Part 2: NLP Constrained Optimization

James G. Shanahan
Independent Consultant and Lecturer UC Santa Cruz
EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com
Final Exam

- Take home final Exam, due on March 24, 2013,

- You will receive it on March 17, 2013 (Happy Saint Patrick’s Day!)
Outline Lecture 10

- Multivariable unconstrained
  - Linear regression, Logistic Regression
- KKT conditions for Constrained Optimization
  - Lagrangian, KKT conditions
- Quadratic Programming
  - Modified Simplex Algorithm
- Convex Programming
  - Frank-Wolfe Algorithm, Penalty or barrier function (e.g., SUMT)
- Nonconvex Programming

- Course review
Newton’s Method in Optimization

- This iterative scheme can be generalized to several dimensions by replacing the derivative with the gradient, $f'(X)$, and the reciprocal of the second derivative with the inverse of the Hessian matrix, $\text{inv}$. One obtains the Newton-iterative scheme

\[
x^{i+1} = -\frac{f'(x^i)}{f''(x^i)} + x^i \quad \text{For one variable}
\]

\[
x^i - \left[ \frac{d^2 f}{dx^2} (x^i) \right]^{-1} f'(x^i) \quad \text{in matrix form}
\]

\[
X^{i+1} = X^i - \left[ f''(X^i) \right]^{-1} f'(X^i) \quad \text{For multivariable}
\]

Newton’s Method in Optimization

• As we have seen above, Newton's method is used to find the roots of equations in one or more dimensions.
• It can also be used to find local maxima and local minima of functions, as these extrema are the roots of the derivative function (i.e., \( f'(x) \)).
• We shall define a series of \( x \)-s, starting from an initial guess \( x_0 \), s.t. the series converges towards \( x^* \) which satisfies \( f'(x^*) = 0 \). This \( x^* \) will also be an extremum, i.e. stationary point, of \( f \).
• Thus, provided that \( f(x) \) is a twice-differentiable function and the initial guess is chosen close enough to \( x^* \), the sequence \((x^n)\) defined as follows will converge on \( x^* \):

\[
x^{i+1} = -\frac{f'(x^i)}{f''(x^i)} + x^i = x^i - \left[f''(x^i)\right]^{-1} f'(x^i)
\]

Iteration function
Gradient: Nabla is the symbol (\(\nabla\)).

- the gradient at a specific point \(x = x'\) is the vector whose elements are the respective partial derivatives evaluated at \(X = x'\), so that

\[
\nabla f(x') = \left(\frac{df}{dx_1}, \frac{df}{dx_2}, \ldots, \frac{df}{dx_n}\right)
\]

- The significance of the gradient is that the (infinitesimal) change in \(x\) that maximizes the rate at which \(f(x)\) increases is the change that is proportional to \(\nabla f(x)\).
Gradient

Because the objective function $f(x)$ is assumed to be differentiable, it possesses a gradient, denoted by $\nabla f(x)$, at each point $x$. In particular, the gradient at a specific point $x = x'$ is the vector whose elements are the respective partial derivatives evaluated at $x = x'$, so that

$$\nabla f(x') = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right) \quad \text{at } x = x'.$$

The significance of the gradient is that the (infinitesimal) change in $x$ that maximizes the rate at which $f(x)$ increases is the change that is proportional to $\nabla f(x)$. To express this idea geometrically, the “direction” of the gradient $\nabla f(x')$ is interpreted as the direction of the directed line segment (arrow) from the origin $(0, 0, \ldots, 0)$ to the point $(\partial f/\partial x_1, \partial f/\partial x_2, \ldots, \partial f/\partial x_n)$, where $\partial f/\partial x_j$ is evaluated at $x_j = x'_j$. Therefore, it may be said that the rate at which $f(x)$ increases is maximized if (infinitesimal) changes in $x$ are in the direction of the gradient $\nabla f(x)$. Because the objective is to find the feasible solution maximizing $f(x)$, it would seem expedient to attempt to move in the direction of the gradient as much as possible.
Significance of the Gradient Vector

• The gradient vector, \( \nabla f(x,y) \), gives the direction of fastest increase of \( f(x, y) \) (assuming a two-variable function here). [Newton-Raphson]
• The gradient vector, \( \nabla f(x,y) \), is orthogonal to the contour lines.
• Imagine climbing an upside-down bowl from below, where I can move in any \( <x, y> \) direction (NOTE I can't move in \( z \); \( x \), and \( y \) are independent variables).
• If I follow the level curve \( (f(x,y)=k) \) then I make no progress to the summit or bottom but if I move perpendicular to the level curve then I make the quickest progress to the summit (of the bowl).
\[ F(x) = F(x^*) + \nabla F(x)^T \bigg|_{x = x^*} (x - x^*) + \ldots \]  

**MultiVariate Taylor**

where

\[ \nabla F(x) \bigg|_{x = x^*} \] is the gradient of \( F(x) \) evaluated at \( x^* \)

**I.E.**

\[
\nabla F(x) = \begin{bmatrix}
\frac{\partial}{\partial x_1} F(x), & \frac{\partial}{\partial x_2} F(x), & \ldots & \frac{\partial}{\partial x_n} F(x)
\end{bmatrix}^T
\]

\[
\nabla F(x) = \begin{bmatrix}
F_{x_1}(x), & F_{x_1}'(x), & F_{x_n}(x),
\end{bmatrix}^T
\]

\[
\nabla F(x) = \begin{bmatrix}
F_{x_1}'(x), & F_{x_1}''(x), & F_{x_n}''(x),
\end{bmatrix}^T
\]

\[
x^{i+1} = x^i - \frac{g(x^i)}{g'(x^i)} \quad \text{Iteration function} \quad \text{where} \quad g = f'(x)
\]

\[
x^{i+1} = x^i - \frac{f'(x^i)}{f''(x^i)} \quad \text{Iteration function} \quad \text{for finding roots of} \quad f(x)
\]
MultiVariate Taylor

\[ F(x) = F(x^*) + \nabla F(x)^T \big|_{x=x^*} (x - x^*) + \frac{1}{2} (x - x^*)^T \nabla^2 F(x) \big|_{x=x^*} (x - x^*) \]

where \( \nabla F(x) \big|_{x=x^*} \) is the gradient of \( F(x) \) evaluated at \( x^* \) and \( \nabla^2 F(x) \big|_{x=x^*} \) is the Hessian of \( F(x) \) evaluated at \( x^* \).

Here

\[ \begin{align*}
x_{i+1} &= x_i - \frac{g(x_i)}{g'(x_i)} & \text{Iteration function} \\
x_{i+1} &= x_i - \frac{f'(x_i)}{f''(x_i)} & \text{Iteration function}
\end{align*} \]

for finding roots of \( f(x) \) using Newton's method.
Multivariate Newton’s Method

Suppose that the objective $f$ is a function of multiple arguments, $f(w_1, w_2, \ldots w_p)$. Let’s bundle the parameters into a single vector, $\bar{w}$. Then the Newton update is

$$\bar{w}_{n+1} = \bar{w}_n - H^{-1}(w_n) \nabla f(\bar{w}_n)$$  \hfill (16)

where $\nabla f$ is the gradient of $f$, its vector of partial derivatives $[\partial f/\partial w_1, \partial f/\partial w_2, \ldots \partial f/\partial w_p]$, and $H$ is the Hessian of $f$, its matrix of second partial derivatives, $H_{ij} = \partial^2 f/\partial w_i \partial w_j$.

Calculating $H$ and $\nabla f$ isn’t usually very time-consuming, but taking the inverse of $H$ is, unless it happens to be a diagonal matrix. This leads to various quasi-Newton methods, which either approximate $H$ by a diagonal matrix, or take a proper inverse of $H$ only rarely (maybe just once), and then try to update an estimate of $H^{-1}(w_n)$ as $w_n$ changes. (See section 8.3 in the textbook for more.)

In R, have a look at

```
?optim  #method=BFGS
```

[Hand, Manilla, Smith, Data Mining, Section 8.3]
Tangent Approximations
A plane from a point and orthogonal vector

- Although a line in space is determined by a point and a direction, a plane in space is more difficult to describe.

- A single vector parallel to a plane is not enough to convey the “direction” of the plane, but a vector perpendicular to the plane does completely specify its direction.

- Thus, a plane in space is determined by a point in the plane and a vector that is orthogonal to the plane. This orthogonal vector is called a normal vector.

\[ ax + by + cz + d = 0 \]

where \( d = -(ax_0 + by_0 + cz_0) \). Equation 7 is called a linear equation in \( x, y, \) and \( z \).
Tangent Planes and Linear Approximation

• Just as we can visualize the line tangent to a curve at a point in 2-space, in 3-space we can picture the plane tangent to a surface at a point.

• Consider the surface given by \( z = f(x,y) \). Let \((x_0, y_0, z_0)\) be any point on this surface.
  
  – If \( f(x,y) \) is differentiable at \((x_0, y_0)\), then the surface has a tangent plane at \((x_0, y_0, z_0)\). The equation of the tangent plane at \((x_0,y_0, z_0)\) is given by:

\[
(z-z_0) = f_x(x_0,y_0)*(x-x_0)+f_y(x_0,y_0) *(y-y_0)
\]

  similar form: \((y-y_0) = f_x(x_0) *(x-x_0)\)

  – where \( f_x(x_0,y_0) \) is the partial derivative of \( f() \) WRT x calculated at \( x_0,y_0 \); similarly for \( f_y(x_0,y_0) \)

http://www.math.hmc.edu/calculus/tutorials/tangentplanes/
Notation

\[ f(x) = f(\alpha) + f'(\alpha)(x - \alpha) \]

\[ F(x) = F(x^*) + \nabla F(x)^T \bigg|_{x = x^*} (x - x^*) + \ldots \]

where

\[ \nabla F(x) \bigg|_{x = x^*} \] is the gradient of \( F(x) \) evaluated at \( x^* \)

I.E.

\[ \nabla F(x) = \left[ \frac{\partial}{\partial x_1} F(x), \frac{\partial}{\partial x_2} F(x), \ldots, \frac{\partial}{\partial x_n} F(x) \right]^T \]

\[ \nabla F(x) = \begin{bmatrix} F_{x_1}(x), F_{x_1}(x), F_{xn}(x) \end{bmatrix}^T \]

\[ \nabla F(x) = \begin{bmatrix} F'_{x_1}(x), F'_{x_1}(x), F'_{xn}(x) \end{bmatrix}^T \]

\[ F(x) = F(x^*) + \nabla F(x^*)^T (x - x^*) \]

1D Linear Approximation

Multi Variable Linear Approx.
(z-z_0) = f_x(x_0,y_0)(x-x_0) + f_y(x_0,y_0)(y-y_0)

**Tangent Plane**

Function of variables

Calculate gradient vector by evaluating partial derivatives at the tangential point

Gradient vector at (1, 1) is (4, 2);

\[ f'(1,1) = (4,2) \]

\[ f(1,1) = 3 \]

Tangent plane at (1, 1, 3) with gradient (4,2)

**EXAMPLE 1** Find the tangent plane to the elliptic paraboloid \( z = 2x^2 + y^2 \) at the point (1, 1, 3).

**SOLUTION** Let \( f(x, y) = 2x^2 + y^2 \). Then

\[ f_x(x, y) = 4x \quad f_y(x, y) = 2y \]

\[ f_x(1, 1) = 4 \quad f_y(1, 1) = 2 \]

Then (2) gives the equation of the tangent plane at (1, 1, 3) as

\[ z - 3 = 4(x - 1) + 2(y - 1) \]

or

\[ z = 4x + 2y - 3 \]

Figure 2(a) shows the elliptic paraboloid and its tangent plane at (1, 1, 3) that we found in Example 1. In parts (b) and (c) we zoom in toward the point (1, 1, 3) by restricting the domain of the function \( f(x, y) = 2x^2 + y^2 \). Notice that the more we zoom in, the flatter the graph appears and the more it resembles its tangent plane.

[Adapted from Multivariable Calculus: Concepts and Contexts, James Stewart]
Tangent Plane Example

In Figure 3 we corroborate this impression by zooming in toward the point \((1, 1)\) on a contour map of the function \(f(x, y) = 2x^2 + y^2\). Notice that the more we zoom in, the more the level curves look like equally spaced parallel lines, which is characteristic of a plane.
EXAMPLE 8  Find the equations of the tangent plane and normal line at the point 
\((-2, 1, -3)\) to the ellipsoid

\[
\frac{x^2}{4} + y^2 + \frac{z^2}{9} = 3
\]

SOLUTION  The ellipsoid is the level surface (with \(k = 3\)) of the function

\[
F(x, y, z) = \frac{x^2}{4} + y^2 + \frac{z^2}{9}
\]

Therefore, we have

\[
F_x(x, y, z) = \frac{x}{2} \quad F_y(x, y, z) = 2y \quad F_z(x, y, z) = \frac{2z}{9}
\]

\[
F_x(-2, 1, -3) = -1 \quad F_y(-2, 1, -3) = 2 \quad F_z(-2, 1, -3) = -\frac{2}{3}
\]

Then Equation 19 gives the equation of the tangent plane at \((-2, 1, -3)\) as

\[-1(x + 2) + 2(y - 1) - \frac{2}{3}(z + 3) = 0\]

which simplifies to \(3x - 6y + 2z + 18 = 0\).

By Equation 20, symmetric equations of the normal line are

\[
\frac{x + 2}{-1} = \frac{y - 1}{2} = \frac{z + 3}{-\frac{2}{3}}
\]
Approximate $\Delta y$ with $dy$ via the tangent

For a function of one variable, $y = f(x)$, we define the differential $dx$ to be an independent variable; that is, $dx$ can be given the value of any real number. The differential of $y$ is then defined as

$$dy = f'(x) \, dx$$

(See Section 3.8.) Figure 6 shows the relationship between the increment $\Delta y$ and the differential $dy$: $\Delta y$ represents the change in height of the curve $y = f(x)$ and $dy$ represents the change in height of the tangent line when $x$ changes by an amount $dx = \Delta x$.

$\Delta y$ is the predicted difference in $(f(x)$ given the linear approximation) Can change $\Delta x$ as much as we like but the bigger the $\Delta x$ the bigger the gap between the tangent approximation and the actual function (and $dy$ and $\Delta y$)
Linear and Quadratic Approximations

- Approximate $f(X)$ for $X$ around point $a$ by the tangent at a point $(a, f(a))$
  
  \[ y - y_1 = m(x - x_1) \]
  \[ f(x) - y_1 = m(x - x_1) \]
  \[ f(x) = y_1 + m(x - x_1) \]

  \[ f(x) = f(a) + f'(a)(x - a) \]

  \[ f(x) = f(a) + f'(a)(x - a) \quad \text{AT} \quad (a, f(a)) \quad \text{slope} = f'(a) \]

- Taylor Series explores different approximations of $f(X)$;
  - the above tangential form is linear approximation

- General Form of a Taylor Series

  \[ f(x) = f(a) + f'(a)(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x - a)^n \]

  More compactly
  \[ f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x - a)^k \]
Total Differential, \( dz \), for \( z = f(x, y) \) in 2D

For a differentiable function of two variables, \( z = f(x, y) \), we define the differentials \( dx \) and \( dy \) to be independent variables; that is, they can be given any values. Then the differential \( dz \), also called the total differential, is defined by

\[
dz = f_x(x, y) \, dx + f_y(x, y) \, dy = \frac{\partial z}{\partial x} \, dx + \frac{\partial z}{\partial y} \, dy
\]

\[ f(\bar{x}) \approx f(\bar{a}) + dz \]
\[ f(\bar{x}) \approx f(\bar{a}) + \nabla f(\bar{a}) \cdot (\bar{x} - \bar{a}) \]

(Compare with Equation 9.) Sometimes the notation \( df \) is used in place of \( dz \).

If we take \( dx = \Delta x = x - a \) and \( dy = \Delta y = y - b \) in Equation 10, then the differential of \( z \) is

\[
dz = f_x(a, b)(x - a) + f_y(a, b)(y - b)
\]

- Estimated change in \( z \) using total differential
- Total Differential in 2D (estimated change in \( z = f(x) \) using a linear approximation)
- This corresponds to the second term (the linear term) in Taylor's expansion

\[
f(x) = f(a) + dz
\]
\[
f(x) = f(a) + f'(a)(x - a)
\]
Total Differential in 2D (estimated change in z)

So, in the notation of differentials, the linear approximation (4) can be written as

\[ f(x, y) \approx f(a, b) + dz \]

Figure 7 is the three-dimensional counterpart of Figure 6 and shows the geometric interpretation of the differential \( dz \) and the increment \( \Delta z \): \( dz \) represents the change in height of the tangent plane, whereas \( \Delta z \) represents the change in height of the surface \( z = f(x, y) \) when \( (x, y) \) changes from \( (a, b) \) to \( (a + \Delta x, b + \Delta y) \).

\[
(z-z_0) = f_x(x_0,y_0)(x-x_0) + f_y(x_0,y_0)(y-y_0)
\]

Where \( L \) is the linear approximation of \( f() \), around the point \( (a,b) \).
Total Differential in 2D (estimated change in z)

So, in the notation of differentials, the linear approximation (4) can be written as

\[ f(x, y) \approx f(a, b) + dz \]

Figure 7 is the three-dimensional counterpart of Figure 6 and shows the geometric interpretation of the differential \( dz \) and the increment \( \Delta z \): \( dz \) represents the change in height of the tangent plane, whereas \( \Delta z \) represents the change in height of the surface \( z = f(x, y) \) when \( (x, y) \) changes from \( (a, b) \) to \( (a + \Delta x, b + \Delta y) \).

The total derivative estimates how much \( z \) changes (estimated based on the tangential plane approximation).

Any \( f(x, y) \) can be approximated \( f(a,b) +\text{total differential for any } (x,y) \) close to \( (a, b) \).
Total Differential in 2D: An Example

(a) If \( z = f(x, y) = x^2 + 3xy - y^2 \), find the differential \( dz \).
(b) If \( x \) changes from 2 to 2.05 and \( y \) changes from 3 to 2.96, compare the values of \( \Delta z \) and \( dz \).

\[ dz = \frac{\partial z}{\partial x} dx + \frac{\partial z}{\partial y} dy = (2x + 3y) dx + (3x - 2y) dy \]

\( z = f_{\text{Tang}(a, b)}(x, y) - f(a, b) \)

The increment of \( z \) is
\[ \Delta z = f(2.05, 2.96) - f(2, 3) = [(2.05)^2 + 3(2.05)(2.96) - (2.96)^2] - [2^2 + 3(2)(3) - 3^2] = 0.6449 \]

Estimated \( z \) difference between \( f(x, y) \) and \( f(a, b) \) 
Actual \( z \) difference (i.e., \( z = f(x, y) - f(a, b) \))

Notice that \( \Delta z \approx dz \) but \( dz \) is easier to compute.
Total Differential and Directional Derivative

The total derivative tells us how much $z$ changes (only estimated as it is based on the tangential plane approximation) when we travel in a particular direction $u$, i.e., $D_u f(x,y) = \nabla f(x,y) \cdot u$

Where $u = (x-a, y-b)$

Any $f(x,y)$ can be approximated by $f(a,b) + \text{total differential}$
Directional Derivative == Total Deriv. == Change in $f(x)$

If $f$ is a real-valued function on $\mathbb{R}^n$, then the partial derivatives of $f$ measure its variation in the direction of the coordinate axes. For example, if $f$ is a function of $x$ and $y$, then its partial derivatives measure the variation in $f$ in the $x$ direction and the $y$ direction. They do not, however, directly measure the variation of $f$ in any other direction, such as along the diagonal line $y = x$. These are measured using directional derivatives. Choose a vector

$$v = (v_1, \ldots, v_n).$$

The \textit{directional derivative} of $f$ in the direction of $v$ at the point $x$ is the limit

$$D_v f(x) = \lim_{h \to 0} \frac{f(x + h v) - f(x)}{h}.$$  

Let $\lambda$ be a scalar. The substitution of $h/\lambda$ for $h$ changes the $\lambda v$ direction’s difference quotient into $\lambda$ times the $v$ direction’s difference quotient. Consequently, the directional derivative in the $\lambda v$ direction is $\lambda$ times the directional derivative in the $v$ direction. Because of this, directional derivatives are often considered only for unit vectors $v$.

If all the partial derivatives of $f$ exist and are continuous at $x$, then they determine the directional derivative of $f$ in the direction $v$ by the formula:

$$D_v f(x) = \sum_{j=1}^{n} v_j \frac{\partial f}{\partial x_j}.$$  

This is a consequence of the definition of the \textit{total derivative}. It follows that the directional derivative is \textit{linear} in $v$.

The same definition also works when $f$ is a function with values in $\mathbb{R}^m$. We just use the above definition in each component of the vectors. In this case, the directional derivative is a vector in $\mathbb{R}^m$.

\[
D_A f(x, y) = \nabla f(x, y) \cdot (x - A_1, y - A_2)
\]
## Necessary and Sufficient Conditions for Optimality

<table>
<thead>
<tr>
<th>Problem</th>
<th>Necessary Conditions for Optimality</th>
<th>Also Sufficient If:</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-variable unconstrained</td>
<td>$\frac{df}{dx} = 0$</td>
<td>$f(x)$ concave</td>
</tr>
<tr>
<td>Multivariable unconstrained</td>
<td>$\frac{\partial f}{\partial x_j} = 0$ ($j = 1, 2, \ldots, n$)</td>
<td>$f(x)$ concave</td>
</tr>
<tr>
<td>Constrained, nonnegativity constraints only</td>
<td>$\frac{\partial f}{\partial x_j} = 0$ ($j = 1, 2, \ldots, n$) (or $\leq 0$ if $x_j = 0$)</td>
<td>$f(x)$ concave</td>
</tr>
<tr>
<td>General constrained problem</td>
<td>Karush-Kuhn-Tucker conditions</td>
<td>$f(x)$ concave and $g_i(x)$ convex ($i = 1, 2, \ldots, m$)</td>
</tr>
</tbody>
</table>

---


KKT Conditions

Theorem. Assume that $f(x)$, $g_1(x)$, $g_2(x)$, \ldots, $g_m(x)$ are differentiable functions satisfying certain regularity conditions. Then

$$x^* = (x_1^*, x_2^*, \ldots, x_n^*)$$

can be an optimal solution for the nonlinear programming problem only if there exist $m$ numbers $u_1$, $u_2$, \ldots, $u_m$ such that all the following KKT conditions are satisfied:

1. $$\frac{\partial f}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x_j} \leq 0$$

$$\text{at } x = x^*, \text{ for } j = 1, 2, \ldots, n.$$  

2. $$x_j^* \left( \frac{\partial f}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x_j} \right) = 0$$

3. $$g_i(x^*) - b_i \leq 0$$

4. $$u_i [g_i(x^*) - b_i] = 0$$

5. $$x_j^* \geq 0,$$  

6. $$u_i \geq 0,$$  

for $i = 1, 2, \ldots, m$.  

for $j = 1, 2, \ldots, n$.  

for $i = 1, 2, \ldots, m$.  

Problem needs to be concave

• ..

However, note that satisfying these conditions does not guarantee that the solution is optimal. As summarized in the rightmost column of Table 12.4, certain additional convexity assumptions are needed to obtain this guarantee. These assumptions are spelled out in the following extension of the theorem.

Corollary. Assume that \( f(x) \) is a concave function and that \( g_1(x), g_2(x), \ldots, g_m(x) \) are convex functions (i.e., this problem is a convex programming problem), where all these functions satisfy the regularity conditions. Then \( x^* = (x_1^*, x_2^*, \ldots, x_n^*) \) is an optimal solution if and only if all the conditions of the theorem are satisfied.
Example. To illustrate the formulation and application of the KKT conditions, we consider the following two-variable nonlinear programming problem:

Maximize \( f(x) = \ln(x_1 + 1) + x_2 \),

subject to

\[ 2x_1 + x_2 \leq 3 \]

and

\[ x_1 \geq 0, \quad x_2 \geq 0, \]

where \( \ln \) denotes the natural logarithm. Thus, \( m = 1 \) (one functional constraint) and \( g_1(x) = 2x_1 + x_2 \), so \( g_1(x) \) is convex. Furthermore, it can be easily verified (see Appendix 2) that \( f(x) \) is concave. Hence, the corollary applies, so any solution that satisfies the KKT conditions will definitely be an optimal solution. Applying the formulas given in the theorem yields the following KKT conditions for this example:

1. (\( j = 1 \)). \[ \frac{1}{x_1 + 1} - 2u_1 \leq 0. \]
2. (\( j = 1 \)). \[ x_1 \left( \frac{1}{x_1 + 1} - 2u_1 \right) = 0. \]
1. (\( j = 2 \)). \[ 1 - u_1 \leq 0. \]
2. (\( j = 2 \)). \[ x_2(1 - u_1) = 0. \]
3. \[ 2x_1 + x_2 - 3 \leq 0. \]
4. \[ u_1(2x_1 + x_2 - 3) = 0. \]
5. \[ x_1 \geq 0, x_2 \geq 0. \]
6. \[ u_1 \geq 0. \]

\[
\begin{align*}
1. \quad & \quad \frac{\partial f}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x_j} \leq 0 \\
2. \quad & \quad x_j^* \left( \frac{\partial f}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x_j} \right) = 0 \quad \text{at } x = x^*, \text{ for } j = 1, 2, \ldots, n. \\
3. \quad & \quad g_i(x^*) - b_i \leq 0 \quad \text{for } i = 1, 2, \ldots, m. \\
4. \quad & \quad u_i[g_i(x^*) - b_i] = 0 \quad \text{for } i = 1, 2, \ldots, m. \\
5. \quad & \quad x_j^* \geq 0, \quad \text{for } j = 1, 2, \ldots, n. \\
6. \quad & \quad u_i \geq 0, \quad \text{for } i = 1, 2, \ldots, m.
\end{align*}
\]
1\((j = 1)\). \[ \frac{1}{x_1 + 1} - 2u_1 \leq 0. \]

2\((j = 1)\). \[ x_1 \left( \frac{1}{x_1 + 1} - 2u_1 \right) = 0. \]

1\((j = 2)\). \[ 1 - u_1 \leq 0. \]

2\((j = 2)\). \[ x_2(1 - u_1) = 0. \]

3. \[ 2x_1 + x_2 - 3 \leq 0. \]

4. \[ u_1(2x_1 + x_2 - 3) = 0. \]

5. \[ x_1 \geq 0, x_2 \geq 0. \]

6. \[ u_1 \geq 0. \]

Example

The steps in solving the KKT conditions for this particular example are outlined below.

1. \(u_1 \geq 1\), from condition 1\((j = 2)\).
   \[ x_1 \geq 0, \] from condition 5.

2. Therefore, \[ \frac{1}{x_1 + 1} - 2u_1 < 0. \]

3. Therefore, \(x_1 = 0\), from condition 2\((j = 1)\).

4. \(u_1 \neq 0\) implies that \(2x_1 + x_2 - 3 = 0\), from condition 4.

5. Steps 3 and 4 imply that \(x_2 = 3\).

6. \(x_2 \neq 0\) implies that \(u_1 = 1\), from condition 2\((j = 2)\).

7. No conditions are violated by \(x_1 = 0, x_2 = 3, u_1 = 1\).

Therefore, there exists a number \(u_1 = 1\) such that \(x_1 = 0, x_2 = 3,\) and \(u_1 = 1\) satisfy all the conditions. Consequently, \(x^* = (0, 3)\) is an optimal solution for this problem.
Shadow prices/dual variables $u_i$

Furthermore, the values of the $u_i$ variables in an optimal solution for the dual problem can again be interpreted as *shadow prices* (see Secs. 4.7 and 6.2); i.e., they give the rate at which the optimal objective function value for the primal problem could be increased by (slightly) increasing the right-hand side of the corresponding constraint. Because duality theory for nonlinear programming is a relatively advanced topic, the interested reader is referred elsewhere for further information.\textsuperscript{14}

You will see another indirect application of the KKT conditions in the next section.
Quadratic Programming

As indicated in Sec. 12.3, the quadratic programming problem differs from the linear programming problem only in that the objective function also includes $x_j^2$ and $x_i x_j$ ($i \neq j$) terms. Thus, if we use matrix notation like that introduced at the beginning of Sec. 5.2, the problem is to find $x$ so as to

$$\text{Maximize } f(x) = cx - \frac{1}{2}x^TQx,$$

subject to

$$Ax \leq b \quad \text{and} \quad x \geq 0,$$

where $c$ is a row vector, $x$ and $b$ are column vectors, $Q$ and $A$ are matrices, and the superscript $T$ denotes the transpose (see Appendix 4). The $q_{ij}$ (elements of $Q$) are given constants such that $q_{ij} = q_{ji}$ (which is the reason for the factor of $\frac{1}{2}$ in the objective function). By performing the indicated vector and matrix multiplications, the objective function then is expressed in terms of these $q_{ij}$, the $c_j$ (elements of $c$), and the variables as follows:

$$f(x) = cx - \frac{1}{2}x^TQx = \sum_{j=1}^{n} c_j x_j - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j.$$

For each term where $i = j$ in this double summation, $x_i x_j = x_j^2$, so $-\frac{1}{2}q_{jj}$ is the coefficient of $x_j^2$. When $i \neq j$, then $-\frac{1}{2}(q_{ij} x_i x_j + q_{ji} x_j x_i) = -q_{ij} x_i x_j$, so $-q_{ij}$ is the total coefficient for the product of $x_i$ and $x_j$. 
Example

To illustrate this notation, consider the following example of a quadratic programming problem.

Maximize \( f(x_1, x_2) = 15x_1 + 30x_2 + 4x_1x_2 - 2x_1^2 - 4x_2^2, \)

subject to

\[ x_1 + 2x_2 \leq 30 \]

and

\[ x_1 \geq 0, \quad x_2 \geq 0. \]

In this case,

\[ c = [15 \ 30], \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad Q = \begin{bmatrix} 4 & -4 \\ -4 & 8 \end{bmatrix}, \]

\[ A = [1 \ 2], \quad b = [30]. \]

Note that

\[ x^TQx = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} 4 & -4 \\ -4 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \]

\[ = [(4x_1 - 4x_2) \ (-4x_1 + 8x_2)] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \]

\[ = 4x_1^2 - 4x_2x_1 - 4x_1x_2 + 8x_2^2 \]

\[ = q_{11}x_1^2 + q_{21}x_2x_1 + q_{12}x_1x_2 + q_{22}x_2^2. \]

Multiplying through by \(-\frac{1}{2}\) gives

\[ -\frac{1}{2}x^TQx = -2x_1^2 + 4x_1x_2 - 4x_2^2, \]
Concave Quadratic Programming

Several algorithms have been developed for the special case of the quadratic programming problem where the objective function is a *concave* function. (A way to verify that the objective function is concave is to verify the equivalent condition that

\[ x^TQx \geq 0 \]

for all \( x \), that is, \( Q \) is a *positive semidefinite* matrix.) We shall describe one\(^\text{15}\) of these algorithms, the *modified simplex method*, that has been quite popular because it requires using only the simplex method with a slight modification. The key to this approach is to construct the KKT conditions from the preceding section and then to reexpress these conditions in a convenient form that closely resembles linear programming. Therefore, before describing the algorithm, we shall develop this convenient form.
The KKT Conditions for Quadratic Programming

For concreteness, let us first consider the above example. Starting with the form given in the preceding section, its KKT conditions are the following:

1. \( (j = 1). \quad 15 + 4x_2 - 4x_1 - u_1 \leq 0. \)
2. \( (j = 1). \quad x_1(15 + 4x_2 - 4x_1 - u_1) = 0. \)
3. \( (j = 2). \quad 30 + 4x_1 - 8x_2 - 2u_1 \leq 0. \)
4. \( (j = 2). \quad x_2(30 + 4x_1 - 8x_2 - 2u_1) = 0. \)
5. \( x_1 \geq 0, \quad x_2 \geq 0. \)
6. \( u_1 \geq 0. \)

To illustrate this notation, consider the following example of a quadratic programming problem.

Maximize \( f(x_1, x_2) = 15x_1 + 30x_2 + 4x_1x_2 - 2x_1^2 - 4x_2^2, \)

subject to

\( x_1 + 2x_2 \leq 30 \)

and

\( x_1 \geq 0, \quad x_2 \geq 0. \)

1. \( \frac{\partial f}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x_j} \leq 0 \)

at \( x = x^*, \) for \( j = 1, 2, \ldots, n. \)

2. \( x_j^* \left( \frac{\partial f}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x_j} \right) = 0 \)

3. \( g_i(x^*) - b_i \leq 0 \)

for \( i = 1, 2, \ldots, m. \)

4. \( u_i [g_i(x^*) - b_i] = 0 \)

5. \( x_j^* \geq 0, \) for \( j = 1, 2, \ldots, n. \)

6. \( u_i \geq 0, \) for \( i = 1, 2, \ldots, m. \)
For concreteness, let us first consider the above example. Starting with the form given in the preceding section, its KKT conditions are the following.

1\( (j = 1) \): \( 15 + 4x_2 - 4x_1 - u_1 \leq 0 \).
2\( (j = 1) \): \( x_1(15 + 4x_2 - 4x_1 - u_1) = 0 \).
1\( (j = 2) \): \( 30 + 4x_1 - 8x_2 - 2u_1 \leq 0 \).
2\( (j = 2) \): \( x_2(30 + 4x_1 - 8x_2 - 2u_1) = 0 \).
3. \( x_1 + 2x_2 - 30 \leq 0 \).
4. \( u_1(x_1 + 2x_2 - 30) = 0 \).
5. \( x_1 \geq 0, \quad x_2 \geq 0 \).
6. \( u_1 \geq 0 \).

To begin reexpressing these conditions in a more convenient form, we move the constants in conditions 1\( (j = 1) \), 1\( (j = 2) \), and 3 to the right-hand side and then introduce nonnegative slack variables (denoted by \( y_1, y_2, \) and \( v_1 \), respectively) to convert these inequalities to equations.

1\( (j = 1) \): \( -4x_1 + 4x_2 - u_1 + y_1 = -15 \)
1\( (j = 2) \): \( 4x_1 - 8x_2 - 2u_1 + y_2 = -30 \)
3. \( x_1 + 2x_2 + v_1 = 30 \)

Note that condition 2\( (j = 1) \) can now be reexpressed as simply requiring that either \( x_1 = 0 \) or \( y_1 = 0 \); that is,

2\( (j = 1) \): \( x_1y_1 = 0 \).

In just the same way, conditions 2\( (j = 2) \) and 4 can be replaced by

2\( (j = 2) \): \( x_2y_2 = 0 \),
4. \( u_1v_1 = 0 \).
Complementarity Constraint

For each of these three pairs—\((x_1, y_1), (x_2, y_2), (u_1, v_1)\)—the two variables are called \textit{complementary variables}, because only one of the two variables can be nonzero. These new forms of conditions 2\((j = 1)\), 2\((j = 2)\), and 4 can be combined into one constraint,

\[ x_1y_1 + x_2y_2 + u_1v_1 = 0, \]

called the \textit{complementarity constraint}. 
After multiplying through the equations for conditions $1(j = 1)$ and $1(j = 2)$ by $-1$ to obtain nonnegative right-hand sides, we now have the desired convenient form for the entire set of conditions shown here:

\[
\begin{align*}
4x_1 - 4x_2 + u_1 - y_1 &= 15 \\
-4x_1 + 8x_2 + 2u_1 - y_2 &= 30 \\
x_1 + 2x_2 + v_1 &= 30 \\
x_1 \geq 0, \quad x_2 \geq 0, \quad u_1 \geq 0, \quad y_1 \geq 0, \quad y_2 \geq 0, \quad v_1 \geq 0 \\
x_1y_1 + x_2y_2 + u_1v_1 &= 0
\end{align*}
\]

This form is particularly convenient because, except for the complementarity constraint, these conditions are linear programming constraints.

For any quadratic programming problem, its KKT conditions can be reduced to this same convenient form containing just linear programming constraints plus one complementarity constraint. In matrix notation again, this general form is

\[
Qx + A^Tu - y = c^T, \\
Ax + v = b, \\
x \geq 0, \quad u \geq 0, \quad y \geq 0, \quad v \geq 0, \\
x^Ty + u^Tv = 0,
\]

where the elements of the column vector $u$ are the $u_i$ of the preceding section and the elements of the column vectors $y$ and $v$ are slack variables.
From Quadratic to LP via KKT conditions

Maximize \( f(x) = cx - \frac{1}{2}x^TQx \),
subject to
\( Ax \leq b \) and \( x \geq 0 \),

\[
Qx + A^Tu - y = c^T, \\
Ax + v = b, \\
x \geq 0, \quad u \geq 0, \quad y \geq 0, \quad v \geq 0, \\
x^Ty + u^Tv = 0,
\]

where the elements of the column vector \( u \) are the \( u_i \) of the preceding section and the elements of the column vectors \( y \) and \( v \) are slack variables.
Find a feasible solution to these constraints

Because the objective function of the original problem is assumed to be concave and because the constraint functions are linear and therefore convex, the corollary to the theorem of Sec. 12.6 applies. Thus, \( x \) is optimal if and only if there exist values of \( y, u, \) and \( v \) such that all four vectors together satisfy all these conditions. The original problem is thereby reduced to the equivalent problem of finding a feasible solution to these constraints.

It is of interest to note that this equivalent problem is one example of the linear complementarity problem introduced in Sec. 12.3 (see Prob. 12.3-6), and that a key constraint for the linear complementarity problem is its complementarity constraint.

\[
Qx + A^T u - y = c^T,
\]
\[
Ax + v = b,
\]
\[
x \geq 0, \quad u \geq 0, \quad y \geq 0, \quad v \geq 0,
\]
\[
x^T y + u^T v = 0,
\]
Modified Simplex Algorithm

- Find a feasible solution to this:

\[
\begin{align*}
Qx + A^T u - y &= c^T, \\
Ax + v &= b, \\
x &\geq 0, \quad u \geq 0, \quad y \geq 0, \quad v \geq 0, \\
x^T y + u^T v &= 0,
\end{align*}
\]
The modified simplex method exploits the key fact that, with the exception of the complementarity constraint, the KKT conditions in the convenient form obtained above are nothing more than linear programming constraints. Furthermore, the complementarity constraint simply implies that it is not permissible for both complementary variables of any pair to be (nondegenerate) basic variables (the only variables $> 0$) when (nondegenerate) BF solutions are considered. Therefore, the problem reduces to finding an initial BF solution to any linear programming problem that has these constraints, subject to this additional restriction on the identity of the basic variables. (This initial BF solution may be the only feasible solution in this case.)

As we discussed in Sec. 4.6, finding such an initial BF solution is relatively straightforward. In the simple case where $c^T \leq 0$ (unlikely) and $b \geq 0$, the initial basic variables are the elements of $y$ and $v$ (multiply through the first set of equations by $-1$), so that the desired solution is $x = 0$, $u = 0$, $y = -c^T$, $v = b$. Otherwise, you need to revise the problem by introducing an artificial variable into each of the equations where $c_j > 0$ (add the variable on the left) or $b_i < 0$ (subtract the variable on the left and then multiply through by $-1$) in order to use these artificial variables (call them $z_1$, $z_2$, and so on) as initial basic variables for the revised problem. (Note that this choice of initial basic variables satisfies the complementarity constraint, because as nonbasic variables $x = 0$ and $u = 0$ automatically.)

Next, use phase 1 of the two-phase method (see Sec. 4.6) to find a BF solution for the real problem; i.e., apply the simplex method (with one modification) to the following linear programming problem

Minimize $Z = \sum_j z_j,$

subject to the linear programming constraints obtained from the KKT conditions, but with these artificial variables included.
Restricted Entry Rule

The one modification in the simplex method is the following change in the procedure for selecting an entering basic variable.

**Restricted-Entry Rule:** When you are choosing an entering basic variable, exclude from consideration any nonbasic variable whose *complementary variable* already is a basic variable; the choice should be made from the other nonbasic variables according to the usual criterion for the simplex method.

This rule keeps the complementarity constraint satisfied throughout the course of the algorithm. When an optimal solution

\[ x^*, u^*, y^*, v^*, z_1 = 0, \ldots, z_n = 0 \]

is obtained for the phase 1 problem, \( x^* \) is the desired optimal solution for the original quadratic programming problem. Phase 2 of the two-phase method is not needed.
Example

The starting point for solving this example is its KKT conditions in the convenient form obtained earlier in the section. After the needed artificial variables are introduced, the linear programming problem to be addressed explicitly by the modified simplex method then is

Minimize \( Z = z_1 + z_2, \)

subject to

\[
\begin{align*}
4x_1 - 4x_2 + u_1 - y_1 &+ z_1 = 15 \\
-4x_1 + 8x_2 + 2u_1 - y_2 &+ z_2 = 30 \\
x_1 + 2x_2 &+ v_1 = 30
\end{align*}
\]

and

\[
\begin{align*}
x_1 &\geq 0, & x_2 &\geq 0, & u_1 &\geq 0, & y_1 &\geq 0, & y_2 &\geq 0, & v_1 &\geq 0, \\
z_1 &\geq 0, & z_2 &\geq 0.
\end{align*}
\]

The additional complementarity constraint

\( x_1y_1 + x_2y_2 + u_1v_1 = 0, \)
### TABLE 12.5 Application of the modified simplex method to the quadratic programming example

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Basic Variable</th>
<th>Eq.</th>
<th>Z</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$u_1$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$v_1$</th>
<th>$z_1$</th>
<th>$z_2$</th>
<th>Right Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$Z$</td>
<td>(0)</td>
<td>−1</td>
<td>0</td>
<td>−4</td>
<td>−3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>−45</td>
</tr>
<tr>
<td></td>
<td>$z_1$</td>
<td>(1)</td>
<td></td>
<td>0</td>
<td>4</td>
<td>−4</td>
<td>1</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>$z_2$</td>
<td>(2)</td>
<td></td>
<td>0</td>
<td>−4</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$v_1$</td>
<td>(3)</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>$Z$</td>
<td>(0)</td>
<td>−1</td>
<td>−2</td>
<td>0</td>
<td>−2</td>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
<td>−30</td>
</tr>
<tr>
<td></td>
<td>$z_1$</td>
<td>(1)</td>
<td></td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>−1</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
<td>1</td>
<td>$\frac{1}{2}$</td>
</tr>
<tr>
<td></td>
<td>$x_2$</td>
<td>(2)</td>
<td></td>
<td>$\frac{1}{2}$</td>
<td>1</td>
<td>$\frac{1}{4}$</td>
<td>0</td>
<td>$\frac{1}{8}$</td>
<td>0</td>
<td>0</td>
<td>$\frac{1}{8}$</td>
<td>$\frac{3}{4}$</td>
</tr>
<tr>
<td></td>
<td>$v_1$</td>
<td>(3)</td>
<td></td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
<td>$\frac{1}{4}$</td>
<td>1</td>
<td>0</td>
<td>$\frac{1}{4}$</td>
</tr>
<tr>
<td>2</td>
<td>$Z$</td>
<td>(0)</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>$\frac{5}{2}$</td>
<td>1</td>
<td>$\frac{3}{4}$</td>
<td>1</td>
<td>0</td>
<td>$\frac{1}{4}$</td>
<td>−7$\frac{1}{2}$</td>
</tr>
<tr>
<td></td>
<td>$z_1$</td>
<td>(1)</td>
<td></td>
<td>0</td>
<td>0</td>
<td>$\frac{5}{2}$</td>
<td>−1</td>
<td>$\frac{3}{4}$</td>
<td>−1</td>
<td>1</td>
<td>$\frac{3}{4}$</td>
<td>7$\frac{1}{2}$</td>
</tr>
<tr>
<td></td>
<td>$x_2$</td>
<td>(2)</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$\frac{1}{8}$</td>
<td>0</td>
<td>$\frac{1}{16}$</td>
<td>$\frac{1}{4}$</td>
<td>0</td>
<td>$\frac{1}{16}$</td>
</tr>
<tr>
<td></td>
<td>$x_1$</td>
<td>(3)</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$\frac{1}{4}$</td>
<td>0</td>
<td>$\frac{1}{8}$</td>
<td>1</td>
<td>0</td>
<td>$\frac{1}{8}$</td>
</tr>
<tr>
<td>3</td>
<td>$Z$</td>
<td>(0)</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$u_1$</td>
<td>(1)</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$\frac{2}{5}$</td>
<td>−$\frac{3}{10}$</td>
<td>$\frac{2}{5}$</td>
<td>$\frac{2}{5}$</td>
<td>$\frac{3}{10}$</td>
</tr>
<tr>
<td></td>
<td>$x_2$</td>
<td>(2)</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$\frac{1}{20}$</td>
<td>$\frac{1}{40}$</td>
<td>$\frac{1}{10}$</td>
<td>$\frac{1}{20}$</td>
<td>$\frac{1}{40}$</td>
</tr>
<tr>
<td></td>
<td>$x_1$</td>
<td>(3)</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$\frac{1}{10}$</td>
<td>$\frac{1}{20}$</td>
<td>$\frac{2}{5}$</td>
<td>$\frac{1}{20}$</td>
<td>$\frac{1}{10}$</td>
</tr>
</tbody>
</table>
Convex Programming

We already have discussed some special cases of convex programming in Secs. 12.4 and 12.5 (unconstrained problems), 12.7 (quadratic objective function with linear constraints), and 12.8 (separable functions). You also have seen some theory for the general case (necessary and sufficient conditions for optimality) in Sec. 12.6. In this section, we briefly discuss some types of approaches used to solve the general convex programming problem [where the objective function $f(x)$ to be maximized is concave and the $g_i(x)$ constraint functions are convex], and then we present one example of an algorithm for convex programming.

There is no single standard algorithm that always is used to solve convex programming problems. Many different algorithms have been developed, each with its own advantages and disadvantages, and research continues to be active in this area. Roughly speaking, most of these algorithms fall into one of the following three categories.

3 families of approaches

•  Sequential Approximation Algorithms
•  Sequential Unconstrained Minimization Techniques (SUMT)
•  Generalized reduced gradient
Sequential Approximation Algorithms

• Sequential-approximation algorithms—including linear approximation and quadratic approximation methods

• These algorithms replace the nonlinear objective function by a succession of linear or quadratic approximations.
  
  – For linearly constrained optimization problems, these approximations allow repeated application of linear or quadratic programming algorithms. This work is accompanied by other analysis that yields a sequence of solutions that converges to an optimal solution for the original problem.

• Can be extended to problems with nonlinear constraint functions by the use of appropriate linear approximations.
Frank–Wolfe Algorithm

• As one example of a sequential-approximation algorithm, consider the Frank-Wolfe algorithm for the case of linearly constrained convex programming
  – With constraints: $Ax \leq b$ and $x \geq 0$ in matrix form).

• This procedure is particularly straightforward;
  – it combines linear approximations of the objective function (enabling us to use the simplex method) with the one-dimensional search procedure of
Frank–Wolfe Algorithm

• The Frank–Wolfe algorithm is a simple iterative first-order optimization algorithm for constrained convex optimization (quadratic programming problems).
  – Also known as the conditional gradient method, reduced gradient algorithm and the convex combination algorithm, the method was originally proposed by Marguerite Frank and Philip Wolfe in 1956.

• In each iteration, the Frank–Wolfe algorithm considers a linear approximation of the objective function, and moves slightly towards a minimizer of this linear function (taken over the same domain).

http://en.wikipedia.org/wiki/Frank%E2%80%93Wolfe_algorithm
• **The basic Idea:**
  1. Use linear functions to approximate both the objective function as well as the constraints (Linearization).
  2. Employ LP algorithms to solve this new linear program.
Introduction

- **Linearization can be achieved in two ways:**
  - Any non-linear function $f(x)$ can be approximated in the vicinity of a point $x^0$ by using Taylor's expansion,

$$f(x) = f(x^0) + \nabla f(x^0)(x - x^0) + O(||x - x^0||^2)$$

$x^0$ is called the linearization point.
  - Using piecewise linear approximations and then applying a modified simplex algorithm (separable programming).
Direct Use of Successive Linear Programs

• Using Taylor’s expansion linearize all problem functions at some selected estimate \( x^i \) of the solution. Result is an LP. \( x^i \) is called the linearization point.

• With some additional precautions the LP solution ought to be an improvement over the linearization point.

• There are two cases to be considered:
  1. Linearly constrained NLP case:

     \[
     \text{Minimize} \quad f(x) \\
     \text{Subject to} \quad A x \leq b \\
     \quad \quad \quad \quad x \geq 0
     \]

  2. General NLP case:

     \[
     \text{Minimize} \quad f(x) \\
     \text{Subject to} \quad g_j(x) \geq 0 \quad j = 1, \ldots, J \\
     \quad \quad \quad \quad h_k(x) = 0 \quad k = 1, \ldots, K \\
     \quad \quad \quad \quad x_{i}^{(U)} \geq x_i \geq x_{i}^{(L)} \quad i = 1, \ldots, N
     \]
The linearly constrained NLP problem that of:

\[
\begin{align*}
\text{Minimize} & \quad f(x) \\
\text{Subject to} & \quad A x \leq b \\
& \quad x \geq 0
\end{align*}
\]

- \( f(x) \) is a nonlinear objective function. Feasible region is a polyhedron, however optimal solution can lie anywhere within the feasible region.
Linearly Constrained NLP case

- Using Taylor’s approximation around the linearization point \( x^0 \) and ignoring the second and higher order terms we obtain the linear approximation of \( f(x) \) around the point \( x^0 \).

\[
\bar{f}(x; x^0) = f(x^0) + \nabla f(x^0)(x - x^0)
\]

- So the linearized version becomes:

\[
\text{Minimize } \bar{f}(x; x^0)
\]
\[
\text{Subject to } Ax \leq b, x \geq 0
\]

- The Solution of the linearized version is \( \tilde{x}^* \). How close is \( \tilde{x}^* \) to \( x^* \) the solution to the original NLP?

- By virtue of minimization it must be true that

\[
\bar{f}(x^0; x^0) > \bar{f}(x^*; x^0)
\]
Linear and Quadratic Approximations

• Using Taylor’s approximation around the linearization point $x_0 = a$

$$f(x) = f(a) + f'(a)(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x - a)^n$$

More compactly

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x - a)^k$$
Total Differential in 2D (estimated change in z)

So, in the notation of differentials, the linear approximation (4) can be written as

\[ f(x, y) \approx f(a, b) + dz \]

**First-order Taylor Series**

Figure 7 is the three-dimensional counterpart of Figure 6 and shows the geometric interpretation of the differential \( dz \) and the increment \( \Delta z \): \( dz \) represents the change in height of the tangent plane, whereas \( \Delta z \) represents the change in height of the surface \( z = f(x, y) \) when \((x, y)\) changes from \((a, b)\) to \((a + \Delta x, b + \Delta y)\).
Given a feasible trial solution $x'$, the linear approximation used for the objective function $f(x)$ is the first-order Taylor series expansion of $f(x)$ around $x = x'$, namely,

$$f(x') \approx f(x') + \sum_{j=1}^{n} \frac{\partial f(x')}{\partial x_j} (x_j - x'_j) = f(x') + \nabla f(x')(x - x'),$$

where these partial derivatives are evaluated at $x = x'$. Because $f(x')$ and $\nabla f(x')x'$ have fixed values, they can be dropped to give an equivalent linear objective function

$$g(x) = \nabla f(x')x = \sum_{j=1}^{n} c_j x_j, \quad \text{where } c_j = \frac{\partial f(x)}{\partial x_j} \quad \text{at } x = x'.$$
Initialization: Find a feasible initial trial solution $x^{(0)}$, for example, by applying linear programming procedures to find an initial BF solution. Set $k = 1$.

Iteration:

1. For $j = 1, 2, \ldots, n$, evaluate
   \[
   \frac{\partial f(x)}{\partial x_j} \quad \text{at} \quad x = x^{(k-1)}
   \]
   and set $c_j$ equal to this value.

2. Find an optimal solution $x_{LP}^{(k)}$ for the following linear programming problem.

   Maximize \[ g(x) = \sum_{j=1}^{n} c_j x_j, \]
   subject to
   \[ Ax \leq b \quad \text{and} \quad x \geq 0. \]

3. For the variable $t$ ($0 \leq t \leq 1$), set
   \[ h(t) = f(x) \quad \text{for} \quad x = x^{(k-1)} + t(x_{LP}^{(k)} - x^{(k-1)}), \]
   so that $h(t)$ gives the value of $f(x)$ on the line segment between $x^{(k-1)}$ (where $t = 0$) and $x_{LP}^{(k)}$ (where $t = 1$). Use some procedure such as the one-dimensional search procedure (see Sec. 13.4) to maximize $h(t)$ over $0 \leq t \leq 1$, and set $x^{(k)}$ equal to the corresponding $x$. Go to the stopping rule.

Stopping rule: If $x^{(k-1)}$ and $x^{(k)}$ are sufficiently close, stop and use $x^{(k)}$ (or some extrapolation of $x^{(0)}, x^{(1)}, \ldots, x^{(k-1)}, x^{(k)}$) as your estimate of an optimal solution. Otherwise, reset $k = k + 1$ and perform another iteration.
Linearly Constrained NLP case

- Using a bit of algebra leads us to the result:
  \[ \nabla f(x^0)(\hat{x}^* - x^0) < 0 \]

- So the vector \( \hat{x}^* - x^0 \) is a descent direction.
- Previously studied that a descent direction can lead to an improved point only if it is coupled with a step adjustment procedure.
- All points between \( \hat{x}^* \) and \( x^0 \) are feasible. Moreover since \( \hat{x}^* \) is a corner point, any point beyond it on the line are outside the feasible region.
- So, to improve upon \( \hat{x}^* \), a line search method is employed in the line segment \( x = x^0 + \alpha(\hat{x}^* - x^0) \quad 0 \leq \alpha \leq 1 \)

- Minimizing \( x \) will find a point \( x^1 \) such that \( f(x^1) \leq f(x^0) \)
Linearly Constrained NLP case

- $x^1$ will not in general be the optimal solution but it will serve as a linearization point for the next approximating LP.
- The text book presents the Frank-Wolfe Algorithm that employs this sequence of alternating LP’s and line searches.
Initialization: Find a feasible initial trial solution $x^{(0)}$, for example, by applying linear programming procedures to find an initial BF solution. Set $k = 1$.

Iteration:
1. For $j = 1, 2, \ldots, n$, evaluate
   \[ \frac{\partial f(x)}{\partial x_j} \quad \text{at} \quad x = x^{(k-1)} \]
   and set $c_j$ equal to this value.
2. Find an optimal solution $x_{LP}^{(k)}$ for the following linear programming problem.

   Maximize \[ g(x) = \sum_{j=1}^{n} c_j x_j, \]

   subject to
   \[ Ax \leq b \quad \text{and} \quad x \geq 0. \]
3. For the variable $t$ ($0 \leq t \leq 1$), set
   \[ h(t) = f(x) \quad \text{for} \quad x = x^{(k-1)} + t(x_{LP}^{(k)} - x^{(k-1)}), \]
   so that $h(t)$ gives the value of $f(x)$ on the line segment between $x^{(k-1)}$ (where $t = 0$) and $x_{LP}^{(k)}$ (where $t = 1$). Use some procedure such as the one-dimensional search procedure (see Sec. 13.4) to maximize $h(t)$ over $0 \leq t \leq 1$, and set $x^{(k)}$ equal to the corresponding $x$. Go to the stopping rule.

Stopping rule: If $x^{(k-1)}$ and $x^{(k)}$ are sufficiently close, stop and use $x^{(k)}$ (or some extrapolation of $x^{(0)}, x^{(1)}, \ldots, x^{(k-1)}, x^{(k)}$) as your estimate of an optimal solution. Otherwise, reset $k = k + 1$ and perform another iteration.
Maximize \[ f(x) = 5x_1 - x_1^2 + 8x_2 - 2x_2^2, \]
subject to
\[ 3x_1 + 2x_2 \leq 6 \]
and
\[ x_1 \geq 0, \quad x_2 \geq 0. \]
Note that
\[ \frac{\partial f}{\partial x_1} = 5 - 2x_1, \quad \frac{\partial f}{\partial x_2} = 8 - 4x_2, \]
\[ \nabla f(x^0)(x^k - x^0) < 0 \]
Given a feasible trial solution $\mathbf{x}'$, the linear approximation used for the objective function $f(\mathbf{x})$ is the first-order Taylor series expansion of $f(\mathbf{x})$ around $\mathbf{x} = \mathbf{x}'$, namely,

$$f(\mathbf{x}') \approx f(\mathbf{x}') + \sum_{j=1}^{n} \frac{\partial f(\mathbf{x}')}{\partial x_j} (x_j - x'_j) = f(\mathbf{x}') + \nabla f(\mathbf{x}') (\mathbf{x} - \mathbf{x}'),$$

where these partial derivatives are evaluated at $\mathbf{x} = \mathbf{x}'$. Because $f(\mathbf{x}')$ and $\nabla f(\mathbf{x}') \mathbf{x}'$ have fixed values, they can be dropped to give an equivalent linear objective function

$$g(\mathbf{x}) = \nabla f(\mathbf{x}') \mathbf{x} = \sum_{j=1}^{n} c_j x_j, \quad \text{where } c_j = \frac{\partial f(\mathbf{x})}{\partial x_j} \quad \text{at } \mathbf{x} = \mathbf{x}'.$
Iteration 1: Because \( x = (0, 0) \) is clearly feasible (and corresponds to the initial BF solution for the linear programming constraints), let us choose it as the initial trial solution \( x^{(0)} \) for the Frank-Wolfe algorithm. Plugging \( x_1 = 0 \) and \( x_2 = 0 \) into the expressions for the partial derivatives gives \( c_1 = 5 \) and \( c_2 = 8 \), so that \( g(x) = 5x_1 + 8x_2 \) is the initial linear approximation of the objective function. Graphically, solving this linear programming problem (see Fig. 12.17a) yields \( x_{LP}^{(1)} = (0, 3) \). For step 3 of the first iteration, the points on the line segment between \( (0, 0) \) and \( (0, 3) \) shown in Fig. 12.17a are expressed by

\[
(x_1, x_2) = (0, 0) + t[(0, 3) - (0, 0)] \quad \text{for } 0 \leq t \leq 1
\]

\[
= (0, 3t)
\]

as shown in the sixth column of Table 12.6. This expression then gives

\[
h(t) = f(0, 3t) = 8(3t) - 2(3t)^2
\]

\[
= 24t - 18t^2,
\]

so that the value \( t = t^* \) that maximizes \( h(t) \) over \( 0 \leq t \leq 1 \) may be obtained in this case by setting

\[
\frac{dh(t)}{dt} = 24 - 36t = 0,
\]

so that \( t^* = \frac{2}{3} \). This result yields the next trial solution

\[
x^{(1)} = (0, 0) + \frac{2}{3}[(0, 3) - (0, 0)]
\]

\[
= (0, 2),
\]
### TABLE 12.6 Application of the Frank-Wolfe algorithm to the example

<table>
<thead>
<tr>
<th>$k$</th>
<th>$x^{(k-1)}$</th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$x_{LP}^{(k)}$</th>
<th>$x$ for $h(t)$</th>
<th>$h(t)$</th>
<th>$t^*$</th>
<th>$x^{(k)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(0, 0)</td>
<td>5</td>
<td>8</td>
<td>(0, 3)</td>
<td>(0, 3$t$)</td>
<td>$24t - 18t^2$</td>
<td>$\frac{2}{3}$</td>
<td>(0, 2)</td>
</tr>
<tr>
<td>2</td>
<td>(0, 2)</td>
<td>5</td>
<td>0</td>
<td>(2, 0)</td>
<td>(2$t$, 2 - 2$t$)</td>
<td>$8 + 10t - 12t^2$</td>
<td>$\frac{5}{12}$</td>
<td>($\frac{5}{6}$, $\frac{7}{6}$)</td>
</tr>
</tbody>
</table>
Iteration 2: To sketch the calculations that lead to the results in the second row of Table 12.6, note that $x^{(1)} = (0, 2)$ gives

$$c_1 = 5 - 2(0) = 5,$$
$$c_2 = 8 - 4(2) = 0.$$

For the objective function $g(x) = 5x_1$, graphically solving the problem over the feasible region in Fig. 12.17a gives $x^{(2)}_{1-2} = (2, 0)$. Therefore, the expression for the line segment between $x^{(1)}$ and $x^{(2)}_{1-2}$ (see Fig. 12.17a) is

$$x = (0, 2) + t[(2, 0) - (0, 2)]$$
$$= (2t, 2 - 2t),$$

so that

$$h(t) = f(2t, 2 - 2t)$$
$$= 5(2t) - (2t)^2 + 8(2 - 2t) - 2(2 - 2t)^2$$
$$= 8 + 10t - 12t^2.$$

Setting

$$\frac{dh(t)}{dt} = 10 - 24t = 0$$

yields $t^* = \frac{5}{12}$. Hence,

$$x^{(2)} = (0, 2) + \frac{5}{12}[(2, 0) - (0, 2)]$$
$$= \left(\frac{5}{6}, \frac{7}{6}\right),$$

which completes the second iteration.
Optimal Solution

(a)
Frank-Wolfe Algorithm Execution: Example 8.1

Minimize \[ f(x) = x_1^{1/4} + \left(\frac{x_2}{x_1}\right)^{1/4} + \left(\frac{64}{x_2}\right)^{1/4} \]

Subject to \[ x_1 \geq 1 \quad x_2 \geq x_1 \quad 64 \geq x_2 \]

Start with \( x^0 = (2, 10) \)
Frank-Wolfe Algorithm Execution: Example 8.1
(Page Number 340)
Frank-Wolfe Algorithm

- Frank-Wolfe algorithm converges to a Kuhn-Tucker point from any feasible starting point.
- No analysis for rate of convergence.
- However, if \( f(x) \) is convex we can obtain estimates on how much remaining improvements can be achieved.
- If \( f(x) \) is convex and it is linearized at a point \( x(t) \), for all \( x \)

\[
f(x) \geq f(x^{(0)}) + \nabla f(x^{(0)})(x - x^{(0)}) = \tilde{f}(x; x^{(0)})
\]

\[
\Rightarrow \quad \min f(x) \geq \min \tilde{f}(x; x^{(0)}) = \tilde{f}(y^{(0)}; x^{(0)})
\]

- Hence after each cycle the difference gives an estimate of the improvement.

\[
f(x^{(t+1)}) - \tilde{f}(y^{(0)}; x^{(0)})
\]
Conclusions: Frank-Wolfe Algorithm

• In conclusion, we emphasize that the Frank-Wolfe algorithm is just one example of sequential-approximation algorithms. Many of these algorithms use quadratic instead of linear approximations at each iteration because quadratic approximations provide a considerably closer fit to the original problem and thus enable the sequence of solutions to converge considerably more rapidly toward an optimal solution than was the case in the linear FW algo.

• For this reason, even though sequential linear approximation methods such as the Frank-Wolfe algorithm are relatively straightforward to use, sequential quadratic approximation methods now are generally preferred in actual applications. Popular among these are the quasi-Newton (or variable metric) methods, which compute a quadratic approximation to the curvature of a nonlinear function without explicitly calculating second (partial) derivatives.

• For further information about convex programming algorithms, see Selected References 4 and 6.
Conclusions: Frank-Wolfe Algorithm

- In conclusion, we emphasize that the Frank-Wolfe algorithm is just one example of sequential-approximation algorithms. Many of these algorithms use quadratic instead of linear approximations at each iteration because quadratic approximations provide a considerably closer fit to the original problem and thus enable the sequence of solutions to converge considerably more rapidly toward an optimal solution than was the case in Fig. 13.17b (H&L Book).

- For this reason, even though sequential linear approximation methods such as the Frank-Wolfe algorithm are relatively straightforward to use, sequential quadratic approximation methods now are generally preferred in actual applications. Popular among these are the quasi-Newton (or variable metric) methods, which compute a quadratic approximation to the curvature of a nonlinear function without explicitly calculating second (partial) derivatives. (For linearly constrained optimization problems, this nonlinear function is just the objective function; whereas with nonlinear constraints, it is the Lagrangian function described in Appendix 3.)

- Some quasi-Newton algorithms do not even explicitly form and solve an approximating quadratic programming problem at each iteration, but instead incorporate some of the basic ingredients of gradient algorithms.
Linearly Constrained NLP case: Frank-Wolfe Algorithm

Given $x^0$, line search, and overall convergence tolerances $\varepsilon > 0$ and $\delta > 0$.

**Step 1.** Calculate $\nabla f(x^{(t)})$. If $\|\nabla f(x^{(t)})\| \leq \varepsilon$, stop. Otherwise, go to step 2.

**Step 2.** Solve the LP subproblem

Minimize $\nabla f(x^{(t)})y$
Subject to $Ay \leq b$
$y \geq 0$

Let $y^{(t)}$ be the optimal solution to the LP problem.

**Step 3.** Find $\alpha^{(t)}$ which solves the problem

$$\min f(x^{(t)}) + \alpha(y^{(t)} - x^{(t)}) \quad 0 \leq \alpha \leq 1$$

**Step 4.** Calculate

$$x^{(t+1)} = x^{(t)} + \alpha^{(t)}(y^{(t)} - x^{(t)})$$

**Step 5.** Convergence check. If

$$\|x^{(t+1)} - x^{(t)}\| < \delta\|x^{(t+1)}\|$$

and if

$$\|f(x^{(t+1)}) - f(x^{(t)})\| \leq \varepsilon\|f(x^{(t+1)})\|$$

then terminate. Otherwise, go to step 1.
MULTI VARIABLE OPTIMIZATION PROCEDURES

• Introduction
• Mutivariable Search Methods Overview
• Unconstrained Multivariable Search Methods
  – Quasi-Newton Methods
  – Conjugate Gradient and Direction Methods
  – Logical Methods
• Constrained Multivariable Search Methods
  – Successive Linear Programming
  – Successive Quadratic Programming
  – Generalized Reduced Gradient Method
  – Penalty, Barrier and Augmented Lagrangian Functions
  – Other Multivariable Constrained Search Methods
  – Comparison of Constrained Multivariable Search Methods
• Stochastic Approximation Procedures
• Closure
• FORTRAN Program for BFGS Search of an Unconstrained Function
• References
• Problems
Sequential Unconstrained Minimization Techniques (SUMT)

We turn now from sequential-approximation algorithms to sequential unconstrained algorithms. As mentioned at the beginning of the section, algorithms of the latter type solve the original constrained optimization problem by instead solving a sequence of unconstrained optimization problems.

A particularly prominent sequential unconstrained algorithm that has been widely used since its development in the 1960s is the sequential unconstrained minimization technique (or SUMT for short). There actually are two main versions of SUMT, one of which is an exterior-point algorithm that deals with infeasible solutions while using a penalty function to force convergence to the feasible region. We shall describe the other version, which is an interior-point algorithm that deals directly with feasible solutions while using a barrier function to force staying inside the feasible region. Although SUMT was originally presented as a minimization technique, we shall convert it to a maximization technique in order to be consistent with the rest of the chapter. Therefore, we continue to assume that the problem is in the form given at the beginning of the chapter and that all the functions are differentiable.
Penalty and Barrier Methods

General classical constrained minimization problem

minimize $f(x)$
subject to
\[ g(x) \leq 0 \]
\[ h(x) = 0 \]

*Penalty methods are motivated by the desire to use unconstrained optimization techniques to solve constrained problems.*

This is achieved by either

- adding a **penalty** for infeasibility and forcing the solution to feasibility and subsequent optimum, or
- adding a **barrier** to ensure that a feasible solution never becomes infeasible.
Penalty Methods

Penalty methods use a mathematical function that will increase the objective for any given constrained violation.

General transformation of constrained problem into an unconstrained problem:

\[
\min T(x) = f(x) + r_k P(x)
\]

where
- \( f(x) \) is the objective function of the constrained problem
- \( r_k \) is a scalar denoted as the penalty or controlling parameter
- \( P(x) \) is a function which imposes penalties for infeasibility (note that \( P(x) \) is controlled by \( r_k \))
- \( T(x) \) is the (pseudo) transformed objective

Two approaches exist in the choice of transformation algorithms:
1) Sequential penalty transformations
2) Exact penalty transformations
Sequential Penalty Transformations

Sequential Penalty Transformations are the oldest penalty methods. Also known as Sequential Unconstrained Minimization Techniques (SUMT) based upon the work of Fiacco and McCormick, 1968.

Consider the following frequently used general purpose penalty function:

\[ T(x) = y(x) + r_k \left\{ \sum_{i=1}^{m} (\max[0, g_i(x)]^2) + \sum_{j=1}^{l} [h_j(x)]^2 \right\} \]

Sequential transformations work as follows:
Choose \( P(x) \) and a sequence of \( r_k \) such that when \( k \) goes to infinity, the solution \( x^* \) is found.

For example, for \( k = 1 \), \( r_1 = 1 \) and we solve the problem. Then, for the second iteration \( r_k \) is increased by a factor 10 and the problem is resolved starting from the previous solution.

Note that an increasing value of \( r_k \) will increase the effect of the penalties on \( T(x) \).

The process is terminated when no improvement in \( T(x) \) is found and all constraints are satisfied.
Two Classes of Sequential Methods

Two major classes exist in sequential methods:

1) First class uses a sequence of infeasible points and feasibility is obtained only at the optimum. These are referred to as penalty function or exterior-point penalty function methods.

2) Second class is characterized by the property of preserving feasibility at all times. These are referred to as barrier function methods or interior-point penalty function methods.

General barrier function transformation

\[ T(x) = y(x) + r_k B(x) \]

where \( B(x) \) is a barrier function and \( r_k \) the penalty parameter which is supposed to go to zero when \( k \) approaches infinity.

Typical Barrier functions are the inverse or logarithmic, that is:

\[ B(x) = - \sum_{i=1}^{m} g_i^{-1}(x) \quad \text{or} \quad B(x) = - \sum_{i=1}^{m} \ln[-g_i(x)] \]
As the name implies, SUMT replaces the original problem by a sequence of unconstrained optimization problems whose solutions converge to a solution (local maximum) of the original problem. This approach is very attractive because unconstrained optimization problems are much easier to solve (see Sec. 12.5) than those with constraints. Each of the unconstrained problems in this sequence involves choosing a (successively smaller) strictly positive value of a scalar $r$ and then solving for $x$ so as to

Maximize $P(x; r) = f(x) - rB(x)$.

Here $B(x)$ is a barrier function that has the following properties (for $x$ that are feasible for the original problem):

1. $B(x)$ is small when $x$ is far from the boundary of the feasible region.
2. $B(x)$ is large when $x$ is close to the boundary of the feasible region.
3. $B(x) \to \infty$ as the distance from the (nearest) boundary of the feasible region $\to 0$.

Thus, by starting the search procedure with a feasible initial trial solution and then attempting to increase $P(x; r)$, $B(x)$ provides a barrier that prevents the search from ever crossing (or even reaching) the boundary of the feasible region for the original problem.

The most common choice of $B(x)$ is

$$B(x) = \sum_{i=1}^{m} \frac{1}{b_i - g_i(x)} + \sum_{j=1}^{n} \frac{1}{x_j}.$$
**SUMT Summary**

*Initialization:* Identify a *feasible* initial trial solution $x^{(0)}$ that is not on the boundary of the feasible region. Set $k = 1$ and choose appropriate strictly positive values for the initial $r$ and for $\theta < 1$ (say, $r = 1$ and $\theta = 0.01$).

*Iteration k:* Starting from $x^{(k-1)}$, apply a multivariable unconstrained optimization procedure (e.g., the gradient search procedure) such as described in Sec. 12.5 to find a local maximum $x^{(k)}$ of

$$P(x; r) = f(x) - r \left( \sum_{i=1}^{m} \frac{1}{b_i - g_i(x)} + \sum_{j=1}^{n} \frac{1}{x_j} \right).$$

*Stopping rule:* If the change from $x^{(k-1)}$ to $x^{(k)}$ is negligible, stop and use $x^{(k)}$ (or an extrapolation of $x^{(0)}$, $x^{(1)}$, $x^{(k-1)}$, $x^{(k)}$) as your estimate of a *local maximum* of the original problem. Otherwise, reset $k = k + 1$ and $r = \theta r$ and perform another iteration.

Finally, we should note that SUMT also can be extended to accommodate *equality* constraints $g_i(x) = b_i$. One standard way is as follows. For each equality constraint,

$$\frac{-[b_i - g_i(x)]^2}{\sqrt{r}}$$

replaces

$$\frac{-r}{b_i - g_i(x)}$$
What to Choose?

• Some prefer barrier methods because even if they do not converge, you will still have a feasible solution.

• Others prefer penalty function methods because
  • You are less likely to be stuck in a feasible pocket with a local minimum.
  • Penalty methods are more robust because in practice you may often have an infeasible starting point.

• However, penalty functions typically require more function evaluations.

• Choice becomes simple if you have equality constraints. (Why?)
SUMT Closing Remarks

Typically, you will encounter sequential approaches.

Various penalty functions $P(x)$ exist in the literature.

Various approaches to selecting the penalty parameter sequence exist. Simplest is to keep it constant during all iterations.

Always ensure that penalty does not dominate the objective function during initial iterations of exterior point method.
Penalty, Barrier Methods

- These methods convert the constrained optimization problem into an unconstrained one.
- The idea is to modify the economic model by adding the constraints in such a manner to have the optimum be located and the constraints be satisfied. There are several forms for the function of the constraints that can be used. These create a penalty to the economic model if the constraints are not satisfied or form a barrier to force the constraints to be satisfied, as the unconstrained search method moves from the starting point to the optimum.
- This approach is related to the method of Lagrange multipliers which is a procedure that modifies the economic model with the constraint equations to have an unconstrained problem. Also, the Lagrangian function can be used with an unconstrained search technique to locate the optimum and satisfy the constraints. In addition, the augmented Lagrangian function combines a penalty function with the Lagrangian function to alleviate computational difficulties associated with boundaries formed by equality constraints when the Lagrangian function is used alone.
Example. To illustrate SUMT, consider the following two-variable problem:

Maximize \[ f(x) = x_1 x_2, \]

subject to

\[ x_1^2 + x_2 \leq 3 \]

and

\[ x_1 \geq 0, \quad x_2 \geq 0. \]

Even though \( g_1(x) = x_1^2 + x_2 \) is convex (because each term is convex), this problem is a nonconvex programming problem because \( f(x) = x_1 x_2 \) is not concave (see Appendix 2). However, the problem is close enough to being a convex programming problem that SUMT necessarily will still converge to an optimal solution in this case. (We will discuss non-convex programming further, including the role of SUMT in dealing with such problems, in the next section.)

For the initialization, \((x_1, x_2) = (1, 1)\) is one obvious feasible solution that is not on the boundary of the feasible region, so we can set \( x^{(0)} = (1, 1) \). Reasonable choices for \( r \) and \( \theta \) are \( r = 1 \) and \( \theta = 0.01 \).

For each iteration,

\[ P(x; r) = x_1 x_2 - r \left( \frac{1}{3 - x_1^2 - x_2} + \frac{1}{x_1} + \frac{1}{x_2} \right). \]
Iterations in SUMT Example

With $r = 1$, applying the gradient search procedure starting from $(1, 1)$ to maximize this expression eventually leads to $x^{(1)} = (0.90, 1.36)$. Resetting $r = 0.01$ and restarting the gradient search procedure from $(0.90, 1.36)$ then lead to $x^{(2)} = (0.983, 1.933)$. One more iteration with $r = 0.01(0.01) = 0.0001$ leads from $x^{(2)}$ to $x^{(3)} = (0.998, 1.994)$. This sequence of points, summarized in Table 12.7, quite clearly is converging to $(1, 2)$. Applying the KKT conditions to this solution verifies that it does indeed satisfy the necessary condition for optimality. Graphical analysis demonstrates that $(x_1, x_2) = (1, 2)$ is, in fact, a global maximum (see Prob. 12.9-13b).

Homework 12.9 – 13.b
See Book website for more worked examples

•

The Worked Examples section of the book’s website provides another example that illustrates the application of SUMT to a convex programming problem in minimization form. You also can go to your OR Tutor to see an additional example. An automatic procedure for executing SUMT is included in IOR Tutorial.
Generalized Reduced Gradient (GRG)

- [http://www.mpri.lsu.edu/textbook/Chapter6-b.htm](http://www.mpri.lsu.edu/textbook/Chapter6-b.htm)

The first category is gradient algorithms, where the gradient search procedure of Sec. 12.5 is modified in some way to keep the search path from penetrating any constraint boundary. For example, one popular gradient method is the generalized reduced gradient (GRG) method. The Excel Solver uses the GRG method for solving convex programming problems. (As discussed in the next section, Premium Solver also includes an Evolutionary Solver option that is well suited for dealing with nonconvex programming problems.)
Nonconvex Programming

- Convert into subproblems

- Find local optimal

- Evolutionary Computing (e.g., genetic algorithms)
  - Run EC, and then use gradient descent
Outline Lecture 10

- Multivariable unconstrained
- Multivariable unconstrained optimization
  - Linear regression, Logistic Regression
- KKT conditions for Constrained Optimization
  - Lagrangian, KKT conditions
- Quadratic Programming
  - Modified Simplex Algorithm
- Convex Programming
  - Frank-Wolfe Algorithm, Penalty or barrier function (e.g., SUMT)
- Nonconvex Programming
- Course review
<table>
<thead>
<tr>
<th>Date</th>
<th>Slides+HW (HW at end of slides)</th>
<th>SupplementaryMaterial</th>
<th>Week</th>
<th>Lecture Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/10/12</td>
<td>Slides+HW Lecture 1</td>
<td>Homework 1 Solutions</td>
<td>1</td>
<td>Introduction to class, Linear Programming Intro, LP via graphical method, Intro R part 1</td>
</tr>
<tr>
<td>1/17/12</td>
<td>Slides+HW Lecture 2</td>
<td>WynDor in Excel</td>
<td>2</td>
<td>LP Assumptions, Simplex Method (Algebraic, Tableau) and intro R part 2</td>
</tr>
<tr>
<td>1/24/12</td>
<td>Slides+HW Lecture 3</td>
<td>Homework 2 Solutions</td>
<td>3</td>
<td>Adapting simplex to other forms (=, ≥, negative b); Two Phase, Big-M and Artificial Variable technique; Sensitivity Analysis; Shadow Prices; Simplex via matrices; Fundamental insight</td>
</tr>
<tr>
<td>1/31/12</td>
<td>Slides + HW Lecture 4</td>
<td>Homework 4 Solutions</td>
<td>4</td>
<td>Sensitivity and Duality, Other LP Methods, LP Applications; Transportation, Assignment and Network Optimization Problems; Case Studies in Digital Advertising and Marketing</td>
</tr>
<tr>
<td>2/7/12</td>
<td>Lecture 5</td>
<td>Homework 5 Solutions</td>
<td>5</td>
<td>Nonlinear programming categories, applications of NLP, Nonlinear Unconstrained, bisection method, Gradient descent Nonlinear Constrained,</td>
</tr>
<tr>
<td>2/14/12</td>
<td>EXAM</td>
<td>EXAM SOLUTIONS</td>
<td>6</td>
<td>Midterm Exam</td>
</tr>
<tr>
<td>2/21/12</td>
<td>Slides and HW Lecture 7</td>
<td>Homework 7 Solutions</td>
<td>7</td>
<td>convex optimization, machine learning and Dynamic Programming Integer Programming, Branch and Bound, Multiple choice Knapsack Problem (MCKP), [Genetic Algorithms] Optimal Bid Determination in Sponsored Search as an MCKP</td>
</tr>
<tr>
<td>2/28/12</td>
<td></td>
<td></td>
<td>8</td>
<td>Metaheuristics and Markov Chains</td>
</tr>
<tr>
<td>3/7/12</td>
<td></td>
<td></td>
<td>9</td>
<td>Game Theory, Decision Analysis and Queueing Theory</td>
</tr>
<tr>
<td>3/14/12</td>
<td></td>
<td></td>
<td>10</td>
<td>Inventory Theory and Simulation</td>
</tr>
<tr>
<td>Wk of 3/18</td>
<td></td>
<td></td>
<td>11</td>
<td>Final Exam</td>
</tr>
</tbody>
</table>
• End of Lecture