Lecture Notes on Engineering Optimization

Fraser J. Forbes and Ilyasse Aksikas

Department of Chemical and Materials Engineering
University of Alberta
aksikas@ualberta.ca
Contents

– Chapter 1 : Introduction - Basic Concepts
– Chapter 2 : Linear Programming
– Chapter 3 : Unconstrained Univariate Nonlinear Optimization
– Chapter 4 : Unconstrained Multivariate Nonlinear Optimization
– Chapter 5 : Constrained Nonlinear Optimization
Chapter I

Introduction - Basic Concepts

- Introductory Example
- General Formulation and Classification
- Development Stages
- Degrees of Freedom
1 Introductory Example

Problem

Schedule your activities for next week-end, given you only have $100 to spend.

Question

How can we go about solving this problem in a coherent fashion?
A systematic approach might be:

**Step 1**: Choose a scheduling objective.

**Step 2**: List all possible activities and pertinent information regarding each activity,

<table>
<thead>
<tr>
<th>activity</th>
<th>fun units/hr $F_i$</th>
<th>cost $C_i$</th>
<th>time limits $L_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 3: Specify other limitations.

\[ t_{\text{available}} = \]
\[ s_{\text{available}} = \]

Step 4: Decide what the decision variables are.

Step 5: Write down any relationships you know of among the variables.
Step 6: Write down all restrictions on the variables.

Step 7: Write down the mathematical expression for the objective.
Step 8: Write the problem out in a standard form.

There is a more compact way to write this problem down.

→ matrix-vector form.
2 General Formulation

Optimize \( P(x) \)  
Objective function

Performance function
Profit (cost) function

Subject to

\[ f(x) = 0 \]  
Equality constraints
Process model

\[ g(x) \leq 0 \]  
Inequality constraints
Operating constraints

\[ x_l \leq x \leq x_u \]  
Variable bounds
Variables limits
3 Development Stages

Stage 1: Problem Definition

- Objectives
  - goal of optimization
  - performance measurement
  - example: max profit, min energy cost, etc
  \[ \max(P(x)) = \min(-P(x)). \]
- Process models
  → assumptions & constants
  → equations to represent relationships
    * material / energy balances
    * thermodynamics
    * kinetics, etc
  → constraints and bounds to represent limitations
    * operating restrictions,
    * bounds / limits

A point which satisfies the process model equations is a \textit{feasible point}. Otherwise, it is an \textit{infeasible point}. 
Stage 2: Problem Formulation

- Standard form

- Degrees of freedom analysis
  
  → over-, under- or exactly specified?
  
  → decision variables (independent vs. dependent)

- Scaling of variables
  
  → units of measure
  
  → scaling factors
Stage 3: Problem Solution

- Technique selection
  → matched to problem type
  → exploit problem structure
  → knowledge of algorithms strengths & weaknesses

- Starting points
  → usually several and compare

- Algorithm tuning
  → termination criteria
  → convergence tolerances
  → step length
Stage 4: Results Analysis

- Solution uniqueness
- Perturbation of optimization problem
  - effects of assumptions
  - variation (uncertainty) in problem parameters
  - variation in prices/costs
Solving an optimization problem is iterative

1. Definition
2. Formulation
3. Solution
4. Results Analysis

Solution of an optimization problem requires all of the steps
– a full understanding is developed by following the complete cycle
– good decisions require a full understanding of the problem
– shortcuts lead to bad decisions.
Classifying Optimization Problems

- Multi-Goal Programming (MGP)
- Vector objective scalar matrix
- Deterministic problem
- Discrete continuity
- Linear constraints
- Linear objective
- Mixed-Integer Linear Programming (MILP)
- Mixed-Integer Non-Linear Programming (MINLP)
- Linear Programming (LP)
- Non-Linear Programming (NLP)
In this course, our problems will have:

- objective functions which are continuously differentiable,
- constraint equations which are continuously differentiable.

Optimization problems with scalar objective functions and vector constraints can be classified:

<table>
<thead>
<tr>
<th>objective function</th>
<th>constraints</th>
<th>problem type</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>linear</td>
<td>LP</td>
</tr>
<tr>
<td>nonlinear</td>
<td>linear</td>
<td>NLP</td>
</tr>
<tr>
<td>linear</td>
<td>nonlinear</td>
<td>NLP</td>
</tr>
<tr>
<td>nonlinear</td>
<td>nonlinear</td>
<td>NLP</td>
</tr>
</tbody>
</table>
4 Degrees of Freedom

To determine the degrees of freedom (the number of variables whose values may be independently specified) in our model we could simply count the number of independent variables (the number of variables which remain on the right-hand side) in our modified equations.

This suggests a possible definition:

\[
\text{degrees of freedom} = \# \text{ variables} - \# \text{ equations}
\]

**Definition**: The degrees of freedom for a given problem are the number of independent problem variables which must be specified to uniquely determine a solution.
Consider the following three equations relating three variables $x_1, x_2$ and $x_3$:

\[
\begin{align*}
  x_1 - x_2 &= 0 \\
  x_1 - 2x_3 &= 0 \\
  x_2 - 2x_3 &= 0
\end{align*}
\]

This seems to indicate that there are no degrees of freedom.

Notice that if we subtract the last from the second equation:

\[
\begin{align*}
  x_1 - 2x_3 &= 0 \\
  - x_2 - 2x_3 &= 0 \\
  \hline
  x_1 - x_2 &= 0
\end{align*}
\]

the result is the first equation.

It seems that we have three different equations, which contain no more information than two of the equations. In fact any of the equations is a linear combination of the other two equations.

We require a clearer, more precise definition for degrees of freedom.
A More Formal Approach:

Suppose we have a set of $m$ equations:

$$h(v) = 0$$

in the set of variables $v$ ($n + m$ elements). We would like to determine whether the set of equations can be used to solve for some of the variables in terms of the others.

In this case we have a system of $m$ equations in $n + m$ unknown variables. The Implicit Function Theorem states that if the $m$ equations are linearly independent, then we can divide our set of variables $v$ into $m$ dependent variables $u$ and $n$ independent variables $x$:

The Implicit Function Theorem goes on to give conditions under which the dependent variables $u$ may be expressed in terms of the independent variables $x$ or:

$$u = g(x)$$
Usually we don’t need to find the set of equations \( u = g(x) \), we only need to know if it is possible. Again the Implicit Function Theorem can help us out:

if \( \text{rank}[\nabla_v h] = m \), all of the model equations are linearly independent and it is possible (at least in theory) to use the set of equations \( h(v) = 0 \) to determine values for all of the \( m \) dependent variables \( u \) given values for the \( n \) independent variables \( x \).

Alternatively we could say that the number of degrees of freedom in this case are the number of independent variables. (Recall that there are \( n \) variables in \( x \)).

We know that:

\[
\text{rank}[\nabla_v h] \leq m.
\]

What does it mean if:

\[
\text{rank}[\nabla_v h] < m?
\]
Let’s investigate a simple set of equations:

\[ h(v) = \begin{bmatrix} v_1 - v_1v_2 - \alpha e^{v_3} \\ v_1 - v_2 - v_3 \end{bmatrix} \]

with \( a = 1/e \). (This could be the material balances for a reactor.) A three dimensional plot of the equations looks like:
The solution of the equations lies at the intersection of the two surfaces, each described by one of the equations.

How many degrees of freedom does this problem have? 
Take a closer look:
Examine the neighbourhood of the point:

\[ v_1 = 1, v_2 = 0, v_3 = 1 \]

What is happening here?

The Jacobian of the equation set \( h(v) = 0 \) is:

\[
\nabla_v h = \begin{bmatrix}
1 - v_2 & -v_1 & -a e^{v_3} \\
1 & -1 & -1
\end{bmatrix}
\]

When \( a=1/e \) at the point \( v = [1, 0, 1]^T \), then:

\[
\nabla_v h = \begin{bmatrix}
1 & -1 & -1 \\
1 & -1 & -1
\end{bmatrix}
\]

What is the rank of this matrix?

\[ rank[\nabla_v h] = 1 < m = 2. \]

This tells us that, at the point \( v = [1, 0, 1]^T \), our system contains only one linearly independent equation.
Then:

degrees of freedom = 3 variables - 1 equation = 2.

→ at the given point two variables must be specified to uniquely determine the solution to the problem.

This example was chosen because it was very easy to see the occurrence of linear dependence within the equation set. Of course the situation can be much more complex for real problems with many more equations and variables. So, clearly we cannot determine degrees of freedom by counting the number of equations in our problem.

As a result, we must modify our definition:

\[
\text{DOF} = \# \text{ variables} - \# \text{ linearly independent equations}
\]
In summary, determination of the degrees of freedom for a specific steady-state set of equations requires:

1. determination of rank $[\nabla_v h]$.
2. if rank $[\nabla_v h] = \text{number of equations (} m \text{)}$, then all of the equations are linearly independent and:

   $$\text{d.o. f} = \# \text{ variables} - \# \text{ equations}.$$  

3. if rank $[\nabla_v h] = \text{number of equations (} m \text{)}$, then all of the equations are not linearly independent and:

   $$\text{d.o. f} = \# \text{ variables} - \# \text{ rank } [\nabla_v h].$$

Remember: For sets of linear equations, the analysis has to be performed only once. Generally, for nonlinear equation sets, the analysis is only valid at the variables values used in the analysis.
Degrees of freedom analysis tells us the maximum number of variables which can be independently specified to uniquely determine a feasible solution to a given problem.

We need to consider degrees of freedom when solving many different types of problems. These include:

- plant design,
- plant flow sheeting,
- model fitting,

and, of course, optimization problems.
Consider the example of isothermal reactor system:

The material balances are:

- **mass** \( F_A - F_R = 0 \)
- **component A** \( F_A - kX_AV\rho - X_AF_R = 0 \)
- **component B** \( kX_AV\rho - X_BF_R = 0 \)
What other information do we have?

By definition, we also know:

\[ 1 - X_A - X_B = 0 \]

If we know the feed-rate \( F_A = 5 \text{kg/min} \), reactor volume \( V = 2 \text{ litres} \), density of the reaction mixture \( \rho = 1 \text{kg/l} \), and reaction rate constant \( k = 7 \text{min}^{-1} \), then the material balances can be re-written:

\[
\begin{align*}
5 - F_R &= 0 \\
5 - 14X_A - X_A F_R &= 0 \\
14X_A - X_B F_R &= 0 \\
1 - X_A - X_B &= 0
\end{align*}
\]

Is this set of equations linear or nonlinear?
In our problem, there are:

3 variables \((X_A, X_B, F_R)\)
4 equations (1 mass balance, 2 component balances, 1 other equation).

What does this mean?

Consider what happens if we add:

component \(A\) + component \(B\) - mass

\[
\Rightarrow F_R - X_A F_R - X_B F_R = 0
\]

or:

\[
F_R (1 - X_A - X_B) = 0
\]

and since \(F_R \neq 0\), then

\[
1 - X_A - X_B = 0
\]
which is our fourth equation. Thus our fourth equations are not linearly independent.

Caution:
Care should be taken when developing a model to avoid this situation of specifying too many linearly dependent equations, since it often leads to difficulty in finding a feasible solution.

We need to eliminate one of the equations from our model. I would suggest eliminating:

\[ 1 - X_A - X_B = 0 \]

This equation could then provide an easy check on any solution we find using the other equations.

Thus our model is reduced to:

\[ 5 - F_R = 0 \]
\[ 5 - 14X_A - X_A F_R = 0 \]
\[ 14X_A - X_B F_R = 0 \]
How many degrees of freedom are there in this model?

For our reactor model

$$\nabla v h = \begin{bmatrix}
-1 & 0 & 0 \\
-X_A & -14 - F_A & 0 \\
-X_B & 14 & -F_R
\end{bmatrix}$$

what is the rank of this matrix? (suppose $F_R = 0$ or $F_R = -14$.)

We could determine the rank by:

1. using software (Maple, Matlab, Mathcad, ...)
2. examining the eigenvalues (which are displayed on the diagonal of a triangular matrix.

$$\text{rank}[\nabla v h] = 3 \quad F_R \neq 0 \text{ or } -14$$

$$\text{rank}[\nabla v h] = 2 \quad F_R = 0 \text{ or } -14$$

3. using the determinant

$$|\nabla v h| = -F_R(14 + F_R)$$
Then for our reactor problem and realistic flow-rates, we have:

3 variables \((F_R, X_A, X_B)\)

3 linearly independent equations (1 mass and 2 component balances).

Therefore:

\[
\text{degrees of freedom} = 3 - 3 = 0.
\]

This says there are no degrees of freedom for optimization and the reactor problem is fully specified.

In fact the unique solution is:

\[
F_R = 5 \text{ kg/min.}
\]

\[
X_A = 5/19 \text{ wt. fraction}
\]

\[
X_B = 14/19 \text{ wt. fraction}
\]
Until now, we have specified the feed-rate \((F_A)\) in our reactor problem. Suppose we decide to include it as an optimization variable:

1. how many degrees of freedom are now available for optimization?

\[
d.o.f = 4 \text{ variables} - 3 \text{ equations} = 1
\]

2. Can we re-introduce the fourth equation:

\[
1 - X_A - X_B = 0
\]

to reduce the degrees of freedom?

No, using \(F_A\) as an optimization variable does not change the linear dependence among the four equations.
Chapter II

Linear Programming

1. Introduction
2. Simplex Method
3. Duality Theory
4. Optimality Conditions
5. Applications (QP & SLP)
6. Sensitivity Analysis
7. Interior Point Methods
1 Introduction

Linear programming became important during World War II:

⇒ used to solve logistics problems for the military.

Linear Programming (LP) was the first widely used form of optimization in the process industries and is still the dominant form today.

Linear Programming has been used to:

→ schedule production,
→ choose feedstocks,
→ determine new product feasibility,
→ handle constraints for process controllers, . . .

There is still a considerable amount of research taking place in this area today.
All Linear Programs can be written in the form:

$$\min_{x_i} \quad c_1 x_1 + c_2 x_2 + c_3 x_3 + \ldots + c_n x_n$$

subject to:

$$a_{11} x_1 + a_{12} x_2 + a_{13} x_3 + \ldots + a_{1n} x_n \leq b_1$$
$$a_{21} x_1 + a_{22} x_2 + a_{23} x_3 + \ldots + a_{2n} x_n \leq b_2$$
$$\vdots$$
$$a_{m1} x_1 + a_{m2} x_2 + a_{m3} x_3 + \ldots + a_{mn} x_n \leq b_m$$

$$0 \leq x_i \leq x_{l,i}$$
$$b_i \geq 0$$

$\Rightarrow$ note that all of the functions in this optimization problem are linear in the variables $x_i$. 
In mathematical short-hand this problem can be re-written:

$$\min_{x} \quad c^T x$$

subject to:

$$Ax \leq b$$
$$0 \leq x \leq x_l$$
$$b \geq 0$$

Note that:

1. the objective function is linear
2. the constraints are linear
3. the variables are defined as non-negative
4. all elements of $b$ are non-negative.
Example

A market gardener has a small plot of land she wishes to use to plant cabbages and tomatoes. From past experience she knows that her garden will yield about 1.5 tons of cabbage per acre or 2 tons of tomatoes per acre. Her experience also tells her that cabbages require 20 lbs of fertilizer per acre, whereas tomatoes require 60 lbs of fertilizer per acre. She expects that her efforts will produce $600/ton of cabbages and $750/ton of tomatoes.

Unfortunately, the truck she uses to take her produce to market is getting old, so she doesn’t think that it will transport anymore than 3 tons of produce to market at harvest time. She has also found that she only has 60 lbs of fertilizer.

**Question** : What combination of cabbages and tomatoes should she plant to maximize her profits?
The optimization problem is:

In the general matrix LP form:
We can solve this problem graphically:

We know that for well-posed LP problems, the solution lies at a vertex of the feasible region. So all we have to do is try all of the constraint intersection points.
Check each vertex:

<table>
<thead>
<tr>
<th>c</th>
<th>t</th>
<th>P(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1800</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1500</td>
</tr>
<tr>
<td>6/5</td>
<td>3/5</td>
<td>1980</td>
</tr>
</tbody>
</table>

So our market gardener should plant 1.2 acres in cabbages and 0.6 acres in tomatoes. She will then make $1980 in profit.

This graphical procedure is adequate when the optimization problem is simple. For optimization problems which include more variables and constraints this approach becomes impractical.
Degeneracy in Linear Programming

Our example was well-posed, which means there was an unique optimum. It is possible to formulate LP problems which are ill-posed or degenerate. There are three main types of degeneracy which must be avoided in Linear Programming:

1. unbounded solutions,
   → not enough independent constraints.

2. no feasible region,
   → too many or conflicting constraints.

3. non-unique solution,
   → too many solutions,
   → dependance between profit function and some constraints.
1. Unbounded Solution:

As we move up either the $x_1$-axis or the constraint the objective function improves without limit.

In industrial-scale problems with many variables and constraints, this situation can arise when:

→ there are too few constraints.
→ there is linear dependence among the constraints.
2. No Feasible Region:

Any value of x violates some set of constraints.

This situation occurs most often in industrial-scale LP problems when too many constraints are specified and some of them conflict.
Non-Unique Solution:

Profit contour is parallel to constraint.
2 Simplex Method

Our market gardener example had the form:

\[
\min_x -[900 \ 1500]x
\]

subject to:

\[
\begin{bmatrix}
1.5 & 2 \\
20 & 60
\end{bmatrix} x \leq \begin{bmatrix}
3 \\
60
\end{bmatrix}
\]

where: \( x = [\text{acres cabbages} \ \ \text{acres tomatoes}] \)

We need a more systematic approach to solving these problems, particularly when there are many variables and constraints.

→ SIMPLEX method (Dantzig).
→ always move to a vertex which improves the value of the objective function.
**SIMPLEX Algorithm**

1. Convert problem into standard form with:
   → positive right-hand sides,
   → lower bounds of zero.

2. Introduce slack/surplus variables:
   → change all inequality constraints to equality constraints.

3. Define an initial feasible basis:
   → choose a starting set of variable values which satisfy all of the constraints.

4. Determine a new basis which improves the objective function:
   → select a new vertex with a better value of $P(x)$.

5. Transform the equations:
   → perform row reduction on the equation set.

6. Repeat steps 4 & 5 until no more improvement in the objective function is possible.
The problem was written:

\[
\max_{c,t} \quad 900c + 1500t
\]

subject to:

\[
\begin{align*}
1.5c + 2t & \leq 3 \\
20c + 60t & \leq 60
\end{align*}
\]

1. in standard form, the problem is

\[
\min_x \quad -[900 \quad 1500]x
\]

subject to:

\[
\begin{bmatrix}
1.5 & 2 \\
20 & 60
\end{bmatrix} x \leq \begin{bmatrix} 3 \\ 60 \end{bmatrix}
\]

where: \( x = [c \quad t]^T \).
2. Introduce slack variables (or convert all inequality constraints to equality constraints):

\[
\begin{align*}
\text{min} & \quad -900c - 1500t \\
\text{subject to:} & \\
1.5c + 2t + s_1 &= 3 \\
20c + 60t + s_2 &= 60 \\
c, t, s_1, s_2 &\geq 0
\end{align*}
\]

or in matrix form:

\[
\begin{align*}
\text{min} & \quad -[900 \quad 1500 \quad 0 \quad 0]x \\
\text{subject to:} & \\
\begin{bmatrix} 1.5 & 2 & 1 & 0 \\ 20 & 60 & 0 & 1 \end{bmatrix} x &= \begin{bmatrix} 3 \\ 60 \end{bmatrix}
\end{align*}
\]

where: \( x = [c \quad t \quad s_1 \quad s_2]^T \geq 0. \)
3. Define an initial basis and set up tableau:

→ choose the slacks as initial basic variables,

→ this is equivalent to starting at the origin.

the initial tableau is:

\[
\begin{array}{c|cc|cc|c}
\hline
 & c & t & s_1 & s_2 & b \\
\hline
 s_1 & 1.5 & 2.0 & 1.0 & 0.0 & 3.0 \\
s_2 & 20.0 & 60.0 & 0.0 & 1.0 & 60.0 \\
 \hline
 -900 & -1500 & 0.0 & 0.0 & 0.0 \\
\hline
\end{array}
\]
4. Determine the new basis:

→ examine the objective function coefficients and choose a variable with a negative weight. (you want to decrease the objective function because you are minimizing. Usually we will choose the most negative weight, but this is not necessary).

→ this variable will be brought into the basis.

→ divide each element of b by the corresponding constraint coefficient of the new basic variable.

→ the variable which will be removed from the basis is in the pivot row (given by the smallest positive ratio of $b_i/a_{ij}$).
## SIMPLEX METHOD

<table>
<thead>
<tr>
<th></th>
<th>( c )</th>
<th>( t )</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_1 )</td>
<td>1.5</td>
<td>2.0</td>
<td>1.0</td>
<td>0.0</td>
<td>3.0</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>20.0</td>
<td>60.0</td>
<td>0.0</td>
<td>1.0</td>
<td>60.0</td>
</tr>
<tr>
<td></td>
<td>-900</td>
<td>-1500</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

New non-basic variable:

- Pivot \( t \)
- Most negative coefficient: bring \( t \) into the basis

\[\frac{b_1}{a_{12}} = \frac{3}{2}\]

\[\frac{b_2}{a_{22}} = 1\]
5. Transform the constraint equations:

→ perform row reduction on the constraint equations to make the pivot element 1 and all other elements of the pivot column 0.

\[
\begin{array}{c|c|c|c|c|c}
 & c & t & s_1 & s_2 & b \\ 
\hline
s_1 & 5/6 & 0.0 & 1.0 & -1/30 & 1.0 \\
t & 1/3 & 1.0 & 0.0 & 1/60 & 1.0 \\
\hline
-400 & 0.0 & 0.0 & 25.0 & 1500 \\
\end{array}
\]

i) new row \#2 = row \#2/60
ii) new row \#1 = row \#1 -2*new row \#2
iii) new row \#3 = row \#3 + 1500*new row \#2
6. Repeat steps 4&5 until no improvement is possible:

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>t</th>
<th>s₁</th>
<th>s₂</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>5/6</td>
<td>0.0</td>
<td>1.0</td>
<td>-1/30</td>
<td>1.0</td>
</tr>
<tr>
<td>t</td>
<td>1/3</td>
<td>1.0</td>
<td>0.0</td>
<td>1/60</td>
<td>1.0</td>
</tr>
<tr>
<td>-400</td>
<td>0.0</td>
<td>0.0</td>
<td>25.0</td>
<td>1500</td>
<td></td>
</tr>
</tbody>
</table>
the new tableau is:

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>t</th>
<th>s₁</th>
<th>s₂</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1.0</td>
<td>0.0</td>
<td>6/5</td>
<td>-1/25</td>
<td>6/5</td>
</tr>
<tr>
<td>t</td>
<td>0.0</td>
<td>1.0</td>
<td>-1/3</td>
<td>3/100</td>
<td>3/5</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>480</td>
<td>9.0</td>
<td>1980</td>
</tr>
</tbody>
</table>

no further improvements are possible, since there are no more negative coefficients in the bottom row of the tableau.
The final tableau is:

- optimal basis contains both cabbages and tomatoes \((c, t)\).
- optimal acres of cabbages is \(6/5\) and of tomatoes is \(3/5\).
- the maximum profit is 1980.
Recall that we could graph the market garden problem:

We can track how the SIMPLEX algorithm moved through the feasible region. The SIMPLEX algorithm started at the origin. Then moved along the tomato axis (this was equivalent to introducing tomatoes into the basis) to the fertilizer constraint. The algorithm then moved up the fertilizer constraint until it found the intersection with the trucking constraint, which is the optimal solution.
Some other considerations

1. Negative lower bounds on variables:
   - SIMPLEX method requires all variables to have non-negative lower bounds (i.e. \( x \geq 0 \)).
   - problems can have variables for which negative values are physically meaningful (e.g. intermediate cross-flows between parallel process units).
   - introduce artificial variables.

If we have bounds on the variable \( x_1 \) such that: \( a \leq x \leq b \), then we introduce two new artificial variables (e.g. \( x_5, x_6 \)) where:

\[
x_1 = x_6 - x_5
\]

and we can require \( x_5 \geq 0 \) and \( x_6 \geq 0 \) while satisfying the bounds on \( x_1 \).
2. Mixed constraints :

- until this point we have always been able to formulate our constraints sets using only $\leq$ constraints (with positive right-hand sides).

- how can we handle constraint sets that also include constraints (with positive right-hand sides) ?

- introduce surplus variables.

Suppose we have the constraint :

$$[a_{i1}...a_{in}] x \geq b_i$$

Then we can introduce a surplus variable $s_i$ to form an equality constraint by setting :

$$a_{i1}x_1 + ... + a_{in}x_n - s_i = b_i$$
We have a problem if we start with an initial feasible basis such that \( x = 0 \) (i.e. starting at the origin). Then:

\[
s_i = -b_i
\]

which violates the non-negativity constraint on all variables (including slack / surplus variables).

We need a way of finding a feasible starting point for these situations.

3. Origin not in the feasible region

The SIMPLEX algorithm requires a feasible starting point. We have been starting at the origin, but in this case the origin is not in the feasible region. There are several different approaches:

\[\rightarrow \textbf{Big M method} \ (\text{Murty, 1983}).\]
**Big M Method**

Consider the minimization problem with mixed constraints:

\[
\min \quad x_1 + 2x_2 \\
\text{subject to:}
\]

\[
3x_1 + 4x_2 \geq 5 \\
x_1 + 2x_3 = 3 \\
x_1 + x_2 \leq 4
\]

1. As well as the usual slack / surplus variables, introduce a separate artificial variable \(a_i\) into each equality and \(\geq\) constraint:

\[
3x_1 + 4x_2 - s_1 + a_1 = 5 \\
x_1 + 2x_3 + a_2 = 3 \\
x_1 + x_2 + s_2 = 4
\]
2. Augment the objective function by penalizing non-zero values of the artificial variables:

\[ P(x) = c^T x + \sum_{i=1}^{m} Ma_i \]

Choose a large value for \( M \) (say 10 times the magnitude of the largest coefficient in the objective function). For the example minimization problem:

\[ P(x) = x_1 + 2x_2 + 20a_1 + 20a_2 \]
3. Set up the tableau as usual. The initial basis will include all of the artificial variables and the required slack variables:

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>3.0</td>
<td>4.0</td>
<td>0.0</td>
<td>-1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>5.0</td>
</tr>
<tr>
<td>$a_2$</td>
<td>1.0</td>
<td>0.0</td>
<td>2.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>3.0</td>
</tr>
<tr>
<td>$s_2$</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>2.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>20.0</td>
<td>20.0</td>
<td>0</td>
</tr>
</tbody>
</table>
4. Phase 1: modify the objective row in the tableau to reduce the coefficients of the artificial variables to zero:

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>3.0</td>
<td>4.0</td>
<td>0.0</td>
<td>-1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>5.0</td>
</tr>
<tr>
<td>$a_2$</td>
<td>1.0</td>
<td>0.0</td>
<td>2.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>3.0</td>
</tr>
<tr>
<td>$s_2$</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>-59</td>
<td>-78</td>
<td>0.0</td>
<td>20.0</td>
<td>0.0</td>
<td>0.0</td>
<td>20.0</td>
<td>-100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>3.0</td>
<td>4.0</td>
<td>0.0</td>
<td>-1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>5.0</td>
</tr>
<tr>
<td>$a_2$</td>
<td>1.0</td>
<td>0.0</td>
<td>2.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>3.0</td>
</tr>
<tr>
<td>$s_2$</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>-79</td>
<td>-78</td>
<td>-40</td>
<td>20</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-160</td>
</tr>
</tbody>
</table>
5. Phase 2: Solve the problem as before starting from the final tableau from Phase 1.

<table>
<thead>
<tr>
<th></th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>3.0</td>
<td>4.0</td>
<td>0.0</td>
<td>-1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>5.0</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>1.0</td>
<td>0.0</td>
<td>2.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>3.0</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>-79</td>
<td>-78</td>
<td>-40</td>
<td>20</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-160</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>1.0</td>
<td>4/3</td>
<td>0.0</td>
<td>-1/3</td>
<td>0.0</td>
<td>1/3</td>
<td>0.0</td>
<td>5/3</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>0.0</td>
<td>-4/3</td>
<td>2.0</td>
<td>1/3</td>
<td>0.0</td>
<td>-1/3</td>
<td>1.0</td>
<td>4/3</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>0.0</td>
<td>-1/3</td>
<td>0.0</td>
<td>1/3</td>
<td>1.0</td>
<td>-1/3</td>
<td>0.0</td>
<td>7/4</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>82/3</td>
<td>-40</td>
<td>-19/3</td>
<td>0.0</td>
<td>79/3</td>
<td>0.0</td>
<td>-85/3</td>
</tr>
</tbody>
</table>
The final tableau is:

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1.0</td>
<td>4/3</td>
<td>0.0</td>
<td>-1/3</td>
<td>0.0</td>
<td>1/3</td>
<td>0.0</td>
<td>5/3</td>
</tr>
<tr>
<td>$x_3$</td>
<td>0.0</td>
<td>-2/3</td>
<td>1.0</td>
<td>1/6</td>
<td>0.0</td>
<td>-1/6</td>
<td>1/2</td>
<td>2/3</td>
</tr>
<tr>
<td>$s_2$</td>
<td>0.0</td>
<td>-1/3</td>
<td>0.0</td>
<td>1/3</td>
<td>1.0</td>
<td>-1/3</td>
<td>0.0</td>
<td>7/3</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>2/3</td>
<td>0.0</td>
<td>1/3</td>
<td>0.0</td>
<td>59/3</td>
<td>20</td>
<td>-5/3</td>
</tr>
</tbody>
</table>
Simplex Algorithm (Matrix Form)

To develop the matrix form of the Linear Programming problem:

\[
\min_x c^T x
\]

subject to:

\[
Ax \leq b \\
x \geq 0
\]

we introduce the slack variables and partition \( x, A \) and \( c \) as follows:

\[
x = \begin{bmatrix}
    x_B \\
    \vdots \\
    x_N
\end{bmatrix}
\]

\[
A = \begin{bmatrix}
    B & N
\end{bmatrix}
\]
Then, the Linear Programming problem becomes:

$$\min_x c_B^T x_B + c_N^T x_N$$

subject to:

$$B x_B + N x_N = b$$

$$x_B, x_N \geq 0$$

Feasible values of the basic variables ($x_B$) can be defined in terms of the values for non-basic variables ($x_N$):

$$x_B = B^{-1}[b - N x_N]$$
The value of the objective function is given by:

\[ P(x) = C_B^T B^{-1} [b - N x_N] + c_N^T x_N \]

or

\[ P(x) = C_B^T B^{-1} b + [c_N^T - c_B^T B^{-1} N] x_N \]

Then, the tableau we used to solve these problems can be represented as:

<table>
<thead>
<tr>
<th></th>
<th>( x_B^T )</th>
<th>( x_N^T )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_B )</td>
<td>( I )</td>
<td>( B^{-1} N )</td>
<td>( B^{-1} b )</td>
</tr>
<tr>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( c_B^T B^{-1} b )</td>
</tr>
</tbody>
</table>
The Simplex Algorithm is:

1. form the $B$ and $N$ matrices. Calculate $B^{-1}$.

2. calculate the shadow prices of the non-basic variables $(x_N)$
   
   $$- [c_N^T - c_B^T B^{-1} N]$$

3. calculate $B^{-1} N$ and $B^{-1} b$.

4. find the pivot element by performing the ratio test using the column corresponding to the most negative shadow price.

5. the pivot column corresponds to the new basic variable and the pivot row corresponds to the new non-basic variable. Modify the $B$ and $N$ matrices accordingly. Calculate $B^{-1}$.

6. repeat steps 2 through 5 until there are no negative
This method is computationally inefficient because you must calculate a complete inverse of the matrix $B$ at each iteration of the SIMPLEX algorithm. There are several variations on the Revised SIMPLEX algorithm which attempt to minimize the computations and memory requirements of the method. Examples of these can be found in:

- Chvatal
- Edgar & Himmelblau (references in 7.7)
- Fletcher
- Gill, Murray and Wright
3 Duality & Linear Programming

Many of the traditional advanced topics in LP analysis and research are based on the so-called method of Lagrange. This approach to constrained optimization:

→ originated from the study of orbital motion.
→ has proven extremely useful in the development of,

- optimality conditions for constrained problems,
- duality theory,
- optimization algorithms,
- sensitivity analysis.
Consider the general Linear programming problem:

$$\min_x c^T x$$

subject to:

$$Ax \geq b$$
$$x \geq 0$$

This problem is often called the Primal problem to indicate that it is defined in terms of the Primal variables ($x$).

You form the Lagrangian of this optimization problem as follows:

$$L(x, \lambda) = c^T x - \lambda^T [Ax - b]$$

Notice that the Lagrangian is a scalar function of two sets of variables: the Primal variables $x$, and the Dual variables (or Lagrange multipliers) $\lambda$. Up until now we have been calling the Lagrange multipliers $\lambda$ the shadow prices.
We can develop the Dual problem first be re-writing the Lagrangian as:

$$L^T(x, \lambda) = x^T c - [x^T A^T - b^T] \lambda$$

which can be re-arranged to yield:

$$L^T(\lambda, x) = b^T \lambda - x^T [A^T \lambda - c]$$

This Lagrangian looks very similar to the previous one except that the Lagrange multipliers $\lambda$ and problem variables $x$ have switched places. In fact this re-arranged form is the Lagrangian for the maximization problem:

$$\max_{\lambda} \quad b^T \lambda$$

subject to:

$$A^T \lambda \leq c$$

$$\lambda \geq 0$$

This formulation is often called the Dual problem to indicate that it is defined in terms of the Dual variables $\lambda$. 
It is worth noting that in our market gardener example, if we calculate the optimum value of the objective function using the Primal variables $x^*$:

\[
P(x^*)|_{\text{primal}} = c^T x^* = 900 \left( \frac{9}{5} \right) + 1500 \left( \frac{3}{5} \right) = 1980
\]

and using the Dual variables $\lambda^*$:

\[
P(\lambda^*)|_{\text{dual}} = b^T \lambda^* = 3 \left( 480 \right) + 60 \left( 9 \right) = 1980
\]

we get the same result. There is a simple explanation for this that we will examine later.
Besides being of theoretical interest, the Dual problem formulation can have practical advantages in terms of ease of solution. Consider that the Primal problem was formulated as:

\[
\min_{x} \quad c^T x \\
\text{subject to:} \\
Ax \geq b \\
x \geq 0
\]

As we saw earlier, problems with inequality constraints of this form often present some difficulty in determining an initial feasible starting point. They usually require a Phase 1 starting procedure such as the Big M method.
The Dual problem had the form:

$$\max_{\lambda} \quad b^T \lambda$$

subject to:

$$A^T \lambda \quad \leq \quad c$$
$$\lambda \quad \geq \quad 0$$

Problems such as these are usually easy to start since $\lambda = 0$ is a feasible point. Thus no Phase 1 starting procedure is required. As a result you may want to consider solving the Dual problem, when the origin is not a feasible starting point for the Primal problem.
Rules for forming the dual

<table>
<thead>
<tr>
<th>Primal</th>
<th>Dual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality constraint</td>
<td>Free variable</td>
</tr>
<tr>
<td>Inequality constraint</td>
<td>Nonnegative variable</td>
</tr>
<tr>
<td>Free variable</td>
<td>Equality constraint</td>
</tr>
<tr>
<td>Nonnegative variable</td>
<td>Inequality constraint</td>
</tr>
</tbody>
</table>
4 Optimality Conditions

Consider the linear programming problem:

\[
\min_x \quad c^T x \\
\text{subject to :}
\begin{bmatrix}
M \\
N
\end{bmatrix} x \geq 
\begin{bmatrix}
b_M \\
b_N
\end{bmatrix} \quad \leftarrow \text{active}
\]

- if equality constraints are present, they can be used to eliminate some of the elements of \( x \), thereby reducing the dimensionality of the optimization problem.

- rows of the coefficient matrix (M) are linearly independent.
In our market gardener example, the problem looked like:

Form the Lagrangian:

\[ L(x, \lambda) = c^T x - \lambda_M [Mx - b_M] - \lambda_N [Nx - b_N] \]

At the optimum, we know that the Lagrange multipliers (shadow prices) for the inactive inequality constraints are zero (i.e. \( \lambda_N = 0 \)). Also, since at the optimum the active inequality constraints are exactly satisfied:

\[ Mx^* - b_M = 0 \]
Then, notice that at the optimum:

\[ L(x^*, \lambda^*) = P(x^*) = c^T x^* \]

At the optimum, we have seen that the shadow prices for the active inequality constraints must all be non-negative (i.e. \( \lambda_M \geq 0 \)). These multiplier values told us how much the optimum value of the objective function would increase if the associated constraint was moved into the feasible region, for a minimization problem.

Finally, the optimum \((x^*, \lambda^*)\) must be a stationary point of the Lagrangian:

\[ \nabla_x L(x^*, \lambda^*) = \nabla_x P(x^*) - (\lambda^*_M)^T g_M(x^*) = c^T - (\lambda^*_M)^T M = 0 \]
\[ \nabla_\lambda L(x^*, \lambda^*) = (g_M(x^*))^T = (M x^* - b_M)^T = 0 \]
Thus, necessary and sufficient conditions for an optimum of a Linear Programming problem are:

- the rows of the active set matrix \((M)\) must be linearly independent,
- the active set are exactly satisfied at the optimum point \(x^*\):
  \[ Mx^* - b_M = 0 \]
- the Lagrange multipliers for the inequality constraints are:
  \[ \lambda_N^* = 0 \text{ and } \lambda_M^* \geq 0 \]
  this is sometimes expressed as:
  \[ (\lambda_M^*)^T[Mx^* - b_M] + (\lambda_N^*)^T[Nx^* - b_N] = 0 \]
- the optimum point \((x^*, \lambda^*)\) is a stationary point of the Lagrangian:
  \[ \nabla_x L(x^*, \lambda^*) = 0 \]
5 Applications

Quadratic Programming Using LP

Consider an optimization problem of the form:

$$\min_x \frac{1}{2} x^T H x + c^T x$$

subject to

$$A x \geq b$$
$$x \geq 0$$

where $H$ is a symmetric, positive definite matrix.

This is a Quadratic Programming (QP) problem. Problems such as these are encountered in areas such as parameter estimation (regression), process control and so forth.
Recall that the Lagrangian for this problem would be formed as:

\[ L(x, \lambda, m) = \frac{1}{2} x^T H x + c^T x - \lambda^T (Ax - b) - m^T x \]

where \( \lambda \) and \( m \) are the Lagrange multipliers for the constraints.

The optimality conditions for this problem are:

\[ \nabla_x L(x, \lambda, m) = x^T H + c^T - \lambda^T A - m^T = 0 \]

\[ Ax - b \geq 0 \]

\[ x, \lambda, m \geq 0 \]

\[ \lambda^T (Ax - b) - m^T x = 0 \]
It can be shown that the solution to the QP problem is also a solution of the following LP problem:

\[
\min_x \frac{1}{2} \left[ c^T x + b^T (Ax - b) \right]
\]

subject to:

\[
Hx + c - A^T \lambda - m = 0
\]

\[
\lambda^T (Ax - b) - m^T x = 0
\]

\[
Ax - b \geq 0
\]

\[
x, \lambda, m \geq 0
\]

This LP problem may prove easier to solve than the original QP problem.
Successive Linear Programming

Consider the general nonlinear optimization problem:

\[
\min_x P(x)
\]

subject to:

\[
h(x) = 0
\]
\[
g(x) \geq 0
\]

If we linearize the objective function and all of the constraints about some point \(x_k\), the original nonlinear problem may be approximated (locally) by:

\[
\min_x P(x_k) + \nabla_x P|_{x_k}(x - x_k)
\]

subject to

\[
h(x_{k+1}) + \nabla_x h|_{x_k}(x - x_k) = 0
\]
\[
g(x_{k+1}) + \nabla_x g|_{x_k}(x - x_k) = 0
\]
Since $x_k$ is known, the problem is linear in the variables $x$ and can be solved using LP techniques.

The solution to the approximate problem is $x^*$ and will often not be a feasible point of the original nonlinear problem (i.e. the point $x^*$ may not satisfy the original nonlinear constraints). There are a variety of techniques to correct the solution $x^*$ to ensure feasibility.

A possibility would be to search in the direction of $x^*$, starting from $x_k$, to find a feasible point:

$$x_{k+1} = x_k + \alpha(x^* - x_k)$$

We will discuss such ideas later with nonlinear programming techniques.
The SLP algorithm is:

1. linearize the nonlinear problem about the current point $x_k$,

2. solve the approximate LP,

3. correct the solution ($x^*$) to ensure feasibility of the new point $x_{k+1}$ (i.e. find a point $x_{k+1}$ in the direction of $x^*$ which satisfies the original nonlinear constraints),

4. repeat steps 1 through 3 until convergence.

The SLP technique has several disadvantages of which slow convergence and infeasible intermediate solutions are the most important. However, the ease of implementation of the method can often compensate for the method’s shortcomings.
6 Sensitivity Analysis

Usually there is some uncertainty associated with the values used in an optimization problem. After we have solved an optimization problem, we are often interested in knowing how the optimal solution will change as specific problem parameters change. This is called by a variety of names, including:

- sensitivity analysis,
- post-optimality analysis,
- parametric programming.

In our Linear Programming problems we have made use of pricing information, but we know such prices can fluctuate. Similarly the demand / availability information we have used is also subject to some fluctuation. Consider that in our market gardener problem, we might be interested in determining how the optimal solution changes when:

- the price of vegetables changes,
- the amount of available fertilizer changes,
- we buy a new truck.
In the market gardener example, we had:

- Changes in the pricing information \( (c^T) \) affects the slope of the objective function contours.

- Changes to the right-hand sides of the inequality constraints \( (b) \), translates the constraints without affecting their slope.

- Changes to the coefficient matrix of the constraints \( (A) \) affects the slopes of the corresponding constraints.
For the general Linear Programming problem:

\[
\min_x \quad c^T x
\]

subject to:

\[
Ax \geq b
\]

we had the Lagrangian:

\[
L(x, \lambda) = P(x) - \lambda^T g(x) = c^T x - \lambda^T [Ax - b]
\]

or in terms of the active and inactive inequality constraints:

\[
L(x, \lambda) = c^T x - \lambda_M^T [Mx - b_M] - \lambda_N^T [Nx - b_N]
\]

Recall that at the optimum:

\[
\nabla_x L(x^*, \lambda^*) = c^T - (\lambda_M^*)^T M = 0
\]

\[
\nabla_\lambda L(x^*, \lambda^*) = [Mx^* - b_M] = 0
\]

\[
(\lambda_N^*)^T = 0
\]
Notice that the first two sets of equations are linear in the variables of interest \((x^*, \lambda^*_M)\) and they can be solved to yield:

\[
\lambda^*_M = (M^T)^{-1} c \quad \text{and} \quad x^* = M^{-1} b_M
\]

Then, as we discussed previously, it follows that:

\[
P(x^*) = c^T x^* = c^T M^{-1} b_M = (\lambda^*_M)^T b_M = b^T_M \lambda^*_M
\]

We now have expressions for the complete solution of our Linear Programming problem in terms of the problem parameters.

In this course we will only consider two types of variation in our nominal optimization problem including:

1. changes in the right-hand side of constraints \((b)\). Such changes occur with variation in the supply / demand of materials, product quality requirements and so forth.

2. pricing \((c)\) changes. The economics used in optimization are rarely known exactly.
For the more difficult problem of determining the effects of uncertainty in the coefficient matrix \( A \) on the optimization results see:


For small (differential) changes in $b_M$, which do not change the active constraint set:

$$\nabla_b P(x^*) = \nabla_b [(\lambda^*)^T b] = (\lambda^*)^T$$

$$\nabla_b x^* = \nabla_b [M^{-1} b_M] = \begin{bmatrix} M^{-1} \\ - - - \\ 0 \end{bmatrix}$$

$$\nabla_b \lambda^* = \nabla_b \begin{bmatrix} M^{-1}c \\ - - - \\ 0 \end{bmatrix} = 0$$

Note that the Lagrange multipliers are not a function of $b_M$ as long as the active constraint set does not change.
For small (differential) changes in $c$, which do not change the active constraint set:

\[ \nabla_c P(x^*) = \nabla_c \left[ c^T x^* \right] = (x^*)^T \]

\[ \nabla_c x^* = \nabla_c \left[ M^{-1} b_M \right] = 0 \]

\[ \nabla_c \lambda^* = \nabla_c \begin{bmatrix} (M^T)^{-1} c \\ - - - \\ 0 \end{bmatrix} = \begin{bmatrix} (M^T)^{-1} \\ - - - \\ 0 \end{bmatrix} \]
Note that $x^*$ is not an explicit function of the vector $c$ and will not change so long as the active set remains fixed. If a change in $c$ causes an element of $\lambda$ to become negative, then $x^*$ will jump to a new vertex.

Also it is worth noting that the optimal values $P(x^*), x^*$ and $\lambda^*$ are all affected by changes in the elements of the constraint coefficient matrix ($M$). This is beyond the scope of the course and the interested student can find some detail in the references previously provided.

There is a simple geometrical interpretation for most of these results. Consider the 2 variable Linear Programming problem:

$$\min_{x_1, x_2} c_1 x_1 + c_2 x_2$$

subject to

$$\begin{bmatrix} g_1(x) \\ g_2(x) \\ \vdots \end{bmatrix} = \begin{bmatrix} a_{11} x_1 + a_{12} x_2 \\ a_{21} x_1 + a_{22} x_2 \\ \vdots \end{bmatrix} = Ax \geq b$$
Note that the set of inequality constraints can be expressed:

\[
Ax = \begin{bmatrix}
M \\
- \\
N
\end{bmatrix} x = \begin{bmatrix}
\nabla x g_1 \\
\nabla x g_2 \\
\vdots
\end{bmatrix} x \geq b
\]

This two variables optimization problem has an optimum at the intersection of the \( g_1 \) and \( g_2 \) constraints, and can be depicted as:
Summary

- The solution of a well-posed LP is uniquely determined by the intersection of the active constraints.

- most LPs are solved by the SIMPLEX method.

- commercial computer codes used the Revised SIMPLEX algorithm (most efficient)

- Optimization studies include:
  * solution to the nominal optimization problem
  * a sensitivity study to determine how uncertainty can affect the solution.
Recall that the SIMPLEX method solved the LP problem by walking around the boundary of the feasible region. The algorithm moved from vertex to vertex on the boundary.

Interior point methods move through the feasible region toward the optimum. This is accomplished by:

1. Converting the Lp problem into an unconstrained problem using Barrier Functions for the constrains at each new point (iterate)
2. Solving the optimality conditions for the resulting unconstrained problem
3. Repeating until converged.

This is a very active research area and some exciting developments have occurred in the last two decades.
Chapter III

Unconstrained Univariate Optimization

- Introduction
- Interval Elimination Methods
- Polynomial Approximation Methods
- Newton’s Method
- Quasi-Newton Methods
1 Introduction

Univariate optimization means optimization of a scalar function of a single variable:

\[ y = P(x) \]

These optimization methods are important for a variety of reasons:

1. there are many instances in engineering when we want to find the optimum of functions such as these (e.g. optimum reactor temperature, etc.),
2. almost all multivariable optimization methods in commercial use today contain a line search step in their algorithm,
3. they are easy to illustrate and many of the fundamental ideas are directly carried over to multivariable optimization.

Of course these discussions will be limited to nonlinear functions, but before discussing optimization methods we need some mathematical background.
Continuity:

In this course we will limit our discussions to continuous functions.

Functions can be continuous but their derivatives may not be.
A function $P(x)$ is continuous at a point $x_0$ iff:

$$P(x_0) \text{ exists and } \lim_{x \to x_0^-} P(x) = \lim_{x \to x_0^+} P(x) = P(x_0)$$

Some discontinuous functions include:

→ price of processing equipment,

→ manpower costs,

→ measurements, ... 

Many of the functions we deal with in engineering are either continuous or can be approximated as continuous functions. However, there is a growing interest in solving optimization problems with discontinuities.
Convexity of Functions

In most optimization problems we require functions which are either convex or concave over some region (preferably globally):

\text{i) for a convex function and } \forall \alpha \in [0, 1],

\[ P(\alpha x + (1 - \alpha)y) \leq \alpha P(x) + (1 - \alpha)P(y) \]

\text{ii) for a concave function and } \forall \alpha \in [0, 1],

\[ P(\alpha x + (1 - \alpha)y) \geq \alpha P(x) + (1 - \alpha)P(y) \]
Convexity is a general idea used in optimization theory which when applied to a function, describes the property of having an optimum. Convexity combines both stationarity and curvature into a single concept. Often both convex and concave functions are described as exhibiting the property of convexity.

These conditions are very inconvenient for testing a specific function, so we tend to test the derivatives of the function at selected points. First derivatives give us a measure of the rate of change of the function. Second derivatives give us a measure of curvature or the rate of change of the second derivatives.
Notice for a convex function:

The first derivatives increase (become more positive) as x increases. Thus, in the region we are discussing, the second derivative is positive. Since the first derivative starts out negative and becomes positive in the region of interest, there is some point where the function no longer decreases in value as x increases and this point is the function minimum.
This is another way of stating that the point $x^*$ is a (local) minimum of the function $P(x)$ iff:

$$P(x^*) \leq P(x), \quad \forall x \in [a, b]$$

If this can be stated as a strict inequality $x^*$ is said to be a unique (local) minimum.

A similar line of argument can be followed for concave functions, with the exception that: the first derivatives decrease (become more negative) with increasing $x$, the second derivative is negative and as result, the optimum is a maximum.

The problem with these ideas is that we can only calculate the derivatives at a point. So looking for an optimum using these fundamental ideas would be infeasible since a very large number points (theoretically all of the points) in the region of interest must be checked.
Necessary and Sufficient Conditions for an Optimum

Recall that for a twice continuously differentiable function $P(x)$, the point $x^*$ is an optimum iff:

\[
\frac{dP}{dx} \bigg|_{x^*} = 0 \quad \text{stationarity}
\]

and

\[
\frac{d^2P}{dx^2} \bigg|_{x^*} > 0 \quad \text{a minimum}
\]

\[
\frac{d^2P}{dx^2} \bigg|_{x^*} < 0 \quad \text{a maximum}
\]

What happens when the second derivative is zero?
Consider the function:

\[ y = x^4 \]

which we know to have a minimum at \( x = 0 \).

at the point \( x = 0 \) :

\[ \left. \frac{dy}{dx} \right|_0 = 4x^3 = 0, \quad \left. \frac{d^2 y}{dx^2} \right|_0 = 12x^2 = 0, \quad \left. \frac{d^3 y}{dx^3} \right|_0 = 24x = 0, \quad \left. \frac{d^4 y}{dx^4} \right|_0 = 24 > 0 \]
We need to add something to our optimality conditions. If at the stationary point $x^*$ the second derivative is zero we must check the higher-order derivatives. Thus, for the first non-zero derivative is odd, that is:

$$\frac{d^n P}{dx^n}|_{x^*} \neq 0 \quad \text{where} \quad n \in \{3, 5, 7, \ldots\}$$

Then $x^*$ is an inflection point. Otherwise, if the first higher-order, non-zero derivative is even:

$$\frac{d^n P}{dx^n}|_{x^*} > 0 \quad \text{a minimum}$$

$$\frac{d^n P}{dx^n}|_{x^*} < 0 \quad \text{a maximum}$$
Some Examples

1. \[
\min_{x} ax^2 + bx + c = 0
\]

stationarity :
\[
\frac{dP}{dx} = 2ax + b \rightarrow x^* = -\frac{b}{2a}
\]

curvature :
\[
\frac{d^2P}{dx^2} = 2a \rightarrow \begin{cases} a > 0 & \text{maximum} \\ a < 0 & \text{minimum} \end{cases}
\]
2. \[ \min_x x^3 - 2x^2 - 5x + 6 \]

stationarity:

\[ \frac{dP}{dx} = 3x^2 - 4x - 5 = 0 \rightarrow x^* = \frac{2 \pm \sqrt{19}}{3} \]

curvature:

\[ \frac{d^2P}{dx^2} |_{x^*} = 6x - 4 = \begin{cases} 2\sqrt{19} > 0 & \text{minimum} \\ -2\sqrt{19} < 0 & \text{maximum} \end{cases} \]

3. \[ \min(x - 1)^n \]
4. A more realistic example. Consider the situation where we have two identical settling ponds in a waste water treatment plant:

These are normally operated at a constant throughput flow of $F$, with an incoming concentration of contaminants $c_i$. The system is designed so that at nominal operating conditions the outgoing concentration $c_o$ is well below the required levels. However, we know that twice a day (7 am and 7 pm) the incoming concentration of contaminants to this section of the plant dramatically increase above these nominal levels. What we would like to determine is whether the peak outgoing levels will remain within acceptable limits.
What assumptions can we make?

1) well mixed,
2) constant density,
3) constant flowrate.

contaminant balance:

\[
\frac{d\rho V c_o}{dt} = F c_1 - F c_o \quad \text{(second pond)}
\]

\[
\frac{d\rho V c_1}{dt} = F c_i - F c_1 \quad \text{(first pond)}
\]

Let the residence time be \( \tau = \frac{\rho V}{F} \). Then the balances become

\[
\tau \frac{dc_o}{dt} = c_1 - c_o \quad \text{(second pond)}
\]

\[
\tau \frac{dc_1}{dt} = c_i - c_1 \quad \text{(first pond)}
\]
Since we wish to know how changes in $c_i$ will affect the outlet concentration $c_o$. Define perturbation variables around the nominal operating point and take Laplace Transforms:

$$\tau s C_o(s) = C_1(s) - C_o(s) \quad \text{(second pond)}$$
$$\tau s C_1(s) = C_i(s) - C_1(s) \quad \text{(first pond)}$$

Combining and simplifying into a single transfer function:

$$C_o(s) = \frac{1}{(\tau s + 1)^2} C_i(s)$$

If we represent the concentration disturbances as ideal impulses:

$$C_i(s) = k$$

Then, transfer function of the outlet concentration becomes:

$$C_o(s) = \frac{k}{(\tau s + 1)^2}$$
In the time-domain this is:

\[ c_o(t) = kte^{-\frac{t}{\tau}} \]

Since we want to determine:

\[ \max_t c_o(t) = kte^{-\frac{t}{\tau}} \]
stationarity:

\[
\frac{dc_o}{dt} = ke^{-\frac{t}{\tau}} \left(1 - \frac{t}{\tau}\right) = 0 \implies 1 - \frac{t}{\tau} = 0 \implies t = \tau
\]

curvature:

\[
\frac{d^2c_o}{dt^2} = \frac{k}{\tau} \left(\frac{t}{\tau} - 2\right)e^{-\frac{t}{\tau}}_{t=\tau} = -\frac{k}{\tau e} < 0 \implies \text{maximum}(k > 0)
\]

Finally, the maximum outlet concentration is:

\[
c_o(\tau) = k\tau e^{-1} \simeq 0.368\frac{k\rho V}{F}
\]
This example serves to highlight some difficulties with the analytical approach. These include:

→ determining when the first derivative goes to zero. This required the solution of a nonlinear equation which is often as difficult as the solution of the original optimization problem.

→ computation of the appropriate higher-order derivatives.

→ evaluation of the higher-order derivatives at the appropriate points.

We will need more efficient methods for optimization of complex nonlinear functions. Existing numerical methods can be classified into two broad types:

1. those that rely solely on the evaluation of the objective function,
2. those that use derivatives (or approximations of the derivatives) of the objective function.
2 Interval Elimination Methods

These methods work by systematically reducing the size of the interval within which the optimum is believed to exist. Since we are only ever working with an interval, we can never find the optimum exactly. There are many interval elimination algorithms; however, they all consist of the same steps:

1. determine an interval in which the optimum is believed to exist,
2. reduce the size of the interval,
3. check for convergence,
4. repeat steps 2 & 3 until converged.

We will examine only one interval elimination method, called the "Golden Section Search".
Scanning & bracketing

As an example, consider that our objective is to minimize a function \( P(x) \). In order to start the optimization procedure we need to determine the upper and lower bounds for an interval in which we believe an optimum will exist. These could be established by:

- physical limits,
- scanning,
  - choosing a starting point,
  - checking the function initially decreases,
  - finding another point of higher value.
Golden Section Search

Consider that we want to break up the interval, in which we believe the optimum exists, into two regions:

\[
\begin{align*}
&l_1 \quad | \\
&\quad | \\
&L \quad | \\
&a \quad | \\
&\quad | \\
&b \quad |
\end{align*}
\]

where: \( \frac{l_1}{l_2} = \frac{l_2}{L} = r \)

Since \( L = l_1 + l_2, l_2 = rL \) and \( l_1 = r^2L \), then:

\[
L = rL + r^2L
\]

or

\[
r^2 + r - 1 = 0
\]
There are two solutions for this quadratic equation. We are only interested in the positive root:

\[ r = \frac{-1 + \sqrt{5}}{2} \approx 0.61803398 \ldots \quad \text{Golden ratio} \]

Which has the interesting property:

\[ r = \frac{1}{r + 1} \]

We will use this Golden ratio to break up our interval as follows:
By breaking up the interval in this way we will be able to limit ourselves to one simple calculation of an $x_i$ and one objective function evaluation per iteration.

The optimization algorithm is:

1. using the interval limits $a_k$ and $b_k$, determine $x_{1}^{k}$ and $x_{2}^{k}$,

2. evaluate $P(x_{1}^{k})$ and $P(x_{2}^{k})$,

3. eliminate the part of the interval in which the optimum is not located,

4. repeat steps 1 through 3 until the desired accuracy is obtained.

How can we decide which part of the interval to eliminate?
There are only three possible situations. For the minimization case, consider:

- $P(x_1^k) > P(x_2^k)$:
  - Eliminate the interval to the left of $x_1^k$. For a convex function, all values to the right of $x_1^k$ are less than those on the left.

- $P(x_1^k) < P(x_2^k)$:
  - Eliminate the interval to the right of $x_2^k$. For a convex function, all values to the left of $x_2^k$ are less than those on the right.

- $P(x_1^k) = P(x_2^k)$:
  - We know that the optimum is between $x_1^k$ and $x_2^k$, so we could eliminate the intervals to the left of $x_1^k$ and to the right of $x_2^k$. In practice, we eliminate only one region for computational ease.
Graphically the algorithm is:

Three things worth noting:

1. only one new point needs to be evaluated during each iteration,
2. the length of the remaining interval after \( k \) iterations (or accuracy to which you know the optimum value of \( x \)) is:

\[
L^k = r^k L^0 \approx (0.618034)^k L^0
\]

3. the new point for the current iteration is:

\[
x_1^k = a^k + b^k - x_2^k
\]

or:

\[
x_2^k = a^k + b^k - x_1^k
\]
Example

Find the solution to within 5% for

\[
\min_x x^2 - 2x + 1
\]

Guess an initial interval for the optimum as \(0 \leq x^* \leq 2\).

<table>
<thead>
<tr>
<th>(k)</th>
<th>(a^k)</th>
<th>(b^k)</th>
<th>(x_1^k)</th>
<th>(x_2^k)</th>
<th>(P(x_1^k))</th>
<th>(P(x_2^k))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000</td>
<td>2.0000</td>
<td>0.7640</td>
<td>1.2360</td>
<td>5.5696E-02</td>
<td>5.5696E-02</td>
</tr>
<tr>
<td>1</td>
<td>0.7640</td>
<td>2.0000</td>
<td>1.2360</td>
<td>1.5280</td>
<td>5.5696E-02</td>
<td>2.7878E-01</td>
</tr>
<tr>
<td>2</td>
<td>0.7640</td>
<td>1.5280</td>
<td>1.0558</td>
<td>1.2360</td>
<td>3.1158E-03</td>
<td>5.5696E-02</td>
</tr>
<tr>
<td>3</td>
<td>0.7640</td>
<td>1.2360</td>
<td>0.9443</td>
<td>1.0558</td>
<td>3.1025E-03</td>
<td>3.1158E-03</td>
</tr>
<tr>
<td>4</td>
<td>0.7640</td>
<td>1.0558</td>
<td>0.8755</td>
<td>0.9443</td>
<td>1.5507E-02</td>
<td>3.1025E-03</td>
</tr>
<tr>
<td>5</td>
<td>0.8755</td>
<td>1.0558</td>
<td>0.9443</td>
<td>0.9869</td>
<td>3.1025E-03</td>
<td>1.7130E-04</td>
</tr>
</tbody>
</table>
Note that the number of calculation we have performed is:

- \((k + 2) = 8\) evaluations of \(P(x_i)\),
- \((k + 2) = 8\) determinations of a new point \(x_i\).

There are a variety of other interval elimination methods including Interval Bisection, Fibonacci Searches, and so forth. They are all simple to implement and not computationally intensive. They tend to be fairly robust and somewhat slow to converge.
There are a number of polynomial approximation methods, which differ only in the choice of polynomial used to locally approximate the function to be optimized. Polynomials are attractive since they are easily fit to data and the optima of low-order polynomials are simple to calculate. We will only discuss Successive Quadratic Approximation as it illustrates the basic ideas and will carry over into multivariable optimization techniques.

Successive Quadratic Approximation

Suppose we have evaluated some test points of a function as follows:
We could approximate the objective function by fitting a quadratic through the test points.

The main advantage of using a quadratic approximation is that we know that optimum is:

\[ x^* = -\frac{b}{2a} \]
The first step is to calculate the approximation constants \((a, b, c)\) in the quadratic from the available points \((x_1, x_2, x_3)\). Using these three evaluation points we can write:

\[
P(x_1) = ax_1^2 + bx_1 + c \\
P(x_2) = ax_2^2 + bx_2 + c \\
P(x_3) = ax_3^2 + bx_3 + c
\]

We have three equations and three unknowns, so in general the constants \((a, b, c)\) can be uniquely determined. In matrix form the set of equations are written:

\[
\begin{bmatrix}
x_1^2 & x_1 & 1 \\
x_2^2 & x_2 & 1 \\
x_3^2 & x_3 & 1 \\
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c \\
\end{bmatrix}
= 
\begin{bmatrix}
P(x_1) \\
P(x_2) \\
P(x_3) \\
\end{bmatrix}
\]
Which has the solution:

\[
\begin{bmatrix}
  a \\
  b \\
  c
\end{bmatrix}
= 
\begin{bmatrix}
  x_1^2 & x_1 & 1 \\
  x_2^2 & x_2 & 1 \\
  x_3^2 & x_3 & 1
\end{bmatrix}
^{-1}
\begin{bmatrix}
  P(x_1) \\
  P(x_2) \\
  P(x_3)
\end{bmatrix}
\]

Some algebra yields the solution for the minimum as:

\[
\hat{x}^* = \frac{1}{2} \frac{(x_2^2 - x_3^2)P(x_1) + (x_3^2 - x_1^2)P(x_2) + (x_1^2 - x_2^2)P(x_3)}{(x_2 - x_3)P(x_1) + (x_3 - x_1)P(x_2) + (x_1 - x_2)P(x_3)}
\]
Graphically, the situation is:

To perform the next iteration, we need to choose which three points to use next. If you’re doing hand calculations choose the point \((x_i)\) with the smallest value of the objective function \(P(x)\) and the points immediately on either side. Note that \(\hat{x}\) will not necessarily have the smallest objective function value.
Finally, the accuracy to which the optimum is known at any iteration is given by the two points which bracket the point with the smallest objective function value.

The procedure is:

1. choose three points that bracket the optimum,
2. determine a quadratic approximation for the objective function based on these three points.
3. calculate the optimum of the quadratic approximation (the predicted optimum $\hat{x}^*$),
4. repeat steps 1 to 3 until the desired accuracy is reached.
Example: revisit the wastewater treatment problem with $\tau = 1$.

$$\max_t te^{-t} = \min_t -te^{-t}$$

<table>
<thead>
<tr>
<th>it</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$P(t_1)$</th>
<th>$P(t_2)$</th>
<th>$P(t_3)$</th>
<th>$t^*$</th>
<th>$P(t^*)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5</td>
<td>1.5</td>
<td>2.5</td>
<td>-0.3303</td>
<td>-0.3347</td>
<td>-0.2052</td>
<td>1.1953</td>
<td>-0.3617</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>1.1953</td>
<td>1.5</td>
<td>-0.3303</td>
<td>-0.3617</td>
<td>-0.3347</td>
<td>1.0910</td>
<td>-0.3664</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>1.0910</td>
<td>1.1953</td>
<td>-0.3303</td>
<td>-0.3664</td>
<td>-0.3617</td>
<td>1.0396</td>
<td>-0.3676</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>1.0396</td>
<td>1.0910</td>
<td>-0.3303</td>
<td>-0.3676</td>
<td>-0.3664</td>
<td>1.0185</td>
<td>-0.3678</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>1.0185</td>
<td>1.0396</td>
<td>-0.3303</td>
<td>-0.3678</td>
<td>-0.3676</td>
<td>1.0083</td>
<td>-0.3679</td>
</tr>
</tbody>
</table>
4 Newton’s Method

This is a derivative-based method that uses first and second derivatives to approximate the objective function. The optimum of the objective function is then estimated using this approximation.

Consider the 2nd-order Taylor Series approximations of our objective function:

\[ P(x) \simeq P(x_k) + \frac{dP}{dx} |_{x_k} (x - x_k) + \frac{1}{2} \frac{d^2P}{dx^2} |_{x_k} (x - x_k)^2 \]

Then we could approximate the first derivative of the objective function as:

\[ \frac{dP}{dx} \simeq \frac{dP}{dx} |_{x_k} + \frac{d^2P}{dx^2} |_{x_k} (x - x_k) \]
Since we require that the first derivative of the objective function to be zero at the optimum, then:

\[
\frac{dP}{dx} \approx \frac{dP}{dx}\bigg|_{x_k} + \frac{d^2P}{dx^2}\bigg|_{x_k} (x - x_k) = 0
\]

A little algebra will allow us to estimate what the optimum value of \( x \) is:

\[
\hat{x}^* = x_{k+1} = x_k - \left[ \frac{dP}{dx}\bigg|_{x_k} \right] \left[ \frac{d^2P}{dx^2}\bigg|_{x_k} \right]
\]

This is called a Newton step and we can iterate on this equation to find the optimum value of \( x \).
Graphically, this is:

Newton’s method attempts to find the stationary point of a function by successive approximation using the first and second derivatives of the original objective function.
Example: revisit the wastewater treatment problem with $\tau = 1$.

$$\max_t te^{-t} = \min_t -te^{-t}$$

<table>
<thead>
<tr>
<th>it</th>
<th>$t^k$</th>
<th>$P'(t^k)$</th>
<th>$P''(t^k)$</th>
<th>$P(t^k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>1.000</td>
<td>-2.000</td>
<td>-0.000</td>
</tr>
<tr>
<td>1</td>
<td>0.500</td>
<td>0.3303</td>
<td>-0.9098</td>
<td>-0.3303</td>
</tr>
<tr>
<td>2</td>
<td>0.8333</td>
<td>0.0724</td>
<td>-0.5070</td>
<td>-0.3622</td>
</tr>
<tr>
<td>3</td>
<td>0.9762</td>
<td>0.0090</td>
<td>-0.3857</td>
<td>-0.3678</td>
</tr>
<tr>
<td>4</td>
<td>0.9994</td>
<td>0.0002</td>
<td>-0.3683</td>
<td>-0.3679</td>
</tr>
<tr>
<td>5</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In summary, Newton’s Method:

1. shows relatively fast convergence (finds the optimum of a quadratic in a single step),
2. requires first and second derivatives of the function,
3. has a computational load of first and second derivative evaluations, and a Newton step calculation per iteration,
4. may oscillate or not converge when there many local optima or stationary points.
5 Quasi-Newton Methods

A major drawback of Newton’s method is that it requires us to have analytically determined both the first and second derivatives of our objective function. Often this is considered onerous, particularly in the case of the second derivative. The large family of optimization algorithms that use finite difference approximations of the derivatives are called ”Quasi-Newton” methods.

There are a variety of ways first derivatives can be approximated using finite differences:

\[ P'(x) \approx \frac{P(x + \Delta x) - P(x - \Delta x)}{2\Delta x} \]

\[ P'(x) \approx \frac{P(x + \Delta x) - P(x)}{\Delta x} \]

\[ P'(x) \approx \frac{P(x) - P(x - \Delta x)}{\Delta x} \]
Approximations for the second derivatives can be determined in terms of either the first derivatives or the original objective function:

\[ P''(x) \approx \frac{P'(x + \Delta x) - P'(x - \Delta x)}{2\Delta x} \]

\[ P''(x) \approx \frac{P'(x + \Delta x) - P'(x)}{\Delta x} \]

\[ P''(x) \approx \frac{P'(x) - P'(x - \Delta x)}{\Delta x} \]

\[ P''(x) \approx \frac{P(x + \Delta x) - 2P(x) + P(x - \Delta x)}{\Delta x^2} \]

\[ P''(x) \approx \frac{P(x + 2\Delta x) - 2P(x + \Delta x) + P(x)}{\Delta x^2} \]

\[ P''(x) \approx \frac{P(x) - 2P(x - \Delta x) + P(x - 2\Delta x)}{\Delta x^2} \]
The major differences within the Quasi-Newton family of
algorithms arises from: the number of function and / or derivatives
that must be evaluated, the speed of converge and the stability of
the algorithm.
Regula Falsi

This method approximates the second derivative as:

$$\frac{d^2 P}{dx^2} \approx \frac{p'(x_q) - P'(x_p)}{x_q - x_p}$$

where $x_p$ and $x_q$ are chosen so that $P'(x_q)$ and $P'(x_p)$ have different signs (i.e. the two points bracket the stationary point).

Substituting this approximation into the formula for a Newton step yields:

$$\hat{x}^* = x_{k+1} = x_q - \left[ \left( \frac{dP}{dx} \right)_{x_q} \right] \left[ \left( \frac{dP}{dx} \right)_{x_q} - \left( \frac{dP}{dx} \right)_{x_p} \right]^{-1}$$
The Regula Falsi method iterates on this formula taking care on each iteration to retain two points:

\[ \hat{x}^* \text{ and } x_p \text{ or } x_q \]

depending upon:

- \( \text{sign}[P'() \hat{x}_k^*] = \text{sign}[P'(x_p)] \implies \text{keep } \hat{x}_k^* \text{ and } x_q \)
- \( \text{sign}[P'() \hat{x}_k^*] = \text{sign}[P'(x_q)] \implies \text{keep } \hat{x}_k^* \text{ and } x_p \)
Graphically:
Example: revisit the wastewater treatment problem with $\tau = 1$.

$$\max_t te^{-t} = \min_t -te^{-t}$$

<table>
<thead>
<tr>
<th>it</th>
<th>$x_p$</th>
<th>$x_q$</th>
<th>$x^*_k$</th>
<th>$P'(x_p)$</th>
<th>$P'(x_q)$</th>
<th>$P'(x^*_k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2.000</td>
<td>1.7616</td>
<td>-1</td>
<td>-0.1353</td>
<td>0.1308</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1.7616</td>
<td>1.5578</td>
<td>-1</td>
<td>0.1308</td>
<td>0.1175</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1.5578</td>
<td>1.3940</td>
<td>-1</td>
<td>0.1175</td>
<td>0.0978</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1.3940</td>
<td>1.2699</td>
<td>-1</td>
<td>0.0978</td>
<td>0.0758</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1.2699</td>
<td>1.1804</td>
<td>-1</td>
<td>0.0758</td>
<td>0.00554</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1.1804</td>
<td>1.1184</td>
<td>-1</td>
<td>0.00554</td>
<td>0.0387</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1.1184</td>
<td>1.0768</td>
<td>-1</td>
<td>0.0387</td>
<td>0.0262</td>
</tr>
</tbody>
</table>

Note: computational load is one derivative evaluation, a sign check and a step calculation per iteration.
In summary, Quasi-Newton methods:

1. show slower convergence rates than Newton’s Method,
2. use some finite difference approximations for derivatives of the objective function,
3. may oscillate or not converge when there are many local optima or stationary points.
6 Summary

There are only two basic ideas in unconstrained univariate optimization:

1. direct search (scanning, bracketing, interval elimination, approximation),
2. derivative based methods (optimality conditions, Newton and Quasi-Newton methods).

The direct search methods work directly on the objective function to successively determine better values of $x$. The derivative based methods try to find the stationary point of the objective function, usually by some iterative procedure.
Choosing which method to use is always a trade-off between:

1. ease of implementation,
2. convergence speed,
3. robustness,
4. computational load (number of function evaluations, derivative evaluations / approximations, etc.).

There are no clear answers as to which univariate method is better. The best choice is problem dependent:

- for nearly quadratic functions, the Newton / Quasi-Newton methods are generally a good choice,
- for very flat functions, interval elimination methods can be very effective.
When the univariate search is buried in a multivariate code, the best choice of univariate search will depend upon the properties and requirements of the multivariate search method.

Finally remember that to solve an unconstrained univariate optimization problem you must:

1. have a function (and derivatives) you can evaluate,
2. choose an optimization method appropriate for the problem,
3. choose a starting point (or interval),
4. select a termination criteria (or accuracy).
Chapter IV

Unconstrained Multivariate Optimization

- Introduction
- Sequential Univariate Searches
- Gradient-Based Methods
- Newton’s Method
- Quasi-Newton Methods
1 Introduction

Multivariate optimization means optimization of a scalar function of several variables:

\[ y = P(x) \]

and has the general form:

\[ \min_x P(x) \]

where \( P(x) \) is a nonlinear scalar-valued function of the vector variable \( x \).
Background

Before we discuss optimization methods, we need to talk about how to characterize nonlinear, multivariable functions such as \( P(x) \).

Consider the 2nd order Taylor series expansion about the point \( x_0 \):

\[
P(x_0) \simeq P(x_0) + \nabla_x P_{|x_0} (x - x_0) + \frac{1}{2} (x - x_0)^T \nabla^2 x P_{|x_0} (x - x_0)
\]

If we let:

\[
a = P(x_0) - \nabla_x P_{|x_0} x_0 + \frac{1}{2} x_0^T \nabla^2 x P_{|x_0} x_0
\]

\[
b^T = \nabla_x P_{|x_0} - x_0^T \nabla^2 x P_{|x_0}
\]

\[
H = \nabla^2 x P_{|x_0}
\]
Then we can re-write the Taylor series expansion as a quadratic approximation for $P(x)$:

$$P(x) = a + b^T x + \frac{1}{2} x^T H x$$

and the derivatives are:

$$\nabla_x P(x) = b^T + x^T H$$
$$\nabla^2_x P(x) = H$$

We can describe some of the local geometric properties of $P(x)$ using its gradient and Hessian. In fact there are only a few possibilities for the local geometry, which can easily be differentiated by the eigenvalues of the Hessian $(H)$.

Recall that the eigenvalues of a square matrix $(H)$ are computed by finding all of the roots $(\lambda)$ of its characteristic equation:

$$|\lambda I - H| = 0$$
The possible geometries are:

1. if $\lambda_i < 0$ ($i = 1, 2, \ldots, n$), the Hessian is said to be negative definite. This object has a unique maximum and is what we commonly refer to as a hill (in three dimensions).
2. if $\lambda_i > 0$ ($i = 1, 2, \ldots, n$), the Hessian is said to be positive definite. This object has a unique minimum and is what we commonly refer to as a valley (in three dimensions).
3. if $\lambda_i < 0$ ($i = 1, 2, \ldots, m$) and $\lambda_i > 0$ ($i = m + 1, \ldots, n$), the Hessian is said to be indefinite. This object has neither a unique minimum or maximum and is what we commonly refer to as a saddle (in three dimensions).
4. if $\lambda_i < 0$ ($i = 1, 2, \ldots, m$) and $\lambda_i = 0$ ($i = m + 1, \ldots, n$), the Hessian is said to be negative semi-definite. This does not have a unique maximum, and is what we commonly refer to as a ridge (in three dimensions).
5. if $\lambda_i > 0$ ($i = 1, 2, \ldots, m$) and $\lambda_i = 0$ ($i = m + 1, \ldots, n$) the Hessian is said to be positive semi-definite. This object does not have a unique minimum and is what we commonly refer to as a trough (in three dimensions).
A well posed problem has a unique optimum, so we will limit our discussions to either problems with positive definite Hessians (for minimization) or negative definite Hessians (for maximization).

Further, we would prefer to choose units for our decision variables \( x \) so that the eigenvalues of the Hessian all have approximately the same magnitude. This will scale the problem so that the profit contours are concentric circles and will condition our optimization calculations.
Necessary and Sufficient Conditions

For a twice continuously differentiable scalar function $P(x)$, a point $x^*$ is an optimum if:

$$\nabla_x P|_{x^*} = 0$$

and:

$$\nabla^2_x P|_{x^*} \text{ is positive definite (a minimum)}$$

$$\nabla^2_x P|_{x^*} \text{ is negative definite (a maximum)}$$

We can use these conditions directly, but it usually involves solving a set of simultaneous nonlinear equations (which is usually just as tough as the original optimization problem).
Consider:

\[ P(x) = x^T A x e^{x^T A x} \]

Then:

\[ \nabla_x P = 2(x^T A) e^{x^T A x} + x^T A x (x^T A) e^{x^T A x} \]
\[ = 2(x^T A) e^{x^T A x} (1 + x^T A x) \]

and stationarity of the gradient requires that:

\[ \nabla_x P = 2(x^T A) e^{x^T A x} (1 + x^T A x) = 0 \]

This is a set of very nonlinear equations in the variables \( x \).
Example

Consider the scalar function:

\[ P(x) = 3 + x_1 + 2x_2 + 4x_1x_2 + x_1^2 + x_2^2 \]

or:

\[ P(x) = 3 + [1 \ 2] x + x^T \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} x \]

where \( x = [x_1 \ x_2]^T \).

Stationarity of the gradient requires:

\[ \nabla_x P = [1 \ 2] + 2x^T \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = 0 \]

\[ x = -\frac{1}{2} \left[ \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \right]^{-1} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0 \end{bmatrix} \]
Check the Hessian to classify the type of stationarity point:

\[ \nabla^2 P = 2 \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix} \]

The eigenvalues of the Hessian are:

\[ \left| \lambda I - \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix} \right| = \left| \begin{array}{cc} \lambda - 2 & -4 \\ -4 & \lambda - 2 \end{array} \right| = (\lambda - 2)^2 - 16 = 0 \]

\[ \lambda_1 = 6 \text{ and } \lambda_2 = -2 \]

Thus the Hessian is indefinite and the stationary point is a saddle point. This is a trivial example, but it highlights the general procedure for direct use of the optimality conditions.
Like the univariate search methods we studied earlier, multivariate optimization methods can be separated into two groups:

i) those which depend solely on function evaluations,

ii) those which use derivative information (either analytical derivatives or numerical approximations).

Regardless of which method you choose to solve a multivariate optimization problem, the general procedure will be:

1) select a starting point(s),
2) choose a search direction,
3) minimize in chosen search direction,
4) repeat steps 2 & 3 until converged.
Also, successful solution of an optimization problem will require specification of a convergence criterion. Some possibilities include:

\[ \| x_{k+1} - x_k \| \leq \gamma \]

\[ \| P(x_{k+1}) - P(x_k) \| \leq \delta \]

\[ \| \nabla_x P|_{x_k} \| \leq \epsilon \]
2 Sequential Univariate Searches

Perhaps the simplest multivariable search technique to implement would be a system of sequential univariate searches along some fixed set of directions. Consider the two dimensional case, where the chosen search directions are parallel to the coordinate axes:

\[ x_1 \]

\[ x_2 \]
In this algorithm, you:

1. select a starting point $x_0$,

2. select a coordinate direction (e.g. $s = [0 1]^T$, or $[1 0]^T$),

3. perform a univariate search,

$$\min_{\alpha} P(x_k + \alpha s)$$

4. repeat steps 2 and 3 alternating between search directions, until converged.

The problem with this method is that a very large number of iterations may be required to attain a reasonable level of accuracy. If we knew something about the ”orientation” of the objective function the rate of convergence could be greatly enhanced.
Consider the previous two dimensional problem, but using the independent search directions $([1 1]^T, [1 -1]^T)$.

Of course, finding the optimum in n steps only works for quadratic objective functions where the Hessian is known and each line search is exact. There are a large number of these optimization techniques which vary only in the way that the search directions are chosen.
Nelder-Mead Simplex

This approach has nothing to do with the SIMPLEX method of linear programming. The method derives its name from an n-dimensional polytope (and as a result is often referred to as the "polytope" method).

A polytope is an n-dimensional object that has n+1 vertices:

<table>
<thead>
<tr>
<th>n</th>
<th>polytope</th>
<th>number of vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>line segment</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>triangle</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>tetrahedron</td>
<td>4</td>
</tr>
</tbody>
</table>

It is an easily implemented direct search method, that only relies on objective function evaluations. As a result, it can be robust to some types of discontinuities and so forth. However, it is often slow to converge and is not useful for larger problems ( > 10 variables).
To illustrate the basic idea of the method, consider the two-dimensional minimization problem:

\[ x_1 \]

\[ x_2 \]
Step 1:

- \( P_1 = \max(P_1, P_2, P_3) \), define \( P_4 \) as the reflection of \( P_1 \) through the centroid of the line joining \( P_2 \) and \( P_3 \).

- \( P_4 \leq \max(P_1, P_2, P_3) \), from a new polytope from the points \( P_2, P_3, P_4 \).

Step 2:

- repeat the procedure in Step 1 to form new polytopes.

We can repeat the procedure until the polytope defined by \( P_4, P_5, P_6 \). Further iterations will cause us to flip between two polytopes. We can eliminate these problems by introducing two further operations into the method:

- **contraction** when the reflection step does not offer any improvement,

- **expansion** to accelerate convergence when the reflection step is an improvement.
For minimization, the algorithm is:

(a) **Order** according to the values at the vertices

\[ P(x_1) \leq P(x_2) \leq \ldots \leq P(x_{n+1}) \]

(b) Calculate \( x_o \), the centre of gravity of all points except \( x_{n+1} \)

\[ x_o = \frac{1}{n} \sum_{i=1}^{n} x_i \]

(c) **Reflection** : Compute reflected point

\[ x_r = x_o + \alpha (x_o - x_{n+1}) \]

If \( P(x_1) \leq P(x_r) < P(x_n) \):

The reflected point is better than the second worst, but not better than the best, then obtain a new simplex by replacing the worst point \( x_{n+1} \) with the reflected point \( x_r \), and go to step (a).
If \( P(x_r) < P(x_1) \)

The reflected point is the best point so far, then go to step (d).

(d) **Expansion**: compute the expanded point

\[
x_e = x_o + \gamma(x_o - x_{n+1})
\]

If \( P(x_e) < P(x_r) \)

The expanded point is better than the reflected point, then obtain a new simplex by replacing the worst point \( x_{n+1} \) with the expanded point \( x_e \), and go to step (a).

Else

Obtain a new simplex by replacing the worst point \( x_{n+1} \) with the reflected point \( x_r \), and go to step (a).
Else

The reflected point is worse than second worst, then continue at step (e).

(e) **Contraction**: Here, it is certain that $P(x_r) \geq P(x_n)$
Compute contracted point

$$x_c = x_{n+1} + \rho(x_o - x_{n+1})$$

If $P(x_c) \leq P(x_{n+1})$

The contracted point is better than the worst point, then obtain a new simplex by replacing the worst point $x_{n+1}$ with the contracted point $x_c$, and go to step (a).

Else

go to step (f).
(f) **Reduction** For all but the best point, replace the point with

\[ x_i = x_1 + \sigma (x_i - x_1), \forall i \in \{2, \ldots, n + 1\} \]

go to step (a).

(g) End of the algorithm.

**Note**: \( \alpha, \gamma \) and \( \sigma \) are respectively the reflection, the expansion, the contraction and the shrink coefficients. Standard values are \( \alpha = 1, \gamma = 2, \rho = 1/2 \) and \( \sigma = 1/2 \).
For the reflection, since $x_{n+1}$ is the vertex with the higher associated value among the vertices, we can expect to find a lower value at the reflection of $x_{n+1}$ in the opposite face formed by all vertices point $x_i$ except $x_{n+1}$.

For the expansion, if the reflection point $x_r$ is the new minimum along the vertices we can expect to find interesting values along the direction from $x_o$ to $x_r$.

Concerning the contraction: If $P(x_r) \geq P(x_n)$ we can expect that a better value will be inside the simplex formed by all the vertices $x_i$.

The initial simplex is important, indeed, a too small initial simplex can lead to a local search, consequently the NM can get more easily stuck. So this simplex should depend on the nature of the problem.

The Nelder-Meade Simplex method can be slow to converge, but it is useful for functions whose derivatives cannot be calculated or approximated (e.g. some non-smooth functions).
Example

\[
\min_x x_1^2 + x_2^2 - 2x_1 - 2x_2 + 2
\]

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(P_1)</th>
<th>(P_2)</th>
<th>(P_3)</th>
<th>(x_r)</th>
<th>(P_r)</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3/3</td>
<td>2/2</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>(2/1)</td>
<td>1</td>
<td>expansion</td>
</tr>
<tr>
<td>1</td>
<td>2/1</td>
<td>2/2</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>(1/1)</td>
<td>0</td>
<td>expansion</td>
</tr>
<tr>
<td>2</td>
<td>2/1</td>
<td>1/1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>(1/0)</td>
<td>1</td>
<td>contraction</td>
</tr>
<tr>
<td>3</td>
<td>2/1</td>
<td>1/1</td>
<td>1</td>
<td>0</td>
<td>13/16</td>
<td>(3/2)</td>
<td>5/16</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3/4</td>
<td>1/1</td>
<td>5/16</td>
<td>0</td>
<td>13/16</td>
<td>(3/2)</td>
<td>5/16</td>
<td></td>
</tr>
</tbody>
</table>

3 Gradient-Based Methods

This family of optimization methods use first-order derivatives to determine a "descent" direction. (Recall that the gradient gives the direction of the quickest increase for the objective function. Thus, the negative of the gradient gives the quickest decrease for the objective function.)
Steepest Descent

It would seem that the fastest way to find an optimum would be to always move in the direction in which the objective function decreases the fastest. Although this idea is intuitively appealing, it is very rarely true; however with a good line search every iteration will move closer to the optimum.

The Steepest Descent algorithm is:

1. choose a starting point $x_0$,

2. calculate the search direction:

$$ s_k = - \left[ \nabla_{x} P |_{x_k} \right] $$

3. determine the next iterate for $x$:

$$ x_{k+1} = x_k + \lambda_k s_k $$
by solving the univariate optimization problem:

$$\min_{\lambda_k} P(x_k + \lambda_k s_k)$$

4. repeat steps 2 & 3 until termination. Some termination criteria to consider:

$$\lambda_k \leq \epsilon$$
$$\|x_{k+1} - x_k\| \leq \epsilon$$
$$\|\nabla_x P|_{x_k}\| \leq \epsilon$$
$$\|P(x_{k+1}) - P(x_k)\| \leq \epsilon$$

where $\epsilon$ is some small positive scalar.
Example 1

\[
\min_x x_1^2 + x_2^2 - 2x_1 - 2x_2 + 2
\]

\[
\nabla_x P^T = \begin{bmatrix} 2x_1 - 2 \\ 2x_2 - 2 \end{bmatrix}
\]

start at \(x_0 = [3 \ 3]^T\)

| \(x_k\) | \(\nabla_x P^T|_{x_k}\) | \(\lambda_k\) | \(P(x_k)\) |
|---|---|---|---|
| 0 | \(\begin{bmatrix} 3 \\ 3 \end{bmatrix}\) | \(\begin{bmatrix} 4 \\ 4 \end{bmatrix}\) | 8 |
| 1 | \(\begin{bmatrix} 1 \\ 1 \end{bmatrix}\) | \(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\) |   |
start at $x_0 = [1 \ 2]^T$

| $x_k$  | $\nabla_x P^T|_{x_k}$ | $\lambda_k$ | $P(x_k)$ |
|--------|-----------------------|-------------|----------|
| 0      | $[1 \ 2]$             | $[0 \ 2]$   | 1        |
| 1      | $[1 \ 1]$             | $[0 \ 0]$   |           |

This example shows that the method of Steepest Descent is very efficient for well-scaled quadratic optimization problems (the profit contours are concentric circles). This method is not nearly as efficient for non-quadratic objective functions with very elliptical profit contours (poorly scaled).
Example 2

Consider the minimization problem:

$$\min_x \frac{1}{2} x^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} x$$

Then

$$\nabla_x P = x^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} = [101x_1 - 99x_2, -99x_1 + 101x_2]$$

To develop an objective function for our line search, substitute

$$x_{k+1} = x_k + \lambda_k s_k$$

into the original objective function:
\[ P(x_k + \lambda_k s_k) = \frac{1}{2} \left( x_k - \lambda_k \nabla_x P|_{x_k} \right)^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \left( x_k - \lambda_k \nabla_x P|_{x_k} \right) \]

\[ = \frac{1}{2} \left( x_k - \lambda_k \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} x_k \right)^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \left( x_k - \lambda_k \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} x_k \right) \]

To simplify our work, let

\[ H = \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \]

which yields:

\[ P(x_k + \lambda_k s_k) = \frac{1}{2} \left[ x_k H x_k - 2 \lambda_k x_k^T H^2 x_k + \lambda_k^2 x_k^T H^3 x_k \right] \]
This expression is quadratic in $\lambda_k$, thus making it easy to perform an exact line search:

$$\frac{dP}{d\lambda_k} = -x_k^T H^2 x_k + \lambda_k x_k^T H^3 x_k = 0$$

Solving for the step length yields:

$$\lambda_k = \frac{x_k^T H^2 x_k}{x_k^T H^3 x_k} = \frac{x_k^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}^2 x_k}{x_k^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}^3 x_k}$$

$$\lambda_k = \frac{x_k^T \begin{bmatrix} 20,002 & -19,998 \\ -19,998 & 20,002 \end{bmatrix} x_k}{x_k^T \begin{bmatrix} 4,000,004 & -3,999,9996 \\ -3,999,996 & 4,000,004 \end{bmatrix} x_k}$$
Starting at $x_0 = [2 \ 1]^T$:

|   | $x_k$       | $P(x_k)$ | $\nabla_x P^T |_{x_k}$ | $\lambda_k$ |
|---|-------------|----------|----------------|-------------|
| 0 | \[2 \ 1\]  | 54.50    | \[103 \ -97\] | 0.0050      |
| 1 | 1.4845      | 4.4104   | \[2.8809 \ 3.0591\] | 0.4591      |
|   | 1.4854      |          |                |             |
| 2 | 0.1618      | 0.3569   | \[8.3353 \ -7.8497\] | 0.0050      |
|   | 0.0809      |          |                |             |
| 3 | 0.1201      | 0.0289   | \[0.2331 \ 0.2476\] | 0.4591      |
|   | 0.1202      |          |                |             |
| 4 | 0.0131      | 0.0023   | \[0.6745 \ -0.6352\] | 0.0050      |
|   | 0.0065      |          |                |             |
| 5 | 0.0097      | $1.8915 \cdot 10^{-4}$ | \[0.0189 \ 0.0200\] | 0.4591      |
|   | 0.0097      |          |                |             |
| 6 | 0.0011      | $1.5307 \cdot 10^{-5}$ | \[0.0547 \ -0.0514\] | 0.0050      |
|   | 0.0005      |          |                |             |
| 7 | \[7.868 \cdot 10^{-4}\] | $1.2387 \cdot 10^{-5}$ | \[0.0015 \ 0.0016\] | 0.4591      |
|   | \[7.872 \cdot 10^{-4}\] |          |                |             |
Conjugate Gradients

Steepest Descent was based on the idea that the minimum will be found provided that we always move in the downhill direction. The second example showed that the gradient does not always point in the direction of the optimum, due to the geometry of the problem. Fletcher and Reeves (1964) developed a method which attempts to account for local curvature in determining the next search direction.

The algorithm is:

1. choose a starting point $x_0$,

2. set the initial search direction

$$s_0 = - \left[ \nabla_x P|_{x_0} \right]$$
3. determine the next iterate for $x$:

$$x_{k+1} = x_k + \lambda_k s_k$$

by solving the univariate optimization problem:

$$\min_{\lambda_k} P(x_k + \lambda_k s_k)$$

4. calculate the next search direction as:

$$s_{k+1} = -\nabla_x P|_{x_k} + s_k \frac{\nabla_x P|_{x_{k+1}} \nabla_x P^T|_{x_{k+1}}}{\nabla_x P|_{x_k} \nabla_x P^T|_{x_k}}$$

5. repeat steps 2 & 3 until termination.
The manner in which the successive search directions are calculated is important. For quadratic functions, these successive search directions are conjugate with respect to the Hessian. This means that for the quadratic function:

\[ P(x) = a + b^T x + \frac{1}{2} x^T H x \]

successive search directions satisfy:

\[ s_k^T H s_{k+1} = 0 \]

As a result, these successive search directions have incorporated within them information about the geometry of the optimization problem.
Example

Consider again the optimization problem:

$$\min_x \frac{1}{2} x^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} x$$

We saw that Steepest Descent was slow to converge on this problem of its poor scaling. As in the previous example, we will use an exact line search. It can be shown that the optimal step length is:

$$\lambda_k = -\frac{x_k^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} s_k}{s_k^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} s_k}$$
start at $x_0 = [2 \ 1]^T$

| $x_k$ | $P(x_k)$ | $\nabla_x P^T_{|x_k}$ | $s_k$ | $\lambda_k$ |
|------|----------|----------------------|-------|------------|
| 0    | $[2 \ 1]$ | $[103 \ -97]$ | $[103 \ -97]$ | |
| 1    | $[1.4845 \ 1.4854]$ | $[2.8809 \ 3.0591]$ | $[-2.9717 \ -2.9735]$ | |
| 2    | $[0 \ 0]$ |        |       | |


Notice the much quicker convergence of the algorithm. For a quadratic objective with exact line searches, the Conjugate Gradients method exhibits quadratic convergence.

The quicker convergence comes at an increased computational requirement. These include:

- a more complex search direction calculation,
- increased storage for maintaining previous search directions and gradients.

These are usually small in relation to the performance improvements.
4 Newton’s Method

The Conjugate Gradients method showed that there was a considerable performance gain to be realized by incorporating some problem geometry into the optimization algorithm. However, the method did this in an approximate sense. Newton’s method does this by using second-derivative information.

In the multivariate case, Newton’s method determines a search direction by using the Hessian to modify the gradient. The method is developed directly from the Taylor Series approximation of the objective function. Recall that an objective function can be locally approximated:

\[ P(x) \sim P(x_k) + \nabla_x P|_{x_k} (x-x_k) + \frac{1}{2} (x-x_k)^T \nabla^2_x P|_{x_k} (x-x_k) \]
Then, stationarity can be approximated as:

\[
\nabla_x P|_{x_k} \sim \nabla_x P|_{x_k} + (x - x_k)^T \nabla^2_x P|_{x_k} = 0
\]

Which can be used to determine an expression for calculating the next point in the minimization procedure:

\[
x_{k+1} = x_k - \left( \nabla^2_x P|_{x_k} \right)^{-1} \left( \nabla_x P|_{x_k} \right)
\]

Notice that in this case we get both the direction and the step length for the minimization. Further, if the objective function is quadratic, the optimum is found in a single step.
At a given point $x_k$, the algorithm for Newton’s method is:

1. calculate the gradient at the current point $\nabla_x P|x_k$.
2. calculate the Hessian at the current point $\nabla_x^2 P|x_k$.
3. calculate the Newton step:
   \[
   \Delta x_k = -\left(\nabla_x^2 P|x_k\right)^{-1}\left(\nabla_x P|x_k\right)^T
   \]
4. calculate the next point:
   \[
   x_{k+1} = x_k + \Delta x_k
   \]
5. repeat steps 1 through 4 until termination.
Example Consider again the optimization problem:

\[
\min_x \frac{1}{2} x^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} x
\]

The gradient and Hessian for this example are:

\[
\nabla_x P = x^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}, \quad \nabla^2_x P = \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}
\]

start at \(x_0 = [2 \ 1]^T\)

| \(x_k\) | \(P(x_k)\) | \(\nabla_x P^T_{|x_k}\) | \(\nabla^2_x P^T_{|x_k}\) |
|---|---|---|---|
| 0 | \[2 \\ 1\] | 54.50 | \[103 \\ -97\] | \[101 & -99 \\ -99 & 101\] |
| 1 | \[0 \\ 0\] | 0 | | |

As expected Newton’s method was very effective, even though the problem was not well scaled. Poor scaling was compensated for by using the inverse of the Hessian.

Steepest Descent was very good for the first two iterations, when we were quite far from the optimum. But it slowed rapidly as:

\[ x \to x^* \quad \nabla_x P_{x_k} \to 0 \]

The Conjugate Gradients approach avoided this difficulty by including some measure of problem geometry in the algorithm. However, the method started out using the Steepest Descent direction and built in curvature information as the algorithm progressed to the optimum.

For higher-dimensional and non-quadratic objective functions, the number of iterations required for convergence can increase substantially.
In general, Newton’s Method will yield the best convergence properties, at the cost of increased computational load to calculate and store the Hessian. The Conjugate Gradients approach can be effective in situations where computational requirements are an issue. However, there some other methods where the Hessian is approximated using gradient information and the approximate Hessian is used to calculate a Newton step. These are called the Quasi-Newton methods.

As we saw in the univariate case, care must be taken when using Newton’s method for complex functions. Since Newton’s method is searching for a stationary point, it can oscillate between such points in the above situation. For such complex functions, Newton’s method is often implemented with an alternative to step 4.

4’. calculate the next point :

$$x_{k+1} = x_k + \lambda_k \Delta x_k$$
where the step length is determined as:

i) $0 < \lambda_k < 1$ (so we don’t take a full step).

ii) perform a line search on $\lambda_k$.

Generally, Newton and Newton-like methods are preferred to other methods, the main difficulty is determining the Hessian.
5 Approximate Newton Methods

As we saw in the univariate case the derivatives can be approximated by finite differences. In multiple dimensions this can require substantially more calculations. As an example, consider forward difference approximation of the gradient:

\[ \nabla_x P|_{x_k} = \left[ \frac{\partial P}{\partial x_i} \right]_{x_k} \sim \frac{P(x_k + \delta_i) - P(x_k)}{\|\delta_i\|} \]

where \( \delta_i \) is a perturbation in the direction of \( x_i \). This approach requires 1 additional objective function evaluation per dimension for forward or back ward differencing, and 2 additional objective function evaluations per dimension for central differencing.
To approximate the Hessian using only finite differences in the objective function could have the form:

\[
\nabla^2 x P|_{x_k} = \left[ \frac{\partial^2 P}{\partial x_i \partial x_j} \right]_{x_k} \sim \frac{[P(x_k + \delta_i) - P(x_k)] - [P(x_k + \delta_j) - P(x_k)]}{\|\delta_i\|\|\delta_j\|}
\]

Finite difference approximation of the Hessian requires at least an additional 2 objective function evaluations per dimension. Some differencing schemes will give better performance than others; however, the increased computational load required for difference approximation of the second derivatives precludes the use of this approach for larger problems.

Alternatively, the Hessian can be approximated in terms of gradient information. In the forward difference case the Hessian can be approximated as:

\[
\nabla^2 x P|_{x_k} = \left[ \frac{\partial^2 P}{\partial x_i \partial x_j} \right]_{x_k} \sim h_i \sim \frac{\nabla x P(x_k + \delta_i) - \nabla x P(x_k)}{\|\delta_i\|}
\]
Then, we can form a finite difference approximation to the Hessian as:

\[ \tilde{H} = [h_i] \]

The problem is that often this approximate Hessian is non-symmetric. A symmetric approximation can be formed as:

\[ \hat{H} = \frac{1}{2} [\tilde{H} + \tilde{H}^T] \]

Whichever approach is used to approximate the derivatives, the Newton step is calculated from them. In general, Newton’s methods based on finite differences can produce results similar to analytical results, providing care is taken in choosing the perturbations \( \delta_i \).

(Recall that as the algorithm proceeds: \( x_k \rightarrow x^* \) and \( \nabla_x P|_{x_k} \rightarrow 0 \). Then small approximation errors will affect convergence and accuracy.) Some alternatives which can require fewer computations, yet build curvature information as the algorithm proceed are the Quasi-Newton family.
5.1 Quasi-Newton Methods

This family of methods are the multivariate analogs of the univariate Secant methods. They build second derivative (Hessian) information as the algorithm proceeds to solution, using available gradient information.

Consider the Taylor series expansion for the gradient of the objective function along the step direction $s_k$:

$$\nabla_{x} P|_{x_{k+1}} = \nabla_{x} P|_{x_{k}} + \nabla^{2}_{x} P|_{x_{k}} (\Delta x_{k}) + \ldots$$

Truncating the Taylor series at the second-order terms an re-arranging slightly yields the so-called Quasi-Newton Condition:

$$\hat{H}_{k+1}(\Delta x_{k}) = \nabla_{x} P|_{x_{k+1}} - \nabla_{x} P|_{x_{k}} = \Delta g_{k}$$
Thus, any algorithm which satisfies this condition can build up curvature information as it proceeds from $x_k$ to $x_{k+1}$ in the direction $\Delta x_k$; however, the curvature information is gained in only one direction. Then as these algorithms proceed the from iteration-to-iteration, the approximate Hessian can differ by only a rank-one matrix

$$\hat{H}_{k+1} = \hat{H}_k + vu^T$$

Combining the Quasi-Newton Condition and the update formula yields:

$$\hat{H}_{k+1}(\Delta x_k) = (\hat{H}_k + vu^T) (\Delta x_k) = \Delta g_k$$

Which yields a solution for the vector $v$ of:

$$v = \frac{1}{u^T \Delta x_k} \left[ \Delta g_k - \hat{H}_k \Delta x_k \right]$$
Providing $\Delta x_k$ and $u^T$ are not orthogonal, the Hessian update formula is given by:

$$\hat{H}_{k+1} = \hat{H}_k + \frac{1}{u^T \Delta x_k} \left[ \Delta g_k - \hat{H}_k \Delta x_k \right] u^T$$

Unfortunately this update formula is not unique with respect to the Quasi-Newton Condition and the step vector $s_k$.

We could add an additional term of $wz^T$ to the update formula, where the vector $w$ is any vector from the null space of $\Delta x_k$. (Recall that if $w \in \text{null}(\Delta x_k)$, then $\Delta x_k^T w = 0$). Thus there are a family of such Quasi-Newton methods which differ only in this additional term $wz^T$. 

Perhaps the most successful member of the family is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update:

\[
\hat{H}_{k+1} = \hat{H}_k + \frac{\Delta g_k \Delta g_k^T}{\Delta g_k^T \Delta x_k} - \frac{\hat{H}_k \Delta x_k \Delta x_k^T \hat{H}_k}{\Delta x_k^T \hat{H}_k \Delta x_k}
\]

The BFGS update has several advantages including that it is symmetric and it can be further simplified if the step is calculated according to:

\[
\hat{H}_k \Delta x_k = -\lambda_k \nabla_x P|_{x_k}
\]

In this case the BFGS update formula simplifies to:

\[
\hat{H}_{k+1} = \hat{H}_k + \frac{\Delta g_k \Delta g_k^T}{\Delta g_k^T \Delta x_k} + \frac{\lambda_k \nabla_x P|_{x_k} \nabla_x P^T|_{x_k}}{\nabla_x P^T|_{x_k} \Delta x_k}
\]
General Quasi-Newton Algorithm

1. Initialization :
   - choose a starting point $x_0$
   - choose an initial Hessian approximation (usually $\hat{H}_0 = I$).

2. At iteration $k$ :
   - calculate the gradient $\nabla_x P | x_k$
   - calculate the search direction :
     \[ s_k = -(\hat{H}_k)^{-1}(\nabla_x P | x_k) \]
   - calculate the next iterate :
     \[ x_{k+1} = x_k + \lambda_k s_k \]
using a line search on $\lambda_k$.

- compute $\nabla_x P_{|x_k}, \Delta g_k, \Delta x_k$.

- update the Hessian ($\hat{H}_{k+1}$) using the method of your choice (e.g. BFGS).

3. Repeat step 2 until termination.
Example

Consider again the optimization problem:

\[
\min_x \frac{1}{2} x^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} x
\]

The gradient and Hessian for this example are:

\[
\nabla_x P = x^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}, \quad \nabla^2_x P = \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}
\]
Start at $x_0 = [2, 1]^T$

| $x_k$     | $P(x_k)$ | $\nabla x P|_{x_k}$ | $\hat{H}_k$ | $s_k$ | $\lambda_k$ |
|-----------|----------|---------------------|-------------|------|-------------|
| $[2 \ 1]$ | 54.5     | $[103 \ -97]$       | $[1 \ 0 \ 1]$ | $[103 \ -97]$ | 0.005       |
| $[1.48 \ 1.48]$ | 4.41    | $[2.88 \ 3.05]$    | $[100.52 \ -99.5 \ 100.46]$ | $[-2.97 \ -2.97]$ | 0.50         |
| $[7.8 \cdot 10^{-4} \ 7.8 \cdot 10^{-4}]$ | $1.2 \cdot 10^{-8}$ | $[1.5 \cdot 10^{-4} \ 1.6 \cdot 10^{-4}]$ | $[101 \ -99 \ 101]$ | $[-7.8 \cdot 10^{-4} \ -7.8 \cdot 10^{-4}]$ | 1             |
| $[0 \ 0]$ |           |                     |             |      |             |

It is worth noting that the optimum is found in 3 steps (although we are very close within 2). Also, notice that we have a very accurate approximation for the Hessian (to 7 significant figures) after 2 updates.
The BFGS algorithm was:

- not quite as efficient as Newton’s method,

- approximately as efficient as the Conjugate Gradients approach,

- much more efficient than Steepest Descent.

This is what should be expected for quadratic functions of low dimension. As the dimension of the problem increases, the Newton and Quasi-Newton methods will usually outperform the other methods.

**Cautionary note:**

The BFGS algorithm guarantees that the approximate Hessian remains positive definite (and invertible) providing that the initial matrix is chosen as positive definite, in theory. However, round-off errors and sometimes the search history can cause the approximate Hessian to become badly conditioned. When this occurs, the approximate Hessian should be reset to some value (often the identity matrix $I$).
5.2 Convergence

During our discussions we have mentioned convergence properties of various algorithms. There are a number of ways with which convergence properties of an algorithm can be characterized. The most conventional approach is to determine the asymptotic behaviour of the sequence of distances between the iterates \((x_k)\) and the optimum \((x^*)\), as the iterations of a particular algorithm proceed. Typically, the distance between an iterate and the optimum is defined as:

\[
\|x_k - x^*\|
\]

We are most interested in the behaviour of an algorithm within some neighbourhood of the optimum; hence, the asymptotic analysis.
The asymptotic rate of convergence is defined as:

$$\lim_{k \to \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^p} = \beta$$

with the asymptotic order $p$ and the asymptotic error $\beta$.

In general, most algorithms show convergence properties which are within three categories:

- linear ($0 \leq \beta < 1, p = 1$),
- super-linear ($\beta = 0, p = 1$),
- quadratic ($\beta > 0, p = 2$).
For the methods we have studied, it can be shown that:

(a) the Steepest Descent method exhibits super-linear convergence in most cases. For quadratic objective functions the asymptotic convergence properties can be shown to approach quadratic as $k(H) \to 1$ and linear as $k(H) \to \infty$.

(b) the Conjugate Gradients method will generally show super-linear (and often nearly quadratic) convergence properties.

(c) Newton’s method converges quadratically.

(d) Quasi-Newton methods can closely approach quadratic rates of convergence.
As in the univariate case, there are two basic approaches:

- direct search (Nelder-Mead Simplex, ...),

- derivative-based (Steepest Descent, Newton’s, ...).

Choosing which method to use for a particular problem is a trade-off among:

- ease of implementation,

- convergence speed and accuracy,

- computational limitations.
Unfortunately there is no best method in all situations. For nearly quadratic functions Newton’s and Quasi-Newton methods will usually be very efficient. For nearly flat or non-smooth objective functions, the direct search methods are often a good choice. For very large problems (more than 1000 decision variables), algorithms which use only gradient information (Steepest Descent, Conjugate Gradients, Quasi-Newton, etc.) can be the only practical methods.

Remember that the solution to a multivariable optimization problem requires:

- a function (and derivatives) that can be evaluated,
- an appropriate optimization method.
Chapter V
Constrained Multivariate Optimization

- Introduction
- Optimality Conditions
- Examples
- Algorithms for Constrained NLP
- Sequential Quadratic Programming
1 Introduction

Until now we have only been discussing NLP that do not have constraints. Most problems that we wish to solve will have some form of constraints.

i) equality constraints (material and energy balances)

ii) inequality constraints (limits on operating conditions).

These type of problems require techniques that rely heavily on the unconstrained methods.
### General Form

Optimize $P(x)$

Subject to

- $f(x) = 0$, Equality constraints
- $g(x) \leq 0$, Inequality constraints
- $x_l \leq x \leq x_u$, Variable bounds

Objective function

Performance function

Profit (cost) function

Process model

Operating constraints

Variables limits
2 Optimality Conditions for Local Optimum

Karush-Kuhn-Tucker (KKT) Conditions:

1. Stationarity
\[ \nabla L(x^*, u, v) = \nabla P(x^*) + \nabla g(x^*)u + \nabla f(x^*)v = 0 \]

2. Feasibility
\[ g(x^*) \leq 0, \quad f(x^*) = 0 \]

3. Complementarity
\[ u^T g(x^*) = 0 \]
4. Non-negativity

\[ u \geq 0 \]

5. Curvature

\[ w^T \nabla^2 Lw \geq 0, \forall \text{ feasible } w \]
Example

\[ \min (x_1 - 5)^2 + (x_2 - 4)^2 \]

subject to

\[ 2x_1 + 3x_2 \leq 12 \]
\[ 6x_1 + 3x_2 \leq 18 \]

Check to see if constraint intersection is optimum, using KKT conditions:

\[ L = (x_1 - 5)^2 + (x_2 - 4)^2 + \lambda_1 (2x_1 + 3x_2 - 12) + \lambda_2 (6x_1 + 3x_2 - 18) \]
The gradients are given by:

$$\nabla_x L = [2(x_1 - 5) + 2\lambda_1 + 6\lambda_2 \quad 2(x_2 - 4) + 3\lambda_1 + 3\lambda_2]$$

$$\nabla_\lambda L = [2x_1 + 3x_2 - 12 \quad 6x_1 + 3x_2 - 18]$$

To find the stationary points, set

$$\nabla_x L = 0 \text{ and } \nabla_\lambda L = 0$$

$$x = \begin{bmatrix} 3/2 \\ 3 \end{bmatrix} \quad \lambda = \begin{bmatrix} -3/4 \\ 17/12 \end{bmatrix} \quad P = 13.25$$

Since one of the multipliers is negative, this is not optimum.
Let us assume that only the second constraint is active at the optimum.

\[
\text{min}(x_1 - 5)^2 + (x_2 - 4)^2
\]

subject to

\[
2x_1 + 3x_2 \leq 12
\]
\[
6x_1 + 3x_2 \leq 18 \text{ (active)}
\]

Then we could modify Lagrangian and check KKT conditions

\[
L = (x_1 - 5)^2 + (x_2 - 4)^2 + \lambda_1(6x_1 + 3x_2 - 18)
\]

\[
\nabla_x L = [2(x_1 - 5) + 6\lambda_1, 2(x_2 - 4) + 3\lambda_1] = 0
\]

\[
\nabla_\lambda L = [6x_1 + 3x_2 - 18] = 0
\]

\[
x = \begin{bmatrix}
9/5 \\
12/5
\end{bmatrix} = \begin{bmatrix}
1.8 \\
2.4
\end{bmatrix} \quad \lambda = 16/15 = 1.0666 \quad P = 12.8
\]
Let us assume that only the second constraint is active at the optimum and right hand side is perturbed a little bit by \( e = 0.001 \).

\[
\min (x_1 - 5)^2 + (x_2 - 4)^2
\]

subject to

\[
\begin{align*}
2x_1 + 3x_2 & \leq 12 \\
6x_1 + 3x_2 & \leq 18 + e \text{ (active)}
\end{align*}
\]

\[
L = (x_1 - 5)^2 + (x_2 - 4)^2 + \lambda_1(6x_1 + 3x_2 - 18.001)
\]

\[
\nabla_x L = [2(x_1 - 5) + 6\lambda_1 \quad 2(x_2 - 4) + 3\lambda_1] = 0
\]

\[
\nabla_\lambda L = [6x_1 + 3x_2 - 18.001] = 0
\]

\[
x = \begin{bmatrix} 1.8001 \\ 2.4001 \end{bmatrix} \quad \lambda = 1.0666 \quad P = 12.7989
\]
since the multiplier is positive, this is optimum.

Notice that $x$ and $\lambda$ changed very slightly from those values of the original solution

$$\min((x_1 - 5)^2 + (x_2 - 4)^2)$$

subject to

$$2x_1 + 3x_2 \leq 12$$
$$6x_1 + 3x_2 \leq 18 + e \text{ (active)}$$

$$\frac{dP}{de} = \frac{P_{original} - P_{new}}{0 - e} = \frac{12.8 - 12.7989}{-0.001} = -1.0666 = -\lambda$$

That is the profit function decreases by an amount of $\lambda$ for every increase in the r.h.s of the active constraint.
Sensitivity analysis tells us how the solution to an optimization problem changes as certain problem parameters change.

We don’t necessarily have to resolve the problem.
3 Algorithms for Constrained NLP

1. Successive (or Sequential) Linear Programming (SLP)
   - Linearize objective function and constraints,
   - Solve a sequence of linearized problems,
   - Limited use.

2. Penalty/Barrier Function Methods
   - Create an unconstrained problem by augmenting the objective function with constraints.
   - Solve the resulting unconstrained problem,
   - Barrier function methods back in vogue.
3. Gradient-Based Methods
   - Steepest, feasible descent direction,
   - Solve the resulting univariate problem,
   - Not currently popular.

4. Successive (or Sequential) Quadratic Programming (SQP)
   - Approximate problem locally as a QP,
   - Solve a sequence of QPs,
   - Currently very popular for process optimization.
4 Sequential Quadratic Programming

The idea is to approximate the NLP with a QP at the current value of $x$ by the problem:

$$\min_{\Delta x} \nabla P(x_k)^T \Delta x + \frac{1}{2} \Delta x^T B_k \Delta x$$

subject to

$$f(x_k) + \nabla f(x_k)^T \Delta x = 0$$
$$g(x_k) + \nabla g(x_k)^T \Delta x = 0$$

where $B_k$ is the current approximation of the Hessian of the Lagrangian and $\Delta x$ is the search direction.

Once the optimal search direction is found a univariate search is performed to determine how far we should proceed in this direction.

The entire procedure is then repeated until the algorithm has converged.
5 Optimization Problems Not Discussed

This workshop only scratched the surface of optimization problems and techniques to solve them. Some of the important topics that were not covered include:

1. Integer, Mixed-Integer, and Logic/Constraint Optimization.

2. Direct Search Optimization Techniques (e.g., Simulated Annealing, Genetic Algorithms).


4. Stochastic, Robust, Uncertain Optimization.

5. Multi-objective, Multi-criteria, and Vector Optimization.


7. Dynamic Optimization.