppose that the set of vectors $\nabla g_i(\mathbf{x}^*)$ ($i \in I$) and $h_j(\mathbf{x}^*)$ ($j = 1, \dots, l$) are linearly independent. Then there exist scalars $\lambda_1, \dots, \lambda_m$ and $\mu_1, \dots, \mu_l$ not all zero so that:*

$$\left\{\begin{aligned} &\nabla f(\mathbf{x}^*) - \sum_{i=1}^m \lambda_i \nabla g_i(\mathbf{x}^*) - \sum_{j=1}^l \mu_j \nabla h_j(\mathbf{x}^*) = \mathbf{0} \\ &\lambda_i g_i(\mathbf{x}^*) = 0 \quad i = 1, \dots, m \\ &\lambda_i \geq 0 \quad i = 1, \dots, m \\ &\mu_j \text{ unrestricted} \quad j = 1, \dots, l \end{aligned}\right. \tag{10.58}$$

THEOREM 10.42. *Consider Problem P, where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is locally concave about $\mathbf{x}^*$ and $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ are locally strictly convex about $\mathbf{x}^*$ and $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$ are locally affine* *about* $\mathbf{x}^*$. *Denote the set* $I = \{i \in \{1, \dots, m\} : g_i(\mathbf{x}^*) = 0\}$. *Suppose that the set of vectors* $\nabla g_i(\mathbf{x}^*)$ *(*$i \in I$*) and* $\nabla h_j(\mathbf{x}^*)$ *are linearly independent. If there exist scalars* $\lambda_1, \dots, \lambda_m$ *and* $\mu_1, \dots, \mu_l$ *not all zero so that:*

$$(10.59) \quad \begin{cases} \nabla f(\mathbf{x}^*) - \displaystyle\sum_{i=1}^{m} \lambda_i \nabla g_i(\mathbf{x}^*) - \sum_{j=1}^{l} \mu_j \nabla h_j(\mathbf{x}^*) = \mathbf{0} \\ \lambda_i g_i(\mathbf{x}^*) = 0 \quad i = 1, \dots, m \\ \lambda_i \geq 0 \quad i = 1, \dots, m \\ \mu_j \ \textit{unrestricted} \quad j = 1, \dots, l \end{cases}$$

then $\mathbf{x}^*$ *is a local maximum of Problem P. Moreover, if the concavity and convexity are global and* h_j *(*$j = 1, \dots, l$*) are globally affine, then* $\mathbf{x}^*$ *is a global maximum.*

6. Quadratic Programming and Active Set Methods

LEMMA 10.43. *Consider the constrained optimization problem:*

$$(10.60) \quad Q_0 \quad \begin{cases} \max \ \mathbf{c}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} \\ s.t. \ \ \mathbf{A}\mathbf{x} = \mathbf{b} \end{cases}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{c} \in \mathbb{R}^{n \times 1}$ *and* $\mathbf{Q} \in \mathbb{R}^{n \times n}$ *is symmetric. Then the KKT conditions for Problem* Q_0 *are:*

$$(10.61) \quad \mathbf{Q}\mathbf{x} - \mathbf{A}^T\boldsymbol{\lambda} = -\mathbf{c}$$

$$(10.62) \quad \mathbf{A}\mathbf{x} = \mathbf{b}$$

□

EXERCISE 80. Prove Lemma 10.43.

LEMMA 10.44. *Suppose that* $\mathbf{A} \in \mathbb{R}^{n \times m}$ *has full row rank. Further, suppose that for all* $\mathbf{x} \in \mathbb{R}^n$ *such that* $\mathbf{A}\mathbf{x} = \mathbf{0}$ *and* $\mathbf{x} \neq \mathbf{0}$, *we have* $\mathbf{x}^T\mathbf{Q}\mathbf{x} < 0$, *then:*

$$(10.63) \quad \mathbf{M} = \begin{bmatrix} \mathbf{Q} & -\mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix}$$

is non-singular.

PROOF. Suppose that we have a solution pair $(\mathbf{x}, \boldsymbol{\lambda})$ so that

$$(10.64) \quad \begin{bmatrix} \mathbf{Q} & -\mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \mathbf{0}$$

From this we conclude that:

$$(10.65) \quad \mathbf{Q}\mathbf{x} - \mathbf{A}^T\boldsymbol{\lambda} = \mathbf{0}$$

$$(10.66) \quad \mathbf{A}\mathbf{x} = \mathbf{0}$$

Thus, $\mathbf{x}$ is in the null space of $\mathbf{A}$. From Equation 10.65 we deduce:

$$(10.67) \quad \mathbf{x}^T\mathbf{Q}\mathbf{x} - \mathbf{x}^T\mathbf{A}^T\boldsymbol{\lambda} = 0$$

But, $\mathbf{x}^T\mathbf{A}^T = \mathbf{0}$ and therefore $\mathbf{x}^T\mathbf{Q}\mathbf{x} = 0$. This can only be true if $\mathbf{x} = \mathbf{0}$ by our assumption. Thus we have shown that necessarily $\mathbf{x} = \mathbf{0}$. Given this, we see that:

$$-\mathbf{A}^T\boldsymbol{\lambda} = \mathbf{0} \tag{10.68}$$

This implies that: $\boldsymbol{\lambda}^T\mathbf{A} = \mathbf{0}^T$. Note that $\boldsymbol{\lambda}^T\mathbf{A}$ is a linear combination of the rows of $\mathbf{A}$, which we assumed to be linearly independent (since $\mathbf{A}$ had full row rank). Thus for Equation 10.68 to hold, we must have $\lambda = \mathbf{0}$. Thus, the null space of $\mathbf{M}$ consists only of the zero-vector and $\mathbf{M}$ is non-singular. □

Lemma 10.45. *Suppose the conditions of the Lemma 10.44 are satisfied. Then Problem Q_0 has a unique solution.*

Proof. It is easy to see that any pair $(\mathbf{x}, \boldsymbol{\lambda})$ satisfying:

$$\begin{bmatrix} \mathbf{Q} & -\mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\mathbf{c} \\ \mathbf{b} \end{bmatrix} \tag{10.69}$$

is a KKT point. Suppose that $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ is a solution to Equation 10.69 and suppose that $z(\mathbf{x}^+) \geq z(\mathbf{x}^*)$ where:

$$z(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} \tag{10.70}$$

Define $\mathbf{p} = \mathbf{x}^+ - \mathbf{x}^*$. Then $\mathbf{A}\mathbf{p} = \mathbf{A}\mathbf{x}^+ - \mathbf{A}\mathbf{x}^* = \mathbf{b} - \mathbf{b} = \mathbf{0}$ and thus $\mathbf{p}$ is in the null space of $\mathbf{A}$. Consider that:

$$\begin{multline} z(\mathbf{x}^+) = z(\mathbf{x}^* + \mathbf{p}) = \mathbf{c}^T(\mathbf{x}^* + \mathbf{p}) + \frac{1}{2}(\mathbf{x}^* + \mathbf{p})^T\mathbf{Q}(\mathbf{x}^* + \mathbf{p}) = \\ z(\mathbf{x}^*) + \mathbf{c}^T\mathbf{p} + \frac{1}{2}\mathbf{p}^T\mathbf{Q}\mathbf{p} + \frac{1}{2}\left(\mathbf{x}^{*T}\mathbf{Q}\mathbf{p} + \mathbf{p}^T\mathbf{Q}\mathbf{x}^*\right) \end{multline} \tag{10.71}$$

We've assumed $\mathbf{Q}$ is symmetric and thus:

$$\frac{1}{2}\left(\mathbf{x}^{*T}\mathbf{Q}\mathbf{p} + \mathbf{p}^T\mathbf{Q}\mathbf{x}^*\right) = \mathbf{p}^T\mathbf{Q}\mathbf{x}^* \tag{10.72}$$

By the necessity of the KKT conditions, we know that $\mathbf{Q}\mathbf{x}^* = -\mathbf{c} + \mathbf{A}^T\boldsymbol{\lambda}^*$. From this we conclude that:

$$\mathbf{p}^T\mathbf{Q}\mathbf{x}^* = -\mathbf{p}^T\mathbf{c} + \mathbf{p}^T\mathbf{A}^T\boldsymbol{\lambda}^*$$

We can substitute this quantity back into Equation 10.71 to obtain:

$$z(\mathbf{x}^+) = z(\mathbf{x}^*) + \mathbf{c}^T\mathbf{p} + \frac{1}{2}\mathbf{p}^T\mathbf{Q}\mathbf{p} - \mathbf{p}^T\mathbf{c} + \mathbf{p}^T\mathbf{A}^T\boldsymbol{\lambda}^* \tag{10.73}$$

We've already observed that $\mathbf{p}$ is in the null space of $\mathbf{A}$ and thus $\mathbf{p}^T\mathbf{A}^T = \mathbf{0}$. Likewise, $\mathbf{c}^T\mathbf{p} = \mathbf{p}^T\mathbf{c}$ and thus:

$$z(\mathbf{x}^+) = z(\mathbf{x}^*) + \frac{1}{2}\mathbf{p}^T\mathbf{Q}\mathbf{p} \tag{10.74}$$

Finally, by our assumption on $\mathbf{Q}$, the fact that $\mathbf{A}\mathbf{p} = \mathbf{0}$ implies that $\mathbf{p}^T\mathbf{Q}\mathbf{p} < 0$ just in case $\mathbf{p} \neq \mathbf{0}$. Thus: $z(\mathbf{x}^+) < z(\mathbf{x}^*)$ and $\mathbf{x}^*$ must be a unique global optimal solution to Problem Q_0. □

Remark 10.46. Maple code for solving equality constrained quadratic programming problems with concave objective functions is shown in Algorithm 23. We also include a helper function to determine whether matrices or vectors are empty in Maple.

```
IsEmpty := overload([
  proc (A::Matrix)::boolean;
    option overload;
    uses LinearAlgebra:
    if evalb(RowDimension(A) = 0 or ColumnDimension(A) = 0) then
      true:
    else
      false:
    end if:
  end proc,
  proc (V::Vector)::boolean;
    option overload;
    uses LinearAlgebra:
    if evalb(Dimension(V) = 0) then
      true:
    else
      false:
    end if
  end proc
]):
```

Algorithm 22. An algorithm to determine whether a matrix is empty in Maple. Notice this is an overloaded method, allowing us to pass in Vectors or Matrices.

REMARK 10.47. We can use the previous lemmas to develop an algorithm for solving convex quadratic programming problems. Before proceeding, we state the following theorem, which we do not prove. See [Sah74].

THEOREM 10.48. *Let*

(1) $\mathbf{Q} \in \mathbb{R}^{n\times n}$,
(2) $\mathbf{A} \in \mathbb{R}^{m\times n}$,
(3) $\mathbf{H} \in \mathbb{R}^{l\times n}$,
(4) $\mathbf{b} \in \mathbb{R}^{m\times 1}$,
(5) $\mathbf{r} \in \mathbb{R}^{l\times 1}$ *and*
(6) $\mathbf{c} \in \mathbb{R}^{n\times 1}$.

Then finding a solution for the problem:

$$\left\{\begin{array}{rl} \max & \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ s.t. & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{H}\mathbf{x} = \mathbf{r} \end{array}\right. \tag{10.75}$$

is NP*-complete for arbitrary parameters.* □

REMARK 10.49. A general algorithm for solving arbitrary quadratic programming problems is given in [BSS06]. It is, essentially, a branch-and-bound style algorithm befitting the fact that the problem is NP-complete.

```
SolveEQQP := proc (c::Vector, Q::Matrix, A::Matrix, b::Vector)::list;
  uses LinearAlgebra:
  local M, r, n, m, y;
  n := RowDimension(Q);
  m := RowDimension(A);
  assert(n = ColumnDimension(A));
  assert(n = Dimension(c));
  assert(m = Dimension(b));

  if IsEmpty(A) then
    M := Q:
    r := -c:
  else
    #Construct some augmented matrices and vectors
    M := <<Q|-Transpose(A)>;<A|ZeroMatrix(m,m)>>:
    r := <-c ; b>
  end if;
  #More efficient than inverting.
  y := LinearSolve(M, r);
  if IsEmpty(A) then
    #Return the solution, there are no Lagrange multipliers.
    [y, []]:
  else
    #Return the solution and the Lagrange multipliers in a list.
    [SubVector(y, [1 .. n]), SubVector(y, [n+1 .. n+m])]
  end if:
end proc:
```

Algorithm 23. An algorithm to solve equality constrained quadratic programming problems, under the forgoing assumptions. **Note we do not test for full row rank.**

Remark 10.50. We now consider an active set method for solving a convex quadratic programming problem, which will be used in the next chapter when developing sequential quadratic programming.

Remark 10.51. The algorithm assumes a problem with the following structure:

$$(10.76)\quad QP\begin{cases}\max & \dfrac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x}+\mathbf{c}^T\mathbf{x}\\ s.t. & \mathbf{a}_i^T\mathbf{x}\leq b_i \quad i\in I\\ & \mathbf{a}_i^T\mathbf{x}=b_i \quad i\in E\end{cases}$$

Obviously, this is identical to the formulation of the problem in (10.75) except we have index sets E and I for the equality and inequality constraints, respectively. Furthermore, we will assume that $\mathbf{Q}$ is negative definite.

Lemma 10.52. *Suppose that* $\mathbf{x}_k$ *is a feasible solution to QP. Let*

$$W_k = \{i \in I : \mathbf{a}_i^T\mathbf{x}_k = b_i\} \cup E$$

Consider the problem:

$$(10.77)\quad QP(\mathbf{x}_k)\begin{cases}\max\ \mathbf{p}^T(\mathbf{c} + \mathbf{Q}\mathbf{x}_k) + \dfrac{1}{2}\mathbf{p}^T\mathbf{Q}\mathbf{p}\\ s.t.\ \ \mathbf{a}_i^T\mathbf{p} = 0 \quad i \in W_k\end{cases}$$

If $\mathbf{p}_k$ *solves this problem, then* $\mathbf{x}_k + \mathbf{p}_k$ *solves:*

$$(10.78)\quad \begin{aligned}\max\ & \mathbf{c}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x}\\ s.t.\ & \mathbf{a}_i^T\mathbf{x} = b_i \quad i \in W_k\end{aligned}$$

Proof. By Lemmas 10.44 and 10.45 and our assumption on $\mathbf{Q}$, there is only one unique solution to each of the problems given in the statement of the theorem. Suppose $\mathbf{p}_k$ is the solution to the first problem. Then since $\mathbf{a}_i^T\mathbf{p}_k = 0$ for all $i \in W_k$ and we know that $\mathbf{a}_i^T\mathbf{x}_k = b_i$ for all $i \in W_k$ it follows that $\mathbf{a}_i^T(\mathbf{x}_k + \mathbf{p}_k) = b_i$ for all $i \in W_k$ and thus $\mathbf{x}_k + \mathbf{p}_k$ is feasible to the second problem in the statement of the theorem. Note that:

$$(10.79)\quad \mathbf{c}^T(\mathbf{x}_k + \mathbf{p}_k) + \frac{1}{2}(\mathbf{x}_k + \mathbf{p}_k)^T\mathbf{Q}(\mathbf{x}_k + \mathbf{p}_k) = z(\mathbf{x}_k) + \mathbf{c}^T\mathbf{p}_k + \frac{1}{2}\mathbf{p}_k^T\mathbf{Q}\mathbf{p}_k + \mathbf{p}_k^T\mathbf{Q}\mathbf{x}_k$$

just as it was in Equation 10.71 where again: $z(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x}$. Thus:

$$(10.80)\quad \mathbf{c}^T(\mathbf{x}_k + \mathbf{p}_k) + \frac{1}{2}(\mathbf{x}_k + \mathbf{p}_k)^T\mathbf{Q}(\mathbf{x}_k + \mathbf{p}_k) = z(\mathbf{x}_k) + \mathbf{p}_k^T(\mathbf{c} + \mathbf{Q}\mathbf{x}_k) + \frac{1}{2}\mathbf{p}_k^T\mathbf{Q}\mathbf{p}_k$$

Since $\mathbf{p}_k$ was chosen to maximize $\mathbf{p}_k^T(\mathbf{c} + \mathbf{Q}\mathbf{x}_k) + \frac{1}{2}\mathbf{p}_k^T\mathbf{Q}\mathbf{p}_k$ subject to the constraints that $\mathbf{a}_i^T\mathbf{p}_k = 0$ for all $i \in W_k$, it follows that $\mathbf{x}_k + \mathbf{p}_k$ must be the unique solution to the second problem in the statement of the theorem, for otherwise, we could have chosen a different $\mathbf{p}_k'$ that satisfied $\mathbf{a}_i^T(\mathbf{x}_k + \mathbf{p}_k') = b_i$ for all $i \in W_k$ with $\mathbf{x}_k + \mathbf{p}_k'$ producing a larger objective function value and this would have been the maximal solution to the first problem. This completes the proof. □

Remark 10.53. Note that the sub-problem:

$$(10.81)\quad QP(\mathbf{x}_k)\begin{cases}\max\ \mathbf{p}^T(\mathbf{c} + \mathbf{Q}\mathbf{x}_k) + \dfrac{1}{2}\mathbf{p}^T\mathbf{Q}\mathbf{p}\\ s.t.\ \ \mathbf{a}_i^T\mathbf{p} = 0 \quad i \in W_k\end{cases}$$

can be solved using simple linear algebra as illustrated in Lemma 10.45 to obtain not just $\mathbf{p}_k$ but also a set of Lagrange multipliers.

Lemma 10.54. *Suppose that* $\mathbf{x}_k$ *is a feasible solution to QP. Let*

$$I_k = \{i \in I : \mathbf{a}_i^T\mathbf{x}_k = b_i\}$$

so that (as in Lemma 10.52) $W_k = I_k \cup E$. *Let* $\boldsymbol{\lambda}$ *be the Lagrange multipliers obtained for the problem:*

$$QP(\mathbf{x}_k)\begin{cases}\max\ \mathbf{p}^T(\mathbf{c} + \mathbf{Q}\mathbf{x}_k) + \dfrac{1}{2}\mathbf{p}^T\mathbf{Q}\mathbf{p}\\ s.t.\ \ \mathbf{a}_i^T\mathbf{p} = 0 \quad i \in W_k\end{cases}$$

when finding the optimal solution $\mathbf{p}_k$. *If* $\mathbf{p}_k = \mathbf{0}$ *and* $\boldsymbol{\lambda}_i \geq 0$ *for all* $i \in I_k$, *then* $\mathbf{x}_k$ *is the optimal solution to QP.*

Exercise 81. Prove Lemma 10.54. [Hint: Clearly, $\boldsymbol{\lambda}$ can act as Lagrange multipliers for all the constraints indexed in W_k. Define the remaining Lagrange multipliers for constraints with index not in W_k to be zero and argue that the KKT conditions for the whole problem are satisfied because they are satisfied for the problem $\max \mathbf{c}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x}$ such that $\mathbf{a}_i^T\mathbf{x} = b_i$ for all $i \in W_k$.]

Lemma 10.55. *Suppose that* $\mathbf{p}_k \neq \mathbf{0}$ *and suppose further that if* $\mathbf{x}_k + \mathbf{p}_k$ *is feasible to QP, then* $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$ *and otherwise, using a minimum ratio test, we set:*

$$\alpha_k = \min\left\{\frac{\mathbf{b}_q - \mathbf{a}_q^T\mathbf{x}_k}{\mathbf{a}_q^T\mathbf{p}_k} : q \in I \setminus I_k, \mathbf{a}_q^T\mathbf{p}_k > 0\right\} \tag{10.82}$$

and define $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$. *Then* $\mathbf{x}_{k+1}$ *is feasible to QP,* $z(\mathbf{x}_{k+1}) > z(\mathbf{z}_k)$ *and* $I_{k+1} = I_k \cup \{q^*\}$ *where* q^* *is the index identified in the minimum ratio test of Expression 10.82.*

Proof. In the case when $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$ is feasible to QP, the result follows immediately from Lemma 10.52. Suppose, therefore, $\mathbf{x}_k + \mathbf{p}_k$ is not feasible to QP.

To prove feasibility, we observe that $\mathbf{a}_i^T\mathbf{x}_{k+1} = b_i$ for all $i \in W_k$ by the proof of Lemma 10.52. Thus, it follows that we wish to ensure that:

$$\mathbf{a}_i^T(\mathbf{x}_k + \alpha_k\mathbf{p}_k) \leq b_i \quad i \in I \setminus I_k \tag{10.83}$$

The previous equation implies that for all $i \in I \setminus I_k$ we have:

$$\alpha_k \leq \frac{\mathbf{b}_q - \mathbf{a}_q^T\mathbf{x}_k}{\mathbf{a}_q^T\mathbf{p}_k} \tag{10.84}$$

Clearly Expression 10.83 can be made true by setting using the minimum ratio test given in Expression 10.82, since if $\mathbf{a}_i^T\mathbf{p}_k < 0$, then we are assured that if $\mathbf{a}_i^T\mathbf{x}_k < b_i$, then so to $\mathbf{a}_i^T\mathbf{x}_{k+1} < b_i$.

To see that $z(\mathbf{x}_{k+1}) > z(\mathbf{x}_k)$, recall we have assumed that $\mathbf{Q}$ is negative definite. Thus, the objective function is strictly concave. We know from Lemma 10.52 that $z(\mathbf{x}_k + \mathbf{p}_k) > z(\mathbf{x}_k)$ (in particular, since $\mathbf{p}_k \neq 0$). Let $\mathbf{y}_k = \mathbf{x}_k + \mathbf{p}_k$. Then there is some $\lambda \in (0, 1)$ so that: $\lambda\mathbf{x}_k + (1-\lambda)\mathbf{y}_k = \mathbf{x}_k + \alpha_k\mathbf{p}_k$. From this we deduce that:

$$\begin{multline} z(\mathbf{x}_{k+1}) = z(\mathbf{x}_k + \alpha_k\mathbf{p}_k) = z(\lambda\mathbf{x}_k + (1-\lambda)\mathbf{y}_k) > \\ \lambda z(\mathbf{x}_k) + (1-\lambda)z(\mathbf{y}_k) > \lambda z(\mathbf{x}_k) + (1-\lambda)z(\mathbf{x}_k) = z(\mathbf{x}_k) \end{multline} \tag{10.85}$$

This completes the proof. □

Remark 10.56. We are now in a position to sketch the active set algorithm for solving convex quadratic programming problems.

(1) Initialize with a feasible point $\mathbf{x}_0$ derived possibly from a Phase I problem on the linear constraints. Set $k = 0$. Compute I_k and W_k.
(2) Solve $QP(\mathbf{x}_k)$ to obtain $\mathbf{p}_k$ and $\boldsymbol{\lambda}_k$.
(3) If $\mathbf{p}_k = \mathbf{0}$ and $\boldsymbol{\lambda}_{k_i} \geq 0$ for all $i \in I_k$, then stop, $\mathbf{x}_k$ is optimal.
(4) If $\mathbf{p}_k = \mathbf{0}$ and for some $i \in I_k$, $\boldsymbol{\lambda}_{k_i} < 0$, then set $i^* = \arg\min\{\boldsymbol{\lambda}_{k_i} : i \in I_k\}$. Remove i^* from I_k to obtain I_{k+1} and set $W_{k+1} = W_k \setminus i^*$. Set $\mathbf{x}_{k+1} = \mathbf{x}_k$. Set $k = k+1$, Goto (2).

(5) If $\mathbf{p}_k \neq \mathbf{0}$ and $\mathbf{x}_k + \mathbf{p}_k$ is feasible, set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$. Compute I_{k+1} and W_{k+1}. Set $k = k + 1$, Goto (2).

(6) If $\mathbf{p}_k \neq \mathbf{0}$ and $\mathbf{x}_k + \mathbf{p}_k$ is not feasible then compute:

$$\alpha_k = \min\left\{\frac{\mathbf{b}_q - \mathbf{a}_q^T\mathbf{x}_k}{\mathbf{a}_q^T\mathbf{p}_k} : q \in I \setminus I_k, \mathbf{a}_q^T\mathbf{p}_k > 0\right\}$$

Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$. Compute I_{k+1} and W_{k+1}. Set $k = k + 1$, Goto (2). Note, I_{k+1} and W_{k+1} can be computed easily as a part of the minimum ratio test.

A reference implementation for the algorithm in Maple is shown in Algorithm 27.

THEOREM 10.57. *The algorithm described in Remark 10.56 and illustrated in Algorithm 27 converges in a finite number of steps to the optimal solution of Problem 10.76 provided that* $\mathbf{Q}$ *is negative definite (or that* $z(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x}$ *is strictly concave).*

SKETCH OF PROOF. It suffices to show that a working (active) set W_k is never repeated. To see this note that if $\mathbf{p}_k = \mathbf{0}$ and we have not achieved optimality, then a binding constraint is removed from W_k to obtain W_{k+1}. This process continues until $\mathbf{p}_k \neq \mathbf{0}$, at which point by Lemma 10.82, the objective function decreases. Since the objective is monotonically decreasing and when it does not decrease, we force a chance in W_k, it is easy to see we can never repeat a working set. The fact that there are a finite number of working sets, it follows that the algorithm must converge after a finite number of iterations to a solution with $\boldsymbol{\lambda}_{k_i} > 0$ for all $i \in I_k$. By Lemmas 10.52 and 10.54 it follows this must be an optimal solution for the Problem 10.76. ☐

REMARK 10.58. The "proof" for the convergence of the active set method given in [**NW06**] is slightly more complete than the sketch of the proof given above, though the major elements of the proof are identical.

REMARK 10.59. The point of introducing this specialized quadratic programming algorithm is that it can be used very efficiently in implementing sequential quadratic programming (SQP) because, through judicious use of the BFGS method, we can ensure that we always have a convex quadratic sub-problem. The SQP method is a generally convergent approach for solving constraint non-linear programming problems that enjoys substantial commercial success. (Matlab's `fmincon` and Maple s `Minimize` functions both use it.) We introduce this technique in the next chapter.

REMARK 10.60. We end this chapter by providing a Maple implementation of the active set method for solving convex quadratic programming problems. This algorithm is long and requires sub-routines for finding an initial point and determining the index of the minimum value in a vector and determining the (initial) set of binding constraints I_0. The actual routine for solving quadratic programs is shown in Algorithm 27.

EXAMPLE 10.61. Consider the following problem:

$$(10.86)\quad \begin{cases} \max\ \dfrac{1}{2}(x-1)^2 + \dfrac{1}{2}(y-1)^2 \\ \quad s.t. x + y \leq 1 \\ \quad\quad 2x + y \geq 1 \\ \quad\quad x, y \geq \dfrac{1}{4} \end{cases}$$

```
MinimumIndex := proc (V::Vector)::integer;
  local leastValue, index, i;
  leastValue := Float(infinity);
  for i to Dimension(V) do
    if evalb(V[i] < leastValue) then
      leastValue := V[i];
      index := i
    end if:
  end do;
  index:
end proc
```

Algorithm 24. An algorithm to determine the index of the minimum value in a Vector in Maple.

```
FindInitialSolution := proc (A::Matrix, b::Vector, Aeq::Matrix, beq::Vector)::
    Vector;
  uses LinearAlgebra, Optimization:
  local N, c;
  N := max(ColumnDimension(A), ColumnDimension(Aeq));
  c := Vector([seq(0, i = 1 .. N)]);
  if ColumnDimension(A) = 0 and ColumnDimension(Aeq) = 0 then
    #Return c, it doesn't matter what you return. There are no constraints.
    c:
  elif ColumnDimension(A) = 0 then
    LPSolve(c, [NoUserValue, NoUserValue, Aeq, beq])[2]
  elif ColumnDimension(Aeq) = 0 then
    LPSolve(c, [A, b])[2]
  else
    LPSolve(c, [A, b, Aeq, beq])[2]
  end if:
end proc:
```

Algorithm 25. A simple routine for initializing the active set quadratic programming method.

If we expand this problem out, constructing the appropriate matrices we see that:

$$\mathbf{Q} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$
$$\mathbf{c} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

```
ReturnWorkingSet := proc (A::Matrix, b::Vector, x::Vector)::list;
  local m, l, i, Ik;
  m := RowDimension(A);
  Ik := [];
  #This is a not equal.
  if evalb(m <> 0) then
    assert(Dimension(x) = ColumnDimension(A));
    assert(m = Dimension(b));
    for i to m do
      if evalb(Row(A, i).x = b[i]) then
        #Append any binding rows to Ik.
        Ik := [op(Ik), i]:
      end if:
    end do:
  end if:
  Ik:
end proc:
```

Algorithm 26. A routine to determine the binding constraints among the inequality constraints.

```
SolveConvexQP := proc (c::Vector, Q::Matrix,
             A::Matrix, b::Vector,
             Aeq::Matrix, beq::Vector)::list;
  uses LinearAlgebra:
  local xnow, bDone, M, Ik, OUT, r, n, ynow, eps, alpha, alphak, cc,
    rebuildM, Ap, i, p, SOLN, lambda, pos, q, pA, pAeq;

  eps := 1/1000000;
  xnow := FindInitialSolution(A, b, Aeq, beq); #Initialize with Linear
      Programming!
  n :=Dimension(xnow);

  rebuildM := true; #A variable that tells us whether we have to rebuild a matrix
      .
  Ik := ReturnWorkingSet(A, b, xnow); #Find the working (active set). Only called
      once.
  bDone := false;

  #Return list...we return more than is necessary.
  #Specifically, the path we take.
  OUT := [];
```

```
while evalb(not bDone) do
  OUT := [op(OUT), convert(xnow, list)];
  if rebuildM then #Rebuild the matrix that gets past to the sub-QP problem.
    if nops(Ik) <> 0 and evalb(not IsEmpty(Aeq)) then
      #Create a augmented matrix...these are the binding constraints.
      #Some of the inequality constraints are binding.
      M := <SubMatrix(A, Ik, [1 .. n]); Aeq>:
    elif evalb(not IsEmpty(Aeq)) then
      M := Aeq
    elif nops(Ik) <> 0 then
      M := SubMatrix(A, Ik, [1 .. n])
    else
      M := Matrix(0, 0)
    end if;
  end if;
  cc := c+Q.xnow;
  r := Vector([seq(0, i = 1 .. RowDimension(M))]);
  #Here's where we solve the sub-problem.
  SOLN := SolveEQQP(cc, Q, M, r);
  p := SOLN[1]; #The direction to move.
  lambda := SOLN[2]; #The Lagrange multipliers.

  if Norm(p) < eps then
    if nops(Ik) = 0 then
      bDone := true
    else
      pos := MinimumIndex(SubVector(lambda, [1 .. nops(Ik)]));
      if 0 <= lambda[pos] then
        #We're done! All Lagrange multipliers that need to be positive
        #are.
        bDone := true
      else
        #Delete the row corresponding to the "most offending" Lagrange
        #multiplier
        M := DeleteRow(M, pos);
        #The next line is a little tortured...it's to remove the right
        #row number from Ik.
        Ik := convert((convert(Ik, set) minus convert({Ik[pos]}, set)), list);
        rebuildM := false;
      end if
    end if
  else
    alphak := 1;
    q := -1;
    for i to RowDimension(A) do
      if evalb(not i in Ik) then
        Ap := Row(A, i).p;
        if 0 < Ap then #Otherwise, it doesn't matter.
          alpha := (b[i]-Row(A, i).xnow)/Ap;
          if alpha < alphak then
            alphak := alpha;
            q := i
          end if;
        end if;
      end if;
    end do;
    if q <> -1 and alphak <= 1 then
      Ik := [op(Ik), q]; #A new constraint becomes binding.
      rebuildM := true:
    else
      alphak := 1: #No new constraints become binding.
    end if;

    xnow := xnow+alphak*p:
  end if:
end do;
#We return two vectors of dual variables, one for the inequality constraints.
#and one for the equality constraints.
pA := Vector(RowDimension(A));
pAeq := Vector(RowDimension(Aeq));
for i to nops(Ik) do
  pA[Ik[i]] := lambda[i]:
end do;
for i from nops(Ik)+1 to Dimension(lambda) do
  pAeq[i-nops(Ik)] := lambda[i]:
end do;
OUT := [[op(OUT), convert(xnow, list)], xnow, pA, pAeq]:
end proc:
```

Algorithm 27. The active set convex quadratic programming algorithm. The algorithm uses all the sub-routines developed for quadratic programming.

Our matrices are:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ -2 & -1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 1 \\ -1 \\ -\frac{1}{4} \\ -\frac{1}{4} \end{bmatrix}$$

$$\mathbf{A}_{\text{eq}} = [] \quad \mathbf{b}_{\text{eq}} = []$$

Since we have no equality constraints, we use empty matrices for these terms. Passing this into our optimization algorithm yields an optimal solution of $x = y = \frac{1}{2}$. The path the algorithm takes is shown in Figure 10.5.

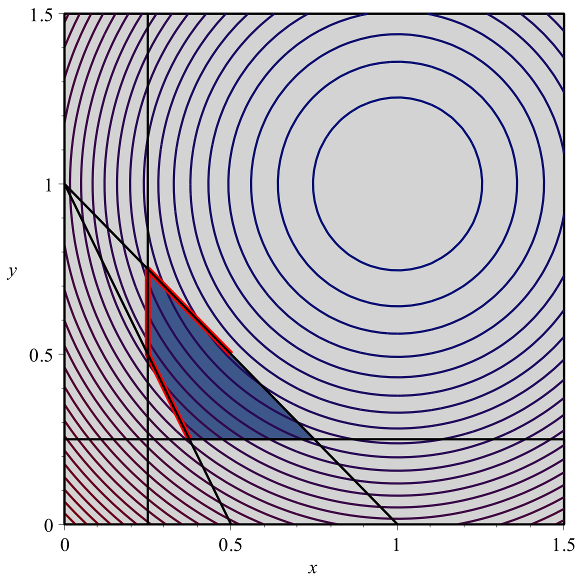


Figure 10.5. The path taken when solving the proposed quadratic programming problem using the active set method. Notice we tend to hug the outside of the polyhedral set.

Exercise 82. Work through the steps of the algorithm by hand for the problem:

$$\left\{\begin{aligned} &\max\ \frac{1}{2}(x-1)^2 + \frac{1}{2}(y-1)^2 \\ &\quad s.t. x + y \leq 1 \\ &\qquad x, y \geq 0 \end{aligned}\right. \tag{10.87}$$

Remark 10.62. As we noted in the previous example, the active set method still tends to hug the boundary of the polyhedral constraint set. Interior point methods are known to converge more quickly. We will discuss these methods for both linear and quadratic programs in the network chapter, after we introduce barrier methods.

CHAPTER 11

Penalty and Barrier Methods, Sequential Quadratic Programming, Interior Point Methods

For simplicity in this chapter, we will *not* study maximization problems, since penalty functions are more easily understood in terms of *minimization* (i.e., we will minimize a penalty). Thus, we will focus on the following problem:

$$(11.1)\quad \begin{cases} \min & f(x_1, \dots, x_n) \\ s.t. & g_1(x_1, \dots, x_n) \leq 0 \\ & \vdots \\ & g_m(x_1, \dots, x_n) \leq 0 \\ & h_1(x_1, \dots, x_n) = 0 \\ & \vdots \\ & h_l(x_1, \dots, x_n) = 0 \end{cases}$$

With a little work, it is easy to see that this problem has as its dual feasibility and complementary slackness conditions:

$$(11.2)\quad \begin{cases} \nabla f(\mathbf{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\mathbf{x}^*) + \sum_{j=1}^{l} \mu_j \nabla h_j(\mathbf{x}^*) = \mathbf{0} \\ \qquad\qquad \lambda_i g_i(\mathbf{x}^*) = 0 \quad i = 1, \dots, m \\ \qquad\qquad \lambda_i \geq 0 \quad i = 1, \dots, m \\ \qquad\qquad \mu_j \text{ unrestricted} \quad j = 1, \dots, l \end{cases}$$

1. Penalty Methods

Proposition 11.1. *Let $g : \mathbb{R}^n \to \mathbb{R}$ and suppose that $r : \mathbb{R} \to \mathbb{R}$ is a strictly convex function with minimum at $x = 0$ so that $r(0) = 0$. Then:*

$$(11.3)\quad p_I(\mathbf{x}) = r\left(\max\{0, g(\mathbf{x})\}\right)$$

is 0 if and only if $\mathbf{x}$ satisfies $g(\mathbf{x}) \leq 0$. Furthermore, suppose that:

$$(11.4)\quad p_E(\mathbf{x}) = r(g(\mathbf{x}))$$

is 0 if and only if $g(\mathbf{x}) = 0$.

Exercise 83. Prove Proposition 11.1

Definition 11.2. Let $g : \mathbb{R}^n \to \mathbb{R}$ and suppose that $r : \mathbb{R} \to \mathbb{R}$ is a strictly convex function with minimum at $x = 0$ so that $r(0) = 0$. For a constraint of type $g(\mathbf{x}) \leq 0$,

$p_I(\mathbf{x})$ defined as in Equation 11.3 is an *inequality penalty function*. For a constraint of type $g(\mathbf{x}) = 0$, $p_E(\mathbf{x})$ defined as in Equation 11.4 is an *equality penalty function*.

Theorem 11.3. *Consider the inequality penalty function with $r(x) = x^2$ and inequality $g(\mathbf{x}) \leq 0$. If $g(\mathbf{x})$ is continuous and differentiable, then $p_I(\mathbf{x})$ is differentiable with derivative:*

$$\frac{\partial p_I(\mathbf{x})}{\partial x_i} = \begin{cases} 2g(\mathbf{x})\frac{\partial g}{\partial x_i} & g(\mathbf{x}) \leq 0 \\ 0 & \textit{otherwise} \end{cases} \tag{11.5}$$

Exercise 84. Prove Theorem 11.3

Remark 11.4. We can use penalty functions to transform constrained optimization problems into unconstrained optimization problems. To see this, consider a simple version of Problem 11.1:

$$\begin{cases} \min & f(x_1, \dots, x_n) \\ s.t. & g(x_1, \dots, x_n) \leq 0 \\ & h(x_1, \dots, x_n) = 0 \end{cases}$$

Then the penalized optimization problem is:

$$\min \quad f_P(x_1, \dots, x_n) = f(x_1, \dots, x_n) + \lambda_I p_I(x_1, \dots, x_n) + \lambda_E p_E(x_1, \dots, x_n) \tag{11.6}$$

where $\lambda_I, \lambda_E \in \mathbb{R}_+$. If we choose $r(x) = x^2$ (e.g.) then as long as the constraints and objective in the original problem are differentiable, then $f_P(\mathbf{x})$ is differentiable and we can apply unconstrained optimization methods that use derivatives. Otherwise, we must use non-differentiable methods.

Remark 11.5. In a penalty function method if λ_I or λ_E are chosen too large, then the penalty functions will overwhelm the objective function. If these values are chosen to small, then the resulting optimal value $\mathbf{x}^*$ for f_P may not be feasible.

Example 11.6. Consider the following simple

2. Sequential Quadratic Programming

3. Barrier Methods

4. Interior Point Simplex as a Barrier Method

5. Interior Point Methods for Quadratic Programs

Bibliography

[Ant94] H. Anton, *Elementary linear algebra*, 7 ed., John Wiley and Sons, 1994.

[Avr03] M. Avriel, *Nonlinear programming: Analysis and methods*, Dover Press, 2003.

[Ber99] D. P. Bertsekas, *Nonlinear Programming*, 2 ed., Athena Scientific, 1999.

[BJS04] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali, *Linear programming and network flows*, Wiley-Interscience, 2004.

[BSS06] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear programming: Theory and algorithms*, John Wiley and Sons, 2006.

[Dem97] J. W. Demmel, *Applied numerical linear algebra*, SIAM Publishing, 1997.

[Ger31] S. Gerschgorin, *Über die Abgrenzung der Eigenwerte einer Matrix*, Izv. Akad. Nauk. USSR Otd. Fiz.-Mat. Nauk **7** (1931), 749–754.

[GR01] C. Godsil and G. Royle, *Algebraic graph theory*, Springer, 2001.

[Gri11] C. Griffin, *Linear programming: Penn state math 484 lecture notes (v 1.8)*, http://www.personal.psu.edu/cxg286/Math484_V1.pdf, 2010-2011.

[HJ61] R. Hooke and T. A. Jeeves, *Direct search solution of numerical and statistical problems.*, J. ACM (1961).

[MT03] J. E. Marsden and A. Tromba, *Vector calculus*, 5 ed., W. H. Freeman, 2003.

[Mun00] J. Munkres, *Topology*, Prentice Hall, 2000.

[NL89] J. Nocedal and D. C. Liu, *On the limited memory method for large scale optimization*, Math. Program. Ser. B **45** (1989), no. 3, 503–528.

[NW06] J. Nocedal and S. Wright, *Numerical optimization*, Springer, 2006.

[PTVF07] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes 3rd edition: The art of scientific computing*, 3 ed., Cambridge University Press, 2007.

[Rud76] W. Rudin, *Principles of mathematical analysis*, McGraw-Hill, 1976.

[Sah74] S. Sahni, *Computationally related problems*, SIAM J. Comput. **3** (1974), 262–279.

[WN99] L. A. Wolsey and G. L. Nemhauser, *Integer and combinatorial optimization*, Wiley-Interscience, 1999.

[ZBLN97] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, *L-bfgs-b: Algorithm 778: Lbfgs-b, fortran routines for large scale bound constrained optimization*, ACM. Trans. Math. Software **23** (1997), no. 4, 550–560.