Optimization Concepts for Data Science

Course Notes

Optimization Concepts for Data Science Course Notes was developed by Robert Blanchard, Mark Hartmann, and Chip Wells. Additional contributions were made by Manoj Chari, Tao Huang, Trevor Kearney, Jay Laramore, Bob Lucas, Rob Pratt, and Jonathan Stephenson. Editing and production support was provided by the Curriculum Development and Support Department.

Optimization Concepts for Data Science Course Notes

Book code E70826, course code LWOPCON/OPCON, prepared date 06Sep2016. LWOPCON_002

ISBN 978-1-62960-699-6

Table of Contents

Course Description

This course focuses on linear, nonlinear, and efficiency optimization concepts. Participants learn how to formulate optimization problems and how to make their formulations efficient by using index sets and arrays. The demonstrations in the course include examples of data envelopment analysis and portfolio optimization. The OPTMODEL procedure is used to solve optimization problems that reinforce concepts introduced in the course.

To learn more…

For information about other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the web at http://support.sas.com/training/ as well as in the Training Course Catalog.

For a list of other SAS books that relate to the topics covered in this course notes, USA customers can contact the SAS Publishing Department at 1-800-727-3228 or send e-mail to sasbook@sas.com. Customers outside the USA, please contact your local SAS office.

Also, see the SAS Bookstore on the web at http://support.sas.com/publishing/ for a complete list of books and a convenient order form.

Prerequisites

Before enrolling in the data science certification program, you should have completed all coursework for the SAS Certified Big Data Professional program or passed the Big Data Certification exams. Before attending this course, you should have

- completed an undergraduate course in operations research that includes linear programming, have recent experience using linear programming, or be comfortable with matrix algebra
- be able to execute SAS programs and create SAS data sets.

You can gain this course-specific knowledge and skill by completing the SAS® Programming 1: Essentials course.

Chapter 1 Introduction to Mathematical Optimization

1.1 Introduction

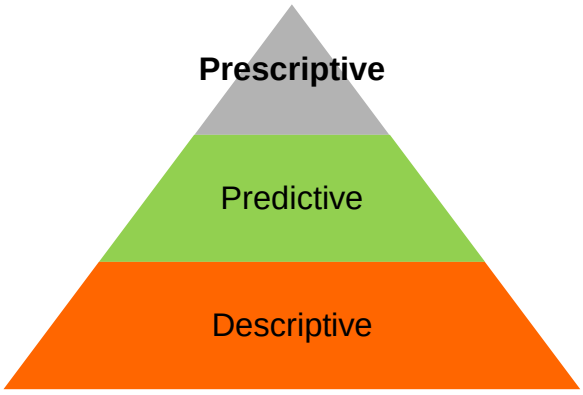
Objectives

- Explain the role of optimization in the decision-making process.
- Define basic terminology and classify mathematical optimization problems.

5

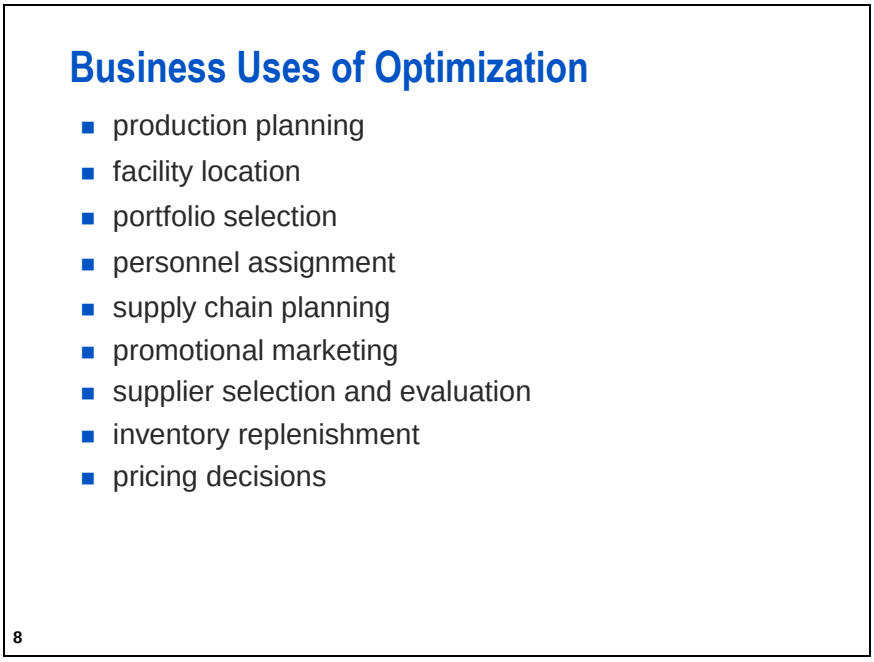
Optimization is a major component of operations research, industrial engineering, and management science. Simply put, optimization is the process of choosing the available actions that produce the best results.

Analytical Categories and Optimization



7

Mathematical optimization moves from "What happened?" (descriptive analytics) and "What will happen?" (predictive analytics) to "What should you do?" (prescriptive analytics). It can also inform "What if?" analysis by making logical supporting decisions, that is, "What should you do if …?"



Mathematical optimization is prevalent in business settings.

Production Planning: Determine which of several possible mixes of products should be produced to achieve the highest profit.

Facility Location: Find the "best" site (for example, for a new factory), in relation to the location of the materials suppliers, distribution centers, and so on.

Portfolio Selection: Maximize return on investment (ROI), while balancing return versus risk.

Personnel Assignment: Match personnel to work requirements so that you can meet current needs and anticipated changes, subject to budget and HR requirements.

Supply Chain Planning: Find the lowest-cost way to move product from factories to distribution centers to stores, and plan for possible disruptions or expansion.

Promotional Marketing: Determine the best combination of promotional offers, delivery channels, and customers to maximize the overall return on marketing investment.

Supplier Selection and Evaluation: Choose which suppliers to deal with so that you can satisfy requirements and maximize leverage. Use a variety of criteria simultaneously to rate suppliers.

Inventory Replenishment: Set inventory policies (reorder levels and maximum stock levels) to meet customer service goals and minimize costs.

Pricing Decisions: Establish and maintain optimal everyday prices based on costs, regional demand patterns, and competitive price information.

Defining Mathematical Optimization Problems

The general form of an optimization problem is as follows:

$$\begin{aligned} &\min|\max && f(\mathbf{x}) \\ &\text{subject to} && c_i(\mathbf{x}) \ \{\leq,=,\geq\} \ b_i \quad (i=1,2,...,m) \\ & && l_j \leq x_j \leq u_j \quad (j=1,2,...,n) \end{aligned}$$

where

- **x** is an array of *decision variables*.
- f(**x**) is an *objective function*.
- $c_1(\mathbf{x})$, ... ,$c_m(\mathbf{x})$ are functions that together with the limits and bounds on **x** determine the *constraints*.

An *optimal solution* **x*** is a set of values that minimize or maximize the objective function subject to the constraints.

13

The decision variables x_j correspond to decisions or choices in the underlying business problem. These choices, about which optimal values are to be determined, are under the control of the decision maker. For some types of mathematical optimization problems, the decision variables are further restricted to be *integers* (…-2,-1,0,1,2, …).

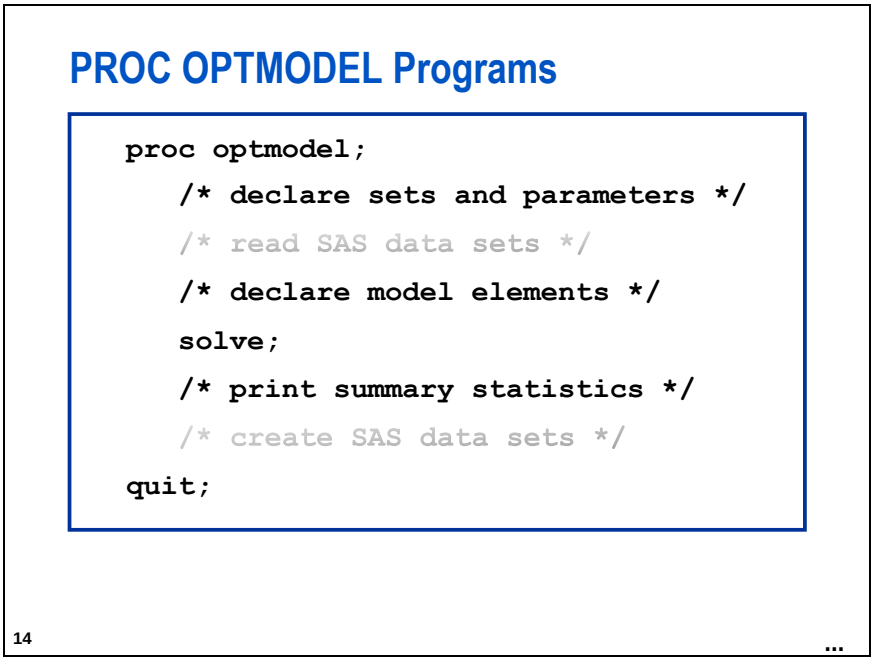
A *feasible solution* **x** is a set of decision variable values that satisfy all of the constraints.

Minimizing f(**x**) or maximizing f(**x**) is referred to as the *objective* of the mathematical optimization problem.

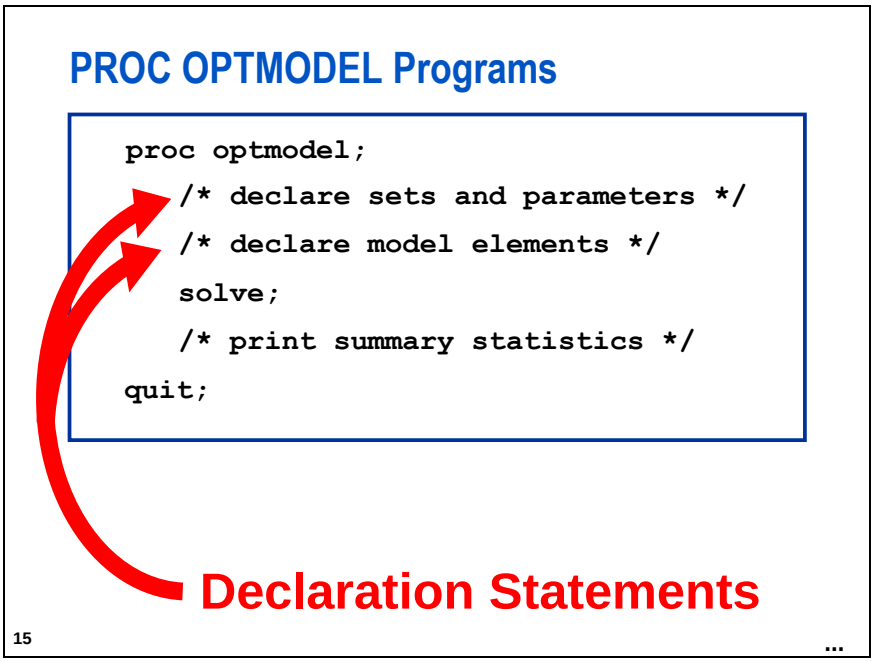
The bounds on **x** can be $\pm\infty$, so that x_j can be restricted to be nonnegative ($l_j = 0$ and $u_j = +\infty$), or for that matter, free ($l_j = -\infty$ and $u_j = +\infty$). Constraints can be structural: contracts, resource availabilities, legal restrictions, and so on.

Next you see the structure of a PROC OPTMODEL program that builds and solves a mathematical optimization problem.

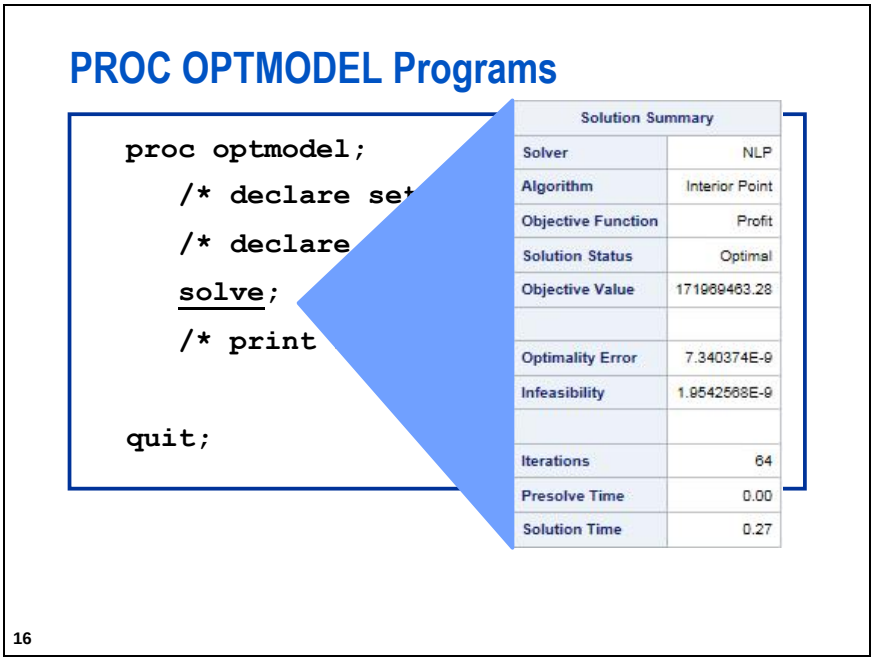
Declaration statements are used to establish model elements (variables, constraints, and objective) and sets and parameters that help determine their defining coefficients, functions, and limits.



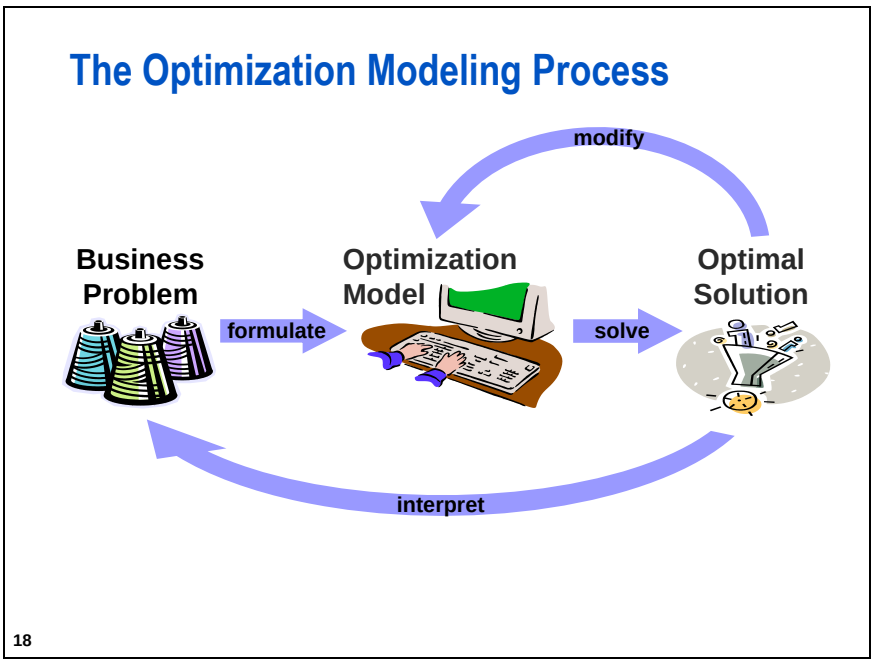
Reading set and parameter values from SAS data sets and writing model output to SAS data sets is an important part of applied optimization analysis with the OPTMODEL procedure. However, this is a one-day class that focuses on solving linear and nonlinear optimization problems, and information about reading and writing SAS data sets is not presented. Information about reading SAS data sets in the OPTMODEL procedure is provided in a self-study section of a subsequent chapter.



Simple mathematical optimization problems might not require sets and parameters.



The SOLVE statement causes the mathematical optimization model to be built from the declaration statements and solved. It produces a solution summary that includes the optimal objective value and solution time.



Formulating a business problem means making the assumptions and approximations that are necessary to represent it as an optimization model that can (ideally) be solved in a reasonable amount of time. If the optimal solution to the optimization problem cannot be interpreted in the context of the business problem, model modifications might be necessary.

Why Optimization?

You can improve the decision-making process by adding the following:

- **Structure:** Know how decisions are motivated
- **Consistency:** Drive all decisions toward the same goals
- **Repeatability:** Make decisions in the same manner over time
- **Adaptability:** Update, but keep the same principles
- **Persistence:** Not tied to individual experience
- **Scalability:** Tackle larger, more complex problems

19

Classification of Mathematical Optimization Problems

Discussed in this course:

- **LP:** $f(\mathbf{x})$ and $c_1(\mathbf{x}), \ldots, c_m(\mathbf{x})$ are linear functions.
- **NLP:** $f(\mathbf{x})$ and $c_1(\mathbf{x}), \ldots, c_m(\mathbf{x})$ are continuous but not ***all*** linear functions.

Not discussed in this course:

- ILP: $f(\mathbf{x})$ is linear, $\mathbf{x}$ must be integer, and $c_1(\mathbf{x}), \ldots, c_m(\mathbf{x})$ are linear functions.
- MILP: $f(\mathbf{x},\mathbf{y})$ is linear, $\mathbf{x}$ must be integer, and $c_1(\mathbf{x},\mathbf{y}), \ldots, c_m(\mathbf{x},\mathbf{y})$ are linear functions, but $\mathbf{y}$ can be fractional.

20

This course provides an introduction to solving linear and nonlinear programming problems using the OPTMODEL procedure in SAS. This chapter provides introductory details in integer and mixed-integer linear programming problems for completeness.

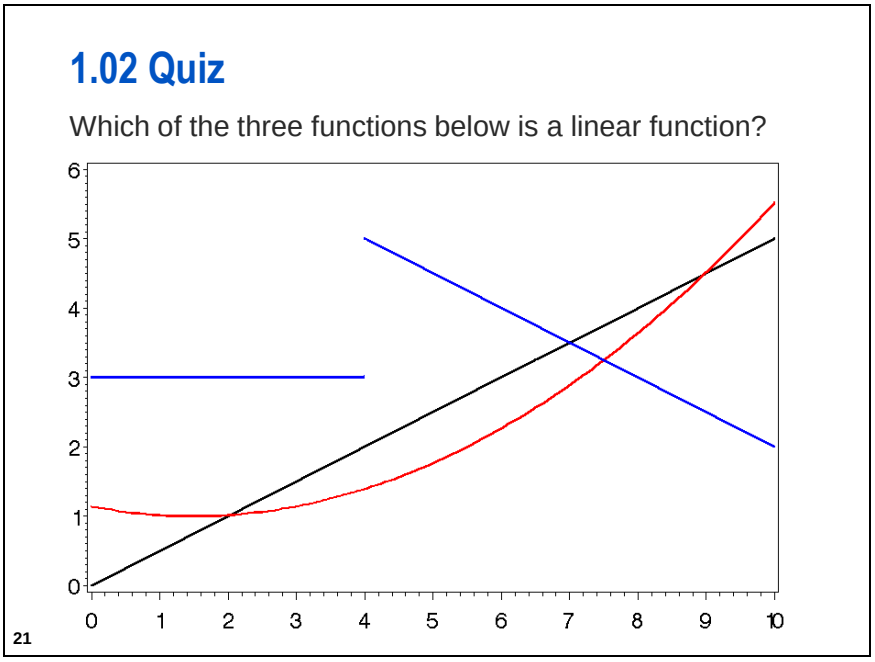
(A graph of a linear, continuous, and discontinuous function appear on the following page.)

- If $f(\mathbf{x})$ and $c_1(\mathbf{x}), \ldots, c_m(\mathbf{x})$ are linear functions, then the mathematical optimization problem is a *linear programming problem*, or *linear program* (*LP*).

- If f(**x**) is linear and the decision variables are required to take on integer values, but otherwise $c_1(\mathbf{x})$, ... ,$c_m(\mathbf{x})$ are linear functions, then the mathematical optimization problem is an *integer linear programming problem*, or *integer linear program* (*ILP*). If only ***some*** of the decision variables are required to take on integer values, then the problem is a *mixed-integer linear programming problem*, or *mixed-integer linear program* (*MILP*).
- If f(**x**) and $c_1(\mathbf{x})$, ... ,$c_m(\mathbf{x})$ are continuous but not ***all*** linear functions, then the mathematical optimization problem is a *nonlinear programming problem*, or *nonlinear program* (*NLP*).

This is not an exhaustive classification. However, problems beyond this set, such as those that combine integer decision variables and nonlinear functions, are solvable only when there is sufficient structure to allow for special-purpose solvers.

The simplex method for solving linear programming problems was designed by G. B. Dantzig in 1947 to solve U.S. Air Force planning problems. Current methods for solving mixed-integer and nonlinear programming problems are more recent, and research is ongoing.



Recall that $f(\mathbf{x})$ is a linear function if $f(\mathbf{x}) = f_1x_1 + f_2x_2 + \ldots + f_nx_n$ for some numbers $f_1, f_2, \ldots, f_n$.

Optimization: Going Up or Going Down?

- **LP:** The default settings work almost every time. You only need to push the correct button.

23

Although the number of iterations (steps) that are required to solve an LP varies with the structure of the problem, it is typically on the order of m*log(n) for an LP with m constraints and n variables.

Optimization: Going Up or Going Down?

- **LP:** The default settings work almost every time. You only need to push the correct button.
- **ILP/MILP:** Some more difficult problems might require different solver options. What happens when you push the red button?

24

The number of iterations that are required to solve an ILP/MILP is difficult to predict. Some are as easy to solve as LPs. For others, the "solution" process approaches a complete enumeration of the possible integer values (whose number can grow exponentially in the number of constraints and variables).

Optimization: Going Up or Going Down?

- **LP:** The default settings work almost every time. You only need to push the correct button.
- **ILP/MILP:** Some more difficult problems might require different solver options. What happens when you push the red button?
- **NLP:** Your starting point might determine where you end up. "I've a feeling we're not in Kansas anymore."

25

When you solve an LP, ILP, or MILP, the solvers provide a bound on the relative or absolute objective gap. (The *objective gap* is the gap between the true optimal objective value and the objective value of the solution that is found.) There is only a bound on the relative or absolute objective gap for special classes of NLPs, such as quadratic programming problems. In addition, the solution that is found might depend on the algorithm that is used and the starting solution that is supplied by the decision maker.

	All continuous variables	Some integer variables	All integer variables
All linear functions	**Linear Programming (LP)**	**Mixed-Integer Linear Programming (MILP)**	**Integer Linear Programming (ILP)**
Some nonlinear functions	**Nonlinear Programming (NLP)**	¿?	?

Exercises

1. Classifying Mathematical Optimization Problems

Use the table above to match the four mathematical optimization problems **a** through **d** with decision variables x, y, and z to the four problem classes: LP (linear programming), ILP (integer linear programming), MILP (mixed-integer linear programming), and NLP (nonlinear programming).

a. minimize $4x + 5y + 2z$
subject to $7x + 8y + 3z \geq 20$
$x \geq 0, y \geq 0, z \geq 0$
x , y, z integers

b. maximize $3x + 5y - 4z$
subject to $6x^2 + 2y^2 + z^2 \leq 17$
$x + y + z = 3$
$-1 \leq x \leq 1$

c. maximize $12x + 19y - 4z$
subject to $x + 3y + z = 225$
$x + y - z \leq 117$
$x \geq 0, y \geq 0, z \geq 0$

d. maximize $4x + 5y$
subject to $x + y + 2z \leq 10$
$2x + 3y - 3z \leq 5$
$x \geq 0, y \geq 0$
x , y integers

End of Exercises

Setup for the Poll

Classify the mathematical optimization problem.

$$\begin{aligned} \text{minimize} \quad & 2xy + 2yz + 2xz \\ \text{subject to} \quad & xyz = 1 \\ & x \geq 0,\ y \geq 0,\ z \geq 0 \end{aligned}$$

28

1.03 Multiple Choice Poll

Classify the mathematical optimization problem.

a. LP
b. ILP
c. MILP
d. NLP

29

Setup for the Poll

Classify the mathematical optimization problem.

$$\begin{aligned} \text{minimize} \quad & z \\ \text{subject to} \quad & x + y - z \leq 10 \\ & 2x + 3y - z \leq 5 \\ & x + z \geq 0 \\ & y + z \geq 0 \end{aligned}$$

31

1.04 Multiple Choice Poll

Classify the mathematical optimization problem.

a. LP
b. ILP
c. MILP
d. NLP

32

1.2 A Simple Example

Objectives

- Explain local properties of functions that are used to solve mathematical optimization problems.

35

Two-Dimensional Example

$$\text{minimize } z = x^2 - x - 2y - xy + y^2$$

This problem minimizes a nonlinear function with

- two decision variables, x and y
- no bounds or other constraints.

Vector calculus can be used to find the solution:

$$\nabla z = [\partial z/\partial x,\ \partial z/\partial y] = [2x - 1 - y,\ -2 - x + 2y] = [0, 0]$$

$$\Rightarrow [x, y] = [4/3, 5/3]$$

$$H(z) = \begin{bmatrix} \frac{\partial^2 z}{\partial x^2} & \frac{\partial^2 z}{\partial x \partial y} \\ \frac{\partial^2 z}{\partial y \partial x} & \frac{\partial^2 z}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$ *(positive definite)*

38

Recall the following two-dimensional Taylor series:

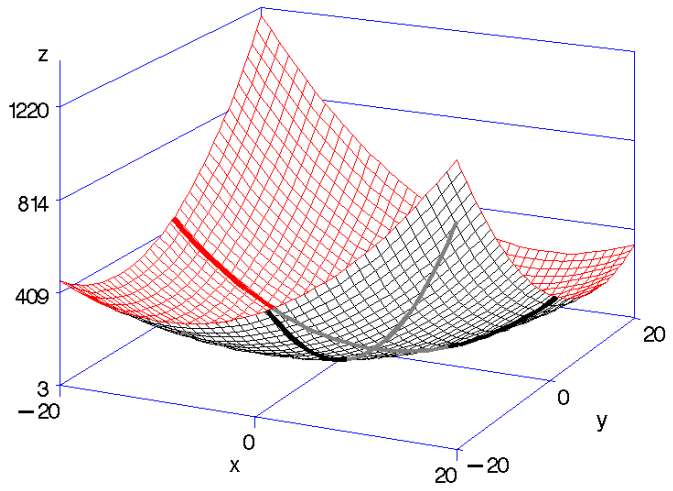
$$z(x+dx,y+dy) \approx z(x,y) + \nabla z^T[dx,dy] + \tfrac{1}{2}\,[dx,dy]^T H(z)[dx,dy]$$

which is exact for a quadratic function.

If the gradient $\nabla z(x,y) \neq [0, 0]$, then either increasing or decreasing x or y slightly decreases z(x,y), so the minimum must occur at a point [x, y] where the gradient is zero.

The Hessian H(z) is positive definite (which means $u^T H(z)u > 0$ for all $u \neq 0$), so [4/3, 5/3] is a minimum, not a maximum or a saddle point.

Graph of Two-Dimensional Example



39

From the graph, you can see that the situation would be more complicated if you added the bounds $x \geq 0$ and $y \leq 0$, which require the solution to lie in the nearest quadrant.

1.05 Poll

Is [x*, y*] = [4/3, 5/3] a feasible solution to the problem of minimizing $z(x,y) = x^2 - x - 2y - xy + y^2$ subject to the bound constraints $x \geq 0$ and $y \leq 0$?

- ❍ Yes
- ❍ No

40

1.06 Poll

Will the optimal solution [x*, y*] to the problem of minimizing $z(x,y) = x^2 - x - 2y - xy + y^2$ subject to the bound constraints $x \geq 0$ and $y \leq 0$ have $x^* > 0$ and $y^* < 0$?

- ❍ Yes
- ❍ No

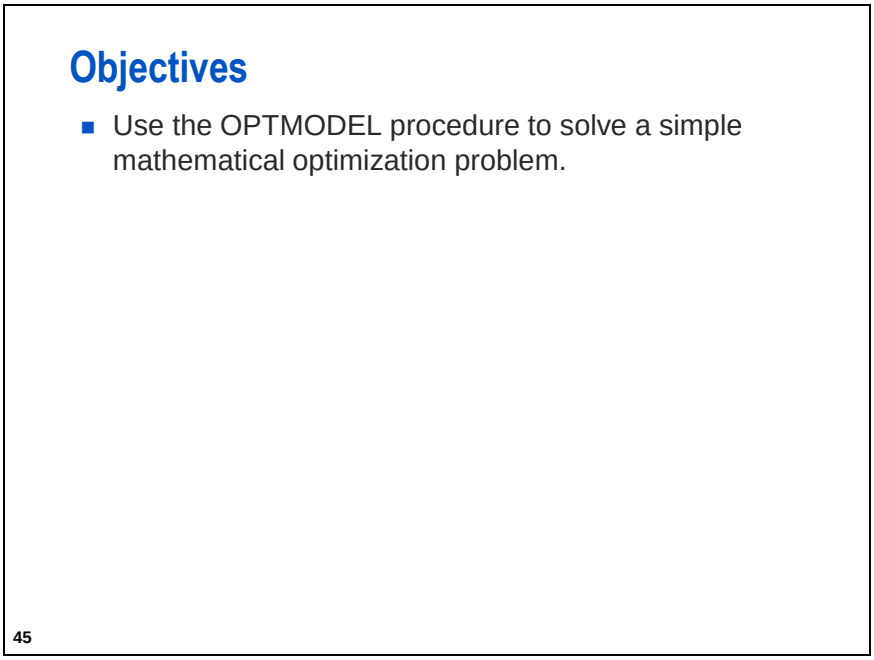
42

How can you find an optimal solution in this case?

The previous graph (without the annotations) can be produced by the following SAS program:

```
data polynomial;
   do x=-20 to 20 by 1;
     do y=-20 to 20 by 1;
       z=x**2-x-2*y-x*y+y**2;
       output;
     end;
   end;
run;

proc g3d data=polynomial;
   plot y*x=z / grid rotate=-25 ctop=red cbottom=black
                yticknum=3 xticknum=3 zmin=3 zmax=1220;
run;
```

1.3 The OPTMODEL Procedure



Solving a mathematical optimization problem means finding an optimal solution, or determining that there is no optimal solution to the mathematical optimization problem.

The OPTMODEL procedure is a full-fledged interactive algebraic modeling language. The ability to solve mathematical optimization problems is a powerful feature of PROC OPTMODEL. It can be used as a building block in the design of integrated solutions to complex business problems.

In the demonstration that follows, PROC OPTMODEL is used to solve the mathematical optimization problem of minimizing the two-dimensional polynomial. An additional PROC OPTMODEL command prints the values of the decision variables.

Solving the Simple Polynomial Example Using PROC OPTMODEL

```
proc optmodel;
   var x, y;

   min z = x**2 - x - 2*y - x*y + y**2;

   solve;

   print x y;

quit;
```

The first and last terms in the objective function can also be entered using the caret symbol (^) for exponentiation. However, this symbol is also a logical NOT operator in Base SAS.

PROC OPTMODEL Output

The OPTMODEL Procedure

Problem Summary

Objective Sense	Minimization
Objective Function	z
Objective Type	Quadratic
Number of Variables	2
Bounded Above	0
Bounded Below	0
Bounded Below and Above	0
Free	2
Fixed	0
Number of Constraints	0
Constraint Coefficients	0

Performance Information

Execution Mode	Single-Machine
Number of Threads	4

Solution Summary

Solver	QP
Algorithm	Interior Point
Objective Function	z
Solution Status	Optimal
Objective Value	-2.333333333
Primal Infeasibility	0
Dual Infeasibility	6.861556E-17
Bound Infeasibility	0
Duality Gap	0
Complementarity	0
Iterations	0
Presolve Time	0.00
Solution Time	0.00

x	y
1.3333	1.6667

PROC OPTMODEL selects an optimization algorithm based on problem characteristics (in this example, an interior point algorithm for quadratic programming problems). No iterations of the interior point algorithm were required to obtain an optimal solution.

The performance information is omitted from subsequent PROC OPTMODEL output.

When you add the bounds $x \geq 0$ and $y \leq 0$, you can see that PROC OPTMODEL selects the same algorithm but finds a different optimal solution.

```
proc optmodel;
   var x >= 0, y <= 0;

   min z = x**2 - x - 2*y - x*y + y**2;

   solve;

   print x y;

quit;
```

PROC OPTMODEL Output

The OPTMODEL Procedure

Problem Summary

Objective Sense	Minimization
Objective Function	z
Objective Type	Quadratic
Number of Variables	2
Bounded Above	1
Bounded Below	1
Bounded Below and Above	0
Free	0
Fixed	0
Number of Constraints	0
Constraint Coefficients	0

Solution Summary

Solver	QP
Algorithm	Interior Point
Objective Function	z
Solution Status	Optimal
Objective Value	-0.250000022
Primal Infeasibility	0
Dual Infeasibility	1.9167889E-9
Bound Infeasibility	8.7045896E-9
Duality Gap	1.9890322E-8
Complementarity	0
Iterations	4
Presolve Time	0.00
Solution Time	0.02

x	y
0.5	8.7046E-09

End of Demonstration

Exercises

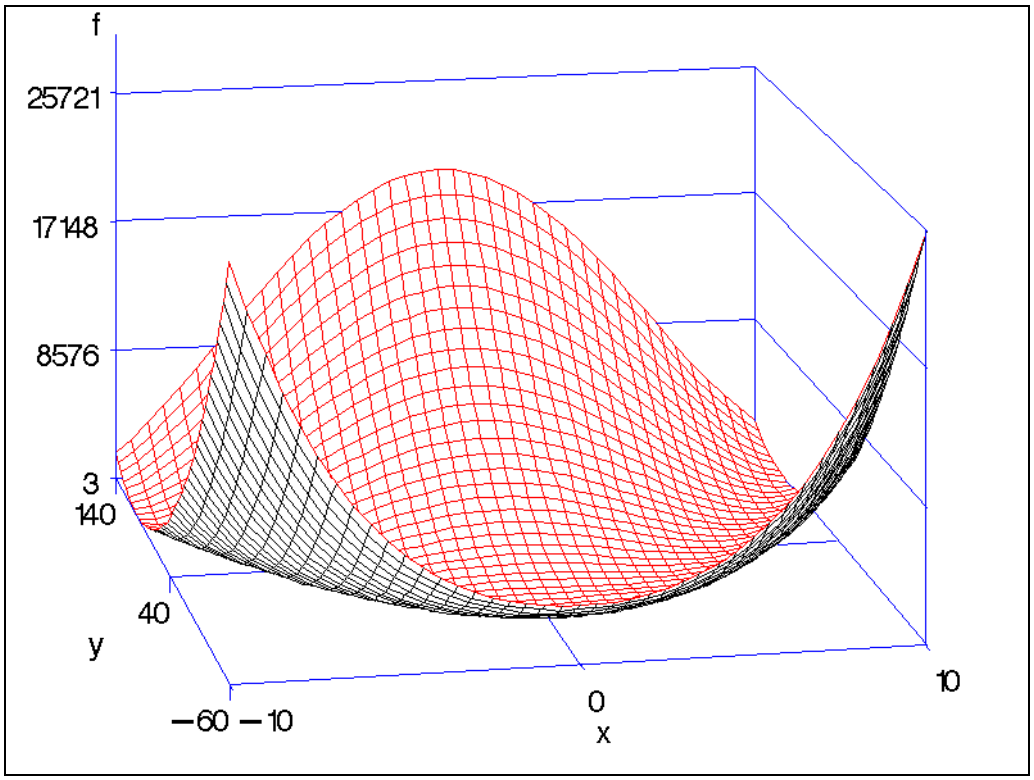
2. Using the OPTMODEL Procedure

Use PROC OPTMODEL to minimize the Rosenbrock banana function.

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

(This is a standard example to show that methods that use only the gradient can fail by repeatedly crossing back and forth over the "valley," whereas methods that use information about curvature can converge in a few iterations.)

Why do you think that the objective function is called a *banana function*?



(This graph can be produced by the SAS program **banana_graph.sas**.)

End of Exercises

1.4 Solutions

Solutions to Exercises

1. Classifying Mathematical Optimization Problems

a. ILP: All variables are required to be integers. This ILP has two optimal solutions: x = 3, y = 0, z = 0, and x = 2, y = 0, z = 2. (Both solutions are feasible and have objective value 12.) The first constraint can be used to show that the objective value is at least 20(4/7) = 11 3/7.

b. NLP: Although the objective function is linear, the inequality constraint contains nonlinear (quadratic) terms. This NLP has a unique optimal solution: x = 0.67689, y = 2.6588, z = –0.33567. (Apart from the bound on x, the feasible region is a cross section of an ellipsoid and hence convex.)

c. LP: Linear constraints can be equations or inequalities. This LP has the unique optimal solution: x = 171, y = 0, z = 54.

d. MILP: Only some of the variables (x and y, but not z) are required to be integer, but otherwise the constraints are linear. Every point with x = 3, y = 2, and $2\frac{1}{3} \leq z \leq 2\frac{1}{2}$ is an optimal solution to the MILP with objective value 22. You can think of z as a parameter that enables you to exchange one resource for another.

2. Using the OPTMODEL Procedure

```
proc optmodel;
   var x, y;

   min f = (y-x**2)**2 + (1-x)**2;

   solve;

   print x y;

quit;
```

The objective function can also be entered using the caret symbol (^) for exponentiation, but this symbol is also a logical NOT operator in Base SAS.

PROC OPTMODEL Output

The OPTMODEL Procedure

Problem Summary

Objective Sense	Minimization
Objective Function	f
Objective Type	Nonlinear
Number of Variables	2
Bounded Above	0
Bounded Below	0
Bounded Below and Above	0
Free	2
Fixed	0
Number of Constraints	0

Solution Summary

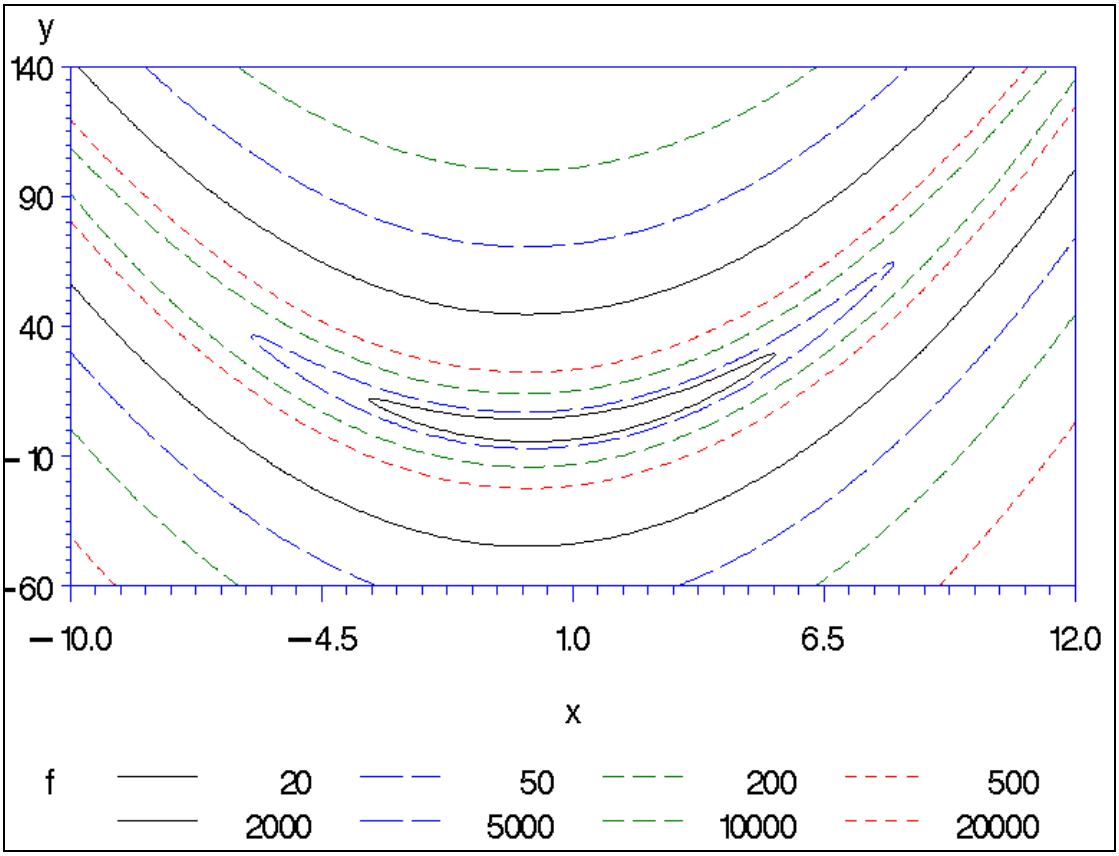
Solver	NLP
Algorithm	Interior Point
Objective Function	f
Solution Status	Optimal
Objective Value	2.962209E-15
Optimality Error	8.9089584E-8
Infeasibility	0
Iterations	5
Presolve Time	0.00
Solution Time	0.03

x	y
1	1

PROC OPTMODEL selected a different algorithm than in the demonstration. In this example, five iterations were required.

The solution x = y = 1 has objective value zero, but the reported value is positive. The default number of significant digits for the PRINT command (PDIGITS=5) is less than the number for the objective value (BEST12.), so the actual values of x and y found by the solver are not both exactly 1.

The objective function is called a banana function because the level sets {[x,y]: f(x,y) = z} for z > 0 have a banana shape. (See the contour plot below.)

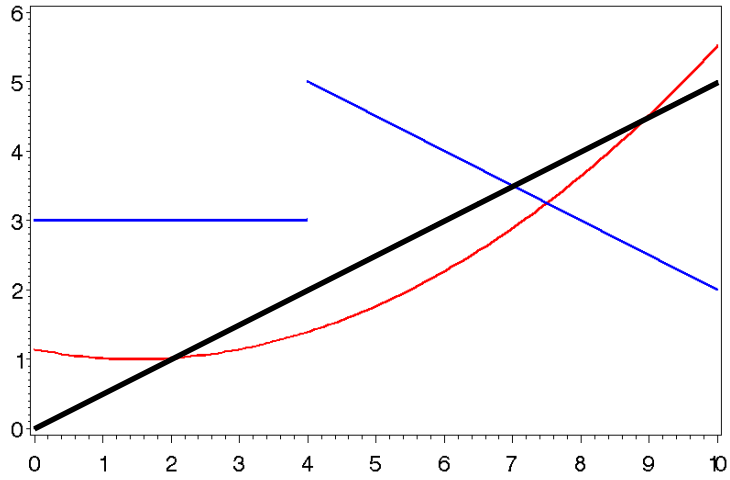


End of Solutions

Solutions to Student Activities (Polls/Quizzes)

1.02 Quiz – Correct Answer

Which of the three functions below is a linear function?



22

The figure shows graphs of a linear function ($y = 0.5x$), a continuous nonlinear function, and a discontinuous function ($y = 3$ for $x \leq 4$ and $y = 7 - 0.5x$ for $x > 4$) of a single variable x.

1.03 Multiple Choice Poll – Correct Answer

Classify the mathematical optimization problem.

a. LP
b. ILP
c. MILP
d. NLP

30

This NLP has the unique optimal solution $x = y = z = 1$. (A cube has the smallest surface area among rectangular prisms with a given volume.)

1.04 Multiple Choice Poll – Correct Answer

Classify the mathematical optimization problem.

a. LP
b. ILP
c. MILP
d. NLP

33

This LP has the unique optimal solution x = y = 5/6 and z = –5/6. This problem can be interpreted as finding the point [x,y] that is farthest from the boundary of the region determined by setting z = 0.

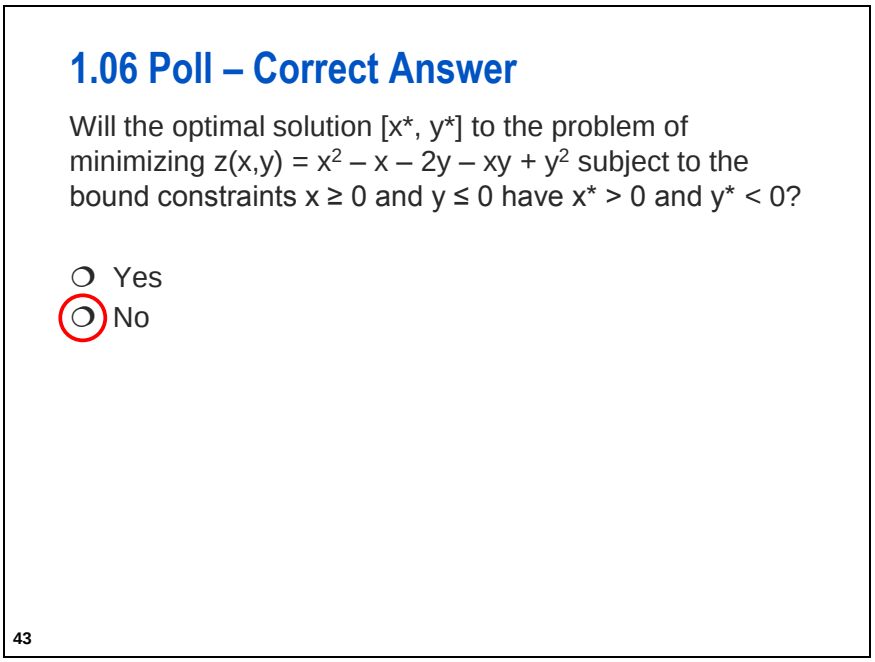
1.05 Poll – Correct Answer

Is $[x^*, y^*] = [4/3, 5/3]$ a feasible solution to the problem of minimizing $z(x,y) = x^2 – x – 2y – xy + y^2$ subject to the bound constraints $x \geq 0$ and $y \leq 0$?

❍ Yes
❍ No

41

The solution has $y^* > 0$.



The gradient $\nabla z(x,y) = [2x - 1 - y, -2 - x + 2y] \neq [0, 0]$ unless $[x,y] = [4/3, 5/3]$. Therefore, no point $[x^*, y^*]$ with $x^* > 0$ and $y^* < 0$ can be an optimal solution. The optimal solution must lie on the boundary $x = 0$ and $y \leq 0$ or $x \geq 0$ and $y = 0$.

Chapter 2 Linear Programming Problems: Basic Ideas

2.1 Introduction to Linear Programming

Objectives

- Introduce the basic ideas in the primal and dual simplex approaches to solving linear programming problems.
- Describe how, geometrically, the primal simplex and dual simplex algorithms solve linear programming problems.
- Introduce the Lagrangian function.
- Use the OPTMODEL procedure to enter and solve simple linear programming problems.

3

The General Framework of Linear Programming

A Linear Programming Problem

$$\begin{aligned} &\min|\max && f_1x_1 + \ldots + f_nx_n \\ &\text{subject to} && \mathbf{Ax} \{\leq, =, \geq\} \mathbf{b} \\ & && l_j \leq x_j \leq u_j \quad (j = 1,2,\ldots,n) \end{aligned}$$

- Each linear constraint can be either an inequality or an equation.
- Bounds can be $\pm\infty$, so x_j can be restricted to be nonnegative ($l_j = 0$ and $u_j = +\infty$) or free ($l_j = -\infty$ and $u_j = +\infty$).

4

If there are m linear constraints, then **A** would be an m x n matrix.

Implicit Assumptions of Linear Programming

$$\begin{aligned} &\min|\max && f_1x_1 + \ldots + f_nx_n \\ &\text{subject to} && \mathbf{Ax}\ \{\leq, =, \geq\}\ \mathbf{b} \\ & && l_j \leq x_j \leq u_j \quad (j = 1,2,\ldots,n) \end{aligned}$$

- **Proportionality:** No economies of scale
- **Additivity:** No synergy or interaction terms
- **Divisibility:** Variables can assume any fractional value.
- **Certainty:** All parameters are known constants.

5

Proportionality: All returns and resource usages change proportionally with the variables, and there are no economies or diseconomies of scale.

Additivity: The total effect of any two variables is the sum of their individual effects without synergy or interaction (that is, no cross-product terms in the constraints or the objective).

Divisibility: Variables can assume any fractional values.

Certainty: All parameters in the model are known constants, not random variables.

The Lagrangian function provides a framework in which values for decision variables that simultaneously optimize the objective function and that satisfy the constraints can be derived.

Lagrangian Function

$$L(\mathbf{x}, \boldsymbol{\lambda}) = F(\mathbf{x}) + \boldsymbol{\lambda}[\mathbf{b} - A(\mathbf{x})]$$

6

The general Lagrangian function has two (possibly vector-valued) arguments. Decision variables are represented by $\mathbf{x}$, and each constraint has a parameter, lambda. A constraint parameter is also called a *shadow price*. (Its role in the optimization is explained in further detail later in this chapter.) The objective, $F(\mathbf{x})$, and constraint, $\mathbf{b} - A(\mathbf{x})$, functions are represented as possibly nonlinear functions of the decision variables. This chapter focuses on ***linear*** programming, and linear in this context means that the objective and constraint functions are linear functions of the decision variables.

Lagrangian Function

$$L(\mathbf{x}, \boldsymbol{\lambda}) = F(\mathbf{x}) + \boldsymbol{\lambda}[\mathbf{b} - A(\mathbf{x})]$$

optimize $\rightarrow$

$$\frac{dL}{dx_i} = 0$$

$$\frac{dL}{d\lambda_i} = 0$$

7

If there are m decision variables and n constraints, then optimizing the Lagrangian function with respect to its arguments results in m+n equations in m+n unknowns. Setting the partial derivatives of the Lagrangian function equal to zero results in the Kuhn-Tucker conditions.

A Two-Dimensional Linear Programming Example with Geometric Interpretation

Two-Dimensional Example: Primal Simplex

The following LP has decision variables x and y:

maximize 12x + 19y

subject to
x + 3y <= 225
x + y <= 117
3x + 4y <= 420
x >= 0, y >= 0

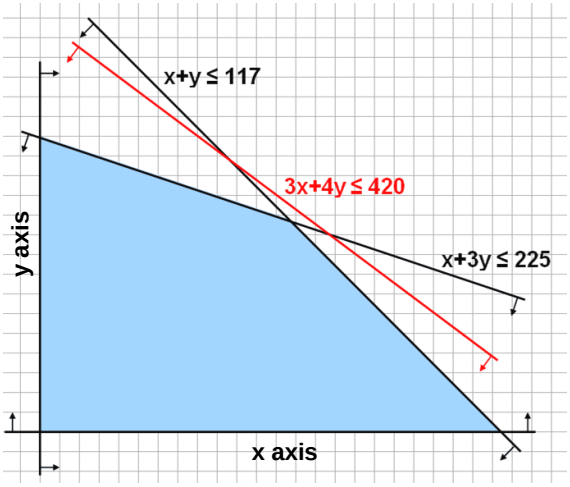
The constraints of the LP determine a *feasible region* in two dimensions.

8

A solution is *feasible* if it satisfies all of the constraints.

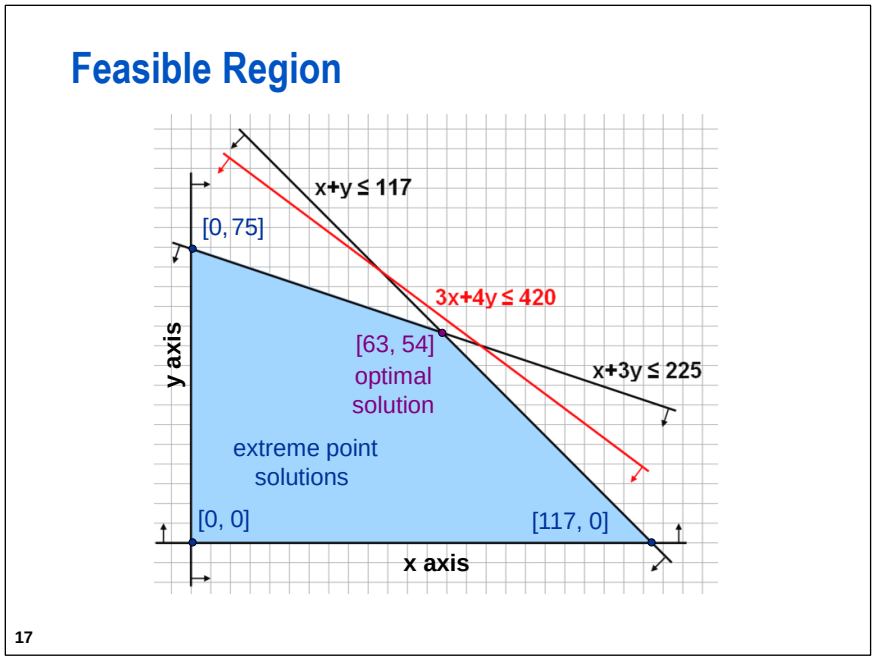
In the figure below, the scale is 1 square = 5 units.

Feasible Region



11 ...

The third constraint 3x + 4y ≤ 420 is redundant. It does not help you determine the feasible region.



An *extreme point* is a "corner" of the feasible region.

An *optimal solution* is a feasible solution that maximizes (or minimizes) the objective function. Because the gradient of a linear function is constant, there is always an extreme point optimal solution if there are any solutions.

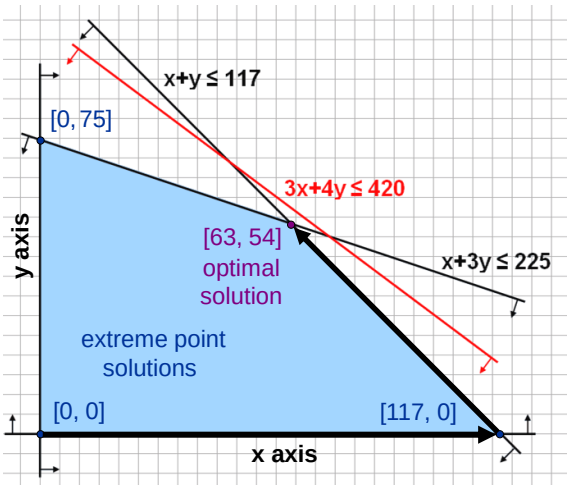
If there are no feasible solutions, the linear programming problem is said to be *infeasible*. For example, if the constraint $3x + 4y \leq 420$ is replaced by the constraint $3x + 4y \geq 420$, the problem is infeasible.

If the objective can be made arbitrarily large (or small in a minimization problem), the linear programming problem is said to be *unbounded*.

If the linear programming problem is unbounded, then the feasible region must also be unbounded, but the converse is not always true. (Consider this two-dimensional linear programming problem without the non-negativity constraints.)

The trajectory of the primal simplex algorithm is the most intuitive and straightforward.

Primal Simplex Trajectory



19

This trajectory is typical for a linear programming problem that starts from an extreme point solution.

2.01 Multiple Answer Poll

How many optimal solutions can a linear programming problem have?

a. no optimal solutions
b. a unique optimal solution
c. more than one, but a finite number of optimal solutions
d. an infinite number of optimal solutions

20

Solving the Two-Dimensional Example

The Lagrangian function can be written as follows:

$$L = 12x+19y + \lambda_1[225 - x - 3y]+\lambda_2[117 - x - y] + \lambda_3[420 - 3x - 4y]$$

9

Optimizing the Lagrangian function is a clever way to optimize the objective function. The product of the constraint and its corresponding shadow price (lambda) parameter is always zero. If a constraint is tight (satisfied), then the shadow price is nonzero. The nonzero shadow price quantifies how much the objective function changes with respect to a unit increase in the right side of the constraint. If the constraint is not satisfied, then the shadow price is zero.

For example, assume that the formulation above is a profit maximization problem, and the second constraint represents the availability of metal in the process of producing x and y. If $x + y = 117$, then the constraint is satisfied, and all of the available metal is used in production. In this case, assume that the shadow price equals 8.5. This indicates that increasing the availability of metal by one unit increases profit by \$8.50. If $x + y < 117$, then all of the available metal is not used in production, and the shadow price is zero. That is, if all available metal is not currently being used in production, then getting an additional unit of metal does not impact profit.

Solving the Two-Dimensional Example

The Kuhn-Tucker conditions:

$$L_x = 12 - \lambda_1 - \lambda_2 - 3\lambda_3 = 0$$
$$L_y = 19 - 3\lambda_1 - \lambda_2 - 4\lambda_3 = 0$$
$$L_{\lambda_1} = 225 - x - 3y = 0$$
$$L_{\lambda_2} = 117 - x - y = 0$$
$$L_{\lambda_3} = 420 - 3x - 4y = 0$$

10

Optimizing the Lagrangian function results in the optimality or Kuhn-Tucker conditions. In this case, there are two decision variables and three constraints, which yield a system of five equations and five unknowns. Values for the decision variables that optimize the objective could be derived fairly quickly with pencil and paper in this instance. However, there is a better tool than these.

Solving a Linear Programming Problem Using PROC OPTMODEL

```
proc optmodel;
   /* declare variables */
   var x >= 0, y >= 0;

   /* declare objective */
   max z = 12*x + 19*y;

   /* declare constraints */
   con x + 3*y <= 225;
   con x + y <= 117;
   con 3*x + 4*y <= 420;

   solve;

   print x y;

quit;
```

The VAR statement is a (decision) variable declaration statement. It can include both upper and lower bounds.

```
   var x >= 0 <= 100;
```

Variable bounds cannot be declared using **ge** or **le**, but these can be used in conditional expressions.

The CON statement (short for CONSTRAINT) is a constraint declaration statement. A constraint can be either an equation or an inequality, and can include both upper and lower bounds (range constraints).

```
   con 100 <= x + 3*y <= 225;
```

When you declare a range constraint, the upper and lower bounds must be numeric expressions.

PROC OPTMODEL Output

```
                    The OPTMODEL Procedure

                       Problem Summary

Objective Sense            Maximization
Objective Function                    z
Objective Type                   Linear

Number of Variables                    2
Bounded Above                          0
Bounded Below                          2
Bounded Below and Above                0
Free                                   0
Fixed                                  0

Number of Constraints                  3
Linear LE (<=)                         3
Linear EQ (=)                          0
Linear GE (>=)                         0
Linear Range                           0
Constraint Coefficients                6
```

```
                    Solution Summary

                Solver                          LP
                Algorithm             Dual Simplex
                Objective Function               z
                Solution Status           Optimal
                Objective Value              1782

                Primal Infeasibility            0
                Dual Infeasibility              0
                Bound Infeasibility             0

                Iterations                      4
                Presolve Time                0.00
                Solution Time                0.00

                          x     y

                         63    54
```

For a problem with linear constraints, the problem summary includes information about the number and type of constraints.

PROC OPTMODEL used the dual simplex algorithm to solve the linear programming problem.

The dual simplex algorithm is the default algorithm for all linear programming problems.

The SOLVE WITH LP statement enables the ALGORITHM= solver option to select a different algorithm such as the primal simplex (PS). This algorithm moves from an extreme point solution to an adjacent extreme point solution until it reaches an optimal solution.

The dual simplex algorithm uses the primal simplex method to solve a dual linear programming problem. The dual problem has a variable for each constraint in the original problem and a constraint for each variable in the original problem. In this case, the dual problem would be a three-dimensional problem. The dual formulation is shown below. Each dual extreme point solution corresponds to the intersection of two inequalities (or bounds) in the two-dimensional linear programming problem.

```
proc optmodel;
   /* declare variables */
   var x >= 0, y >= 0;

   /* declare objective */
   max z = 12*x + 19*y;

   /* declare constraints */
   con x + 3*y <= 225;
   con x + y <= 117;
   con 3*x + 4*y <= 420;

   solve with lp / algorithm=ps;

   print x y;
   print _con_.dual;

quit;
```

The **_con_.dual** option in the PRINT statement outputs the estimates of the shadow prices to the Solution Summary.

PROC OPTMODEL Output (omitting the problem summary)

The OPTMODEL Procedure

Solution Summary

Solver	LP
Algorithm	Primal Simplex
Objective Function	z
Solution Status	Optimal
Objective Value	1782
Primal Infeasibility	0
Dual Infeasibility	0
Bound Infeasibility	0
Iterations	4
Presolve Time	0.00
Solution Time	0.00

x	y
63	54

	CON.
[1]	DUAL
1	3.5
2	8.5
3	0.0

The shadow price on the third constraint is zero. This implies that the third constraint was not satisfied, or $3x + 4y < 425$.

End of Demonstration

2.02 Multiple Choice Poll

What is the default algorithm for linear programming problems?

a. primal simplex algorithm
b. dual simplex algorithm
c. interior point algorithm
d. depends on problem characteristics

23

Exercises

1. Changing the Limit of a Tight Constraint

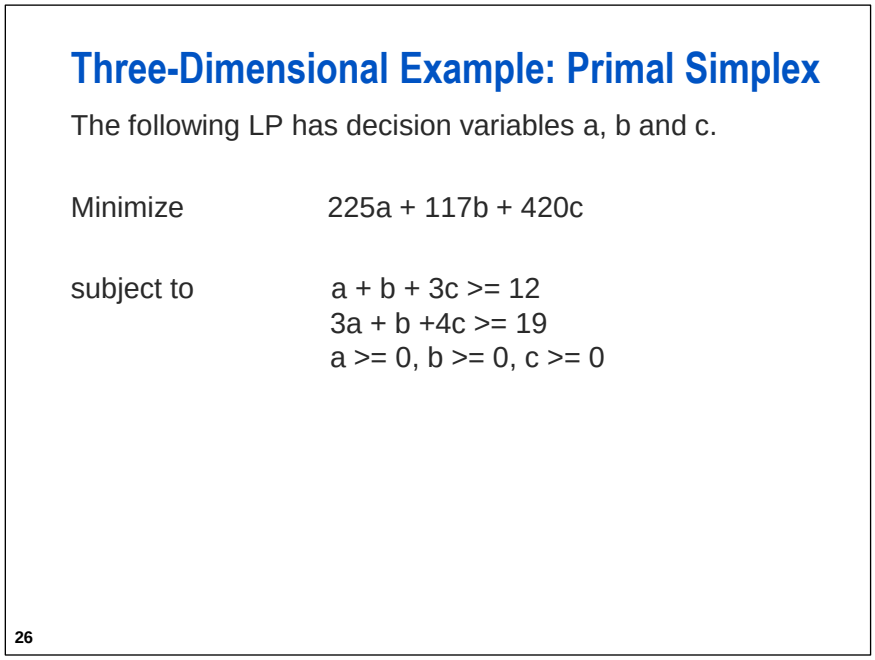
The optimal solution to the two-dimensional linear programming problem has integer values for the decision variables. This does not need to be the case. Change the limit of the first constraint in the demonstration program above. Re-solve the problem. The optimal solution should change. Why?

2. Detecting Infeasible Linear Programming Problems

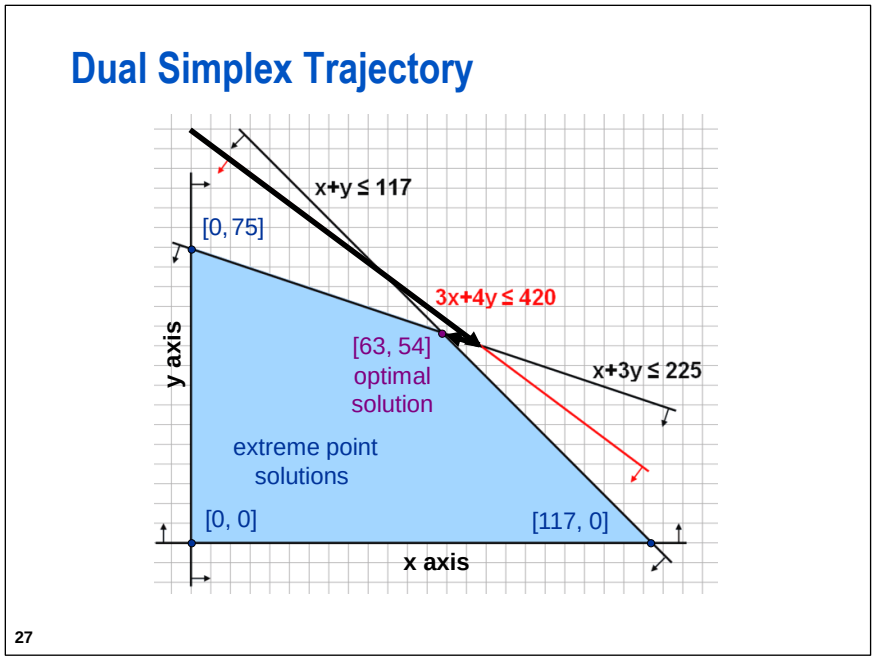
Change the sense of the constraint $3x + 4y \leq 420$ from less-than-or-equal ($\leq$) to greater-than-or-equal ($\geq$). Re-solve the problem. How does PROC OPTMODEL indicate that the resulting problem has no feasible solutions?

End of Exercises

The Dual (Self-Study)



The primal simplex problem shown above is the counterpoint to the primal simplex problem shown earlier in this chapter. Every maximization problem has a counterpoint minimization problem. The furniture-making problem presented below is described in the context of profit maximization. Because the optimal values in the primal and the dual are identical, the problem could be described equivalently as a cost minimization subject to attaining some fixed level of profit. An alternative but equivalent formulation can be useful to applied analysts, because the alternative might represent the problem in a way that is more relevant to business users, and be easier to formulate and solve.



The graph above illustrates solving the first, two-dimensional, primal simplex problem using the dual simplex algorithm. Iterates of the primal simplex algorithm are feasible, but iterates of the dual simplex algorithm are not feasible until an optimal solution is reached.

Solving the Dual Linear Programming Problem Using PROC OPTMODEL (Self-Study)

The problem is recast below in the dual formulation. The dual formulation of the first, two-dimensional, primal simplex problem is solved directly below. (The primal simplex algorithm is used.)

```
proc optmodel;
   /* declare variables */
   var a >= 0, b >= 0, c>= 0;

   /* declare objective */
   min d = 225*a + 117*b + 420*c;

   /* declare constraints */
   con a + b + 3*c >= 12;
   con 3*a + b + 4*c >= 19;

   solve with lp / algorithm=ps;
   print a b c;
   print _con_.dual;

quit;
```

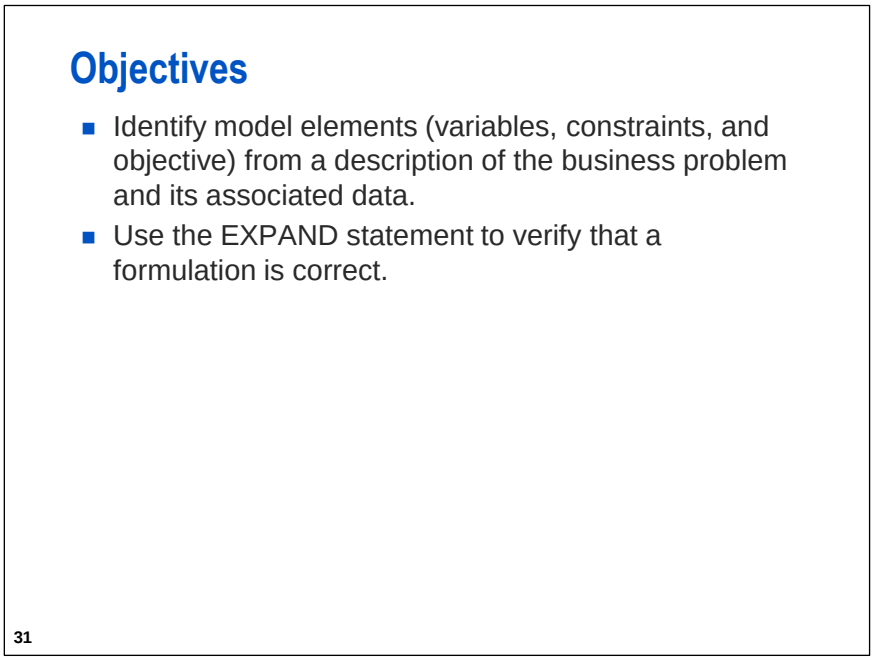
PROC OPTMODEL Output

```
                    The OPTMODEL Procedure

                      Solution Summary

                Solver                            LP
                Algorithm              Primal Simplex
                Objective Function                 d
                Solution Status              Optimal
                Objective Value                 1782

                Primal Infeasibility               0
                Dual Infeasibility                 0
                Bound Infeasibility                0

                Iterations                         5
                Presolve Time                   0.00
                Solution Time                   0.00

                   a      b    c

                 3.5    8.5    0

                          _CON_.
                  [1]       DUAL

                  1           63
                  2           54
```

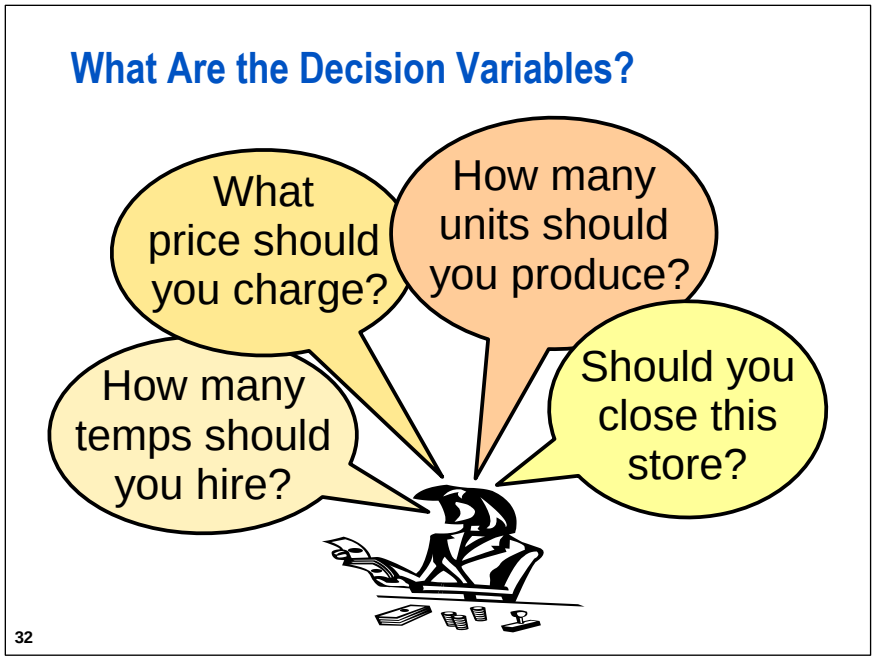
Notice that the optimized value for the objective function is identical to the objective value in the primal simplex formulation, above. Also notice that values for decision variables that optimize the objective in the dual formulation are identical to the shadow price estimates in the primal formulation.

End of Demonstration

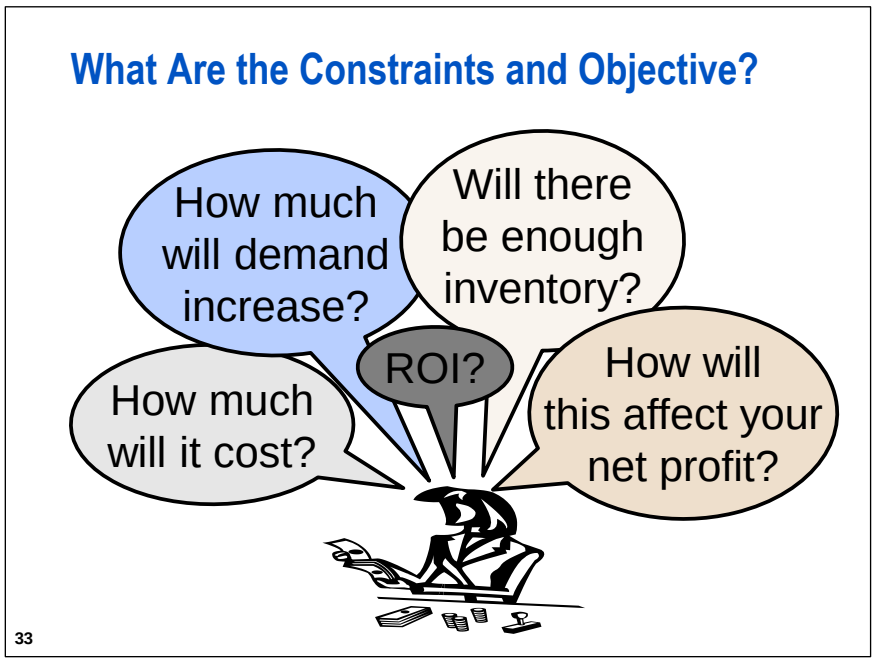
2.2 Formulating and Solving Linear Programming Problems Using the OPTMODEL Procedure



To formulate a linear programming problem, you must first identify the decision variables, the constraints, and the objective. Because the constraints and objective involve functions of the decision variables, the decision variables are, in some sense, the most basic model elements.



Some decision variables are (or can be assumed to be) continuous. Others must necessarily be binary or integers.



In order for the formulation to be solved as a mathematical optimization problem, data must be available at the granularity of the decision variables. This can limit the choice of optimization models.

A Furniture-Making Problem

A furniture-making company can manufacture desks, chairs, bookcases, and bedframes. All of these require various person-hours of labor and units of metal and wood. This information is given in the table below.

	Labor (hrs)	Metal (lbs)	Wood (ft^3)	Selling Price ($)
Desks	2	1	3	94
Chairs	1	1	3	79
Bookcases	3	1	4	125
Bedframes	2	1	4	109

34

A Furniture-Making Problem

The cost and availability of labor, metal, and wood are as follows:

	Labor (hrs)	Metal (lbs)	Wood (ft^3)
Cost ($)	14	20	11
Availability	225	117	420

Assuming all furniture can be sold, how many desks, chairs, bookcases, and bedframes should the company produce per day so that its profit is as large as possible?

35

To formulate the furniture-making problem, you need to identify decision variables first.

A Furniture-Making Problem

The cost and availability of labor, metal, and wood are as follows:

	Labor (hrs)	Metal (lbs)	Wood (ft^3)
Cost ($)	14	20	11
Availability	225	117	420

Assuming all furniture can be sold, how many desks, chairs, bookcases, and bedframes should the company produce per day so that its profit is as large as possible?

- **What should the decision variables be?**

36

The problem statement answers this question.

A Furniture-Making Problem

The cost and availability of labor, metal, and wood are as follows:

	Labor (hrs)	Metal (lbs)	Wood (ft^3)
Cost ($)	14	20	11
Availability	225	117	420

Assuming all furniture can be sold, how many **desks**, **chairs**, **bookcases**, and **bedframes** should the company produce per day so that its profit is as large as possible?

- **What should the decision variables be?**

37

The decision variables can be **Desks**, **Chairs**, **Bookcases**, and **Bedframes**. These names follow the usual SAS naming conventions and are not case sensitive.

2.03 Multiple Answer Poll

Which of the following statements are true of SAS names?

a. SAS names can be up to 32 characters long.
b. SAS names cannot start with a number.
c. SAS names are case sensitive.

38

The data table can be used to help formulate the objective function and the constraints.

Furniture-Making Problem Data

	Labor (hrs)	Metal (lbs)	Wood (ft^3)	Selling Price ($)
Desks	2	1	3	94
Chairs	1	1	3	79
Bookcases	3	1	4	125
Bedframes	2	1	4	109
Cost ($)	14	20	11	
Availability	225	117	420	

- **What should the objective be?**

40 ...

Replace the numbers in the table with the variable expressions that give the amount of each resource that is used to produce desks, chairs, bookcases, and bedframes, and the revenue that they generate.

Furniture-Making Problem Data

	Labor (hrs)	Metal (lbs)	Wood (ft^3)	Selling Price ($)
Desks	2*Desks	Desks	3*Desks	94*Desks
Chairs	Chairs	Chairs	3*Chairs	79*Chairs
Bookcases	3	1	4	125
Bedframes	2	1	4	109
Cost ($)	14	20	11	
Availability	225	117	420	

- **What should the objective be?**

Maximize NetProfit = Revenue – Cost

43

As a typographical convention, the first letter in each word of the names of decision variables and objectives is capitalized, and underscores are not used in the names.

The structure of a typical formulation is basically the same for the two approaches.

Mathematical Optimization Formulations

The basic structure of a typical mathematical optimization problem formulation is shown here:

min|max	objective function
subject to	constraints
	variable bounds

The formulation is easier to understand if it is followed by definitions for the decision variables, sets, and parameters.

44

The corresponding PROC OPTMODEL program uses this structure but presents its elements in reverse order.

PROC OPTMODEL Programs

The basic structure of a PROC OPTMODEL program that solves a mathematical optimization problem is shown here:

```
proc optmodel;
   /* declare sets and parameters */
   /* declare variables */
   /* declare constraints */
   /* declare objective */
   solve;
   /* print solution */
quit;
```

45

PROC OPTMODEL programs have more flexibility than this structure indicates. For example, the objective can be declared before the constraints. Not all programs have all elements.

Formulation Using Named Variables

```
proc optmodel;
   /* declare variables */
      Desks, Chairs, Bookcases, Bedframes
   /* declare constraints */
      availability of labor, metal, wood
   /* declare objective */
      maximize NetProfit = Revenue - Cost
   solve;
   /* print solution */
quit;
```

46

Using PROC OPTMODEL to Solve the Furniture-Making Problem

```
proc optmodel;
   /* declare variables */
   var Desks >= 0, Chairs >= 0, Bookcases >= 0, Bedframes >= 0;

   /* declare constraints */
   con Labor: 2*Desks + Chairs + 3*Bookcases + 2*Bedframes <= 225;
   con Metal: Desks + Chairs + Bookcases + Bedframes <= 117;
   con Wood: 3*Desks + 3*Chairs + 4*Bookcases + 4*Bedframes <= 420;

   /* declare objective */
   max NetProfit =
     94*Desks + 79*Chairs + 125*Bookcases + 109*Bedframes
     - 14 * (2*Desks + Chairs + 3*Bookcases + 2*Bedframes)
     - 20 * (Desks + Chairs + Bookcases + Bedframes)
     - 11 * (3*Desks + 3*Chairs + 4*Bookcases + 4*Bedframes);

   expand;

   solve;

   print Desks Chairs Bookcases Bedframes;

quit;
```

The names follow the usual SAS naming conventions. That is, they can be a maximum of 32 characters, are ***not*** case sensitive, and cannot begin with a number. Avoid names that begin with an underscore. Notice that, in addition to the objective, the constraints are also named.

Model elements must be declared before they are used in subsequent statements. (PROC OPTMODEL executes interactively, unlike many other SAS procedures, and has no RUN statement.)

The EXPAND statement causes PROC OPTMODEL to output a listing of the formulation.

The `%`LET macro statement can be used for recurring expressions in the constraints and objective.

```
%let Hrs = 2*Desks + Chairs + 3*Bookcases + 2*Bedframes;
%let Lbs = Desks + Chairs + Bookcases + Bedframes;
%let Ft3 = 3*Desks + 3*Chairs + 4*Bookcases + 4*Bedframes;
```

PROC OPTMODEL Output

```
                                The OPTMODEL Procedure

Var Desks >= 0
Var Chairs >= 0
Var Bookcases >= 0
Var Bedframes >= 0
Maximize NetProfit=17*Bedframes + 19*Bookcases + 12*Chairs + 13*Desks
Constraint Labor: 2*Bedframes + 3*Bookcases + Chairs + 2*Desks <= 225
Constraint Metal: Bedframes + Bookcases + Chairs + Desks <= 117
Constraint Wood: 4*Bedframes + 4*Bookcases + 3*Chairs + 3*Desks <= 420
```

Problem Summary

Objective Sense	Maximization
Objective Function	NetProfit
Objective Type	Linear
Number of Variables	4
Bounded Above	0
Bounded Below	4
Bounded Below and Above	0
Free	0
Fixed	0
Number of Constraints	3
Linear LE (<=)	3
Linear EQ (=)	0
Linear GE (>=)	0
Linear Range	0
Constraint Coefficients	12

Solution Summary

Solver	LP
Algorithm	Dual Simplex
Objective Function	NetProfit
Solution Status	Optimal
Objective Value	1827
Primal Infeasibility	5.684342E-14
Dual Infeasibility	0
Bound Infeasibility	0
Iterations	5
Presolve Time	0.00
Solution Time	0.00

Desks	Chairs	Bookcases	Bedframes
0	48	39	30

Notice that the EXPAND statement simplified the objective function. The displayed coefficients are the net profits for the four activities. For example, the net profit for making a single desk is as follows:

(Selling Price) – (Cost of Labor) – (Cost of Wood) – (Cost of Metal)

= \$94 – (\$14*2) – (\$20) – (\$11*3) = \$94 – \$81 = \$13

PROC OPTMODEL also enables you to declare ***implicit*** variables for expressions that are used repeatedly in a model. These expressions function much the same as the expressions that are defined using the %LET macro statement.

```
proc optmodel;
   /* declare variables */
   var Desks >= 0, Chairs >= 0, Bookcases >= 0, Bedframes >= 0;

   impvar Hrs = 2*Desks + Chairs + 3*Bookcases + 2*Bedframes;
   impvar Lbs = Desks + Chairs + Bookcases + Bedframes;
   impvar Ft3 = 3*Desks + 3*Chairs + 4*Bookcases + 4*Bedframes;

   /* declare constraints */
   con Labor: Hrs <= 225, Metal: Lbs <= 117, Wood: Ft3 <= 420;

   /* declare objective */
   max NetProfit =
     94*Desks + 79*Chairs + 125*Bookcases + 109*Bedframes
     - 14*Hrs - 20*Lbs - 11*Ft3;

   expand;

   solve;

   print Desks Chairs Bookcases Bedframes;

   print Hrs Lbs Ft3;

quit;
```

PROC OPTMODEL Output (omitting the solution summary)

```
                                The OPTMODEL Procedure

Var Desks >= 0
Var Chairs >= 0
Var Bookcases >= 0
Var Bedframes >= 0
Impvar Hrs = 2*Bedframes + 3*Bookcases + Chairs + 2*Desks
Impvar Lbs = Bedframes + Bookcases + Chairs + Desks
Impvar Ft3 = 4*Bedframes + 4*Bookcases + 3*Chairs + 3*Desks
Maximize NetProfit=- 11*Ft3 - 20*Lbs - 14*Hrs + 109*Bedframes + 125*Bookcases + 79*Chairs +
94*Desks
Constraint Labor: Hrs <= 225
Constraint Metal: Lbs <= 117
Constraint Wood: Ft3 <= 420

                                  Problem Summary

                          Objective Sense           Maximization
                          Objective Function           NetProfit
                          Objective Type                  Linear

                          Number of Variables                4
                          Bounded Above                      0
                          Bounded Below                      4
                          Bounded Below and Above            0
                          Free                               0
                          Fixed                              0
```

```
                          Number of Constraints                3
                          Linear LE (<=)                       3
                          Linear EQ (=)                        0
                          Linear GE (>=)                       0
                          Linear Range                         0

                          Constraint Coefficients             12

                        Desks    Chairs    Bookcases    Bedframes

                            0        48           39           30

                                    Hrs    Lbs    Ft3

                                    225    117    420
```

The problem submitted to the solver is the same with or without implicit variables, because the implicit variables are removed from the problem by substitution before it is solved. Printing the values of the implicit variables shows that all of the available resources are used in the optimal solution.

The SAS log gives additional information about the solution of the linear programming problem.

```
NOTE: The LP solver is called.
NOTE: The Dual Simplex algorithm is used.
                          Objective
      Phase Iteration        Value         Time
       D 1          1    0.000000E+00         0
       D 2          2    6.107188E+03         0
       D 2          5    1.827000E+03         0
NOTE: Optimal.
NOTE: Objective = 1827.
NOTE: The Dual Simplex solve time is 0.00 seconds.
```

The LOGFREQ=*k* solver option controls the frequency of printing to the SAS log. A line is printed in the first iteration of each phase, every *k* iterations, and the last iteration. The default for the interior point algorithm is *k*=1 (every iteration). The default for the primal and dual simplex algorithms is determined dynamically by problem size.

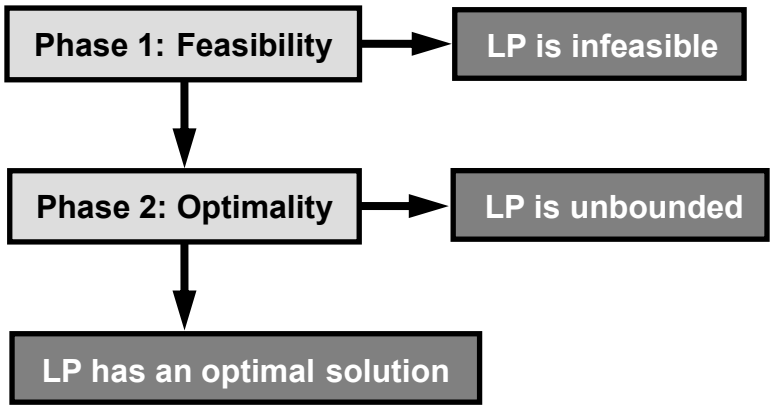
Setting LOGFREQ=1 also yields additional detailed information about the solution process.

```
NOTE: The LP solver is called.
NOTE: The Dual Simplex algorithm is used.
                          Objective                Entering         Leaving
      Phase Iteration        Value         Time     Variable        Variable
       D 1          1    0.000000E+00         0
       D 2          2    6.107188E+03         0    Bedframes       Metal (S)
       D 2          3    2.139760E+03         0       Chairs       Labor (S)
       D 2          4    1.944000E+03         0    Bookcases        Wood (S)
       D 2          5    1.827000E+03         0
NOTE: Optimal.
NOTE: Objective = 1827.
NOTE: The Dual Simplex solve time is 0.00 seconds.
```

In the SAS log, the (S) denotes a slack variable. These are added in the primal and dual simplex algorithms to convert inequalities to equations.

Entering Variable and **Leaving Variable** refer to the set of *basic variables* that are maintained in both the primal and dual simplex algorithms. In the two-dimensional, linear programming problem, basic variables indicate which constraints determine (intersect to form) the extreme point solution in the current iteration.

Phase refers to the two-phase simplex method. Its conceptual flow chart is pictured below for the primal simplex algorithm.



For the dual simplex algorithm, Phase 1 attempts to find a solution that satisfies the optimality conditions (together with the corresponding dual variables). Phase 2 attempts to find a solution that is also feasible (and hence optimal).

End of Demonstration

EXPAND Statement

The EXPAND statement has options to list only a part of the linear programming formulation. This is the statement's syntax:

EXPAND *[identifier-expression] [/ options]*;

Identifier-expression is the name of a variable, objective, or constraint. Options include the following:

VAR outputs decision variables.
IMPVAR outputs referenced implicit variables.
OBJECTIVE|OBJ outputs objectives.
CONSTRAINT|CON outputs constraints.

48

In a PROC OPTMODEL syntax box, italicized brackets denote optional elements.

The simple EXPAND statement produces the same output as this statement:

```
expand / var impvar obj con;
```

A single constraint can be expanded using the following statement:

```
expand Labor;
```

Exercises

3. Adding a Budget Constraint to the Furniture-Making Problem

Add a budget constraint to the previous demonstration program. Specify that the total cost of all the resources that are used in the furniture-making problem is at most $10,000. Expand only this new constraint. Does the optimal objective value change?

End of Exercises

2.3 Using Index Sets and Arrays in the OPTMODEL Procedure

Objectives

- Formulate linear programming problems using index sets and parameter arrays.

52

Next, use arrays and index sets to formulate the furniture-making problem. This requires the problem data to be stored in parameter arrays.

Named Variables versus Arrays and Index Sets

These are approaches to formulating linear programming problems:

Named Variables
■ explicit
■ enter data
■ intuitive

Arrays & Index Sets
■ compact
■ read data
■ flexible

53

Furniture-Making Problem Data

	Labor (hrs)	Metal (lbs)	Wood (ft^3)	Selling Price ($)
Desks	2	1	3	94
Chairs	1	1	3	79
Bookcases	3	1	4	125
Bedframes	2	1	4	109
Cost ($)	14	20	11	
Availability	225	117	420	

Focus on getting the data into PROC OPTMODEL.

Data → **Parameter Arrays** → **Formulation**

54

Parameters and parameter arrays are ubiquitous in PROC OPTMODEL programs.

Parameters and Parameter Arrays

Parameters can be used in expressions defining implicit variables, constraints, and objectives.

- Values can be assigned or read from a SAS data set.
- Values can be listed using the PRINT statement.
- Values are not changed by the solver.

55

By convention, the names of parameters and parameter arrays are in lowercase with underscores to separate words.

Typographical Conventions

SAS names are case insensitive, but conventions can help you distinguish between the various model elements.

Expression	Capitalization
Variable names	First letter of each word
Objective names	First letter of each word
Constraint names	First letter
Parameter names	All lowercase
Set names	All uppercase

In addition, underscores are not used in the names of decision variables, implicit variables, or objectives.

56

2.04 Multiple Choice Poll

Which potential index set for the parameter array **selling_price** is most indicative of the problem context?

	Selling Price ($)
Desks	94
Chairs	79
Bookcases	125
Bedframes	109

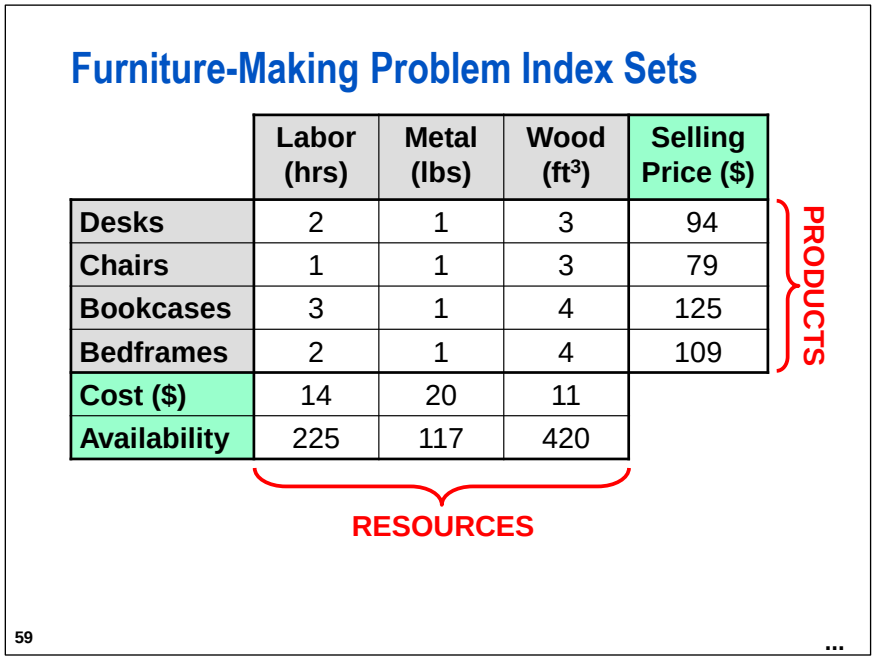
Index Set

a. {1,2,3,4}
b. {'P1','P2','P3','P4'}
c. {'desks','chairs','bookcases','bedframes'}

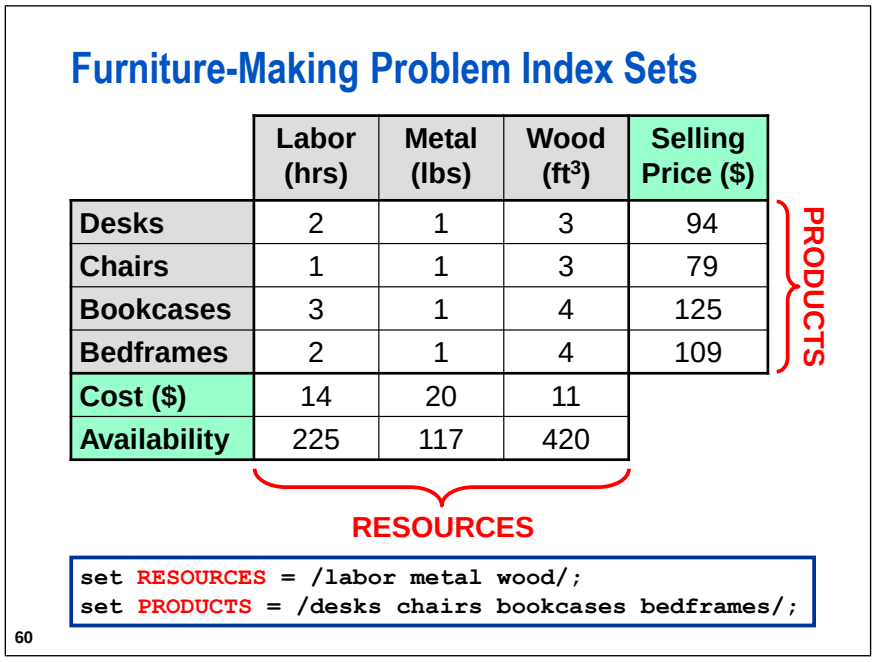
57

Indexing in PROC OPTMODEL is more flexible than array indexing in the SAS DATA step. Index sets can be sets of numbers or strings, and are not required to fit into a rigid sequence.

Index Sets



The sets of products and resources are declared before the parameter arrays.



The sets are initialized using a *set literal expression*, which explicitly lists the elements of a set. Unlike an array, the members of a set cannot be referenced by location. (For example, there is no first or last element of a set.)

Parameters and Parameter Arrays

Furniture-Making Problem Parameters

	Labor (hrs)	Metal (lbs)	Wood (ft^3)	Selling Price ($)
Desks	2	1	3	94
Chairs	1	1	3	79
Bookcases	3	1	4	125
Bedframes	2	1	4	109
Cost ($)	14	20	11	
Availability	225	117	420	

PRODUCTS

RESOURCES

```
set PRODUCTS = /desks chairs bookcases bedframes/;
num selling_price {PRODUCTS} = [94 79 125 109];
```

61

You can say that the parameter array **selling_price** is indexed by **PRODUCTS**. This is another way of saying that there is a selling price for each product.

You use square brackets to initialize the parameter array **selling_price**. The selling price of desks is referenced as **selling_price['desks']**.

Printing a Parameter Array

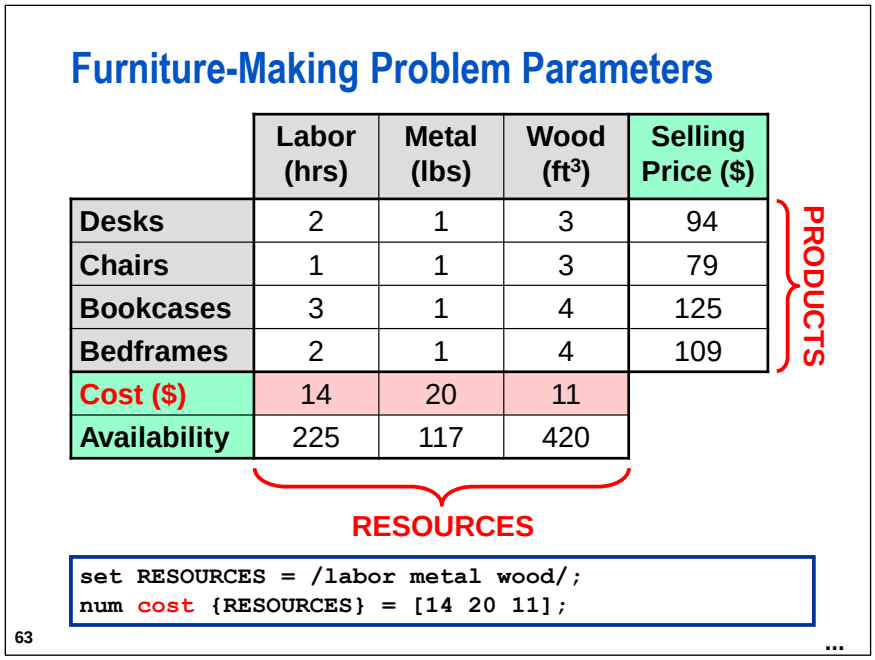
The PRINT statement can be used with arrays.

```
print selling_price;
```

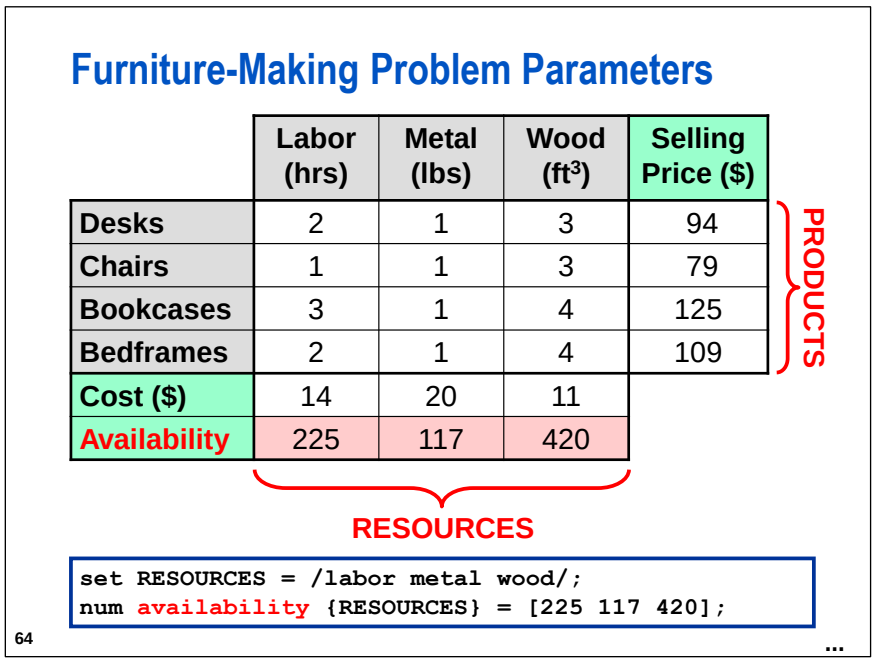
PROC OPTMODEL Output

```
                    selling_
[1]                    price

bedframes                109
bookcases                125
chairs                    79
desks                     95
```

62



The parameter array **cost** is indexed by **RESOURCES**. This is another way of saying that there is a cost for (or associated with) each resource. The set **RESOURCES** is declared only once in the program.



The parameter array **availability** is also indexed by **RESOURCES**. How would you reference the availability of labor?

Furniture-Making Problem Parameters

	Labor (hrs)	Metal (lbs)	Wood (ft^3)
Desks	2	1	3
Chairs	1	1	3
Bookcases	3	1	4
Bedframes	2	1	4

PRODUCTS

RESOURCES

```
set RESOURCES = /labor metal wood/;
set PRODUCTS = /desks chairs bookcases bedframes/;
num required {PRODUCTS, RESOURCES} =
  [2 1 3 1 1 3 3 1 4 2 1 4];
```

65

- Here **required** is a two-dimensional parameter array indexed by **PRODUCTS** and **RESOURCES**.
- The parameter array is initialized rowwise (more precisely, in the order in which the indices appear).
- These values can also be divided into four lines to make the statement easier to parse.

How are parameters declared more generally in PROC OPTMODEL?

Declaring Parameters in PROC OPTMODEL

Parameters must be declared in PROC OPTMODEL ***before*** they are used.

- **NUMBER** or **NUM** declares numeric parameters.

```
num n;
num pi = constant('pi');
num mat {1..n, 1..n} init 0;
```

66

The index set for the two-dimensional parameter array **mat** uses *range expressions* **1..n**, which define a set of consecutive integers. (More general range expressions are possible.)

All except the first parameter are initialized. A value that is assigned using an equal sign in a declaration statement ***cannot*** be modified. A value that is assigned using the INIT keyword ***can*** be modified later in the program.

Assignment Statements in PROC OPTMODEL

identifier-expression = *expression*;

Assignment statements can be used to assign values to decision variables and parameters.

```
num n;
num pi = constant('pi');
num mat {1..n, 1..n} init 0;

n = 1;
mat[1,1] = pi;
n = n - 1;
```

67

Declaring Parameters in PROC OPTMODEL

Parameters must be declared in PROC OPTMODEL ***before*** they are used.

- **NUMBER** or **NUM** declares numeric parameters.

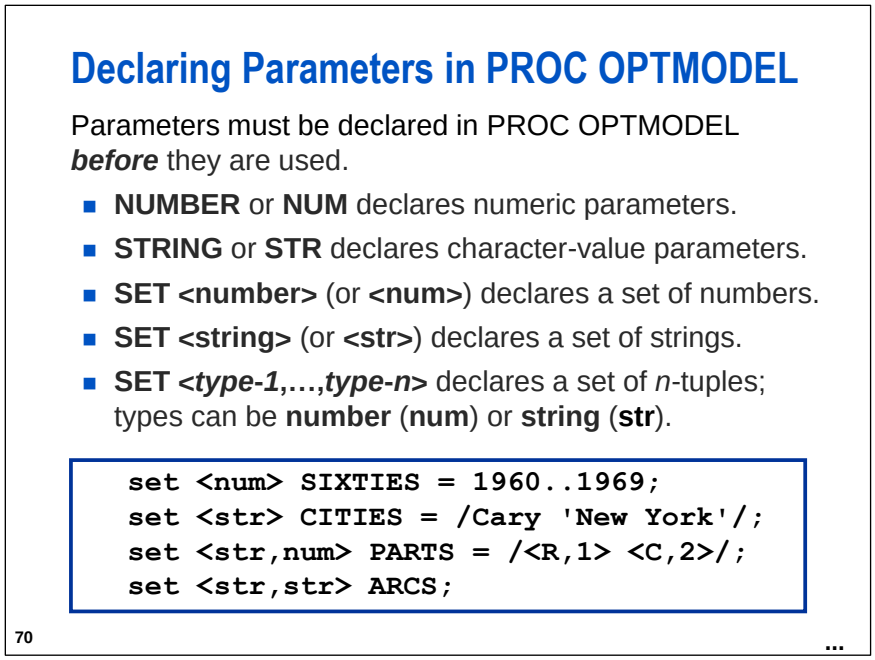
```
num n;
num pi = constant('pi');
num mat {1..n, 1..n} init 0;
```

- **STRING** or **STR** declares character-value parameters.

```
str name;
str day {1..5} = [Mo Tu We Th Fr];
str type {RESOURCES};
```

69

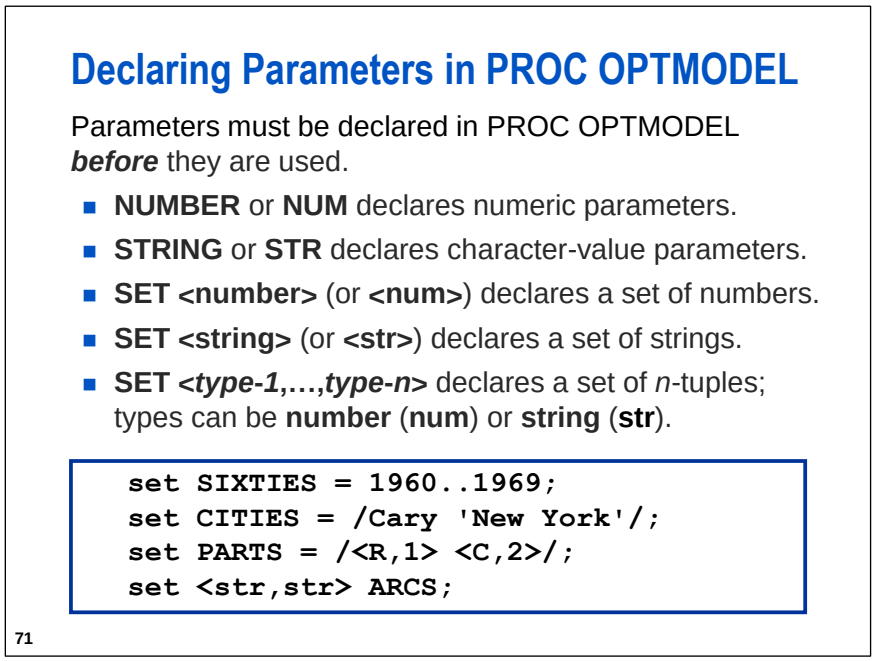
Braces are used to enclose a set, and ***brackets*** are used to denote array locations.



SET is used very differently in PROC OPTMODEL than in SAS DATA step programming.

Set literal expressions are used to initialize the second and third sets.

Angle brackets around set elements are necessary only for tuples.



The *type* of a set does not need to be declared if the set is initialized. (The default type for sets is **<num>**.)

Arrays of sets can also be declared in PROC OPTMODEL.

2.05 Multiple Answer Poll

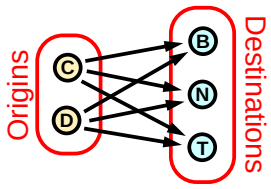
Which of the following statements are true of parameters and parameter arrays in PROC OPTMODEL?

a. Parameters must be declared before they are used.
b. The type of a set must be declared.
c. Parameters must be initialized when they are declared.
d. All parameter values can be modified after they are declared.
e. Parameter values cannot be modified by the solver.

72

Example: Transportation Problem

Given supply values at a set of origins, demand values at destinations, and unit shipping costs, the *transportation problem* is to determine the amount to transport from each origin to each destination to meet demand at minimum cost.



	Boston	Newark	Toronto	Supply
Chicago	40	25	15	200
Detroit	30	15	10	150
Demand	50	150	100	

75

...

The decision variables are the amount to ship from each origin to each destination. Here, an approach using index sets and arrays is more natural than one that uses named variables.

Example: Transportation Problem

Given supply values at a set of origins, demand values at destinations, and unit shipping costs, the *transportation problem* is to determine the amount to transport from each origin to each destination to meet the demand at minimum cost.

	Boston	Newark	Toronto	Supply
Chicago	40	25	15	200
Detroit	30	15	10	150
Demand	50	150	100	

ORIGINS

DESTINATIONS

```
set ORIGINS = /Chicago Detroit/;
set DESTINATIONS = /Boston Newark Toronto/;
```

77

These index sets are used to index the parameter arrays for the transportation problem.

2.06 Transportation Problem

The parameter declaration statements are as follows:

```
set ORIGINS = /Chicago Detroit/;
set DESTINATIONS = /Boston Newark Toronto/;
num supply {ORIGINS} = [200 150];
num demand {DESTINATIONS} = [50 150 100];
num unit_cost {ORIGINS, DESTINATIONS} =
  [40 25 15 30 15 10];
```

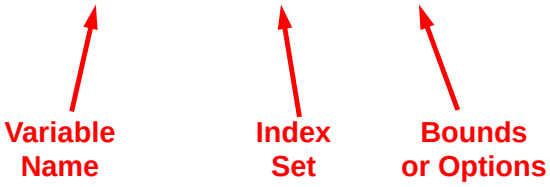
78

Decision Variables

Declaring an Array of Decision Variables

The syntax for declaring an array of decision variables follows the same pattern as declaring a parameter array.

```
var NumProd {PRODUCTS} >= 0;
```



In the formulation that uses arrays and index sets, the decision variable **Desks** becomes **NumProd['desks']**.

79

The index set is enclosed in braces. The non-negativity is applied to the entire array of decision variables. Options for decision variables include declaring integrality (ILP/MILP) and setting initial values (NLP).

A decision variable is declared for every element of the set **PRODUCTS**. If an element is later added to the set **PRODUCTS**, then the corresponding decision variable is declared by this statement. This is convenient for column-generation methods.

Declaring Decision Variables

VAR *var-declaration [, … , var-declaration]* **;**

The syntax for a *var-declaration* is

name [**{** *index-set* **}** *] [var-option(s)]*

The available *var-options* are as follows:

- *>= expression*
- *<= expression*
- **INIT** *expression*
- **INTEGER**
- **BINARY**

80

Decision variables must be declared before they are used.

Formulation Using Arrays and Index Sets

```
proc optmodel;
   /* declare sets and parameters */
     RESOURCES, PRODUCTS, cost …
   /* declare variables */
     NumProd {PRODUCTS}
   /* declare constraints */
   /* declare objective */
   solve;
   /* print solution */
quit;
```

81

The other declaration statements, including implicit variables, generally require summations.

Implicit Variable Details, Summation, and Local Dummy Parameters

SUM Aggregation Operator

The SUM aggregation operator can be used to add numeric or variable expressions.

```
print ( sum {k in 1..24} k**2 );
```

PROC OPTMODEL Output

```
4900
```

In this sum, **k** is a *local dummy parameter*. Local dummy parameters do ***not*** need to be declared.

83 …

The parentheses in the PRINT statement signal PROC OPTMODEL to evaluate the expression to compute and output the value 4900.

SUM Aggregation Operator

The SUM aggregation operator can be used to add numeric or variable expressions.

```
print ( sum {k in 1..24} k**2 );

impvar Revenue = sum {p in PRODUCTS}
   selling_price[p] * NumProd[p];
```

Parentheses,
Exponents,
Multiplication and
Division,
Addition and
Subtraction

The SUM aggregation operator goes here in the order of operations.

84

In the variable expression, **p** is a local dummy parameter.

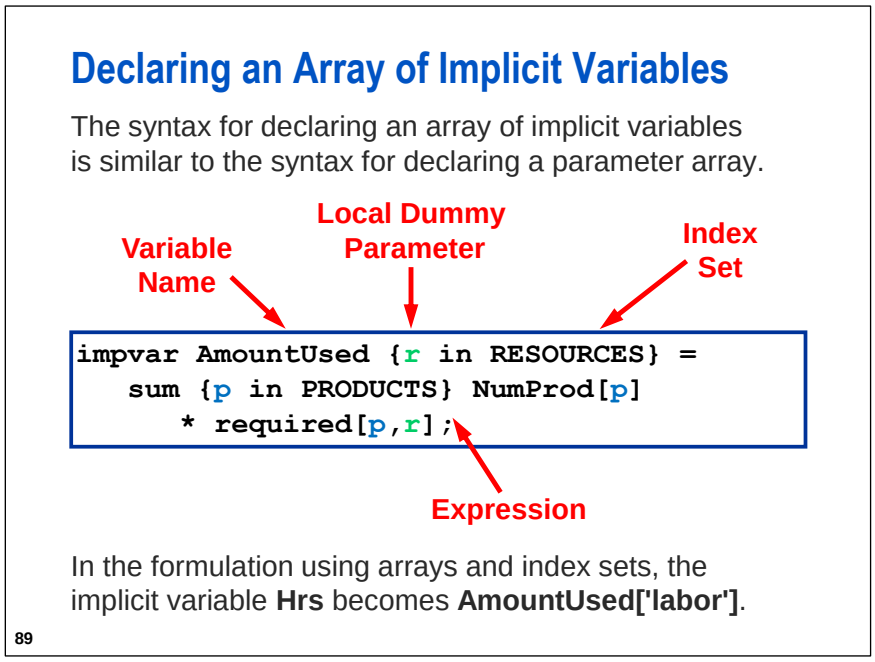
2.07 Multiple Choice Poll

Which of the following does ***not*** need to be declared in PROC OPTMODEL?

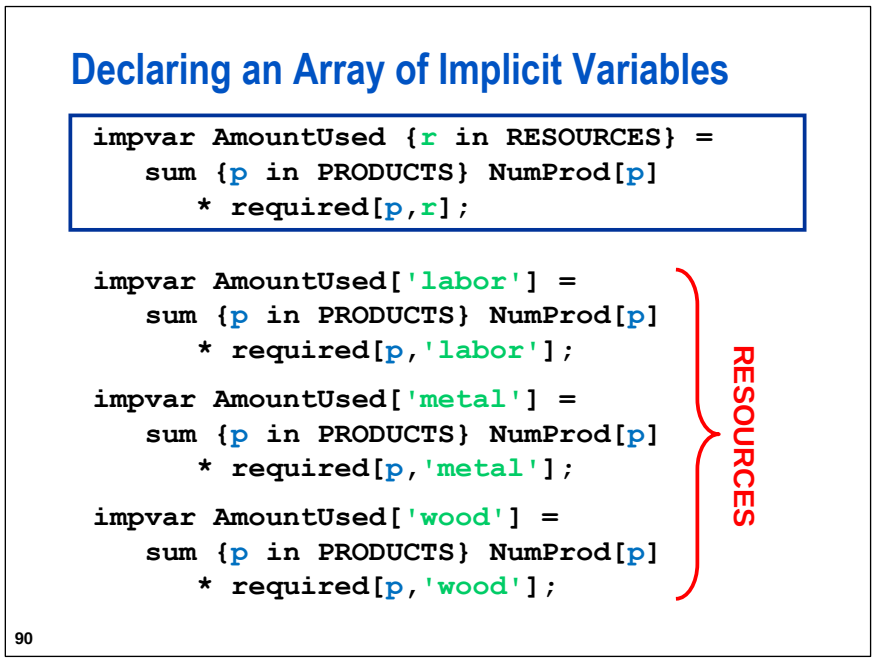
a. parameter arrays
b. decision variables
c. implicit variables
d. local dummy parameters

86

You can also declare an array of implicit variables.



The index set is enclosed in braces. The local dummy parameter **r** is required to make the expression apply to a particular resource, whereas the local dummy parameter **p** is an index of summation.



An implicit variable is declared for every element of the set **RESOURCES**. If an element is later added to the set **RESOURCES**, the corresponding implicit variable is declared by this statement.

Implicit variables can be used to simplify variable expressions and for reporting quantities of interest. They are eliminated from the model by substitution before the problem is solved, and their values are computed after the problem is solved.

Declaring Implicit Variables

IMPVAR *name* [**{** *index-set* **}**] = *expression* **;**

The expression can involve decision variables and other (previously declared) implicit variables.

The expression for an array of implicit variables can be defined recursively, as long as the recursion is not circular.

91

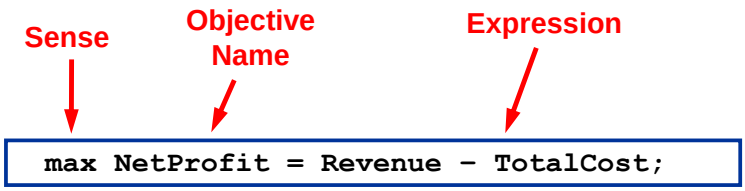
Implicit variables must be declared before they are used.

Declaring the Objective Function

The implicit variables declared for the furniture-making problem are used to help declare the objective and constraints.

Declaring an Objective

The objective function for the furniture-making problem can be declared as the difference between the implicit variables **Revenue** and **TotalCost**.



92

Without implicit variables, the variable expression in this objective would be more complicated. It would involve summations, parameters, and decision variables.

```
max NetProfit = sum {p in PRODUCTS} ( selling_price[p] -
    sum {r in RESOURCES} required[p,r] * cost[r] ) * NumProd[p];
```

The linear programming problem solved is the same with or without implicit variables.

Declaring Objectives

MAX|MIN *name* [**{** *index-set* **}**] = *expression* **;**

Multiple objective functions are permitted. The most recent objective is used unless an objective is specified using the OBJECTIVE (abbreviated OBJ) option.

SOLVE OBJECTIVE|OBJ *name* **;**

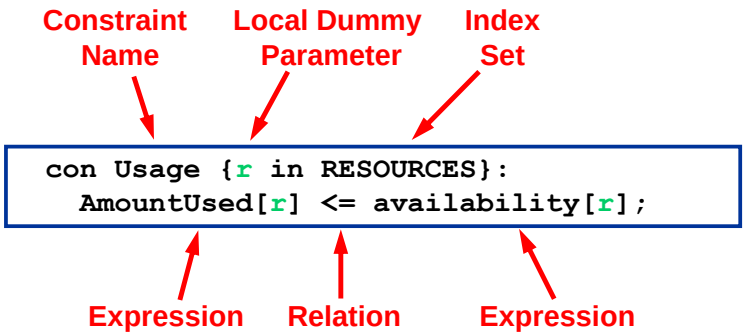
You can reference objective names in the same way as implicit variables.

93

Objectives must be declared before they are used. Arrays of objectives can arise in "What-if?" analysis.

Declaring Constraints

Declaring a Family of Constraints



In the formulation using arrays and index sets, the constraint **Labor** becomes **Usage['labor']**.

95

The local dummy parameter **r** is used to make the constraint particular to a specific resource.

Declaring a Family of Constraints

```
con Usage {r in RESOURCES}:
   AmountUsed[r] <= availability[r];
```

```
con Usage['labor']:
   AmountUsed['labor']
      <= availability['labor'];
con Usage['metal']:
   AmountUsed['metal']
      <= availability['metal'];
con Usage['wood']:
   AmountUsed['wood']
      <= availability['wood'];
```

RESOURCES

96

A constraint is declared for every element of the set **RESOURCES**. If an element is later added to the set **RESOURCES**, then the corresponding constraint is declared by this statement. This is a convenient way to implement methods that involve constraint generation.

Declaring Constraints

CONSTRAINT *constraint [, … , constraint]* **;**

CONSTRAINT can be abbreviated as **CON**.
The allowed forms of *constraint* are shown here:

[name [**{** *index-set* **}** *] :] expression = expression*
[name [**{** *index-set* **}** *] :] expression relation expression*
[name [**{** *index-set* **}** *] :] bound relation body relation bound*

For range constraints, *relation* must be either <= or >= and the same in both positions. If no name is provided, a default name of the form **_ACON_ [*n*]** is assigned.

97

Constraints must be declared before they are used.

PROC OPTMODEL automatically declares symbols that are aliases for model components. All decision variables and constraints in a model can be referred to anonymously, as **_VAR_[n]** for **n** in **{1.._NVAR_}** and **_CON_[n]** for **n** in **{1.._NCON_}**. The current objective can also be referred to as **_OBJ_**.

Using Arrays and Index Sets in PROC OPTMODEL to Solve the Furniture-Making Problem

```
proc optmodel;
   /* declare sets and parameters */
   set RESOURCES = /labor metal wood/;
   set PRODUCTS = /desks chairs bookcases bedframes/;
   num selling_price {PRODUCTS} = [94 79 125 109];
   num cost {RESOURCES} = [14 20 11];
   num availability {RESOURCES} = [225 117 420];
   num required {PRODUCTS, RESOURCES} =
     [2 1 3 1 1 3 3 1 4 2 1 4];

   /* declare variables */
   var NumProd {PRODUCTS} >= 0;

   impvar Revenue = sum {p in PRODUCTS}
      selling_price[p] * NumProd[p];

   impvar AmountUsed {r in RESOURCES} =
      sum {p in PRODUCTS} NumProd[p] * required[p,r];

   impvar TotalCost = sum {r in RESOURCES}
      cost[r] * AmountUsed[r];

   /* declare constraints */
   con Usage {r in RESOURCES}:
      AmountUsed[r] <= availability[r];

   /* declare objective */
   max NetProfit = Revenue - TotalCost;

   expand / var impvar;
   expand NetProfit;
   expand Usage;

   solve;

   print NumProd;
   print AmountUsed;

quit;
```

Notice the index set in the constraint declaration and its absence in the last EXPAND statement. The local dummy parameter in the constraint declaration statement is necessary to make each constraint in the family refer to a specific index (in this case, a specific resource).

In the output from the EXPAND statements, the indices are not enclosed in quotation marks. Inside the PROC OPTMODEL program, the quotation marks are necessary to distinguish between a name expression (no quotation marks) and a character string (quotation marks).

PROC OPTMODEL Output

The OPTMODEL Procedure

```
Var NumProd[desks] >= 0
Var NumProd[chairs] >= 0
Var NumProd[bookcases] >= 0
Var NumProd[bedframes] >= 0
Impvar Revenue = 94*NumProd[desks] + 79*NumProd[chairs] + 125*NumProd[bookcases] +
109*NumProd[bedframes]
Impvar AmountUsed[labor] = 2*NumProd[desks] + NumProd[chairs] + 3*NumProd[bookcases] +
2*NumProd[bedframes]
Impvar AmountUsed[metal] = NumProd[desks] + NumProd[chairs] + NumProd[bookcases] +
NumProd[bedframes]
Impvar AmountUsed[wood] = 3*NumProd[desks] + 3*NumProd[chairs] + 4*NumProd[bookcases] +
4*NumProd[bedframes]
Impvar TotalCost = 14*AmountUsed[labor] + 20*AmountUsed[metal] + 11*AmountUsed[wood]
Maximize NetProfit=- TotalCost + Revenue
Constraint Usage[labor]: AmountUsed[labor] <= 225
Constraint Usage[metal]: AmountUsed[metal] <= 117
Constraint Usage[wood]: AmountUsed[wood] <= 420
```

Problem Summary

Objective Sense	Maximization
Objective Function	NetProfit
Objective Type	Linear
Number of Variables	4
Bounded Above	0
Bounded Below	4
Bounded Below and Above	0
Free	0
Fixed	0
Number of Constraints	3
Linear LE (<=)	3
Linear EQ (=)	0
Linear GE (>=)	0
Linear Range	0
Constraint Coefficients	12

Solution Summary

Solver	LP
Algorithm	Dual Simplex
Objective Function	NetProfit
Solution Status	Optimal
Objective Value	1827
Primal Infeasibility	0
Dual Infeasibility	0
Bound Infeasibility	0
Iterations	6
Presolve Time	0.00
Solution Time	0.00

[1]	Num Prod
bedframes	30
bookcases	39
chairs	48
desks	0

[1]	Amount Used
labor	225
metal	117
wood	420

Compare this to the output from the variable names formulation in the demonstration above.

PROC OPTMODEL Output (omitting the problem and solution summaries)

The OPTMODEL Procedure

```
Var Desks >= 0
Var Chairs >= 0
Var Bookcases >= 0
Var Bedframes >= 0
Impvar Hrs = 2*Bedframes + 3*Bookcases + Chairs + 2*Desks
Impvar Lbs = Bedframes + Bookcases + Chairs + Desks
Impvar Ft3 = 4*Bedframes + 4*Bookcases + 3*Chairs + 3*Desks
Maximize NetProfit=- 11*Ft3 - 20*Lbs - 14*Hrs + 109*Bedframes + 125*Bookcases + 79*Chairs +
94*Desks
Constraint Labor: Hrs <= 225
Constraint Metal: Lbs <= 117
Constraint Wood: Ft3 <= 420
```

Desks	Chairs	Bookcases	Bedframes
0	48	39	30

Hrs	Lbs	Ft3
225	117	420

End of Demonstration

Exercises

4. Formulating a Transportation Problem

Consider the transportation problem with unit shipping costs, supply, and demand in the table below.

DESTINATIONS

ORIGINS		Boston	Newark	Toronto	Supply
	Chicago	40	25	15	200
	Detroit	30	15	10	150
	Demand	50	150	100	

a. Complete the parameter declaration statements in part **4a** of the program **ch2ex.sas**. Replace each *INDEX-SET* with the appropriate (declared) index set and *INITIALIZERS* with the appropriate data items from the table. This assigns values to a two-dimensional array **unit_cost** of unit costs, an array **supply** of supplies, and an array **demand** of demands. How does PROC OPTMODEL format the output of the PRINT statement in the SAS program?

b. Complete the variable declaration statements in part **4b** of the program **ch2ex.sas**. Replace each *INDEX-SET* with the appropriate (declared) index set and *EXPRESSION* with the appropriate variable expression. (The decision variables are initialized to 50 in order to print values for the implicit variables.)

c. Complete the constraint and objective declaration statements in part 4c of the program ch2ex.sas and solve the transportation problem using PROC OPTMODEL. Use the EXPAND statement to check your formulation.

Hint: The optimal objective value is $5,750.

End of Exercises

2.4 Dual Values and Reduced Costs in the Simplex Method (Self-Study)

Objectives

- Explain the interpretation of dual values in linear programming.
- Describe how dual values are used in the primal simplex algorithm and how pricing options influence the behavior of the simplex algorithms (Self-Study).

102

The dual value of a constraint measures the per-unit change in the optimal objective value as the limit of the constraint is changed.

Interpretation of Dual Values

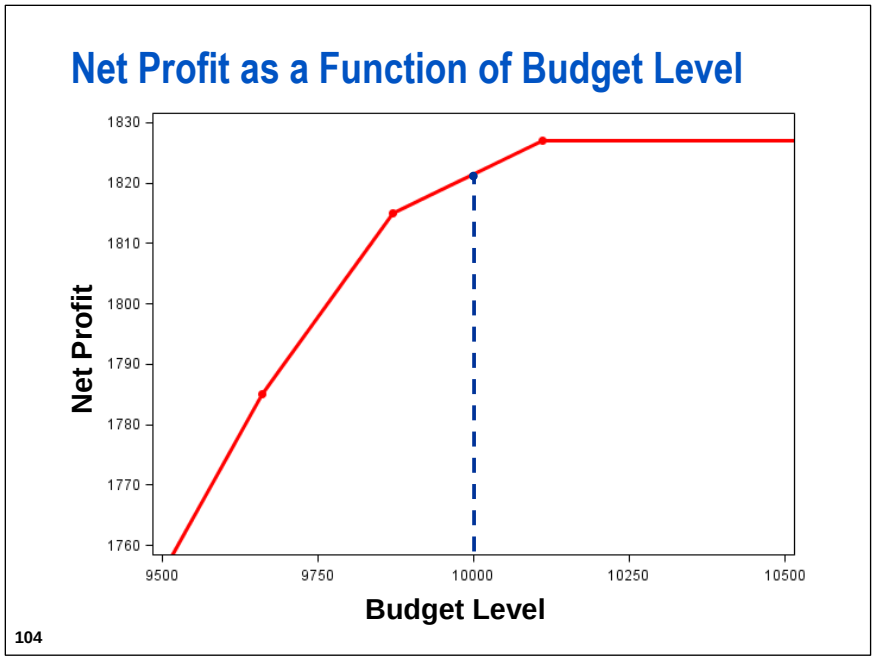
- The *dual value* of a constraint is defined as follows:

$$\text{Dual Value} = \frac{\text{Change in optimal objective}}{\text{Unit increase in constraint limit}}$$

- This assumes that the extreme point determining the optimal solution is not overdetermined.

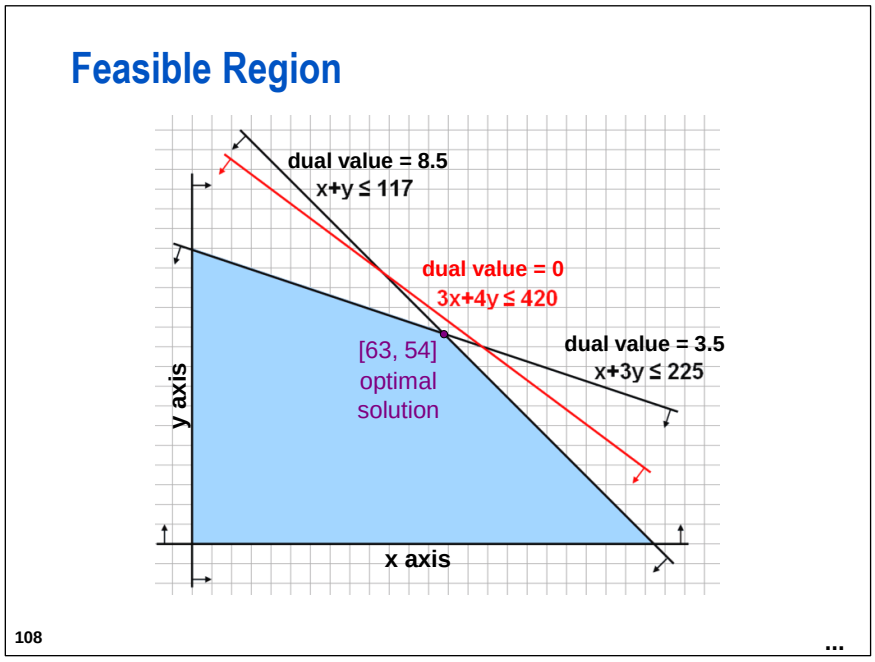
106

At the apex of the pyramid, four planes intersect in a single point (one more than is necessary).



The dual value of the budget constraint gives the slope of the graph at the current value of 10,000.

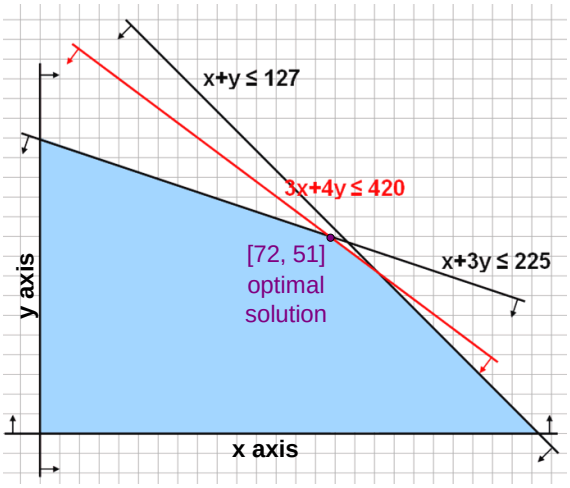
The dual values for the two-dimensional linear programming problem are pictured below.



The constraint that is not tight (satisfied with equality) at the optimal solution has a dual value of zero.

How does the dual value of the constraint $x + y \leq 117$ predict the change in the optimal objective value if the limit of the constraint is increased to 127?

Feasible Region



114

The predicted change in the objective value is accurate only in a neighborhood of the optimal solution. However, it does ***bound*** the change in the objective value:

$$12(72) + 19(51) = 1833 \leq 1782 + 8.5(10) = 1867$$

The latter quantity is the objective value of the point [x,y] = [78,49], which is the intersection of the lines $x + y = 127$ and $x + 3y = 225$.

Dual Values for the Furniture-Making Problem

Dual values can be listed using the PRINT statement.

```
print Usage.dual;

print Budget.dual;
```

PROC OPTMODEL Output

[1]	Usage. DUAL
labor	1.30
metal	0.00
wood	2.45

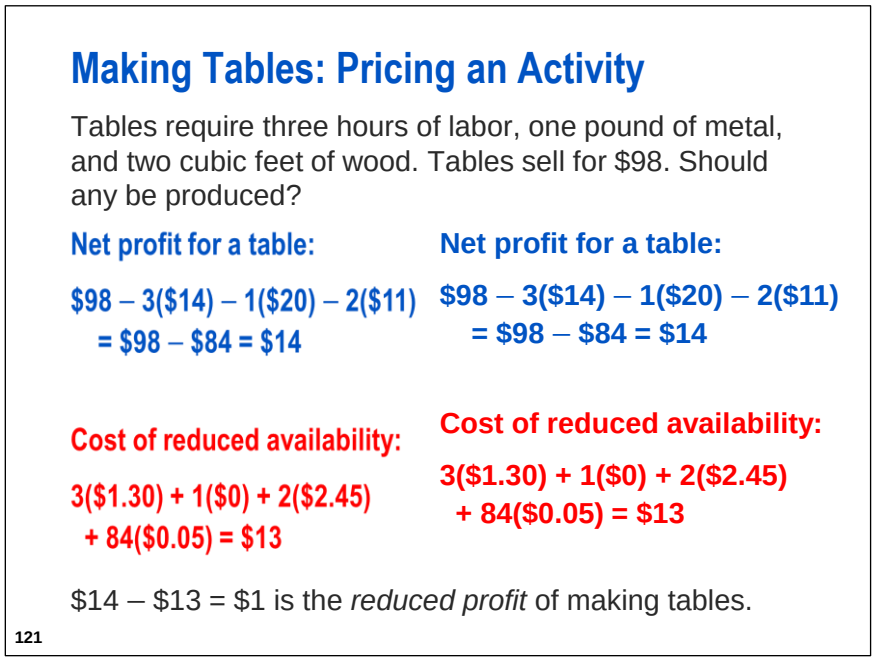
Budget. DUAL
0.05

If additional overtime hours are available for $21 (time-and-a-half), would they be used?

No. For an additional hour at $21 = $14 + $7, the optimal objective value changes by (at most) $1.30 – $7 = –$5.70.

118

Interpreting dual values can be tricky. Here, increasing the limit of the resource usage constraints increases only the ***availability*** of the resources at their current cost.



The numbers in parentheses that are used to compute the cost of reduced availability are the dual values for the availability of labor, metal, and wood, and the budget constraint. For labor, the dual value $1.30 is multiplied by 3 because each table requires three hours of labor.

Making tables appears to be a profitable activity. However, the dual values themselves cannot inform the decision about the most profitable mix of tables and other products.

Due to a series of unfortunate events, the canonical form of a linear programming problem ***minimizes*** the objective function, so the computed quantity is also referred to as the *reduced cost* of making tables. Here, it is **profit** that is reduced.

For a variable in the model that is at its upper or lower bound, the reduced profit or cost is the amount that the variable's objective function coefficient must ***decrease*** before the variable can change from its current value. For example, in the furniture-making problem, the reduced profit of **NumProd['desks']** is −1, so the selling price of desks must increase by $1 (decrease by −$1) before any desks are produced.

Dual Values and Reduced Profits in the Furniture-Making Problem

```
%let budget_limit = 10000;

proc optmodel;
   /* declare sets and parameters */
   set RESOURCES = /labor metal wood/;
   set PRODUCTS = /desks chairs bookcases bedframes/;
   num selling_price {PRODUCTS} = [94 79 125 109];
   num cost {RESOURCES} = [14 20 11];
   num availability {RESOURCES} = [225 117 420];
   num required {PRODUCTS, RESOURCES} =
     [2 1 3 1 1 3 3 1 4 2 1 4];

   /* declare variables */
   var NumProd {PRODUCTS} >= 0;

   impvar Revenue = sum {p in PRODUCTS}
      selling_price[p] * NumProd[p];

   impvar AmountUsed {r in RESOURCES} =
      sum {p in PRODUCTS} NumProd[p] * required[p,r];

   impvar TotalCost = sum {r in RESOURCES}
      cost[r] * AmountUsed[r];

   /* declare constraints */
   con Usage {r in RESOURCES}:
      AmountUsed[r] <= availability[r];
      /* Note, Budget constraint from the exercise is added */
      con Budget: TotalCost <= &budget_limit;

   /* declare objective */
   max NetProfit = Revenue - TotalCost;

   solve;

   print NumProd NumProd.dual;
   print AmountUsed availability Usage.dual;
   print TotalCost Budget.ub Budget.dual;
   print _VAR_.name _VAR_.dual;
   print _CON_.name _CON_.body _CON_.ub _CON_.dual;
quit;
```

PROC OPTMODEL Output

The OPTMODEL Procedure

Solution Summary

Solver	LP
Algorithm	Dual Simplex
Objective Function	NetProfit
Solution Status	Optimal
Objective Value	1821.5
Primal Infeasibility	5.684342E-14
Dual Infeasibility	0
Bound Infeasibility	0
Iterations	6
Presolve Time	0.00
Solution Time	0.00

[1]	Num Prod	Num Prod. DUAL
bedframes	57.5	0
bookcases	28.0	0
chairs	26.0	0
desks	0.0	-1

[1]	Amount Used	availability	Usage. DUAL
labor	225.0	225	1.30
metal	111.5	117	0.00
wood	420.0	420	2.45

Total Cost	Budget.UB	Budget. DUAL
10000	10000	0.05

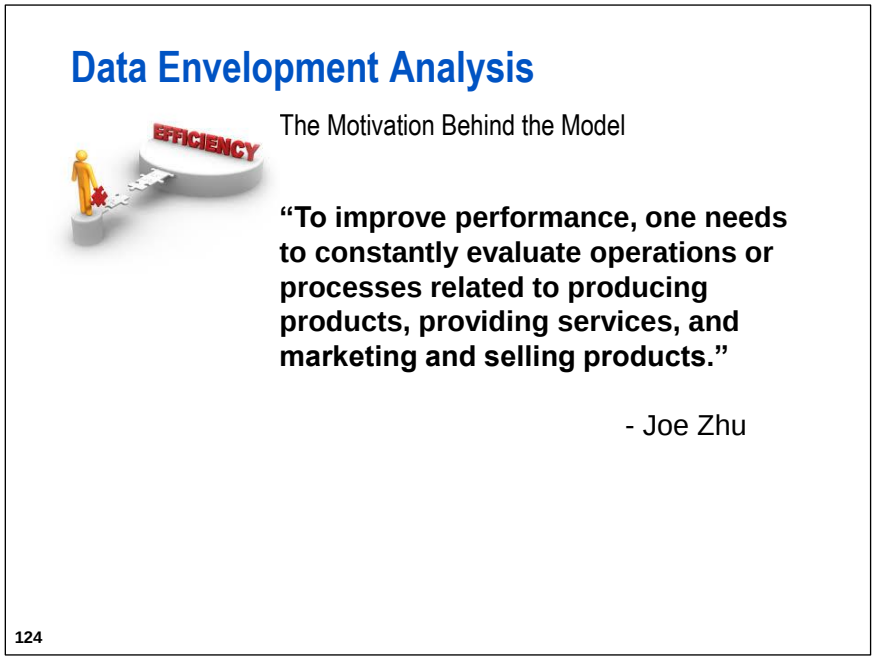
[1]	_VAR_.NAME	_VAR_. DUAL
1	NumProd[desks]	-1
2	NumProd[chairs]	0
3	NumProd[bookcases]	0
4	NumProd[bedframes]	0

The OPTMODEL Procedure

[1]	_CON_.NAME	_CON_. BODY	_CON_.UB	_CON_. DUAL
1	Usage[labor]	225.0	225	1.30
2	Usage[metal]	111.5	117	0.00
3	Usage[wood]	420.0	420	2.45
4	Budget	10000.0	10000	0.05

End of Demonstration

2.5 Applied Data Envelopment Analysis



For years, economists examined the relationship between the outputs produced by firms and the resources available for production, which is a concept known as the *production function*. New fields of study related to operations research grew to focus on mathematical processes that support business objectives and inform business decisions. Companies continuously apply mathematical techniques to answer two valuable questions:

1. What is the ***best*** way to operate?
2. Is our current way of doing business, the ***best*** way of doing business?

Answering these two questions can help an organization or company improve performance to maintain their competitiveness. As Joe Zhu states in his 2009 book, *Quantitative Models for Performance Evaluations and Benchmarking*:

"Managers are often under great pressure to improve the performance of their organizations. To improve performance, one needs to constantly evaluate operations or processes related to producing products, providing services, and marketing and selling products."

Notice that the word *best* is used in both questions and for good reason. *Best*, as used above, changes with technology and is unique to each industry, company, or organization. Changing technologies and unknown relationships can make it difficult to specify a functional form of the relationship between inputs and outputs. To further complicate the answer, most problems or processes involve a multitude of inputs or decisions that need to be changed in a way that yields the Pareto optimal solution for multiple considerations or goals. For these reasons, the word *best* generalizes well to encompass all dynamic interactions involved in operating an organization. However, if you seek to answer the questions, *best* is simply too vague. It should be replaced with a more specific set of terms that can be used to develop surrogate questions that you can answer.

A solution is Pareto Optimal if you cannot do better at accomplishing one goal without doing worse on at least one other goal.

Questions Companies Ask

1) "What is the ***most efficient*** way to operate?"

2) "Is our current way of doing business, the ***most efficient*** way of doing business?

126

Narrow your focus to *efficiency* and specify both questions with a replacement of the word *best* with the words *most efficient*. With this change, you can direct your efforts to solving the question of efficiency.

Define Efficiency

$$E_r = \frac{y_r}{y_R}$$

y_r = Measurement under evaluation

y_R = Possible maximum

127

You can define efficiency using a simple ratio between the actual achieved and the possible maximum. Consider a chemistry example that captures the essentials of this discussion. A reaction can produce as much as a *theoretical yield*, but the actual yield produced by a reaction might fall short because of competing or "side" reactions that occur during the process. The reaction process can be evaluated by taking the yield produced by the process and dividing by the theoretical yield. The resulting ratio is a measure of efficiency for the reaction process.

Understanding theoretical yield in the context of chemistry is not important to the discussion. It is important that the reader understand efficiency to be a ratio of actual to possible or it can be relative in cases where a ratio comparing one entity to another is used.

Comparing Efficiency

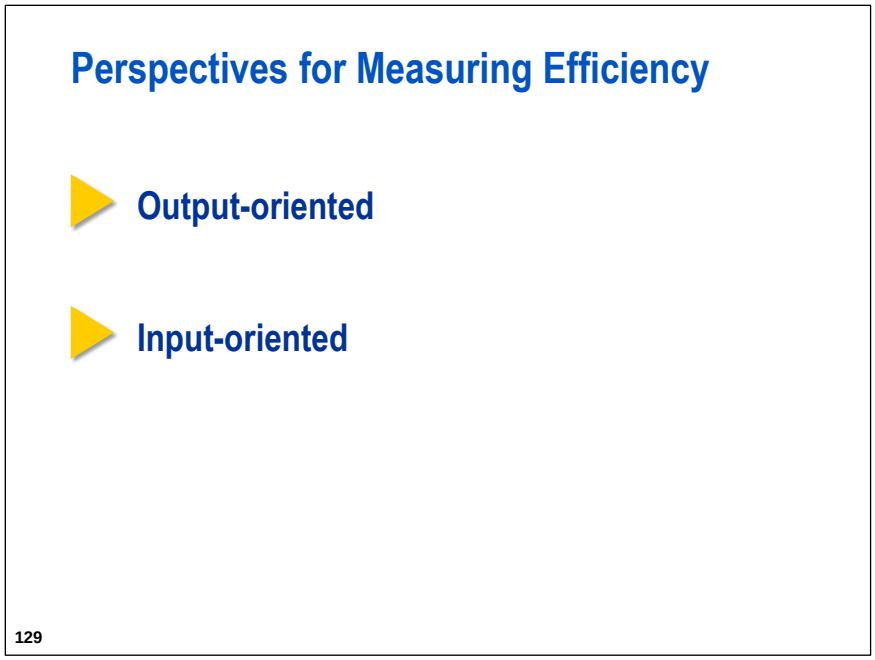
$$E_r = \frac{z_r}{z_R}$$

z_r = Unit under evaluation

z_R = Comparison unit

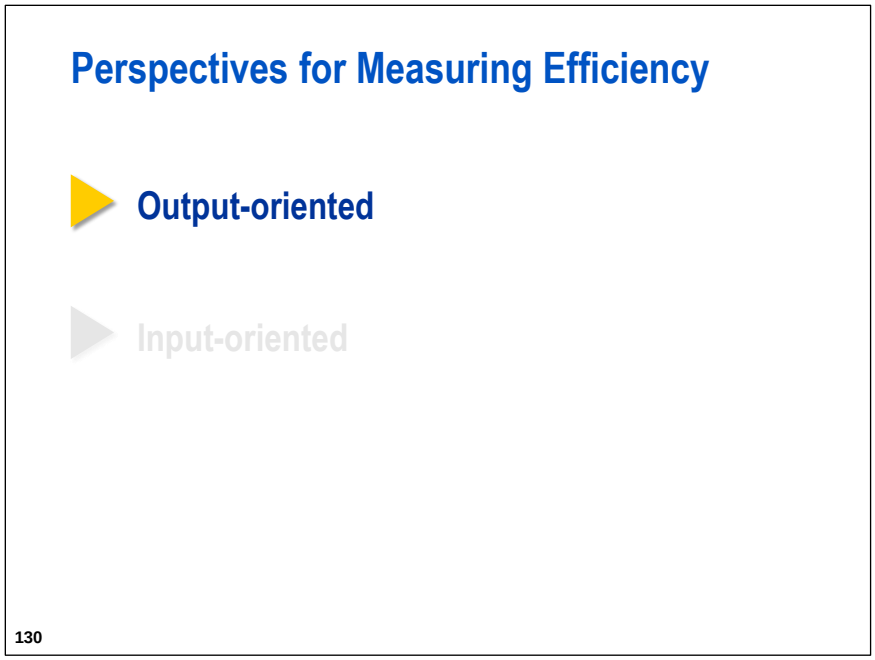
128

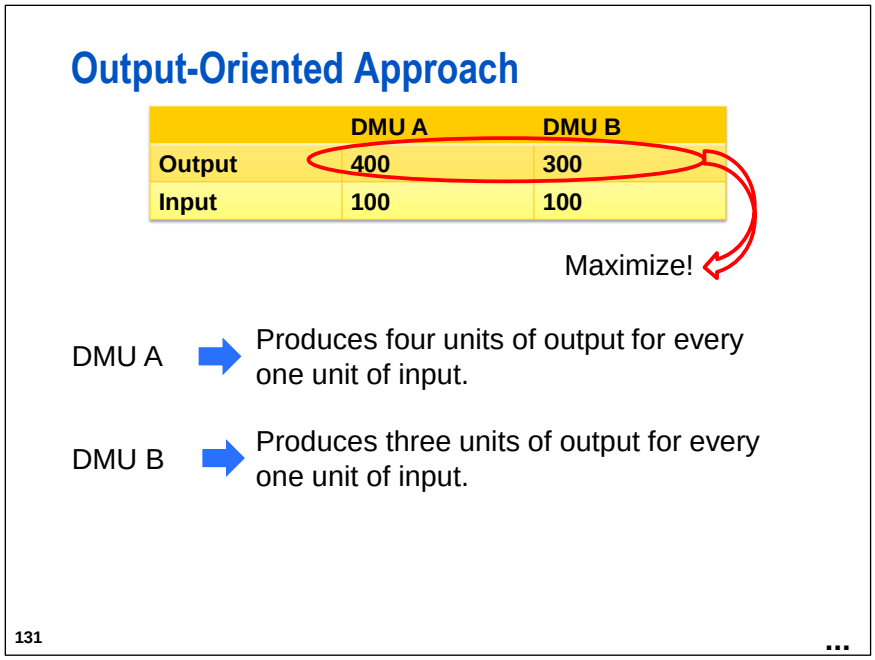
You can extend this idea of efficiency to compare more than one process or approach for completing a task. Here, you are concerned with *relative* or *production* efficiency, such that the measure of efficiency is relative to the processes or approaches being compared, rather than some extraneous maximum. Consider an example where you (the reader) and the author are both producers of output. In this example, assume both use 10 units of input to produce some level of output. Suppose the author produces 30 units of output and you produce 40 units of output. Simply dividing the outputs by the inputs gives the output/input ratios. In this example, you turned one unit of input into four units of output, whereas the author turned only one unit of input into three units of output. You can see that the author is 75% as efficient as you are in turning inputs into outputs (3/4 = .75). You can also say that you are producing 1.33 times more than the author is producing (4/3 = 1.33). This example is from the perspective of an *output-oriented* approach for measuring production efficiency.



When you compare efficiency of various organizational units, you typically refer to these units as *decision-making units* or *DMUs* for short. You can measure the production efficiency from two different, but related perspectives.

- The first perspective is an *output-oriented* approach that looks at maximizing outputs using a given combination of inputs. The general idea of the *output-oriented* approach would say that DMU A is more efficient than DMU B, if DMU A can ***produce more output*** than DMU B produces, without using more of any input than DMU B uses for production.
- The second perspective is an *input-oriented* approach that looks at minimizing inputs to produce a given combination of outputs. The general idea of the *input-oriented* approach would say that DMU A is more efficient than DMU B, if DMU A ***uses fewer inputs*** than DMU B and still produces at least as much of every output as DMU B.





You can imagine a simple example of an output-oriented model in which you compare two producers who use the same amount of one input to produce as much of one output as possible. Producer or DMU A can produce four units of output for every one unit of input. DMU B can produce three units of output for every one unit of input. A ratio comparing output for DMU A to output produced by DMU B reveals that DMU B can expand output by 1.33 times its current level.

Output-Oriented Approach

	DMU A	DMU B
Output	400	300
Input	100	100

Ratio of Outputs Produced

$$\frac{\text{DMU A}}{\text{DMU B}} \Rightarrow \frac{4}{3} = 1.33$$

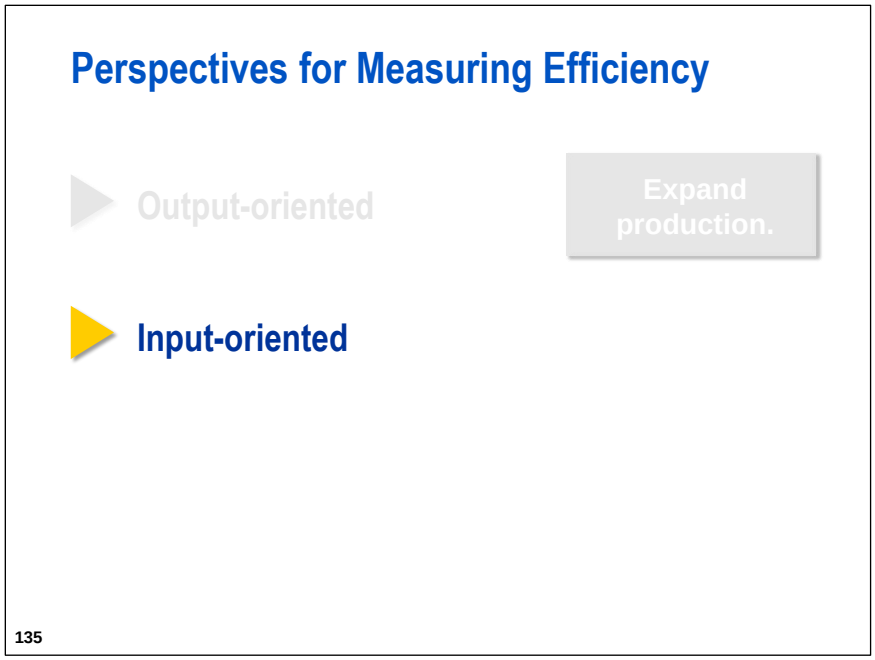
133

The output-oriented approach above examines whether a DMU can expand production, when it is compared to the production of similar DMUs.

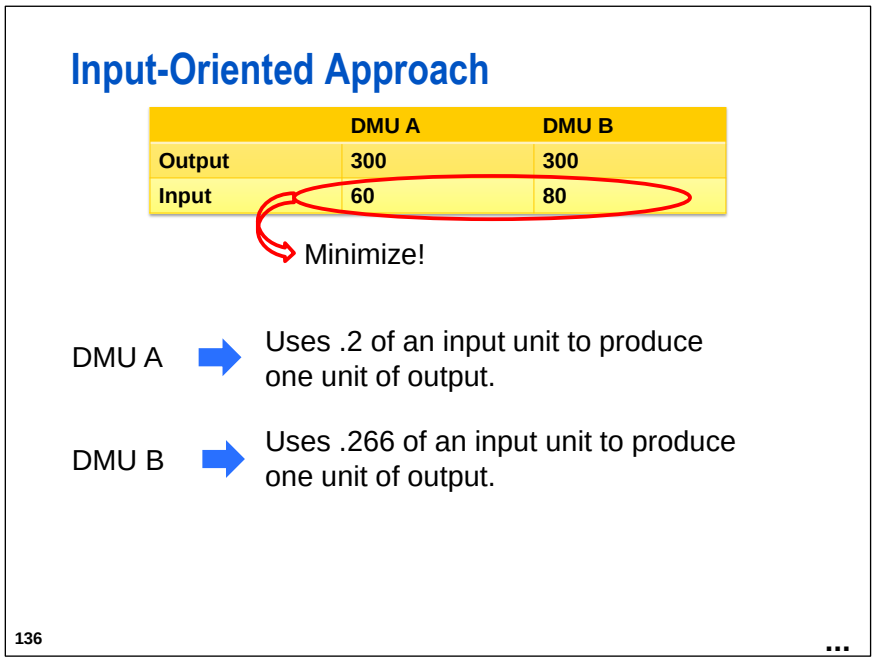
Perspectives for Measuring Efficiency

- **Output-oriented** — **Expand production.**
- Input-oriented

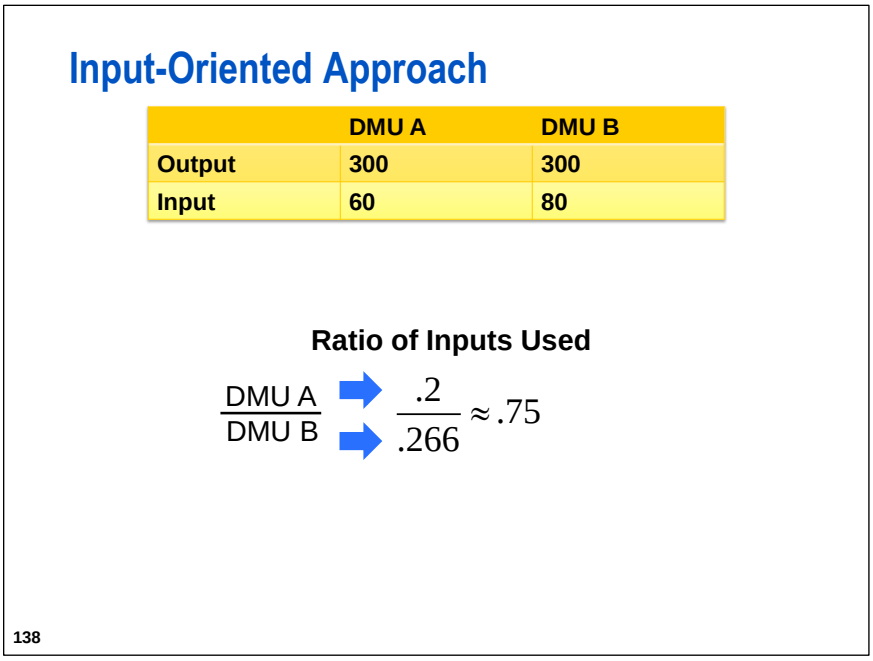
134



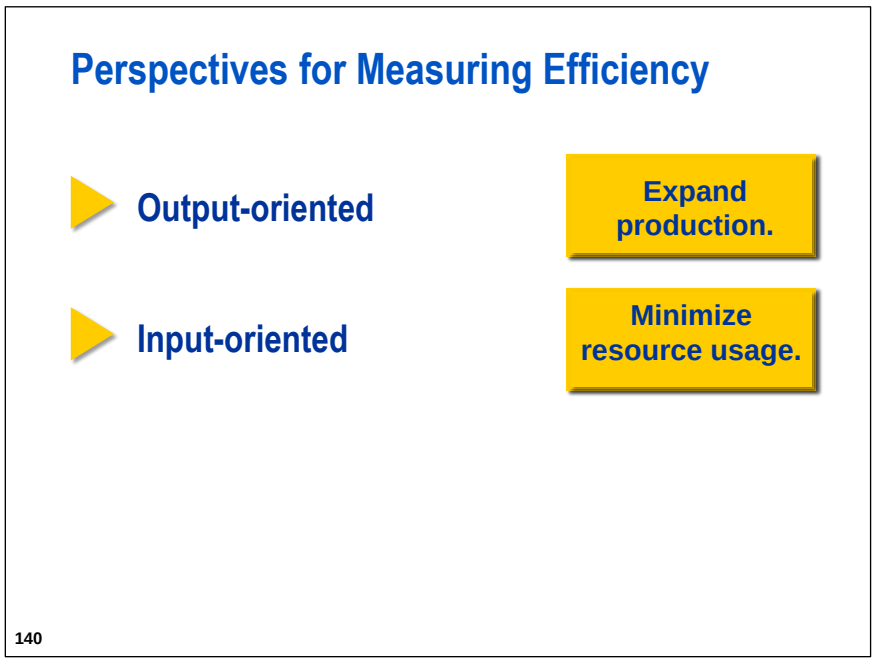
Just as you can examine an output-oriented model, you can also discuss an input-oriented approach where the focus is on minimizing inputs. Again imagine a simple example, but this time of an input-oriented model in which you compare two producers that produce the same amount of one output using as little of one input as possible.



If producer or DMU A uses .2 of a unit of input to produce one unit of output, whereas DMU B uses .2667 of an input unit to produce the same amount of output, you can say that DMU A uses only 75% of the inputs of DMU B. Thus, the relative efficiency of DMU B is 75% when compared to DMU A.

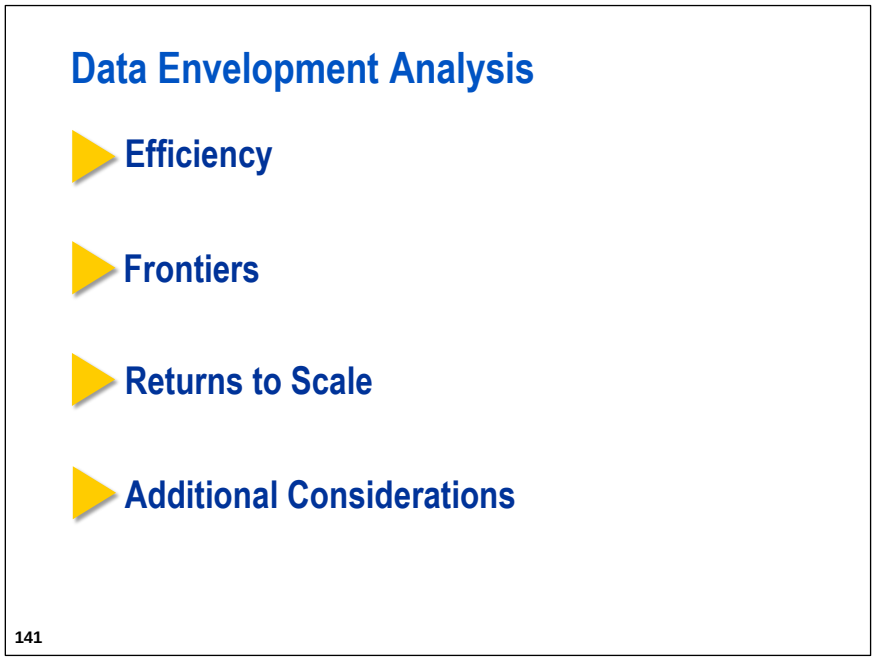


The input-oriented approach above examines whether a DMU can reduce resources used in production, when compared to the production of similar DMUs.



These simple examples can help you understand comparisons of different producers or DMUs. However, real examples are much more involved because businesses often face problems with complexities far beyond this simplistic example. For example, during the 1970s, the US federal government wanted to assess the effectiveness of Program Follow Through, which was created to help disadvantaged kids in public schools. However, the analysis comparing the schools was complicated by the charge from the federal government to assess the schools on more than one measure, including academic performance, extracurricular activities, and the psychological wellness of the students. Traditional statistical approaches seemed to be poorly equipped for the analysis, and the researchers decided on a different approach, which they developed and called *data envelopment analysis*.

Researchers studying Program Follow Through found a regression to be an inadequate model for a few reasons. First, a regression models the average and not the extreme or frontier. Secondly, a regression model is only one outcome or a weighted sum where an *a priori* structure or the relationship between outcomes is assumed to be known.



Data envelopment analysis (DEA) is a linear programming method for modeling efficiency among similar decision-making units. The strength of the analysis is gained from the following:

- Data envelopment analysis can assess efficiency for similar decision-making units that use multiple inputs to produce one or ***more*** outputs.
- Data envelopment analysis is a nonparametric model that does not require a specified functional form or relationship assessing similar decision-making units, which use multiple inputs to produce multiple outputs.

Efficiency for a single unit can be can be measured by defining efficiency as follows:

$$E = \frac{u_1 y_1 + ... + u_r y_r}{v_1 y_1 + ... + v_i x_i}$$

where

- u_r are output weights determined from analysis.
- v_i are input weights determined from analysis.
- y_r are outputs obtained from historical data.
- x_i are inputs obtained from historical data.

Data Envelopment Analysis

Efficiency

$$\text{Max } E = \frac{\textit{weighted sum of outputs}}{\textit{weighted sum of inputs}}$$

Linear Form

Max $E =$ *weighted sum of outputs*

s.t.
- *weighted sum of inputs = 1*
- *weighted sum of outputs - weighted sum of inputs* ≤ 0

143 ...

Dividing the weighted sum of outputs by the weighted sum of inputs makes sense if you are interested in evaluating the performance of a single DMU, but how might you assess multiple DMUs?

The answer is to linearize the problem and then pull together a hybrid DMU that uses a combination of DMUs that yield the best value for your objective.

Data Envelopment Analysis

Efficiency

Comparing Multiple Decision Making Units

Input-Oriented

$$\begin{aligned} &\min \theta \\ &\text{s.t.} \quad X\lambda \leq \theta X_o \\ &\qquad Y\lambda \geq Y_o \\ &\qquad \theta \text{ free}, \ \lambda \geq 0 \end{aligned}$$

Output-Oriented

$$\begin{aligned} &\max \phi \\ &\text{s.t.} \quad X\lambda \leq X_o \\ &\qquad Y\lambda \geq \phi Y_o \\ &\qquad \phi \text{ free}, \ \lambda \geq 0 \end{aligned}$$

144 ...

To linearize the problem, you can transform the following:

$$\max \frac{\sum_{r}^{n} u_r y_{ro}}{\sum_{i}^{n} v_i y_{io}}, \quad \frac{\sum_{r}^{n} u_r y_{rj}}{\sum_{i}^{n} v_i y_{ij}} \leq 1 \text{ for } j = 0,1,...,n, \quad u_r, v_i \geq 0$$

into the equivalent linear programming problem:

$$\begin{aligned} &\max z = u^{\mathrm{T}} Y_o, \\ &s.t. \qquad v^{\mathrm{T}} X_o = 1, \\ &\qquad\quad u^{\mathrm{T}} Y - v^{\mathrm{T}} X \leq 0, \\ &\qquad\quad u^{\mathrm{T}} \geq 0, \\ &\qquad\quad v^{\mathrm{T}} \geq 0, \end{aligned}$$

whose linear programming (LP) dual problem is as follows:

$$\begin{aligned} &\min \theta, \\ &s.t. \qquad X\lambda \leq \theta X_o, \\ &\qquad\quad Y\lambda \geq Y_o, \\ &\qquad\quad \theta \text{ free}, \lambda \geq 0. \end{aligned}$$

You should note that the lambdas in this specific LP dual problem do not represent shadow prices, but instead represent the allocation each *peer* DMU contributes to the hybrid DMU.

Peer DMUs are the DMUs to which the DMU under evaluation is being compared.

For the output-oriented equivalent, you solve for the following:

$$\begin{aligned} &\max \phi, \\ &s.t. \qquad X\lambda \leq X_o, \\ &\qquad\quad Y\lambda \geq \phi Y_o, \\ &\qquad\quad \theta \text{ free}, \lambda \geq 0. \end{aligned}$$

2.08 Multiple Choice Poll

Is the following PROC OPTMODEL an input-oriented or output-oriented DEA model?

```
VAR  dmu_1 >=0, dmu_2 >=0, H;

MIN  resources=H;

CON  inputs: dmu_1*70 + dmu_2*57 <=70*H;
CON  outputs:dmu_1*91 + dmu_2*99 >=91;
```

a. input-oriented

$$\begin{aligned} &\min \theta \\ &\text{s.t.} \quad X\lambda \le \theta X_o \\ &\qquad Y\lambda \ge Y_o \\ &\qquad \theta \text{ free}, \lambda \ge 0 \end{aligned}$$

b. output-oriented

$$\begin{aligned} &\max \phi \\ &\text{s.t.} \quad X\lambda \le X_o \\ &\qquad Y\lambda \ge \phi Y_o \\ &\qquad \phi \text{ free}, \lambda \ge 0 \end{aligned}$$

145

2.09 Multiple Choice Poll

Which of the following correctly defines the DEA model coded below?

```
VAR  dmu_1 >=0, dmu_2 >=0, H;

MAX  production=H;

CON  inputs: dmu_1*70 + dmu_2*57 <=57;
CON  outputs:dmu_1*91 + dmu_2*99 >=99*H;
```

a. input-oriented DEA model evaluating the efficiency of DMU_1

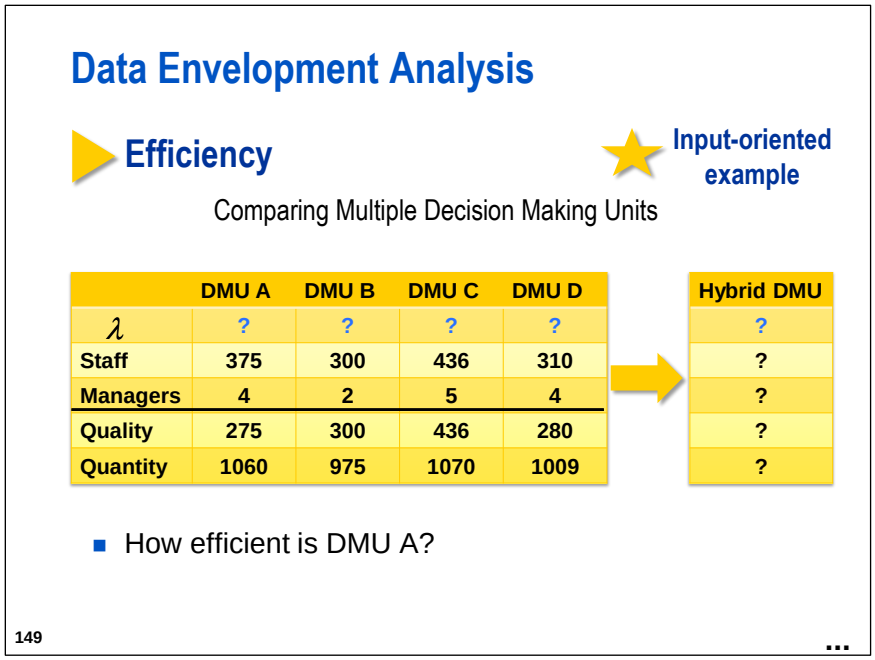
b. output-oriented DEA model evaluating the efficiency of DMU_1

c. output-oriented DEA model evaluating the efficiency of DMU_2

147

Look at another simple input-oriented example to illustrate the approach. There are four DMUs in the example. Each DMU uses two inputs (Staffs and Managers) to produce two outputs (Quality and Quantity). The DMUs are named DMU A, DMU B, DMU C, and DMU D.

An output-oriented model's objective is to maximize outputs that are produced. An input-oriented model seeks to minimize inputs that are used.



You can use PROC OPTMODEL to solve for the optimal lambdas, which minimize resource usage. Positive lambda values represent the *peer(s)* DMU for the DMU under evaluation. The allocations from each peer DMU are used to create a hybrid or virtual DMU that is compared against the DMU under evaluation. The programmatic formulation is shown below.

```
/* simple input-oriented DEA example */
proc optmodel;

   /* Declare Variables */
var DMU_A >=0, DMU_B >=0, DMU_C >=0, DMU_D >=0;
var H;

   /* Declare Objective Function */
min Resources=H;

   /* Declare Constraints */
con Staff_IP: DMU_A*375 + DMU_B*300 + DMU_C*436 + DMU_D*310 <= 375*H;
con Managers_IP: DMU_A*4 + DMU_B*2 + DMU_C*5 + DMU_D*4 <= 4*H;

con Quality_OP: DMU_A*275 + DMU_B*300 + DMU_C*436 + DMU_D*280 >= 275;
con Quantity_OP: DMU_A*1060 + DMU_B*975 + DMU_C*1070 + DMU_D*1009 >= 1060;

   /* Call the Solver */
solve;

   /* Print Solutions */
print DMU_A DMU_B DMU_C DMU_D;

quit;
```

In the case of an input-oriented model, the objective value in the output indicates the percentage equivalent of DMU A's inputs that are used by the hybrid DMU.

Data Envelopment Analysis

The SAS System

The OPTMODEL Procedure

Solution Summary	
Solver	LP
Algorithm	Dual Simplex
Objective Function	eff
Solution Status	Optimal
Objective Value	0.8689138577
Primal Infeasibility	0
Dual Infeasibility	0
Bound Infeasibility	0
Iterations	7
Presolve Time	0.00
Solution Time	0.00

DMU_A	DMU_B	DMU_C	DMU_D
0	0.38951	0	0.67416

- The objective value provides the efficiency percentage.
- The DMU variable values represent the percentage from each DMU that is allocated to the hybrid DMU.

157 ...

The DMU variable values represent the percentage from each DMU that is allocated to the hybrid DMU.

Data Envelopment Analysis

Efficiency

Input-oriented example

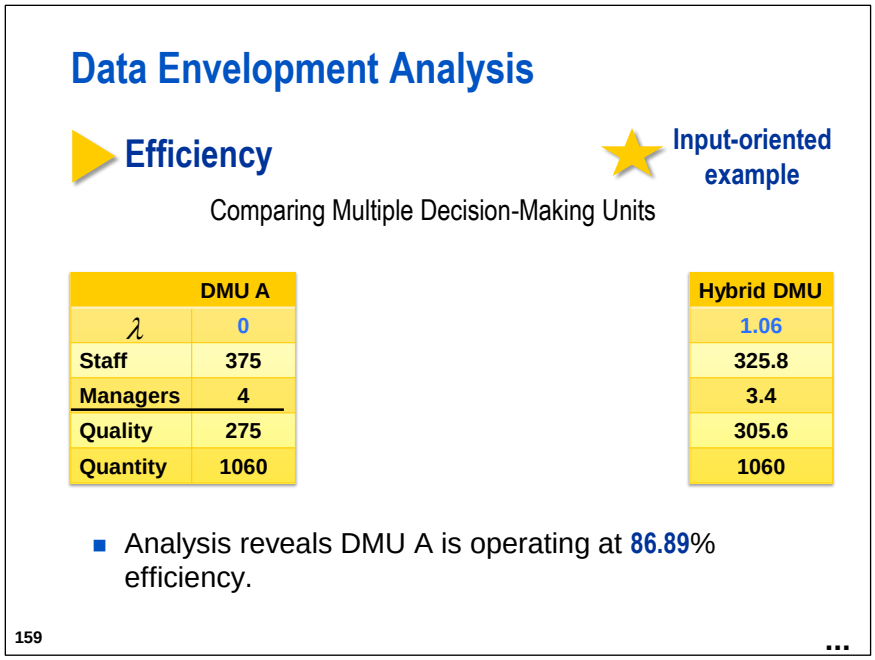
Comparing Multiple Decision-Making Units

	DMU A	DMU B	DMU C	DMU D	Hybrid DMU
λ	0	.3895	0	.67416	1.06
Staff	375	300	436	310	325.8
Managers	4	2	5	4	3.4
Quality	275	300	436	280	305.6
Quantity	1060	975	1070	1009	1060

158

The hybrid DMU uses approximately 326 (300*.3895 + 310*.67416) staff members and approximately 3.4 (2*.3895 + 4*.67416) managers to produce a quantity of 1,060 (975*.3895 + 1009*.67416) units at a quality score of approximately 305.6 (300*.3895 + 280*.67416).

The hybrid DMU effectively produces at least as much output as DMU A, but uses fewer inputs. In fact, the hybrid DMU uses only approximately 87% of the inputs used by DMU A.

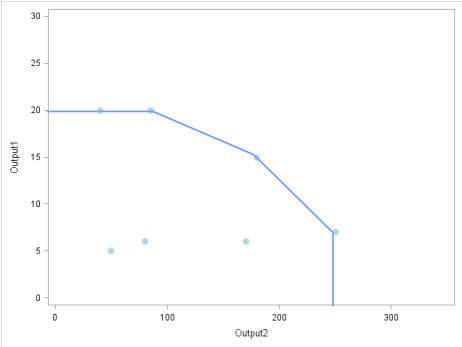


The hybrid DMU uses only approximately 325.8 staff members and about 3.4 managers. Having 3.4 managers is not realistic, so you can restrict staff and manager inputs to be integers. (Integer programming is not discussed in this course.)

Using data envelopment analysis has a distinct advantage over techniques such as a regression because data envelopment analysis models the production frontier, as opposed to the average.

Data Envelopment Analysis

Frontiers



- Data envelopment analysis creates an empirical piecewise linear frontier.
- DMUs that reside on the frontier are considered to be efficient.

162 ...

The frontier created by data envelopment analysis is an empirical piecewise linear frontier.

Data Envelopment Analysis

Efficiency

Frontiers

Returns to Scale

Additional Considerations

164 ...

The primary discussion is about two types of returns to scale: constant and variable.

Data Envelopment Analysis

Returns to Scale

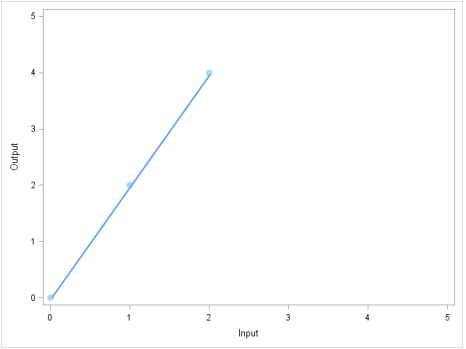
- Constant returns to scale.
- Variable returns to scale.
  - Increasing returns to scale.
  - Constant returns to scale.
  - Decreasing returns to scale.

165 ...

The original form of data envelopment analysis, introduced by Charnes, Cooper, and Rhodes (1978), assumed constant returns to scale (CRS). CRS says that if inputs are increased, you can expect outputs to increase by the same proportion. For example, if you double inputs under the CRS assumption, you expect outputs to double as well. The CRS assumption lead many economists to be critical of DEA early in DEA's introduction. Below, you can see an example of CRS graphed by three points on a scatter plot.

Data Envelopment Analysis

Returns to Scale



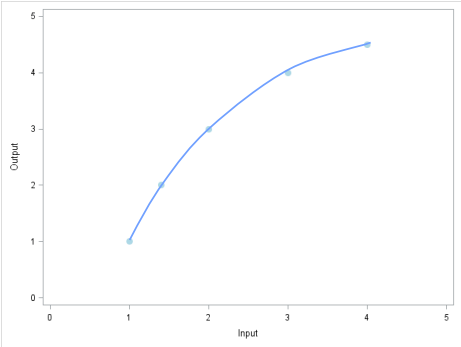
- *Constant returns to scale envelopment* assumes that output and input change by equivalent proportions.
  - For example, if inputs double, you expect outputs to double.

166 ...

In the CRS approach, θ is free. A variation of DEA was later developed by Banker, Chames, and Cooper (1984). It allowed for variable returns to scale (VRS). VRS enables DEA to model any DMU returns to scale frontier, including increasing returns to scale (IRS), constant returns to scale (CRS), and decreasing returns to scale (DRS). For VRS, you constrain $\theta = 1$.

Data Envelopment Analysis

Returns to Scale



- *Variable returns to scale* identifies the frontier for DMUs exhibiting increasing returns to scale (IRS), constant returns to scale (CRS), and decreasing returns to scale (DRS).

167 ...

You can also constrain $\theta \leq 1$ for specifically modeling increasing returns to scale or constrain $\theta \geq 1$ for decreasing returns to scale.

Data Envelopment Analysis

Efficiency

Frontiers

Returns to Scale

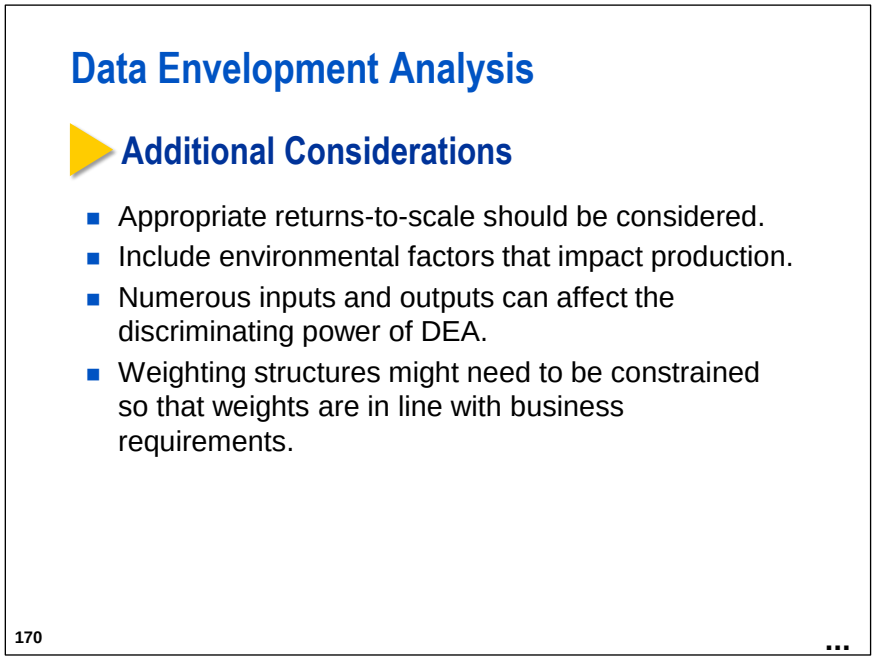
Additional Considerations

168

There are a few additional considerations that should be made when using data envelopment analysis.

- First, it is advised that the analyst identify the appropriate returns to scale for the industry and use the appropriate constraint (θ=1, $\theta \geq 1$, $\theta \leq 1$, and so on) when assessing DMUs. Seiford and Zhu (1999a) demonstrate three methods for identifying the appropriate returns to scale.

- Second, it is important that the environmental factors that impact production be accounted for as inputs or outputs in the model. If quantifying the environmental factors is infeasible, consider using surrogate representations.
- Third, using numerous inputs, outputs, or both can reduce DEA's discriminating power because the flexibility in choosing optimal weights can lead all DMUs to be classified as efficient. For example, you want to have no less than nine DMUs in an analysis for which you have three inputs and three outputs. For this reason, you should set a lower bound on the DMU count to be the product of inputs and outputs. Principal components analysis can also be used to reduce the dimension when a large number of inputs are present.



Lastly, weighting structures might need to be constrained further so that weights are in line with business requirements. You saw an example of additional constraints that were needed previously when DEA produced a hybrid that recommended that you use 3.4 managers. Perhaps defining DMU weights as integers would solve the problem of partial labor in this example.

Exercises

5. Building Data Envelopment Analysis Models to Solve an Operations Problem

An insurance company has local offices that are spread throughout the country. The offices service middle-market (mid-sized companies) accounts. When analyzing employee ages, you noticed that a large number of employees in these local offices are approaching retirement age. You presented this information to your senior leadership. In response, they asked you to determine whether the company should hire replacements for those that retire, or use the natural attrition as a means of becoming more efficient.

Two regional offices cover each region, so there are a total of twelve offices. An office has account managers (AM) and account executives (AE) who service new and current policies. In addition, the regional offices also handle policy modifications.

a. If not already opened, open **ch2ex.sas** and run the code that generates an insurance operations data set. Next, run the code under the commented line that begins with "5a." The OPTMODEL code is evaluating the South East1 region using an input-oriented DEA model. Is the South East 1 region operating efficiently? If the answer is no, by what percentage can this office reduce staff?

b. Copy and paste the 5a code and modify the code to assess the South West 1 region. Is the South West 1 region operating efficiently according to the input-oriented DEA model?

c. Copy and paste the OPTMODEL code and modify the model so that you are now using an output-oriented DEA model to assess offices. Using the output-oriented DEA model, assess the office labeled Mid-West2. Is Mid-West2 operating efficiently according to the output-oriented model? If the answer is no, by what percentage can this office increase production?

Data Set View:

Office	AEs	AMs	New_Policies	Old_Policies	Modified_Policies
South_East1	4	15	32	51	31
South_East2	5	7	25	77	21
South_West1	4	4	33	98	24
South_West2	2	10	34	58	21
North_East1	3	10	28	53	22
North_East2	1	4	27	71	45
North_West1	2	4	34	78	30
North_West2	2	4	25	91	37
Mid_East1	3	3	14	75	24
Mid_East2	1	11	22	94	36
Mid_West1	4	12	19	90	36
Mid_West2	3	8	20	91	43

End of Exercises

2.6 Reading SAS Data Sets (Self-Study)

Objectives

- Read data from (multiple) SAS data sets to formulate linear programming problems.

173

Reading Data from SAS Data Sets: Example

How can you read the height, weight, and age of students into the arrays **height**, **weight**, and **age**?

SAS Data Set: **opt.class** (only the first four rows)

	Name	Height	Weight	Age
1	Alfred	69	112.5	14
2	Alice	56.5	84	13
3	Barbara	65.3	98	13
4	Carol	62.8	102.5	14

174 ...

The READ DATA statement saves the student names as the elements of the set **STUDENTS**, which must already be declared (as must the parameter arrays **height**, **weight**, and **age**).

Reading Data from SAS Data Sets: Example

How can you read the height, weight, and age of students into the arrays **height**, **weight**, and **age**?

SAS Data Set: **opt.class** (only the first four rows)

	Name	Height	Weight	Age
1	Alfred	69	112.5	14
2	Alice	56.5	84	13
3	Barbara	65.3	98	13
4	Carol	62.8	102.5	14

```
read data opt.class into STUDENTS=[Name]
     height weight age;
```

175

These other data set options might be used when you read or create SAS data sets from PROC OPTMODEL:

FIRSTOBS=	To skip observations at the start of the SAS data set
PW=	To access a password-protected SAS data set
READ=	To assign a password to a SAS data set
RENAME=	To change the name of SAS data set variables*
WHERE=	To read or create a SAS data set that contains only observations meeting certain criteria

* SAS data set variables can be read into parameter arrays with different names in PROC OPTMODEL.

Reading the Students Data Set in PROC OPTMODEL

```
proc optmodel;
   /* declare variables */
   set <str> STUDENTS;
   num height {STUDENTS}, weight {STUDENTS}, age {STUDENTS};

   /* read data from SAS data sets */
   read data opt.class into STUDENTS=[Name] height weight age;

   print height weight age;

quit;
```

Before submitting the program, you must first create a SAS library and assign the name **opt** as the libref.

PROC OPTMODEL Output

The OPTMODEL Procedure

[1]	height	weight	age
Alfred	69.0	112.5	14
Alice	56.5	84.0	13
Barbara	65.3	98.0	13
Carol	62.8	102.5	14
Henry	63.5	102.5	14
James	57.3	83.0	12
Jane	59.8	84.5	12
Janet	62.5	112.5	15
Jeffrey	62.5	84.0	13
John	59.0	99.5	12
Joyce	51.3	50.5	11
Judy	64.3	90.0	14
Louise	56.3	77.0	12
Mary	66.5	112.0	15
Philip	72.0	150.0	16
Robert	64.8	128.0	12
Ronald	67.0	133.0	15
Thomas	57.5	85.0	11
William	66.5	112.0	15

SAS Log

```
NOTE: There were 19 observations read from the data set OPT.CLASS.
```

The OBS data set option is useful for checking the syntax of a READ DATA statement by printing the first few observations.

```
   /* read data from SAS data sets */
   read data opt.class(obs=4) into STUDENTS=[Name] height weight age;
```

PROC OPTMODEL Output

The OPTMODEL Procedure

[1]	height	weight	age
Alfred	69.0	112.5	14
Alice	56.5	84.0	13
Barbara	65.3	98.0	13
Carol	62.8	102.5	14

SAS Log

```
NOTE: There were 4 observations read from the data set OPT.CLASS.
```

End of Demonstration

Reading Data: READ DATA Statement

READ DATA *SAS-data-set* [**NOMISS**] **INTO**
[set-name=] **[** *read-key-columns* **]** *[read-columns]* **;**

- *SAS-data-set* specifies the input data set.
- *read-key-columns* provide the index values for array destinations.
- The optional *set-name* saves index values as a set.
- *read-columns* specify the data values to read and destination locations.
- The optional NOMISS keyword suppresses the assignment of missing values.

179

The syntax for a read column parameter has several forms, including an iterated form.

identifier-expression [/ trim-option]

identifier-expression = *name [/ trim-option]*

identifier-expression = **COL(***name-expression***)** *[/ trim-option]*

- *identifier-expression* specifies the target.
- The name of the input data set variable can be specified with *name* or a COL expression (which is evaluated at read time). Otherwise, the name part of *identifier-expression* is used.
- *trim-option* controls how leading and trailing blanks are treated for strings. The default trim option removes leading and trailing blanks.

{ *index-set* **}** **<** *read-columns* **>**

- This form iterates *read-columns* for each element of *index-set*. Distinct input columns can be specified using dummy parameters from *index-set* in the COL expressions.

Iterated read columns ***cannot*** be nested.

Next, you read the data for the furniture-making problem.

Furniture-Making Problem SAS DATA Steps

```
data resource_data;
   input Resource $ Cost Amount_Available;
   datalines;
labor 14 225
metal 20 117
wood  11 420
run;

data product_data;
   length Item $9;
   input Item $ Selling_Price labor metal wood;
   datalines;
desks      94 2 1 3
chairs     79 1 1 3
bookcases 125 3 1 4
bedframes 109 2 1 4
run;
```

181

Furniture-Making Problem SAS Data Sets

SAS Data Set: **work.resource_data**

	Resource	Cost	Amount_Available
1	labor	14	225
2	metal	20	117
3	wood	11	420

SAS Data Set: **work.product_data**

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3
3	bookcases	125	3	1	4
4	bedframes	109	2	1	4

182

There are other ways that the same data can be stored in SAS data sets.

Reading the Furniture-Making Data Sets

```
proc optmodel;
   /* declare sets and parameters */
   set <str> PRODUCTS, RESOURCES;
   num cost {RESOURCES}, availability {RESOURCES};
   num selling_price {PRODUCTS};
   num required {PRODUCTS, RESOURCES};

   read data resource_data into RESOURCES=[Resource]
      cost availability=Amount_Available;

   read data product_data into PRODUCTS=[Item]
      {r in RESOURCES} <required[Item,r]=col(r)>
      selling_price;
```

183 ...

These READ DATA statements are examined step by step.

Reading the Furniture-Making Data Sets

```
proc optmodel;
   /* declare sets and parameters */
   set <str> PRODUCTS, RESOURCES;
   num cost {RESOURCES}, availability {RESOURCES};
   num selling_price {PRODUCTS};
   num required {PRODUCTS, RESOURCES};

   read data resource_data into RESOURCES=[Resource]
      cost availability=Amount_Available;

   read data product_data into PRODUCTS=[Item]
      {r in RESOURCES} <required[Item,r]=col(r)>
      selling_price;
```

- Declare index sets. (With no initialization expression, **<str>** is necessary because the default is **<num>**.)

184 ...

The index sets are populated by the READ DATA statements.

Reading the Furniture-Making Data Sets

```
proc optmodel;
   /* declare sets and parameters */
   set <str> PRODUCTS, RESOURCES;
   num cost {RESOURCES}, availability {RESOURCES};
   num selling_price {PRODUCTS};
   num required {PRODUCTS, RESOURCES};

   read data resource_data into RESOURCES=[Resource]
      cost availability=Amount_Available;

   read data product_data into PRODUCTS=[Item]
      {r in RESOURCES} <required[Item,r]=col(r)>
      selling_price;
```

- Declare parameter arrays indexed by the (unpopulated) index sets **PRODUCTS** and **RESOURCES**.

185

Most information in PROC OPTMODEL is stored symbolically and resolved only when necessary.

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num cost {RESOURCES}, availability {RESOURCES};

read data resource_data into RESOURCES=[Resource]
   cost availability=Amount_Available;
```

read key column
(SAS data set variable name)

SAS Data Set: **work.resource_data**

	Resource	Cost	Amount_Available
1	labor	14	225
2	metal	20	117
3	wood	11	420

186 ...

The values in the read key column of the SAS data set should be unique. If not, a warning message is generated in the SAS log file. Subsequent observations overwrite earlier observations with the same value in the read key column.

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num cost {RESOURCES}, availability {RESOURCES};

read data resource_data into RESOURCES=[Resource]
   cost availability=Amount_Available;
```

set name
(OPTMODEL index set)

SAS Data Set: **work.resource_data**

	Resource	Cost	Amount_Available
1	labor	14	225
2	metal	20	117
3	wood	11	420

187 ...

The set name does not need to be the plural form of the read key column. The former is a PROC OPTMODEL name and the latter is a SAS data set variable name. They can be the same or completely unrelated.

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num cost {RESOURCES}, availability {RESOURCES};

read data resource_data into RESOURCES=[Resource]
   cost availability=Amount_Available;
```

read columns
(OPTMODEL array name *[=SAS data set variable name]*)

SAS Data Set: **work.resource_data**

	Resource	Cost	Amount_Available
1	labor	14	225
2	metal	20	117
3	wood	11	420

188

In the first case, no SAS data set variable name is supplied, so it is assumed to be **cost** (**Cost**).

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num cost {RESOURCES}, availability {RESOURCES};

read data resource_data into RESOURCES=[Resource]
   cost availability=Amount_Available;
```

The read columns in the READ DATA statement can be seen as an abbreviated version of the read columns in the following equivalent READ DATA statement:

```
set <str> PRODUCTS, RESOURCES;
num cost {RESOURCES}, availability {RESOURCES};

read data resource_data into RESOURCES=[Resource]
   cost[Resource]=Cost
   availability[Resource]=Amount_Available;
```

189

This form of the read columns syntax more closely resembles SAS DATA step syntax.

The name of the read key column functions as a local dummy parameter, which can be used to refer to array locations in the read columns. This is necessary in the READ DATA statement for the product data.

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num selling_price {PRODUCTS};
num required {PRODUCTS, RESOURCES};

read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

read key column (SAS data set variable name)

SAS Data Set: **work.product_data**

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3
3	bookcases	125	3	1	4
4	bedframes	109	2	1	4

190 ...

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num selling_price {PRODUCTS};
num required {PRODUCTS, RESOURCES};

read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

set name (OPTMODEL index set)

SAS Data Set: **work.product_data**

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3
3	bookcases	125	3	1	4
4	bedframes	109	2	1	4

191 ...

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num selling_price {PRODUCTS};
num required {PRODUCTS, RESOURCES};

read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

read column (OPTMODEL array name)

SAS Data Set: **work.product_data**

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3
3	bookcases	125	3	1	4
4	bedframes	109	2	1	4

192 ...

No SAS data set variable name is supplied, so it is assumed to be **selling_price** (**Selling_Price**).

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num selling_price {PRODUCTS};
num required {PRODUCTS, RESOURCES};

read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

iterated read column (array destination=COL expression)

SAS Data Set: **work.product_data**

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3
3	bookcases	125	3	1	4
4	bedframes	109	2	1	4

193

Local dummy parameters **r** and **Item** are used to specify array locations. (String functions can be applied to the local dummy parameter in the COL expression to match SAS data set variable names.)

The iterated read column can be replaced by three read columns.

A Closer Look at the Iterated Read Column

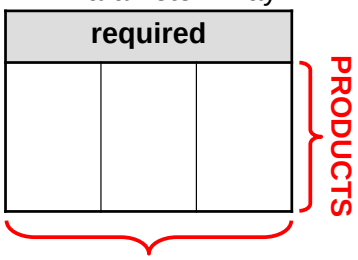
```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>;
```

```
read data product_data into PRODUCTS=[Item]
   required[Item,'labor']=labor
   required[Item,'metal']=metal
   required[Item,'wood']=wood;
```

SAS Data Set (last columns)

	labor	metal	wood
1	2	1	3
2	1	1	3
3	3	1	4
4	2	1	4

2-D Parameter Array



194

These read columns illustrate the difference between string values and names. (See the COL expression in the iterated read column.)

Space in OPTMODEL: Broad Overview

Declaration statements:

- Types are assigned, so the compiler can check syntax.
- Storage for arrays is allocated one element at a time, when the element is first referenced or assigned.

READ DATA statements:

- These statements populate sets and parameter arrays.

PRINT statements:

- These statements cause storage to be allocated.

SOLVE or EXPAND statements:

- The model (or partial model) is built from index sets and parameter arrays.
- Storage is allocated for suffixed parameter arrays.

195

The following sequence of slides shows how the last READ DATA statement for the furniture-making problem populates the first two rows of the parameter arrays **required** and **selling_price**. The set **RESOURCES** is already populated by the first READ DATA statement.

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required	selling_price
2	

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

196

...

For the first observation, the character string in the **Item** column is added to the set **PRODUCTS**. The value in the **labor** column is assigned to a location in the parameter array **required**.

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required		selling_price
2	1	

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

197 ...

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required			selling_price
2	1	3	

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

198 ...

After the set **RESOURCES** is iterated, the read column is processed.

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required			selling_price
2	1	3	94

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

199 ...

The value in the **Selling_Price** column of the SAS data set is assigned to a location in the parameter array **selling_price**.

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required			selling_price
2	1	3	94
1			

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

200 ...

For the second observation, the character string in the **Item** column is added to the set **PRODUCTS**, and the process continues.

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required			selling_price
2	1	3	94
1	1		

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

201 ...

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required			selling_price
2	1	3	94
1	1	3	

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

202 ...

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required			selling_price
2	1	3	94
1	1	3	79

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

203

The order in which the columns are read from the SAS data set is not important.

Reading SAS Data Sets in PROC OPTMODEL for the Furniture-Making Problem

```
data resource_data;
   input Resource $ Cost Amount_Available;
   datalines;
labor 14 225
metal 20 117
wood  11 420
run;

data product_data;
   length Item $9;
   input Item $ Selling_Price labor metal wood;
   datalines;
desks      94 2 1 3
chairs     79 1 1 3
bookcases 125 3 1 4
bedframes 109 2 1 4
run;

%let budget_limit = 10000;

proc optmodel;
   /* declare sets and parameters */
   set <str> PRODUCTS, RESOURCES;
   num cost {RESOURCES}, availability {RESOURCES};
   num selling_price {PRODUCTS};
   num required {PRODUCTS, RESOURCES};

   /* read data from SAS data sets */
   read data resource_data into RESOURCES=[Resource]
      cost availability=Amount_Available;

   read data product_data into PRODUCTS=[Item]
      {r in RESOURCES} <required[Item,r]=col(r)>
      selling_price;

   /* print parameter arrays */
   print cost dollar. availability;
   print selling_price dollar. required;

   /* declare variables */
   var NumProd {PRODUCTS} >= 0;

   impvar Revenue = sum {p in PRODUCTS}
      selling_price[p] * NumProd[p];

   impvar AmountUsed {r in RESOURCES} =
      sum {p in PRODUCTS} NumProd[p] * required[p,r];
```

```
   impvar TotalCost = sum {r in RESOURCES}
      cost[r] * AmountUsed[r];

   /* declare constraints */
   con Usage {r in RESOURCES}:
      AmountUsed[r] <= availability[r];

   con Budget: TotalCost <= &budget_limit;

   /* declare objective */
   max NetProfit = Revenue - TotalCost;

   solve;

   print NumProd;
   print AmountUsed;

quit;
```

The SAS macro language can be used to input the values of individual parameters, with one macro variable per parameter.

Notice the format applied to the **cost** and **selling_price** parameter arrays in the PRINT statement.

PROC OPTMODEL Output (only the first two PRINT statements)

The OPTMODEL Procedure

[1]	cost	availability
labor	$14	225
metal	$20	117
wood	$11	420

[1]	selling_ price
bedframes	$109
bookcases	$125
chairs	$79
desks	$94

	required			
	bedframes	bookcases	chairs	desks
labor	2	3	1	2
metal	1	1	1	1
wood	4	4	3	3

The problem summary, solution summary, and the output of the last PRINT statements are omitted, because they are identical to the output of the previous furniture-making demonstration program from a previous section.

If the product data is read from the SAS data set below, it is not necessary to read the data from the additional column.

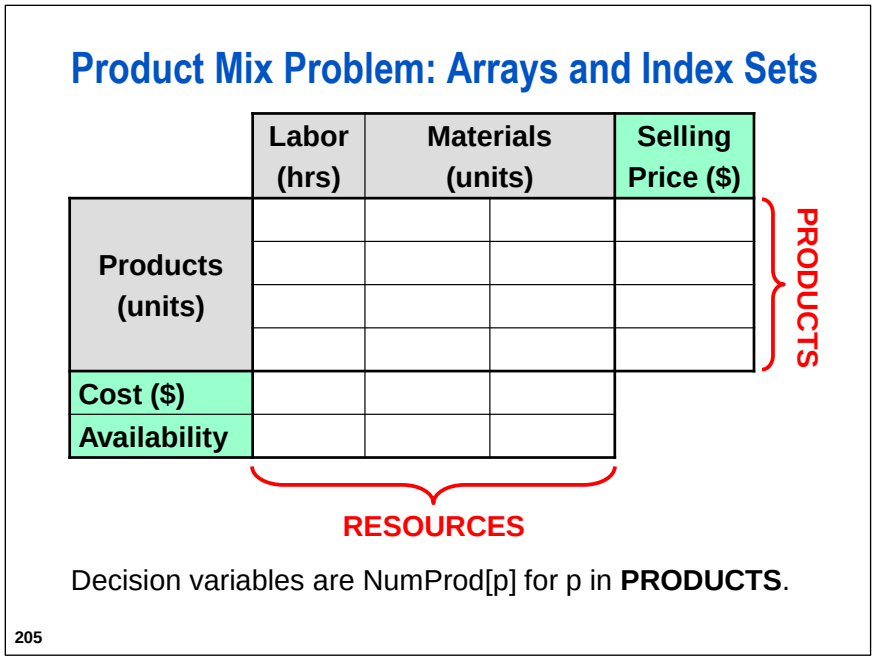
opt.product_data

	Item	Setup_Time	Selling_Price	labor	metal	wood
1	desks	0.3	94	2	1	3
2	chairs	0.6	79	1	1	3
3	bookcases	0.5	125	3	1	4
4	bedframes	0.8	109	2	1	4

Not Used

Notice that, except for the SAS DATA steps, nothing in the PROC OPTMODEL program is specific to the furniture-making problem. In fact, the model is generic and can be used to solve a class of problems called *product mix problems*.

End of Demonstration



These problems might or might not include a budget constraint.

2.7 Solutions

Solutions to Exercises

1. Changing the Limit of a Tight Constraint

```
   con x + 3*y <= 224;
```

PROC OPTMODEL Output

```
                          The OPTMODEL Procedure

                            Problem Summary

                Objective Sense            Maximization
                Objective Function                    z
                Objective Type                   Linear

                Number of Variables                   2
                Bounded Above                         0
                Bounded Below                         2
                Bounded Below and Above               0
                Free                                  0
                Fixed                                 0

                Number of Constraints                 3
                Linear LE (<=)                        3
                Linear EQ (=)                         0
                Linear GE (>=)                        0
                Linear Range                          0

                Constraint Coefficients               6

                          Solution Summary

                  Solver                             LP
                  Algorithm                Dual Simplex
                  Objective Function                  z
                  Solution Status               Optimal
                  Objective Value                1778.5

                  Primal Infeasibility                0
                  Dual Infeasibility                  0
                  Bound Infeasibility                 0

                  Iterations                          5
                  Presolve Time                    0.00
                  Solution Time                    0.00

                                x       y

                             63.5    53.5
```

The previous optimal solution with [x,y] = [63,54] has x + 3y = 63 + 162 = 225, so it is no longer feasible. An inequality that is satisfied with equality by an optimal solution is said to be *tight*.

2. Detecting Infeasible Linear Programming Problems

```
   con 3*x + 4*y >= 420;
```

PROC OPTMODEL Output

```
                          The OPTMODEL Procedure

                            Problem Summary

                  Objective Sense            Maximization
                  Objective Function                    z
                  Objective Type                   Linear

                  Number of Variables                   2
                  Bounded Above                         0
                  Bounded Below                         2
                  Bounded Below and Above               0
                  Free                                  0
                  Fixed                                 0

                  Number of Constraints                 3
                  Linear LE (<=)                        2
                  Linear EQ (=)                         0
                  Linear GE (>=)                        1
                  Linear Range                          0

                  Constraint Coefficients               6

                            Solution Summary

                    Solver                           LP
                    Algorithm                Dual Simplex
                    Objective Function                z
                    Solution Status         Infeasible

                    Iterations                         0
                    Presolve Time                   0.00
                    Solution Time                   0.00

                                 x    y

                                 0    0
```

The solution status indicates that the problem is infeasible.

3. Adding a Budget Constraint to the Furniture-Making Problem

```
proc optmodel;
   /* declare variables */
   var Desks >= 0, Chairs >= 0, Bookcases >= 0, Bedframes >= 0;

   impvar Hrs = 2*Desks + Chairs + 3*Bookcases + 2*Bedframes;
   impvar Lbs = Desks + Chairs + Bookcases + Bedframes;
   impvar Ft3 = 3*Desks + 3*Chairs + 4*Bookcases + 4*Bedframes;

   /* declare constraints */
```

```
   con Labor: Hrs <= 225, Metal: Lbs <= 117, Wood: Ft3 <= 420,
       Budget:  14*Hrs + 20*Lbs + 11*Ft3 <= 10000 ;

   /* declare objective */
   max NetProfit =
      94*Desks + 79*Chairs + 125*Bookcases + 109*Bedframes
      - 14*Hrs - 20*Lbs - 11*Ft3;

   expand Budget;

   solve;

   print Desks Chairs Bookcases Bedframes;

   print Hrs Lbs Ft3;

quit;
```

4. Formulating a Transportation Problem

a. The following are the declaration statements:

```
proc optmodel;
   /* declare sets and parameters */
   set ORIGINS = /Chicago Detroit/;
   set DESTINATIONS = /Boston Newark Toronto/;
   num supply {ORIGINS} = [200 150];
   num demand {DESTINATIONS} = [50 150 100];
   num unit_cost {ORIGINS, DESTINATIONS} =
       [40 25 15 30 15 10];

   /* print parameter arrays */
   print unit_cost supply demand;

quit;
```

In the declaration statement for **unit_cost**, the values can be divided into two lines to make it easier to parse the initializers.

PROC OPTMODEL Output

```
                        The OPTMODEL Procedure

                              unit_cost
                          Boston    Newark    Toronto

                Chicago       40        25         15
                Detroit       30        15         10

                     [1]       supply    demand

                     Boston                    50
                     Chicago      200
                     Detroit      150
                     Newark                   150
                     Toronto                  100
```

The PRINT statement combines **supply** and **demand** as a listing, whereas **unit_cost** is formatted as a separate table because it is a two-dimensional array.

b. The declaration statements are as follows:

```
proc optmodel;
   /* declare sets and parameters */
   set ORIGINS = /Chicago Detroit/;
   set DESTINATIONS = /Boston Newark Toronto/;
   num supply {ORIGINS} = [200 150];
   num demand {DESTINATIONS} = [50 150 100];
   num unit_cost {ORIGINS, DESTINATIONS} =
       [40 25 15 30 15 10];

   /* declare variables */
   var NumShip {ORIGINS, DESTINATIONS} >= 0 init 50;

   impvar FlowOut {i in ORIGINS} =
      sum {j in DESTINATIONS} NumShip[i,j];

   impvar FlowIn {j in DESTINATIONS} =
      sum {i in ORIGINS} NumShip[i,j];

   expand FlowOut['Chicago'];
   expand FlowIn['Boston'];

   expand / impvar;

   print FlowOut;
   print FlowIn;

quit;
```

PROC OPTMODEL Output

```
                              The OPTMODEL Procedure

Impvar FlowOut[Chicago] = NumShip[Chicago,Boston] + NumShip[Chicago,Newark] +
NumShip[Chicago,Toronto]
Impvar FlowIn[Boston] = NumShip[Chicago,Boston] + NumShip[Detroit,Boston]

                                          Flow
                                [1]        Out

                                Chicago    150
                                Detroit    150

                                          Flow
                                [1]         In

                                Boston     100
                                Newark     100
                                Toronto    100
```

The (individual) implicit variables are expanded, but the EXPAND/IMPVAR statement triggers the current model to be built. It does not produce any output because the implicit variables do not enter into the constraints or objective function of the optimization problem.

c. The formulation using index sets is as follows:

```
proc optmodel;
   /* declare sets and parameters */
   set ORIGINS = /Chicago Detroit/;
   set DESTINATIONS = /Boston Newark Toronto/;
   num supply {ORIGINS} = [200 150];
   num demand {DESTINATIONS} = [50 150 100];
   num unit_cost {ORIGINS, DESTINATIONS} =
      [40 25 15 30 15 10];

   /* declare variables */
   var NumShip {ORIGINS, DESTINATIONS} >= 0;

   impvar FlowOut {i in ORIGINS} =
      sum {j in DESTINATIONS} NumShip[i,j];

   impvar FlowIn {j in DESTINATIONS} =
      sum {i in ORIGINS} NumShip[i,j];

   /* declare constraints */
   con Supply_con {i in ORIGINS}:
      FlowOut[i] <= supply[i];

   con Demand_con {j in DESTINATIONS}:
      FlowIn[j] >= demand[j];

   /* declare objective */
   min TotalCost = sum {i in ORIGINS, j in DESTINATIONS}
      unit_cost[i,j] * NumShip[i,j];

   expand;

   solve;

   print NumShip;

quit;
```

PROC OPTMODEL Output

```
                          The OPTMODEL Procedure

Var NumShip[Chicago,Boston] >= 0
Var NumShip[Chicago,Newark] >= 0
Var NumShip[Chicago,Toronto] >= 0
Var NumShip[Detroit,Boston] >= 0
Var NumShip[Detroit,Newark] >= 0
Var NumShip[Detroit,Toronto] >= 0
Impvar FlowOut[Chicago] = NumShip[Chicago,Boston] + NumShip[Chicago,Newark] +
NumShip[Chicago,Toronto]
Impvar FlowOut[Detroit] = NumShip[Detroit,Boston] + NumShip[Detroit,Newark] +
NumShip[Detroit,Toronto]
Impvar FlowIn[Boston] = NumShip[Chicago,Boston] + NumShip[Detroit,Boston]
```

```
Impvar FlowIn[Newark] = NumShip[Chicago,Newark] + NumShip[Detroit,Newark]
Impvar FlowIn[Toronto] = NumShip[Chicago,Toronto] + NumShip[Detroit,Toronto]
Minimize TotalCost=40*NumShip[Chicago,Boston] + 25*NumShip[Chicago,Newark] +
15*NumShip[Chicago,Toronto] + 30*NumShip[Detroit,Boston] + 15*NumShip[Detroit,Newark] +
10*NumShip[Detroit,Toronto]
Constraint Supply_con[Chicago]: FlowOut[Chicago] <= 200
Constraint Supply_con[Detroit]: FlowOut[Detroit] <= 150
Constraint Demand_con[Boston]: FlowIn[Boston] >= 50
Constraint Demand_con[Newark]: FlowIn[Newark] >= 150
Constraint Demand_con[Toronto]: FlowIn[Toronto] >= 100
```

Problem Summary

Objective Sense	Minimization
Objective Function	TotalCost
Objective Type	Linear
Number of Variables	6
Bounded Above	0
Bounded Below	6
Bounded Below and Above	0
Free	0
Fixed	0
Number of Constraints	5
Linear LE (<=)	2
Linear EQ (=)	0
Linear GE (>=)	3
Linear Range	0
Constraint Coefficients	12

Solution Summary

Solver	LP
Algorithm	Dual Simplex
Objective Function	TotalCost
Solution Status	Optimal
Objective Value	5750
Primal Infeasibility	0
Dual Infeasibility	0
Bound Infeasibility	0
Iterations	7
Presolve Time	0.00
Solution Time	0.00

NumShip

	Boston	Newark	Toronto
Chicago	50	0	100
Detroit	0	150	0

The output of the PRINT statement for **NumShip** is formatted as a table because it is a two-dimensional array. PROC OPTMODEL formats sparse two-dimensional arrays as lists. (The PMATRIX option controls the tolerance level.)

5. Building Data Envelopment Analysis Models to Solve an Operations Problem

a. Run an input-oriented DEA to evaluate the efficiency of the regional, "South East 1" office. You run the code that is provided and view the results of PROC OPTMODEL.

You see that the South East 1 office has an efficiency score of .31, which means that this office is operating inefficiently and should reduce inputs that are used to approximately 32% of current levels.

```
                                The OPTMODEL Procedure

                                   Solution Summary

                                Solver                            LP
                                Algorithm                 Dual Simplex
                                Objective Function           Resources
                                Solution Status                Optimal
                                Objective Value          0.3106796117

                                Primal Infeasibility                0
                                Dual Infeasibility                  0
                                Bound Infeasibility                 0

                                Iterations                          6
                                Presolve Time                    0.00
                                Solution Time                    0.00

  SE1     SE2     SW1     SW2     NE1       NE2         NW1     NW2     ME1     ME2     MW1     MW2

    0       0       0       0       0    1.0874     0.07767       0       0       0       0       0
```

b. Edit the PROC OPTMODEL code so that DEA evaluates the South West 1 region office.

```
proc optmodel;

   /* Declare Variables */
   var SE1 >=0, SE2 >=0, /*Regions: South_East1 and South_East2*/
      SW1 >=0, SW2 >=0,  /*Regions: South_West1 and South_West2*/
      NE1 >=0, NE2 >=0,  /*Regions: North_East1 and North_East2*/
      NW1 >=0, NW2 >=0,  /*Regions: North_West1 and North_West2*/
      ME1 >=0, ME2 >=0,  /*Regions: Mid_East1 and Mid_East2*/
      MW1 >=0, MW2 >=0;  /*Regions: Mid_West1 and Mid_West2*/
   var H;                 /*Objective variable*/

   /* Declare Objective Function */
   min Resources=H;

   /*****************************/
   /* Declare Input Constraints */
   /*****************************/
   /*Account Executives*/
   con AEs: SE1*4 + SE2*5 + SW1*4 + SW2*2 +
            NE1*3 + NE2*1 + NW1*2 + NW2*2 +
            ME1*3 + ME2*1 + MW1*4 +MW2*3 <= 4*H;
```

```
   /*Account Managers*/
   con AMs: SE1*15 + SE2*7 + SW1*4 + SW2*10 +
            NE1*10 + NE2*4 + NW1*4 + NW2*4 +
            ME1*3 + ME2*11 + MW1*12 +MW2*8 <= 4*H;
   /*****************************/
   /* Declare Output Constraints*/
   /*****************************/

   /*New Policies*/
   con NP:  SE1*32 + SE2*25 + SW1*33 + SW2*34 +
            NE1*28 + NE2*27 + NW1*34 + NW2*25 +
            ME1*14 + ME2*22 + MW1*19 +MW2*20 >= 33;
   /*Old Policies*/
   con OP:  SE1*51 + SE2*77 + SW1*98 + SW2*58 +
            NE1*53 + NE2*71 + NW1*78 + NW2*91 +
            ME1*75 + ME2*94 + MW1*90 +MW2*91 >= 98;
      /*Modified Policies*/
   con MP:  SE1*31 + SE2*21 + SW1*24 + SW2*21 +
            NE1*22 + NE2*45 + NW1*30 + NW2*37 +
            ME1*24 + ME2*36 + MW1*36 +MW2*43 >= 24;

   /* Call the Solver */
   solve;

   /* Print Solutions */
   print SE1 SE2 SW1 SW2 NE1 NE2 NW1 NW2 ME1 ME2 MW1 MW2;

quit;
```

The objective value in the output indicates that the South West 1 office is operating efficiently.

```
                          The OPTMODEL Procedure

                             Solution Summary

                        Solver                            LP
                        Algorithm                 Dual Simplex
                        Objective Function           Resources
                        Solution Status              Optimal
                        Objective Value                    1

                        Primal Infeasibility    1.776357E-15
                        Dual Infeasibility                 0
                        Bound Infeasibility                0

                        Iterations                         8
                        Presolve Time                   0.00
                        Solution Time                   0.00


        SE1     SE2     SW1     SW2     NE1     NE2     NW1     NW2     ME1     ME2     MW1     MW2

          0       0       1       0       0       0       0       0       0       0       0       0
```

c. Edit the PROC OPTMODEL code so that you are now using an output-oriented DEA model to evaluate the Mid-West 2 region office.

```
proc optmodel;

   /* Declare Variables */
   var SE1 >=0, SE2 >=0, /*Regions: South_East1 and South_East2*/
      SW1 >=0, SW2 >=0,  /*Regions: South_West1 and South_West2*/
      NE1 >=0, NE2 >=0,  /*Regions: North_East1 and North_East2*/
      NW1 >=0, NW2 >=0,  /*Regions: North_West1 and North_West2*/
      ME1 >=0, ME2 >=0,  /*Regions: Mid_East1 and Mid_East2*/
      MW1 >=0, MW2 >=0;  /*Regions: Mid_West1 and Mid_West2*/
   var H;                /*Objective variable*/

   /* Declare Objective Function */
   Max Production=H;

   /*****************************/
   /* Declare Input Constraints */
   /*****************************/
   /*Account Executives*/
   con AEs: SE1*4 + SE2*5 + SW1*4 + SW2*2 +
            NE1*3 + NE2*1 + NW1*2 + NW2*2 +
            ME1*3 + ME2*1 + MW1*4 +MW2*3 <= 3;
   /*Account Managers*/
   con AMs: SE1*15 + SE2*7 + SW1*4 + SW2*10 +
            NE1*10 + NE2*4 + NW1*4 + NW2*4 +
            ME1*3 + ME2*11 + MW1*12 +MW2*8 <= 8;
    /*****************************/
   /* Declare Output Constraints*/
   /*****************************/

   /*New Policies*/
   con NP:  SE1*32 + SE2*25 + SW1*33 + SW2*34 +
            NE1*28 + NE2*27 + NW1*34 + NW2*25 +
            ME1*14 + ME2*22 + MW1*19 +MW2*20 >= 20*H;
   /*Old Policies*/
   con OP:  SE1*51 + SE2*77 + SW1*98 + SW2*58 +
            NE1*53 + NE2*71 + NW1*78 + NW2*91 +
            ME1*75 + ME2*94 + MW1*90 +MW2*91 >= 91*H;
   /*Modified Policies*/
   con MP:  SE1*31 + SE2*21 + SW1*24 + SW2*21 +
            NE1*22 + NE2*45 + NW1*30 + NW2*37 +
            ME1*24 + ME2*36 + MW1*36 +MW2*43 >= 43*H;

   /* Call the Solver */
   solve;

   /* Print Solutions */
   print SE1 SE2 SW1 SW2 NE1 NE2 NW1 NW2 ME1 ME2 MW1 MW2;

quit;
```

The objective value in the output indicates that the Mid-West 2 office is not operating efficiently and can expand production by approximately 78%.

The OPTMODEL Procedure

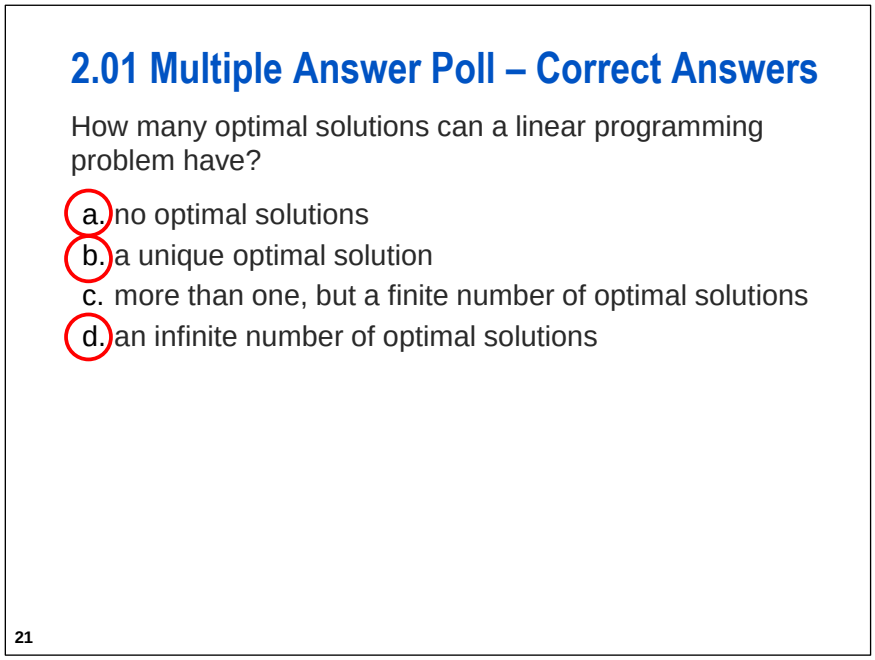
Solution Summary

Solver	LP
Algorithm	Dual Simplex
Objective Function	Production
Solution Status	Optimal
Objective Value	1.7802197802
Primal Infeasibility	0
Dual Infeasibility	0
Bound Infeasibility	0
Iterations	6
Presolve Time	0.00
Solution Time	0.00

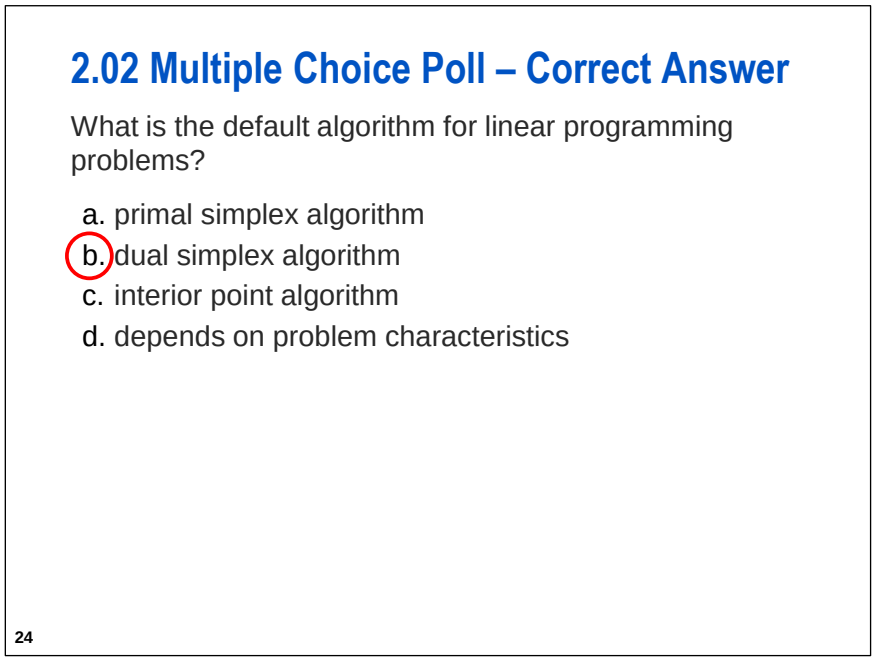
SE1	SE2	SW1	SW2	NE1	NE2	NW1	NW2	ME1	ME2	MW1	MW2
0	0	0	0	0	1	0	1	0	0	0	0

End of Solutions

Solutions to Student Activities (Polls/Quizzes)



A linear programming problem has a finite number of extreme point solutions, but these are not the only feasible solutions. Any point on the line between two optimal solutions is an optimal solution.



For linear programming problems, the default algorithm does not depend on problem characteristics.

The LP solver must be specified so that you can set solver options.

2.03 Multiple Answer Poll – Correct Answers

Which of the following statements are true of SAS names?

a. SAS names can be up to 32 characters long.
b. SAS names cannot start with a number.
c. SAS names are case sensitive.

39

SAS names are ***not*** case sensitive.

2.04 Multiple Choice Poll – Correct Answer

Which potential index set for the parameter array **selling_price** is most indicative of the problem context?

	Selling Price ($)
Desks	94
Chairs	79
Bookcases	125
Bedframes	109

PRODUCTS

a. {1,2,3,4}
b. {'P1','P2','P3','P4'}
c. {'desks','chairs','bookcases','bedframes'}

58

The index set **PRODUCTS** can have the product names as elements. (In this case, it is a set of strings.)

2.05 Multiple Answer Poll – Correct Answers

Which of the following statements are true of parameters and parameter arrays in PROC OPTMODEL?

a. Parameters must be declared before they are used.
b. The type of a set must be declared.
c. Parameters must be initialized when they are declared.
d. All parameter values can be modified after they are declared.
e. Parameter values cannot be modified by the solver.

73

There is a default type for sets if none is declared and there are no initializers. Even parameters that are used in defining index sets for other parameters, such as the parameter **n** in the examples, need only to be declared, and not assigned values.

2.07 Multiple Choice Poll – Correct Answer

Which of the following does ***not*** need to be declared in PROC OPTMODEL?

a. parameter arrays
b. decision variables
c. implicit variables
d. local dummy parameters

87

2.08 Multiple Choice Poll – Correct Answer

Is the following PROC OPTMODEL an input-oriented or output-oriented DEA model?

```
VAR  dmu_1 >=0, dmu_2 >=0, H;

MIN  resources=H;

CON  inputs: dmu_1*70 + dmu_2*57 <=70*H;
CON  outputs:dmu_1*91 + dmu_2*99 >=91;
```

a. input-oriented

$$\begin{aligned} &\min \theta \\ &\text{s.t.} \quad X\lambda \le \theta X_o \\ &\qquad Y\lambda \ge Y_o \\ &\qquad \theta \text{ free}, \lambda \ge 0 \end{aligned}$$

b. output-oriented

$$\begin{aligned} &\max \phi \\ &\text{s.t.} \quad X\lambda \le X_o \\ &\qquad Y\lambda \ge \phi Y_o \\ &\qquad \phi \text{ free}, \lambda \ge 0 \end{aligned}$$

146

2.09 Multiple Choice Poll – Correct Answer

Which of the following correctly defines the DEA model coded below?

```
VAR  dmu_1 >=0, dmu_2 >=0, H;

MAX  production=H;

CON  inputs: dmu_1*70 + dmu_2*57 <=57;
CON  outputs:dmu_1*91 + dmu_2*99 >=99*H;
```

a. input-oriented DEA model evaluating the efficiency of DMU_1

b. output-oriented DEA model evaluating the efficiency of DMU_1

c. output-oriented DEA model evaluating the efficiency of DMU_2

148

Chapter 3 Nonlinear Programming Problems

3.1 Introduction to Nonlinear Programming

Objectives

- Describe how, conceptually and geometrically, iterative improvement algorithms solve nonlinear programming problems.
- Explain which types of nonlinear programming problems can be solved in general.
- Explain the optimality conditions for nonlinear programming problems.

3

Nonlinear Programming Problems

The general form of a nonlinear programming problem is

$$\begin{aligned} &\min|\max \quad f(\mathbf{x}) \\ &\text{subject to} \quad c_i(\mathbf{x}) \ \{\leq,=,\geq\} \ b_i \ \ (i=1,2,...,m) \\ &\qquad\qquad l_j \leq x_j \leq u_j \ \ (j=1,2,...,n) \end{aligned}$$

where $f(\mathbf{x})$ and $c_1(\mathbf{x}), \ldots, c_m(\mathbf{x})$ are continuous functions. NLPs can be classified according to their constraints.

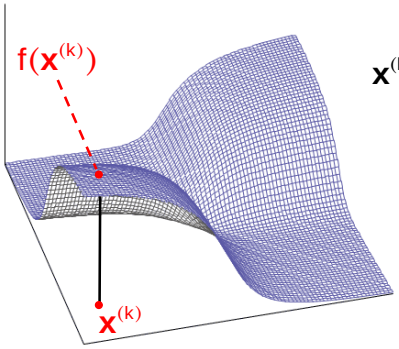
- **unconstrained**: no constraints or bounds
- **bound constrained**: no constraints other than bounds
- **linearly constrained**: $c_1(\mathbf{x}), \ldots, c_m(\mathbf{x})$ are linear functions.
- **nonlinearly constrained**: One or more of the functions $c_i(\mathbf{x})$ is nonlinear.

4

The general strategy is illustrated below for a bound-constrained nonlinear programming problem.

General Strategy for Solving NLPs

Most of the algorithms for solving NLPs are iterative.



$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{s}^{(k)}$$

$\mathbf{x}^{(k)}$ iterate

$\mathbf{s}^{(k)}$ search direction

$\alpha^{(k)}$ step length

Local information at $\mathbf{x}^{(k)}$ is used to find the direction $\mathbf{s}^{(k)}$, so essentially $f(\mathbf{x})$ is approximated by a quadratic at $\mathbf{x}^{(k)}$.

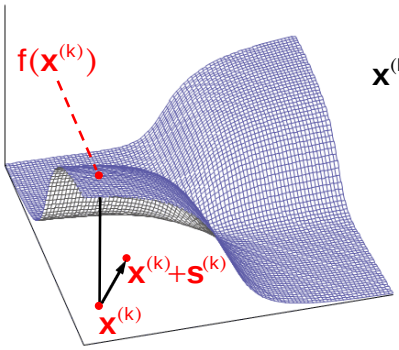
5 ...

To minimize this objective function, a straightforward choice of search direction would be the negative gradient of the objective at the current iterate (corresponding to the initial direction that a marble would roll from this point).

The NLP algorithms in PROC OPTMODEL use more information about the objective function to determine the search direction.

General Strategy for Solving NLPs

Most of the algorithms for solving NLPs are iterative.



$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{s}^{(k)}$$

$\mathbf{x}^{(k)}$ iterate

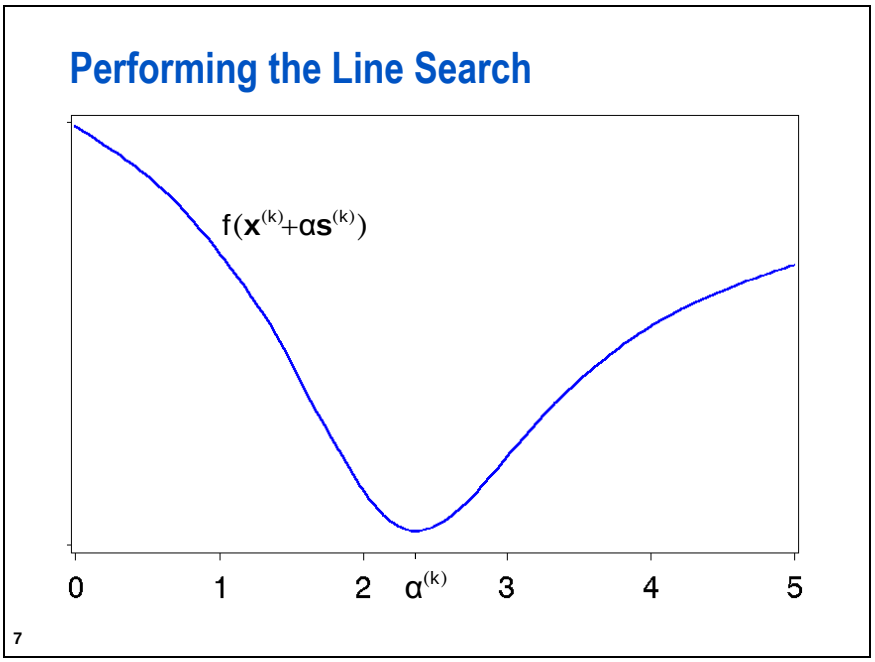
$\mathbf{s}^{(k)}$ search direction

$\alpha^{(k)}$ step length

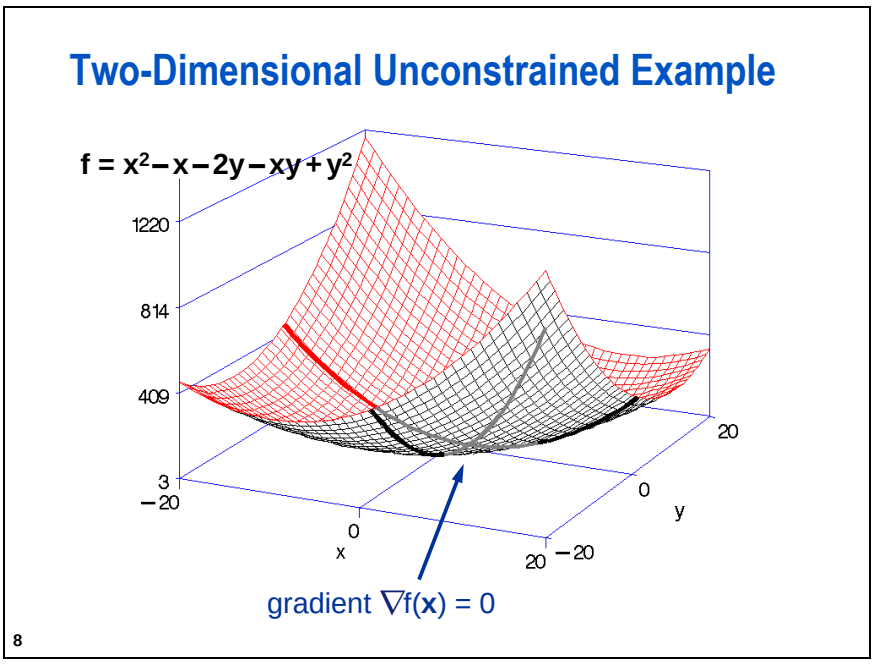
Local information at $\mathbf{x}^{(k)}$ is used to find the direction $\mathbf{s}^{(k)}$, so essentially $f(\mathbf{x})$ is approximated by a quadratic at $\mathbf{x}^{(k)}$.

6

The one-dimensional optimization problem of minimizing or maximizing $f(\mathbf{x}^{(k+1)}) = f(\mathbf{x}^{(k)} + \alpha\mathbf{s}^{(k)})$ with respect to α to find the step length $\alpha^{(k)}$ is called the *line search* and is often only solved approximately.



In Chapter 1, PROC OPTMODEL is used to minimize a simple polynomial function of two variables with no constraints or bounds. The graph of this unconstrained problem appears below.

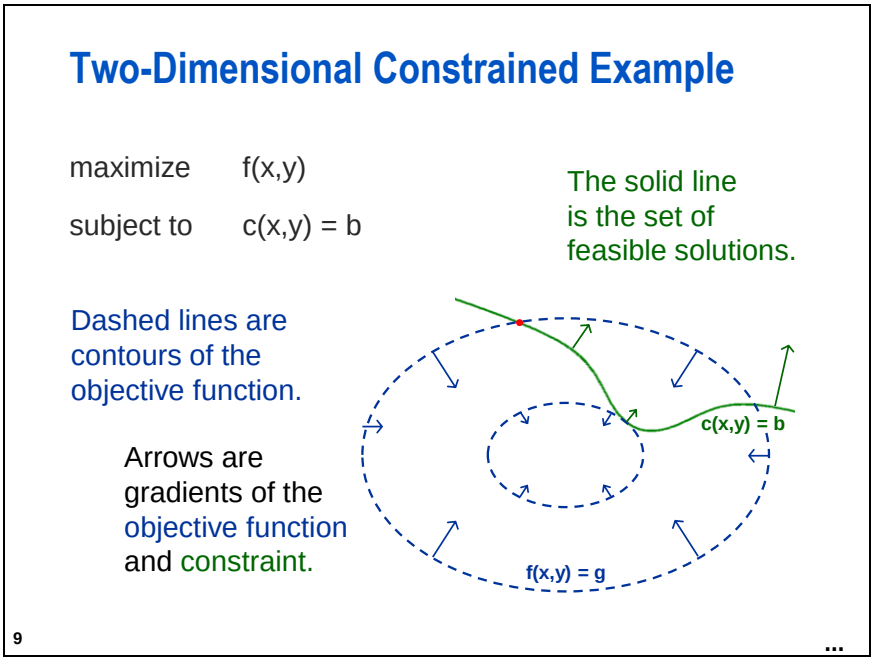


At the optimal solution [x,y] = [1⅓,1⅔], the gradient of the objective function is zero. At any other point, the negative gradient of the objective function serves as a search direction.

For an unconstrained nonlinear programming problem, solvers search for a point where the gradient is zero.

When linear equations are added to the problem, the gradient can be projected onto the feasible region. For linear inequalities such as the simple bound constraints $x \geq 0$ and $y \leq 0$, active set methods (which maintain an active set of tight inequality constraints) can project the gradient to find a search direction in the space of active constraints. Solving the line search problem can then indicate additional constraints to add to the active set. If the projected gradient is zero, some constraints might need to be removed from the active set to find a search direction.

The optimality conditions for nonlinear programming problems can be explained using a simple nonlinearly constrained nonlinear programming problem.



The point shown at the intersection of the solid line and the outer contour line is a feasible solution to the constrained problem, but not the optimal solution. Where is the optimal solution?

Two-Dimensional Constrained Example

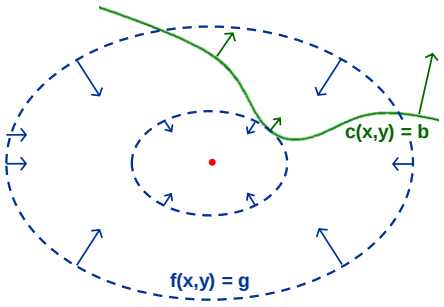
maximize f(x,y)

subject to ~~c(x,y) = b~~

The solid line is the set of feasible solutions.

Dashed lines are contours of the objective function.

Arrows are gradients of the objective function and constraint.



10 ...

The solution to the unconstrained problem is at the "top of the hill," but the value of c(x,y) is less than b at that point. One approach to constrained optimization is to penalize those points with c(x,y) < b, and add a multiple of the constraint violation to the objective function. What does this do to the optimal solution?

Two-Dimensional Constrained Example

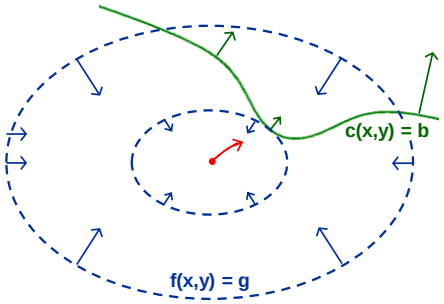
maximize $f(x,y) - \lambda[c(x,y) - b]$

subject to ~~c(x,y) = b~~

The solid line is the set of feasible solutions.

Dashed lines are contours of the objective function.

Arrows are gradients of the objective function and constraint.



11 ...

As the penalty is increased, the trajectory of the optimal solution approaches the set of feasible solutions.

Two-Dimensional Constrained Example

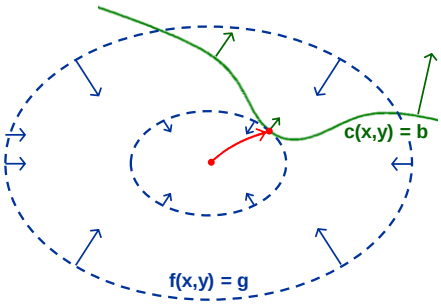
maximize $f(x,y) - \lambda^*[c(x,y) - b]$

subject to ~~$c(x,y) = b$~~

The solid line is the set of feasible solutions.

Dashed lines are contours of the objective function.

Arrows are gradients of the objective function and constraint.



12

The non-linearity of the problem does not change the Lagrangian. The form of the function is identical to the Lagrangian shown for the Linear Programming problem. The constraint, or second term, in the Lagrangian function "penalizes" points that do not satisfy the constraint.

Optimality Conditions: Lagrangian Function

For multipliers $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_m]$, the *Lagrangian function*

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i [c_i(\mathbf{x}) - b_i]$$

can be used to state optimality conditions for NLPs.

Necessary Conditions: An optimal solution **x*** must satisfy

$$\nabla L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = 0$$
$$\lambda_i^* [c_i(\mathbf{x}^*) - b_i] = 0 \quad (i = 1,2,\ldots,m)$$

for some *Lagrange multipliers* **λ***. For inequality constraints, each λ_i^* must be nonnegative or nonpositive (depending on whether the objective sense is ***min*** or ***max***).

14

The first necessary condition says that the gradient of the objective function is a linear combination of the gradients of the bodies of the constraints. The second necessary condition (known as the *complementarity* condition) ensures that the Lagrangian has the same value as the objective at **x***.

The nonlinear programming solvers all find both an optimal solution **x*** and Lagrange multipliers **λ*** that are a (first-order) "certificate" of the optimality of **x***.

Minimizing the Banana Function Using PROC OPTMODEL

```
proc optmodel;
   var x >= 0, y <= 0;

   min f = (y-x**2)**2 + (1-x)**2;

   solve;

   print x x.dual y y.dual;

quit;
```

PROC OPTMODEL Output

The OPTMODEL Procedure

Problem Summary

Objective Sense	Minimization
Objective Function	f
Objective Type	Nonlinear
Number of Variables	2
Bounded Above	1
Bounded Below	1
Bounded Below and Above	0
Free	0
Fixed	0
Number of Constraints	0

Solution Summary

Solver	NLP
Algorithm	Interior Point
Objective Function	f
Solution Status	Optimal
Objective Value	0.2892740938
Optimality Error	5E-7
Infeasibility	0
Iterations	4
Presolve Time	0.00
Solution Time	0.01

x	x.DUAL	y	y.DUAL
0.58975	0.000001571	-9.6298E-07	-0.69562

The true Lagrange multipliers for the constraints $x \geq 0$ and $y \leq 0$ are 0 and –0.69562.

Are the complementarity conditions satisfied for the solution [x*,y*] = [0.58975,0]?

$0(x^* - 0) = 0(0.58976) = 0$

$-0.69562(y^* - 0) = -0.69562(0) = 0$

What is the Lagrangian function?

$f - 0(x - 0) - (-0.69562)(y - 0) = f + 0.69562y$

The necessary conditions imply that up to second-order terms,

$$f(\mathbf{x}^*+\mathbf{dx}) \approx f(\mathbf{x}^*) + \sum_{i=1}^{m} \lambda_i^* [c_i(\mathbf{x}^*+\mathbf{dx}) - b_i]$$

for any point x* that satisfies the gradient and complementarity conditions. This helps explain the sign conditions on the Lagrange multipliers for inequality constraints.

End of Demonstration

Interpretation of Lagrange Multipliers

- Like dual values, the (nonzero) Lagrange multiplier of a constraint can be interpreted as follows:

$$\lambda_i^* = \frac{\text{Change in optimal objective}}{\text{Unit increase in constraint limit}}$$

- This assumes that the boundary point determining the optimal solution is not over-determined.
- Exercise caution because boundaries can now be curved.

16

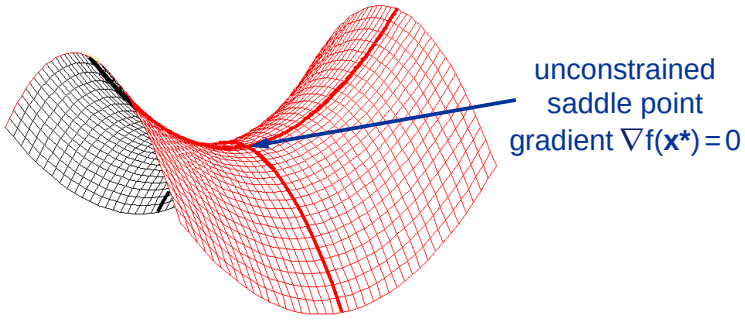
Recall that, in the linear case, this interpretation was good only until moving the constraint (by changing the limit) made the optimal solution violate another constraint.

In the nonlinear case, the curved nature of the boundary means that the effect of a unit change in the limit can depend on the values of the decision variables, as in the law of diminishing returns.

Most methods terminate when the first-order conditions are approximately satisfied. However, the first-order conditions are not sufficient to guarantee the optimality of the solution, even for an unconstrained nonlinear programming problem.

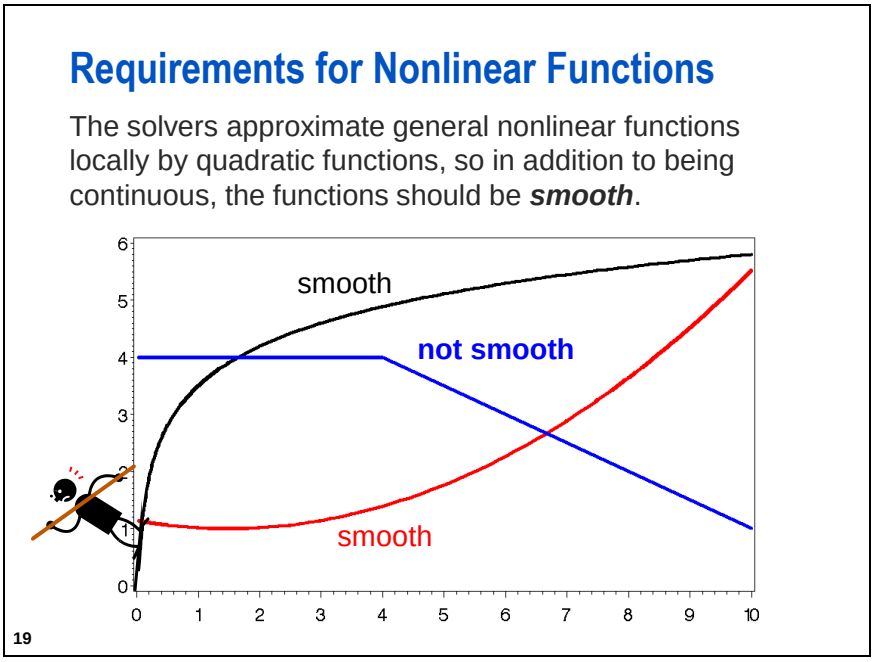
Second-Order Conditions: The Hessian

Second-order conditions are necessary to avoid saddle points. They involve the Hessian $\nabla^2 f(\mathbf{x}^*)$ or $\nabla^2 L(\mathbf{x}^*, \boldsymbol{\lambda}^*)$.



17

At a saddle point, the function is locally flat. This can cause problems for methods that do not use the Hessian matrix of second-order partial derivatives of the objective function.

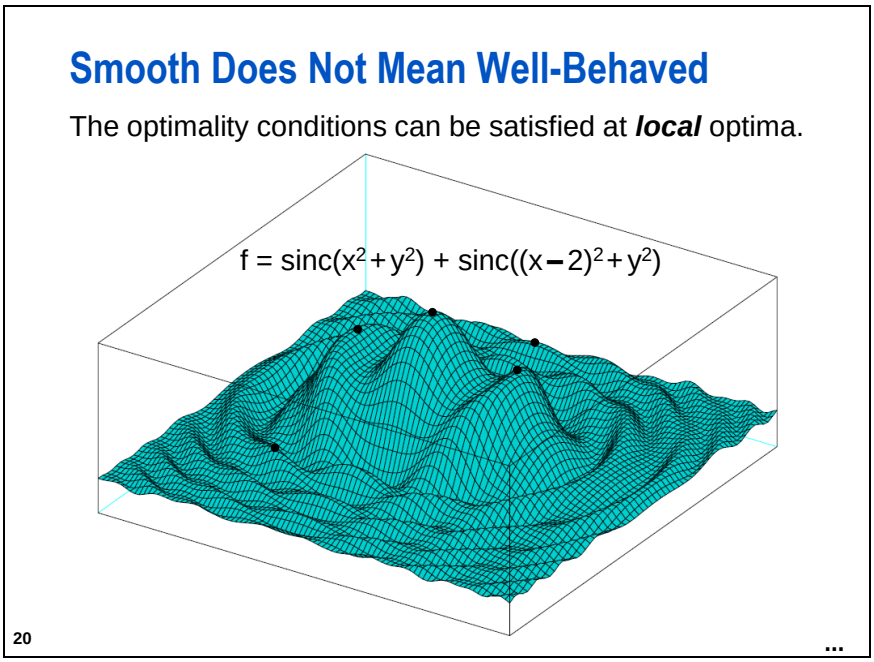


More specifically:

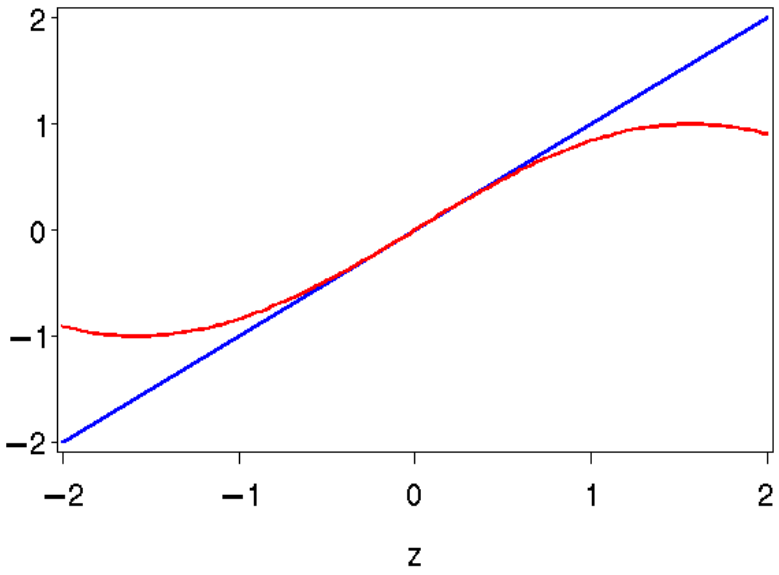
- The objective function f(**x**) should be finite for all points in the feasible region, and in some cases, for all values of the decision variables that satisfy the bound constraints. (One of the functions above could present a problem.)
- The gradient ∇f(**x**) should be defined and continuous for all points in the feasible region, as should the Jacobian (the matrix whose rows are the gradients of the constraints).
- The solvers use or approximate the Hessian (the matrix of second partial derivatives of the objective function) and might require it to be continuous.

Computing the gradient takes more CPU time than computing the objective function value. Computing the Hessian matrix can take much more CPU time and memory than computing the gradient, especially when there are many decision variables.

Generally speaking, optimization techniques that do not use the Hessian usually require more iterations than techniques that do use the Hessian. The former tend to be slower and less reliable. However, the techniques that use the Hessian can be prohibitively slow for larger problems.



This bound-constrained nonlinear programming problem has a single global maximum but at least 72 other local maxima. The sampling function is defined as sinc(0)=1 and sinc(z) = sin(z)/z for $z \neq 0$.



sin(z)

Solving the Sinc Problem Using PROC OPTMODEL

```
proc optmodel;
   /* declare variables */
   var x >= -4 <= 6 init 0;
   var y >= -5 <= 5 init 0;

   /* declare objective */
   max SincQuad = if (x = 0 and y = 0) then
       1 + sin((x-2)**2+y**2)/((x-2)**2+y**2)
      else if (x = 2 and y = 0) then
         sin(x**2+y**2)/(x**2+y**2) + 1
      else sin(x**2+y**2)/(x**2+y**2) +
         sin((x-2)**2+y**2)/((x-2)**2+y**2);

   solve;

   print x x.dual y y.dual;

   /* alternate initial point */
   x = 0.275;
   y = 4.937;

   solve;

   print x x.dual y y.dual;

quit;
```

Notice that IF-THEN/ELSE statements are used to define the objective function in case [x,y] = [0,0] or [2,0], where the limiting value is used instead of the formula to prevent division by zero.

Using the analytic (limiting) formula yields sinc'(0) = 0 and sinc''(0) = –⅓. Replacing sinc(z) by 1 at z = 0 should not adversely affect the NLP algorithms.

PROC OPTMODEL Output (omitting the second problem summary)

The OPTMODEL Procedure

Problem Summary

Objective Sense	Maximization
Objective Function	SincQuad
Objective Type	Nonlinear
Number of Variables	2
Bounded Above	0
Bounded Below	0
Bounded Below and Above	2
Free	0
Fixed	0
Number of Constraints	0

Solution Summary

Solver	NLP
Algorithm	Interior Point
Objective Function	SincQuad
Solution Status	Optimal
Objective Value	1.6829419696
Optimality Error	5E-7
Infeasibility	0
Iterations	3
Presolve Time	0.00
Solution Time	0.01

x	x.DUAL	y	y.DUAL
1	-6.6613E-16	0	0

Solution Summary

Solver	NLP
Algorithm	Interior Point
Objective Function	SincQuad
Solution Status	Optimal
Objective Value	0.0586579039
Optimality Error	5E-7
Infeasibility	0
Iterations	3
Presolve Time	0.00
Solution Time	0.01

x	x.DUAL	y	y.DUAL
1	-3.8416E-07	5	0.47508

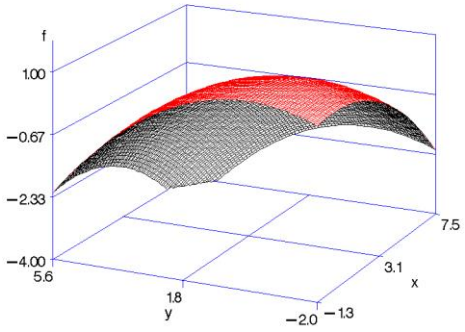
Notice the following:

- The second solution cannot be the global optimal solution (because $0.058658 < 1.682942$).
- The bound $y \leq 5$ is satisfied with equality.
- The corresponding Lagrange multiplier is positive.

End of Demonstration

3.01 Quiz

Is the objective function $f(x,y) = 1 - 0.05x^2 - 0.1y^2$ of the two-dimensional inequality constrained problem concave?

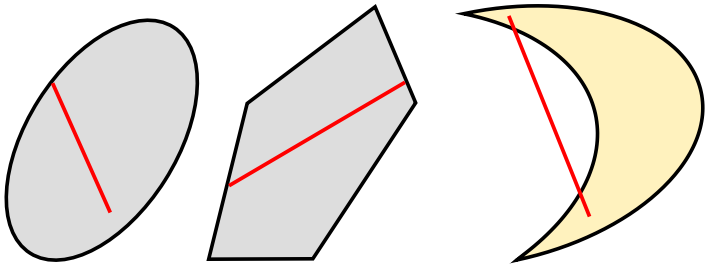


24

Remember the adage "concave looks like a cave."

Conditions That Guarantee Global Optima

If the feasible region is ***convex***, the local minimum of a ***convex*** function (or the local maximum of a ***concave*** function) is globally optimal.



two convex sets **non-convex set**

27

A set of points is convex if it contains every line that connects two points in the set. In the example with the sinc functions, the feasible region is ***convex*** (the feasible region is a rectangle because the problem is bound constrained), but the objective function is not ***concave***.

The following is a basic fact about convex feasible regions:

- Problems with linear constraints or inequalities $c_i(\mathbf{x}) \leq b_i$ whose body $c_i(\mathbf{x})$ is convex have convex feasible regions.

The ellipsoid on the left could be the feasible region of a problem with a single inequality constraint whose body is a convex quadratic function. The pentagon in the middle could be the feasible region of a problem with five linear inequalities. The crescent on the right could be the feasible region of a problem with two inequality constraints whose bodies are quadratic functions: one convex and the other concave.

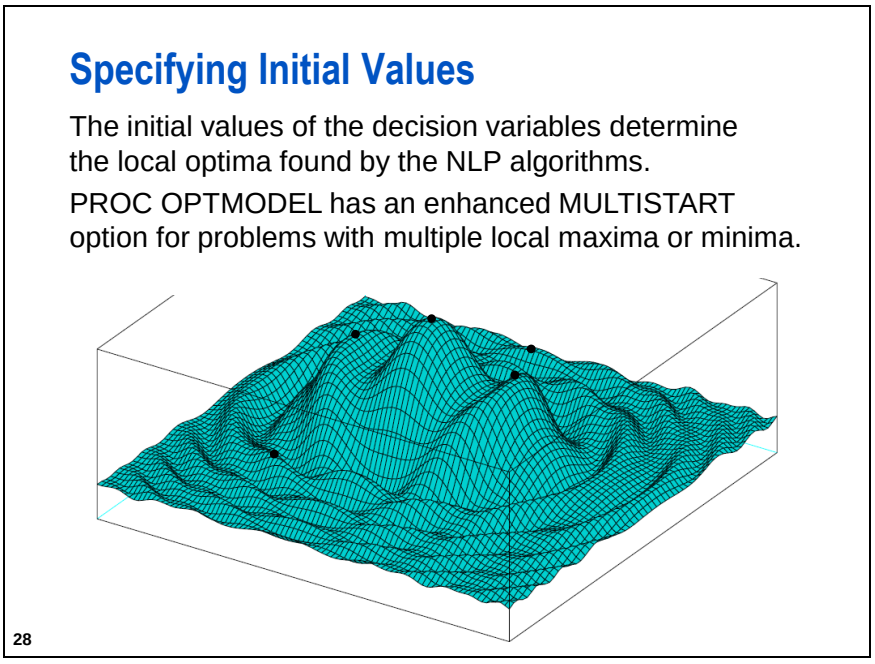
Some basic facts about convex functions are as follows:

- $f(x)$ is convex if $f''(x) \geq 0$ (for example, $f(x) = \exp\{ax\}$ is convex).
- If $f(x)$ and $g(x)$ are convex, then so is the sum $f(x) + g(x)$.
- $\mathbf{x}^T\mathbf{P}\mathbf{x}$ is convex if the matrix $\mathbf{P}$ is positive semi-definite.
- Any norm $\|\mathbf{x}\|_p = (\sum_i x_i^p)^{1/p}$ for $p \geq 1$ is convex.
- $f(\mathbf{x})$ is convex if the Hessian is positive semi-definite.
- If $f(\mathbf{y})$ is convex, then so is $f(\mathbf{A}\mathbf{x} + \mathbf{b})$ as a function of $\mathbf{x}$.

All you ever need to know about concave functions is the following:

- $f(\mathbf{x})$ is concave if and only if $-f(\mathbf{x})$ is convex.

MULTISTART Options



Determining which initial values lead to a particular local optima, which is somewhat analogous to the domain of attraction for chaotic systems, is complicated by varying step sizes.

MULTISTART Solver Option

If the MULTISTART solver option is specified, the solver randomly samples the feasible region and generates a number of starting points. The points are separated both from each other and from known local optima.

- The best solution that is found is reported as the optimal solution.
- The starting point that leads to the optimal solution can be accessed using the **.msinit** suffix.
- A separate multistart iteration log is produced.

29

The MULTISTART solver option takes advantage of multiple threads to execute optimization runs from multiple starting points in parallel.

MULTISTART Suboptions

Suboption	Description
MAXSTARTS=	Maximum number of starting points (default 100)
MAXTIME=	Upper limit in seconds of real time in multistart mode, checked after each completion of the local solver
BNDRANGE=	Maximum range of values for each decision variable, centered at the initial value (default 200)
DISTTOL=	Distance tolerance for distinct local optima (default 1e-6)
LOGLEVEL=	Amount of information in iteration log

30

Solving the Sinc Problem Using PROC OPTMODEL MULTISTART Options

```
proc optmodel;
   /* declare variables */
   var x >= -4 <= 6 ;
   var y >= -5 <= 5 ;

   /* declare objective */
   max SincQuad = if (x = 0 and y = 0) then
                  1 + sin((x-2)**2+y**2)/((x-2)**2+y**2)
      else if (x = 2 and y = 0) then
         sin(x**2+y**2)/(x**2+y**2) + 1
      else sin(x**2+y**2)/(x**2+y**2) +
         sin((x-2)**2+y**2)/((x-2)**2+y**2);

   solve with nlp / multistart=(maxstarts=100) seed=12345;

   print x x.dual y y.dual;

quit;
```

The syntax above is similar to the syntax shown in the previous demonstration. Initial value options in the VARIABLE declaration statements are removed, and MULTISTART options are added to the SOLVE statement.

The optimum found by starting at initial values x=0 and y=0 in the previous demonstration appears to be the global optimum.

The OPTMODEL Procedure

Solution Summary

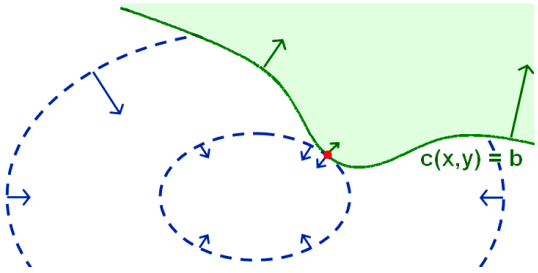
Solver	Multistart NLP
Algorithm	Interior Point
Objective Function	SincQuad
Solution Status	Optimal
Objective Value	1.6829419696
Number of Starts	39
Number of Sample Points	1600
Number of Distinct Optima	19
Random Seed Used	12345
Optimality Error	5E-7
Infeasibility	0
Presolve Time	0.00
Solution Time	18.05

x	x.DUAL	y	y.DUAL
1	2.3536E-11	-7.2777E-13	8.7672E-13

End of Demonstration

3.02 Quiz

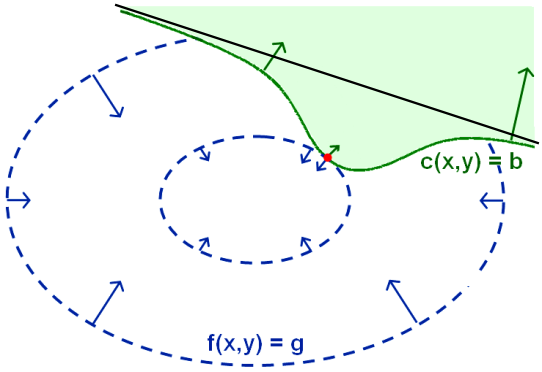
Is the feasible region of the two-dimensional inequality constrained problem convex?



32

Sufficient (Not Necessary) Conditions

The two-dimensional inequality constrained problem does ***not*** satisfy the conditions, but has no other local maxima.



34

The set of points with c(x,y) = b is smoothly asymptotic to the line pictured, so there are no other local maxima. The conditions that guarantee that all local optimal solutions are globally optimal are *sufficient*, but not *necessary* conditions.

3.2 Solving Nonlinear Programming Problems Using the OPTMODEL Procedure

Objectives

- Solve nonlinear programming problems using the OPTMODEL procedure.
- Interpret the information written to the SAS log during the solution of a nonlinear programming problem.

36

Overview of the OPTMODEL Procedure

How does PROC OPTMODEL solve NLPs?

- PROC OPTMODEL recognizes linear and quadratic functions in the objective and constraints.
- The QP solver is used to minimize convex (maximize concave) quadratic problems with linear constraints.
- The INTERIORPOINT algorithm is used to solve all other classes of nonlinear programming problems.
- The ACTIVESET algorithm can also be selected.
- Automatic differentiation is used to compute the gradient and the Jacobian and Hessian matrices.
- All algorithms take full advantage of sparsity in the objective function (Hessian) and the constraints (Jacobian).

38

PROC OPTMODEL can also determine whether a quadratic function is convex or concave.

NLP Solver Iteration Log

In addition to the solution summary, the NLP algorithms (INTERIORPOINT and ACTIVESET) produce an iteration log that contains the following:

- iteration number
- objective value
- infeasibility
- optimality error

The LOGFREQ=*k* option controls the frequency of printing to the iteration log (every *k* iterations).

39

For all NLP algorithms, the default is LOGFREQ=1.

Termination Criteria for the NLP Solver

Solver Option	Description
FEASTOL=	Tolerance for constraint violation at an optimal solution (default 1e-6)
OPTTOL=	Tolerance for Lagrangian gradient norm at an optimal solution (default 1e-6)
OBJLIMIT=	Upper limit on objective for detecting unboundedness (default 1e+20)
MAXITER=	Maximum number of iterations (default 5000)
MAXTIME=	Upper limit in seconds, checked at the end of each iteration (default infinity)

40

Both tolerances must be met for the NLP solver to terminate with an optimal solution.

Example: Portfolio Optimization Problem

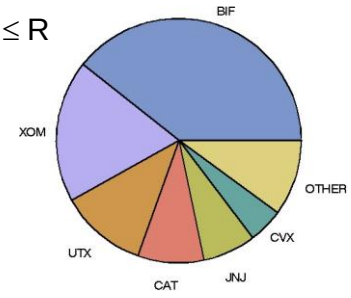
Select a portfolio that maximizes expected return while not exceeding a specified risk R (measured by variance).

maximize $\sum_{j=1}^{n} r_j x_j$

subject to $\sum_{i=1}^{n}\sum_{j=1}^{n} x_i \sigma_{ij} x_j \le R$

$\sum_{j=1}^{n} x_j = 1$

$x_1 \ge 0, \ldots, x_n \ge 0$



x_j proportion of stock j

r_j expected return of stock j

σ_{ij} covariance between stocks i and j

41

The expected returns for this problem are based on historical stock market data from 1998 through 2008. The covariance matrix is estimated using the shrinkage estimator of Ledoit and Wolf (2003).

The body of the risk constraint, which is the variance of the portfolio's annual return, is a convex function. This implies that the feasible region is convex, so the optimal solution is a global maximum.

The target risk value used in the demonstration is R = 200.

Solving a Portfolio Optimization Problem Using PROC OPTMODEL

```
%let expected_risk = 200;
proc optmodel;
   /* declare sets and parameters */
   set ASSETS = /MMM AA AXP AIG T BAC BA CAT CVX C/;
   num return {ASSETS} =
      [9.85 10.90 7.32 1.31 5.74 4.58 11.15 16.21 11.92 7.37];
   num covariance {ASSETS, ASSETS} =
[427.77 318.95 170.78 143.20 90.28 105.25 203.89  252.32 108.30 200.50
 318.95 1443.56 291.85 183.88 226.06 219.65 302.50 489.91 242.84 408.71
 170.76 291.85 620.01 241.68 162.42 220.33 269.85 243.93 136.24 413.14
 143.20 183.88 241.68 806.94 136.94 144.73 187.12 192.30 127.08 294.13
  90.28 226.06 162.42 136.94 597.30 109.05 149.69 185.79 108.41 233.04
 105.25 219.65 220.33 144.73 109.05 570.09 125.59 180.80 122.89 383.32
 203.89 302.50 269.85 187.12 149.69 125.58 781.05 280.31 173.51 251.25
 252.32 489.91 243.93 192.30 185.79 180.80 280.31 838.37 220.19 331.84
 108.30 242.84 136.24 127.08 108.41 122.89 173.51 220.19 427.53 175.78
200.50 408.71 413.14  294.13 233.04 383.32 251.25 331.84 175.78
1358.79]
;

/* declare variables */
   var Prop {ASSETS} >= 0;

/* declare constraints */
   con Portfolio: sum {j in ASSETS} Prop[j] = 1;

   con Variance: sum {i in ASSETS, j in ASSETS}
     covariance[i,j] * Prop[i] * Prop[j] <= &expected_risk;

/* declare objective */
   max ExpectedReturn = sum {j in ASSETS} return[j] * Prop[j];

   solve ;

   print {j in ASSETS: Prop[j]>1e-4} Prop
     {j in ASSETS: Prop[j]>1e-4} return;

   print Variance.ub Variance.dual;


quit;
```

PROC OPTMODEL Output

The OPTMODEL Procedure

Problem Summary

Objective Sense	Maximization
Objective Function	ExpectedReturn
Objective Type	Linear
Number of Variables	10
Bounded Above	0
Bounded Below	10
Bounded Below and Above	0
Free	0
Fixed	0
Number of Constraints	2
Linear LE (<=)	0
Linear EQ (=)	1
Linear GE (>=)	0
Linear Range	0
Nonlinear LE (<=)	1
Nonlinear EQ (=)	0
Nonlinear GE (>=)	0
Nonlinear Range	0

Performance Information

Execution Mode	Single-Machine
Number of Threads	4

```
                                The SAS System       10:39 Sunday, October 18, 2015   2

                              The OPTMODEL Procedure

                                 Solution Summary

                          Solver                            NLP
                          Algorithm              Interior Point
                          Objective Function     ExpectedReturn
                          Solution Status                Optimal
                          Objective Value          8.9805347586

                          Optimality Error                 5E-7
                          Infeasibility             6.8167706E-8

                          Iterations                          6
                          Presolve Time                    0.00
                          Solution Time                    0.00

                                [1]        Prop    return

                                AIG    0.030272      1.31
                                AXP    0.040532      7.32
                                BA     0.052203     11.15
                                BAC    0.129927      4.58
                                CAT    0.018218     16.21
                                CVX    0.301072     11.92
                                MMM    0.274334      9.85
                                T      0.153439      5.74

                                 Variance.    Variance.
                                        UB         DUAL

                                       200      0.10913
```

The optimal portfolio contains only eight stocks, and the expected annual return is 8.98. Notice that the NLP solver takes advantage of multiple processors for evaluation.

Information that is put in the log indicates that the problem required six iterations (away from the initial values for the portfolio proportions) to converge to an optimum, and that the Interior Point algorithm is used.

```
Using analytic derivatives for objective.
Using analytic derivatives for nonlinear constraints.
Using 4 threads for nonlinear evaluation.
The NLP solver is called.
The Interior Point algorithm is used.
Initial point was changed to be interior to bounds.
                        Objective                          Optimality
           Iter             Value     Infeasibility             Error
              0        0.86350000        0.90000000        0.98630409
              1       10.36744854        9.15489293        9.15489293
              2        9.79983415       12.74359642       12.74359642
              3        9.15520164        0.01377535        0.01377535
              4        8.98727249        0.04483575        0.04483575
              5        8.98053457   0.0000000735412        0.00002691
              6        8.98053476   0.0000000681677   0.0000005000000
Optimal.
Objective = 8.9805347586.
```

End of Demonstration

3.03 Quiz

Predict (approximately) the expected return if the bound on the variance is decreased (by 10) to 190.

```
Objective Function    ExpectedReturn
Solution Status              Optimal
Objective Value           8.9805347586

       Variance.    Variance.
              UB         DUAL

             200        0.10913
```

43

Solution Status on Termination

The solution status is one of the following:

- Optimal
- Conditional Optimal
- Best Feasible
- Infeasible
- Unbounded
- Infeasible or Unbounded
- Bad Problem Type
- Iteration Limit Reached
- Time Limit Reached
- Failed

45

Conditional Optimal means that the optimality of the solution cannot be proven, usually because one of the tolerances is not met.

Best Feasible means that the solver converged to a local optimum. However, a feasible solution (that does not satisfy the optimality conditions) with a better objective value is found and hence is returned. In this case, missing values are reported in place of the Lagrange multipliers.

Bad Problem Type means that the selected solver cannot be used to solve the problem type.

Failed means that the solver failed to converge, possibly due to numerical issues.

PROC OPTMODEL updates a macro variable _OROPTMODEL_ each time a problem is solved. The macro variable includes values from the solution summary table. It can be used in the conditions of control flow statements using the SAS function SYMGET. The value can be listed using the %PUT statement.

```
%put &_OROPTMODEL_;
```

PROC OPTMODEL Output (sample)

```
STATUS=OK SOLUTION_STATUS=OPTIMAL OBJECTIVE=9 ITERATIONS=1 PRESOLVE_TIME=0 SOLUTION_TIME=0
```

PROC OPTMODEL also updates a pre-declared string-valued parameter _SOLUTION_STATUS_ whose value is the SOLUTION_STATUS= component of the _OROPTMODEL_. The value can be listed using the PRINT statement.

PROC OPTMODEL Output (sample)

```
                                    _SOLUTION_
                                    STATUS_

                                    OPTIMAL
```

Nonlinear Optimization Methods in the OPTMODEL Procedure

NLP Algorithms (Starting in SAS/OR 9.3)

At each iteration, a system of nonlinear equations is solved to determine the search direction. These primal-dual algorithms iteratively generate better approximations of both the decision variables and the Lagrange multipliers. Two algorithms are available.

- INTERIORPOINT: trust region interior point method
- ACTIVESET: active set trust region method

The INTERIORPOINT algorithm uses barrier functions to ensure that iterates satisfy the bound constraints.

The ACTIVESET algorithm explicitly enforces the bound constraints and attempts to identify those inequalities that are satisfied with equality by the optimal solution.

48

Convergence of both algorithms is achieved by using a trust region line-search framework that guides the iterates toward the optimal solution.

NLP Solver Options

Solver Option	Description
ALGORITHM=	Specifies the algorithm to be used (default =INTERIORPOINT)
LOGFREQ=	Frequency of printing to the iteration log (default =1)
SOLTYPE=	Specifies whether the optimal solution returned should be locally optimal (=0) or can be the best feasible (default =1)
HESSTYPE=	Hessian-vector products (=PRODUCT) or the full Hessian (default =FULL)

49

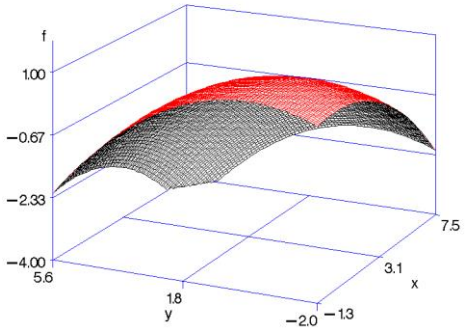
When the solver uses only Hessian-vector products to find a search direction, it usually uses less memory, especially when the problem is large and the Hessian is not sparse. When the full Hessian is used, the problem can create a better preconditioner to solve the problem in less CPU time.

3.3 Solutions

Solutions to Student Activities (Polls/Quizzes)

3.01 Quiz – Correct Answer

Is the objective function $f(x,y) = 1 - 0.05x^2 - 0.1y^2$ of the two-dimensional inequality constrained problem concave?

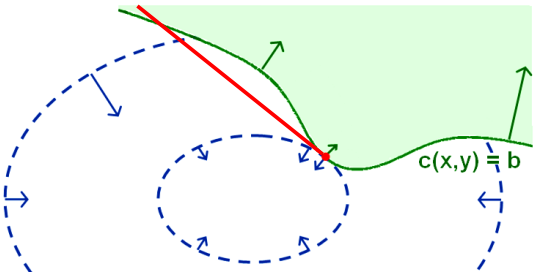


Yes, the Hessian = diag(–0.1,–0.2) is negative definite.

25

3.02 Quiz – Correct Answer

Is the feasible region of the two-dimensional inequality constrained problem convex?



The feasible region is *not* convex. (See the line above.)

33

This does not mean that the nonlinear programming problem has alternate local maxima.

3.03 Quiz – Correct Answer

Predict (approximately) the expected return if the bound on the variance is decreased (by 10) to 190.

Objective Function	ExpectedReturn
Solution Status	Optimal
Objective Value	8.9805347586

Variance. UB	Variance. DUAL
200	0.10913

The predicted return is 8.98 – 10 x 0.10903 = 7.9.

44

Appendix A References

A.1 References

Chapter 2

The furniture-making problem is from Chvátal, V. 1980. *Linear Programming*. New York: W.H. Freeman and Company.

Other references:

Banker, R. D., A. Charnes, and W. W. Cooper. 1984. "Some Models for Estimating Technical and Scale Inefficiencies in Data Envelopment Analysis." *Management Science* 30(9): 1078-1092.

Charnes, A., W. W. Cooper, and E. Rhodes. 1978. "Measuring the efficiency of decision making units." *European Journal of Operational Research*. 6: 429-444.

Seiford, L. M. and J. Zhu. 1999. "Profitability and Marketability of the Top 55 US Commercial Banks." *Management Science* 45(9): 1270-1288.

Zhu, Joe. 2002. Quantitative Models for Performance Evaluation and Benchmarking. New York: Springer.

Chapter 3

The covariance matrix for the portfolio optimization problem is generated using the shrinkage estimator from Ledoit, O. and M. Wolf. 2004. "Honey, I Shrunk the Covariance Matrix." *Journal of Portfolio Management*. 30(4):110-119.