

# **SAS/OR<sup>®</sup> 13.2 User's Guide: Mathematical Programming Legacy Procedures The LP Procedure**



This document is an individual chapter from *SAS/OR® 13.2 User's Guide: Mathematical Programming Legacy Procedures*.

The correct bibliographic citation for the complete manual is as follows: SAS Institute Inc. 2014. *SAS/OR® 13.2 User's Guide: Mathematical Programming Legacy Procedures*. Cary, NC: SAS Institute Inc.

Copyright © 2014, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

**For a hard-copy book:** No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

**For a Web download or e-book:** Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

**U.S. Government License Rights; Restricted Rights:** The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a) and DFAR 227.7202-4 and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

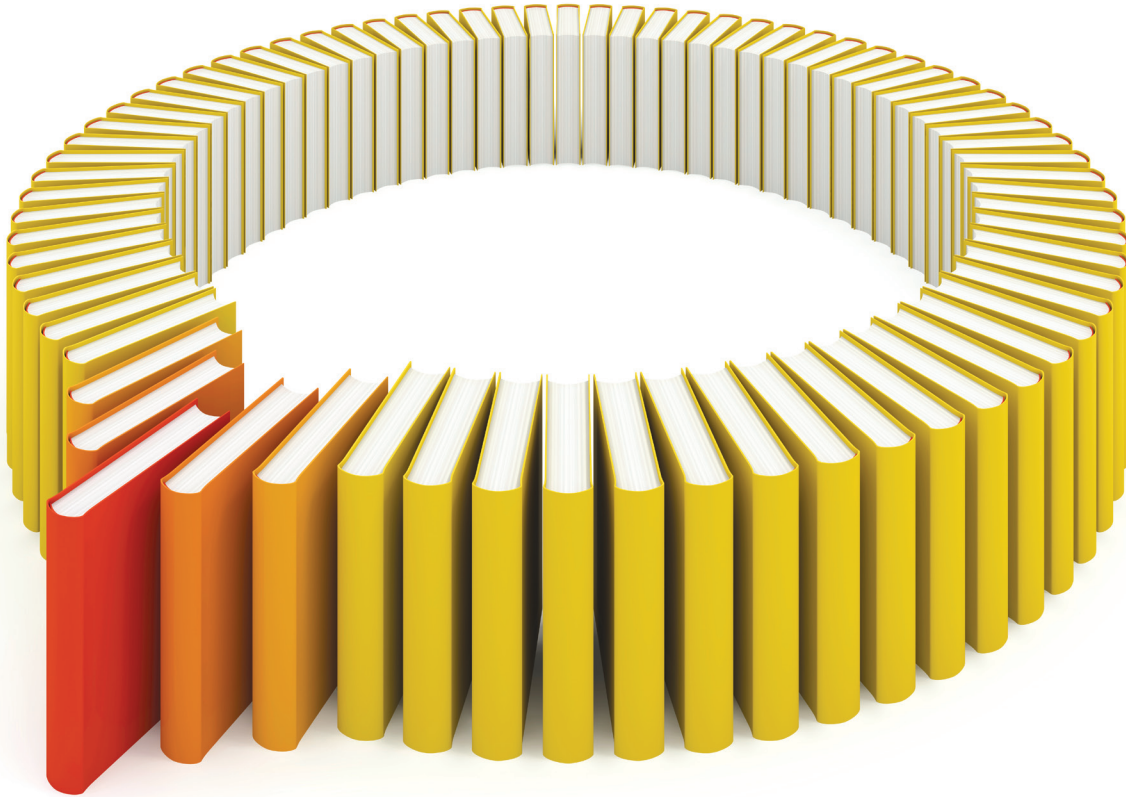
SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

August 2014

SAS provides a complete selection of books and electronic products to help customers use SAS® software to its fullest potential. For more information about our offerings, visit [support.sas.com/bookstore](http://support.sas.com/bookstore) or call 1-800-727-3228.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.



# Gain Greater Insight into Your SAS<sup>®</sup> Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.



[support.sas.com/bookstore](https://support.sas.com/bookstore)  
for additional books and resources.







# Chapter 5

## The LP Procedure

### Contents

---

|                                                                       |            |
|-----------------------------------------------------------------------|------------|
| Overview: LP Procedure . . . . .                                      | <b>168</b> |
| Getting Started: LP Procedure . . . . .                               | <b>170</b> |
| An Introductory Example . . . . .                                     | 171        |
| An Integer Programming Example . . . . .                              | 176        |
| An MPS Format to Sparse Format Conversion Example . . . . .           | 177        |
| Syntax: LP Procedure . . . . .                                        | <b>178</b> |
| Functional Summary . . . . .                                          | 179        |
| PROC LP Statement . . . . .                                           | 183        |
| COEF Statement . . . . .                                              | 192        |
| COL Statement . . . . .                                               | 192        |
| ID Statement . . . . .                                                | 193        |
| IPIVOT Statement . . . . .                                            | 193        |
| PIVOT Statement . . . . .                                             | 193        |
| PRINT Statement . . . . .                                             | 193        |
| QUIT Statement . . . . .                                              | 195        |
| RANGE Statement . . . . .                                             | 195        |
| RESET Statement . . . . .                                             | 196        |
| RHS Statement . . . . .                                               | 197        |
| RHSEN Statement . . . . .                                             | 197        |
| ROW Statement . . . . .                                               | 197        |
| RUN Statement . . . . .                                               | 198        |
| SHOW Statement . . . . .                                              | 198        |
| TYPE Statement . . . . .                                              | 198        |
| VAR Statement . . . . .                                               | 200        |
| Details: LP Procedure . . . . .                                       | <b>201</b> |
| Missing Values . . . . .                                              | 201        |
| Dense Data Input Format . . . . .                                     | 201        |
| Sparse Data Input Format . . . . .                                    | 201        |
| Converting Any PROC LP Format to an MPS-Format SAS Data Set . . . . . | 203        |
| Converting Standard MPS Format to Sparse Format . . . . .             | 204        |
| The Reduced Costs, Dual Activities, and Current Tableau . . . . .     | 206        |
| Macro Variable _ORLP_ . . . . .                                       | 207        |
| Pricing . . . . .                                                     | 208        |
| Scaling . . . . .                                                     | 209        |
| Preprocessing . . . . .                                               | 209        |
| Integer Programming . . . . .                                         | 210        |

|                                                                                   |            |
|-----------------------------------------------------------------------------------|------------|
| Sensitivity Analysis . . . . .                                                    | 217        |
| Range Analysis . . . . .                                                          | 219        |
| Parametric Programming . . . . .                                                  | 220        |
| Interactive Facilities . . . . .                                                  | 221        |
| Memory Management . . . . .                                                       | 223        |
| Output Data Sets . . . . .                                                        | 223        |
| Input Data Sets . . . . .                                                         | 226        |
| Displayed Output . . . . .                                                        | 226        |
| ODS Table and Variable Names . . . . .                                            | 230        |
| Memory Limit . . . . .                                                            | 231        |
| Examples: LP Procedure . . . . .                                                  | <b>232</b> |
| Example 5.1: An Oil Blending Problem . . . . .                                    | 233        |
| Example 5.2: A Sparse View of the Oil Blending Problem . . . . .                  | 238        |
| Example 5.3: Sensitivity Analysis: Changes in Objective Coefficients . . . . .    | 241        |
| Example 5.4: Additional Sensitivity Analysis . . . . .                            | 244        |
| Example 5.5: Price Parametric Programming for the Oil Blending Problem . . . . .  | 247        |
| Example 5.6: Special Ordered Sets and the Oil Blending Problem . . . . .          | 251        |
| Example 5.7: Goal-Programming a Product Mix Problem . . . . .                     | 254        |
| Example 5.8: A Simple Integer Program . . . . .                                   | 260        |
| Example 5.9: An Infeasible Problem . . . . .                                      | 263        |
| Example 5.10: Restarting an Integer Program . . . . .                             | 266        |
| Example 5.11: Alternative Search of the Branch-and-Bound Tree . . . . .           | 272        |
| Example 5.12: An Assignment Problem . . . . .                                     | 277        |
| Example 5.13: A Scheduling Problem . . . . .                                      | 282        |
| Example 5.14: A Multicommodity Transshipment Problem with Fixed Charges . . . . . | 289        |
| Example 5.15: Converting to an MPS-Format SAS Data Set . . . . .                  | 292        |
| Example 5.16: Migration to OPTMODEL: Assignment . . . . .                         | 294        |
| Example 5.17: Migration to OPTMODEL: Multicommodity Transshipment . . . . .       | 296        |
| References . . . . .                                                              | <b>299</b> |

---

## Overview: LP Procedure

The LP procedure solves linear programs, integer programs, and mixed-integer programs. It also performs parametric programming, range analysis, and reports on solution sensitivity to changes in the right-hand-side constants and price coefficients.

The LP procedure provides various control options and solution strategies. It also provides the functionality to produce various kinds of intermediate and final solution information. The procedure's interactive features enable you to take control of the problem solving process. During linear or integer iterations, for example, you can stop the procedure at intermediate stages and examine current results. If necessary, you can change options or strategies and resume the execution of the procedure.

The LP procedure is used to optimize a linear function subject to linear and integer constraints. Specifically, the LP procedure solves the general mixed-integer program of the form

$$\begin{array}{ll}
\text{minimize} & c^T x \\
\text{subject to} & Ax \{ \geq, =, \leq \} b \\
& \ell \leq x \leq u \\
& x_i \text{ is integer, } i \in \mathcal{S}
\end{array}$$

where

- $A$  is an  $m \times n$  matrix of technological coefficients
- $b$  is an  $m \times 1$  matrix of right-hand-side (RHS) constants
- $c$  is an  $n \times 1$  matrix of objective function coefficients
- $x$  is an  $n \times 1$  matrix of structural variables
- $\ell$  is an  $n \times 1$  matrix of lower bounds on  $x$
- $u$  is an  $n \times 1$  matrix of upper bounds on  $x$
- $\mathcal{S}$  is a subset of the set of indices  $\{1, \dots, n\}$

Linear programs (when  $\mathcal{S}$  is empty) are denoted by (LP). For these problems, the procedure employs the two-phase revised simplex method, which uses the Bartels-Golub update of the LU decomposed basis matrix to pivot between feasible solutions (Bartels 1971). In phase 1, PROC LP finds a basic feasible solution to (LP), while in phase 2, PROC LP finds an optimal solution,  $x^{opt}$ . The procedure implicitly handles unrestricted variables, lower-bounded variables, upper-bounded variables, and ranges on constraints. When no explicit lower bounds are specified, PROC LP assumes that all variables are bounded below by zero.

When a variable is specified as an integer variable,  $\mathcal{S}$  has at least one element. The procedure then uses the branch-and-bound technique for optimization.

The relaxed problem (the problem with no integer constraints) is solved initially using the primal algorithm described previously. Constraints are added in defining the subsequent descendant problems in the branch-and-bound tree. These problems are then solved using the dual simplex algorithm. Dual pivots are referred to as phase 3 pivots.

The preprocessing option enables the procedure to identify redundant and infeasible constraints, fix variables, and reduce the feasible region before solving a problem. For linear programs, the option often can reduce the number of constraints and variables, leading to a quicker elapsed solution time and improved reliability. For integer programs, it often reduces the gap between an integer program and its relaxed linear program, which will likely lead to a reduced branch-and-bound tree and a quicker CPU time. In general, it provides users an alternative to solving large, complicated operations research problems.

The LP procedure can also analyze the sensitivity of the solution  $x^{opt}$  to changes in both the objective function and the right-hand-side constants. There are three techniques available for this analysis: sensitivity analysis, parametric programming, and range analysis. Sensitivity analysis enables you to examine the size of a perturbation to the right-hand-side or objective vector by an arbitrary change vector for which the basis of the current optimal solution remains optimal.

Parametric programming, on the other hand, enables you to specify the size of the perturbation beforehand and examine how the optimal solution changes as the desired perturbation is realized. With this technique,

the procedure pivots to maintain optimality as the right-hand-side or objective vector is perturbed beyond the range for which the current solution is optimal. Range analysis is used to examine the range of each right-hand-side value or objective coefficient for which the basis of the current optimal solution remains optimal.

The LP procedure can also save both primal and dual solutions, the current tableau, and the branch-and-bound tree in SAS data sets. This enables you to generate solution reports and perform additional analyses with the SAS System. Although PROC LP reports solutions, this feature is particularly useful for reporting solutions in formats tailored to your specific needs. Saving computational results in a data set also enables you to continue executing a problem not solved because of insufficient time or other computational problems.

The LP procedure uses the Output Delivery System (ODS), a SAS subsystem that provides capabilities for displaying and controlling the output from SAS procedures. ODS enables you to modify the headers, column names, data formats, and layouts of the output tables in PROC LP.

There are no restrictions on the problem size in the LP procedure. The number of constraints and variables in a problem that PROC LP can solve depends on the host platform, the available memory, and the available disk space for utility data sets.

You can also solve LP problems by using the OPTLP procedure. The OPTLP procedure requires a linear program to be specified using a SAS data set that adheres to the MPS format, a widely accepted format in the optimization community. You can use the `MPSOUT=` option in the LP procedure to convert typical PROC LP format data sets into MPS-format SAS data sets.

---

## Getting Started: LP Procedure

PROC LP expects the definition of one or more linear, integer, or mixed-integer programs in an input data set. There are two formats, a dense format and a sparse format, for this data set.

In the dense format, a model is expressed in a similar way as it is formulated. Each SAS variable corresponds to a model's column, and each SAS observation corresponds to a model's row. A SAS variable in the input data set is one of the following:

- a `type` variable
- an `id` variable
- a `structural` variable
- a `right-hand-side` variable
- a `right-hand-side sensitivity analysis` variable or
- a `range` variable

The type variable tells PROC LP how to interpret the observation as a part of the mathematical programming problem. It identifies and classifies objectives, constraints, and the rows that contain information of variables like types, bounds, and so on. PROC LP recognizes the following keywords as values for the type variable: MIN, MAX, EQ, LE, GE, SOSEQ, SOSLE, UNRSTRT, LOWERBD, UPPERBD, FIXED, INTEGER, BINARY, BASIC, PRICESEN, and FREE. The values of the id variable are the names of the rows in the model. The other variables identify and classify the columns with numerical values.

The sparse format to PROC LP is designed to enable you to specify only the nonzero coefficients in the description of linear programs, integer programs, and mixed-integer programs. The SAS data set that describes the sparse model must contain at least four SAS variables:

- a **type** variable
- a **column** variable
- a **row** variable and
- a **coefficient** variable

Each observation in the data set associates a type with a row or a column, or defines a coefficient or a numerical value in the model, or both. In addition to the keywords in the dense format, PROC LP also recognizes the keywords RHS, RHSEN, and RANGE as values of the type variable. The values of the row and column variables are the names of the rows and columns in the model. The values of the coefficient variables give the coefficients or other numerical data. The SAS data set can contain multiple pairs of row and coefficient variables. In this way, more information about the model can be specified in each observation in the data set. See the section “[Sparse Data Input Format](#)” on page 201 for further discussion.

With both the dense and sparse formats for model specification, the observation order is not important. This feature is particularly useful when using the sparse model input.

---

## An Introductory Example

A simple blending problem illustrates the dense and sparse input formats and the use of PROC LP. A step in refining crude oil into finished oil products involves a distillation process that splits crude into various streams. Suppose there are three types of crude available: Arabian light, Arabian heavy, and Brega. These types of crude are distilled into light naphtha, intermediate naphtha, and heating oil. These in turn are blended into jet fuel using one of two recipes. What amounts of the three crudes maximize the profit from producing jet fuel? A formulation to answer this question is as follows:

$$\text{maximize} \quad -175 a_{\text{light}} - 165 a_{\text{heavy}} - 205 b_{\text{rega}} + 300 \text{jet}_1 + 300 \text{jet}_2$$

```

subject to .035 a_light + .03 a_heavy + .045 brega = naphthal
           .1 a_light + .075 a_heavy + .135 brega = naphthai
           .39 a_light + .3 a_heavy + .43 brega = heatingo
           .3 naphthai + .7 heatingo = jet_1
           .2 naphthal + .8 heatingo = jet_2
           a_light ≤ 110
           a_heavy ≤ 165
           brega ≤ 80
           a_light, a_heavy, brega, naphthai,
           naphthal, heatingo, jet_1, jet_2 ≥ 0

```

The following data set gives the representation of this formulation. Notice that the variable names are the structural variables, the rows are the constraints, and the coefficients are given as the values for the structural variables.

```

data;
  input _id_ $17.
        a_light a_heavy brega naphthal naphthai
        heatingo jet_1 jet_2
        _type_ $ _rhs_;
  datalines;
profit          -175 -165 -205  0  0  0 300 300 max      .
naphtha_l_conv  .035 .030 .045 -1  0  0  0  0 eq       0
naphtha_i_conv  .100 .075 .135  0 -1  0  0  0 eq       0
heating_o_conv  .390 .300 .430  0  0 -1  0  0 eq       0
recipe_1        0    0    0  0 .3 .7 -1  0 eq       0
recipe_2        0    0    0 .2  0 .8  0 -1 eq       0
available       110  165   80  .  .  .  .  . upperbd .
;

```

The same model can be specified in the sparse format, as follows. This format enables you to omit the zero coefficients.

```

data;
  format _type_ $8. _col_ $8. _row_ $16. ;
  input _type_ $ _col_ $ _row_ $ _coef_ ;
  datalines;
max      .          profit      .
eq       .          napha_l_conv .
eq       .          napha_i_conv .
eq       .          heating_oil_conv .
eq       .          recipe_1     .
eq       .          recipe_2     .
upperbd  .          available    .
.        a_light    profit      -175
.        a_light    napha_l_conv .035
.        a_light    napha_i_conv .100

```

```

.      a_light      heating_oil_conv      .390
.      a_light      available            110
.      a_heavy      profit                -165
.      a_heavy      napha_l_conv          .030
.      a_heavy      napha_i_conv          .075
.      a_heavy      heating_oil_conv      .300
.      a_heavy      available            165
.      brega        profit                -205
.      brega        napha_l_conv          .045
.      brega        napha_i_conv          .135
.      brega        heating_oil_conv      .430
.      brega        available            80
.      naphthal     napha_l_conv          -1
.      naphthal     recipe_2              .2
.      naphthai     napha_i_conv          -1
.      naphthai     recipe_1              .3
.      heatingo     heating_oil_conv      -1
.      heatingo     recipe_1              .7
.      heatingo     recipe_2              .8
.      jet_1        profit                300
.      jet_1        recipe_1              -1
.      jet_2        profit                300
.      jet_2        recipe_2              -1
.      _rhs_        recipe_1              0
;

```

Because the input order of the model into PROC LP is unimportant, this model can be specified in sparse input in arbitrary row order. [Example 5.2](#) in the section “[Examples: LP Procedure](#)” on page 232 demonstrates this.

The dense and sparse forms of model input give you flexibility to generate models using the SAS language. The dense form of the model is solved with the statements

```

proc lp;
run;

```

The sparse form is solved with the statements

```

proc lp sparsedata;
run;

```

[Example 5.1](#) and [Example 5.2](#) in the section “[Examples: LP Procedure](#)” on page 232 continue with this problem.

## Problem Input

As default, PROC LP uses the most recently created SAS data set as the problem input data set. However, if you want to input the problem from a specific SAS data set, use the [DATA=](#) option. For example, if the previous dense form data set has the name DENSE, the PROC LP statements can be written as

```

proc lp data=dense;
run;

```

## Problem Definition Statements

In the previous dense form data set, the `_ID_`, `_TYPE_`, and `_RHS_` variables are special variables in PROC LP. They stand for id variable, type variable, and right-hand-side variable. If you replace those variable names with, for example, `ROWNAME`, `TYPE`, and `RHS`, you need the problem definition statements (`ID`, `TYPE` and `RHS`) in PROC LP:

```
proc lp;
  id rowname;
  type type;
  rhs rhs;
run;
```

Other special variables for the dense format are `_RHSEN_` and `_RANGE_`, which identify the vectors for the right-hand-side sensitivity and range analyses. The corresponding statements are the `RHSEN` and `RANGE` statements. (Notice that a variable name can be identical to a statement name.)

In the same way, if you replace the variables `_COL_`, `_ROW_`, `_TYPE_`, and `_COEF_` in the previous sparse form data set by `COLUMN`, `ROW`, `TYPE`, and `COEF`, you need the problem definition statements (`COL`, `ROW`, `TYPE`, and `COEF`) in PROC LP.

```
proc lp sparsedata;
  col column;
  row row;
  type type;
  coef coef;
run;
```

In the sparse form data set, the value `'_RHS_'` under the variable `_COL_` is a special column name, which represents the model's right-hand-side column. If you replace it by a value `'R'`, the PROC LP statements would be

```
proc lp sparsedata;
  rhs r;
run;
```

Other special column names for the sparse format are `'_RHSEN_'` and `'_RANGE_'`. The corresponding statements are the `RHSEN` and `RANGE` statements.

PROC LP is case insensitive to variable names and all character values, including the row and column names in the sparse format. The order of the problem definition statements is not important.

For the dense format, a model's row names appear as character values in a SAS data set. For the sparse format, both the row and the column names of the model appear as character values in the data set. Thus, you can put spaces or other special characters in the names. When referring to these names in the problem definition statement or other LP statements, you must use single or double quotes around them. For example, if you replace `'_RHS_'` by `'R H S'` in the previous sparse form data set, the PROC LP statements would become



```
proc lp sparsedata;
    rhs "r h s";
run;
```

## LP Options

The specifications **SPARSEDATA** and **DATA=** in the previous examples are PROC LP options. PROC LP options include

- data set options
- display control options
- interactive control options
- preprocessing control options
- branch-and-bound control options
- sensitivity/parametric/ranging control options
- simplex algorithm control options

## Interactive Processing

Interactive control options include **READPAUSE**, **ENDPAUSE**, and so forth. You can run PROC LP interactively using those options. For example, for the blending problem example in the dense form, you can first pause the procedure before iterations start with the **READPAUSE** option. The PROC LP statements are

```
proc lp readpause;
run;
```

When the procedure pauses, you run the **PRINT** statement to display the initial technological matrix and see if the input is correct. Then you run the **PIVOT** statement to do one simplex pivot and pause. After that you use the **SHOW** statement to check the current solution status. Then you apply the **RESET** statement to tell the procedure to stop as soon as it finds a solution. Now you use the **RUN** statement to continue the execution. When the procedure stops, you run the **PRINT** statement again to do a price range analysis and **QUIT** the procedure. Use a SAS **%PUT** statement to display the contents of PROC LP's macro variable, **\_ORLP\_**, which contains iterations and solution information. What follows are the complete statements in batch mode:

```
proc lp readpause;
run;
print matrix(,); /* display all rows and columns. */
pivot;
show status;
reset endpause;
run;
print rangeprice;
quit;
%put &_orlp_;
```

**NOTE:** You can force PROC LP to pause during iterations by using the CTRL-BREAK key.

## An Integer Programming Example

The following is a simple mixed-integer programming problem. Details can be found in [Example 5.8](#) in the section “[Examples: LP Procedure](#)” on page 232.

```
data;
  format _row_ $10.;
  input _row_ $ choco gumdr ichoco igumdr _type_ $ _rhs_;
  datalines;
object      .25      .75      -100      -75 max      .
cooking      15      40      0      0 1e      27000
color      0 56.25      0      0 1e      27000
package     18.75      0      0      0 1e      27000
condiments   12      50      0      0 1e      27000
chocolate    1      0 -10000      0 1e      0
gum          0      1      0 -10000 1e      0
only_one     0      0      1      1 eq      1
binary      .      .      1      2 binary .
;
```

The row with ‘binary’ type indicates that this problem is a mixed-integer program and all the integer variables are binary. The integer values of the row set an ordering for PROC LP to pick the branching variable when `VARSELECT=PRIOR` is chosen. Smaller values will have higher priorities. The `_ROW_` variable here is an alias of the `_ID_` variable.

This problem can be solved with the following statements:

```
proc lp canselect=lifo backtrack=obj varselect=far endpause;
run;
quit;
%put &_orlp_;
```

The options `CANSELECT=`, `BACKTRACK=`, and `VARSELECT=` specify the rules for picking the next active problem and the rule to choose the branching variable. In this example, the values LIFO, OBJ and FAR serve as the default values, so the three options can be omitted from the PROC LP statement. The following is the output from the %PUT statement:

```
STATUS=SUCCESSFUL PHASE=3 OBJECTIVE=285 P_FEAS=YES D_FEAS=YES INT_ITER=3
INT_FEAS=2 ACTIVE=0 INT_BEST=285 PHASE1_ITER=0 PHASE2_ITER=5
PHASE3_ITER=5
```

## Preprocessing

Using the `PREPROCESS=` option, you can apply the preprocessing techniques to pre-solve and then solve the preceding mixed-integer program:

```
proc lp preprocess=1 endpause;
run;
quit;
%put &_orlp_;
```

The preprocessing statistics are written to the SAS log file as follows:

```
NOTE: Preprocessing 1 ...
NOTE:      2 upper bounds decreased.
NOTE:      2 coefficients reduced.
NOTE: Preprocessing 2 ...
NOTE:      2 constraints eliminated.
NOTE: Preprocessing done.
```

The new output `_ORLP_` is as follows:

```
STATUS=SUCCESSFUL PHASE=3 OBJECTIVE=285 P_FEAS=YES D_FEAS=YES INT_ITER=0
      INT_FEAS=1 ACTIVE=0 INT_BEST=285 PHASE1_ITER=0 PHASE2_ITER=5
      PHASE3_ITER=0
```

In this example, the number of integer iterations (INT\_ITER=) is zero, which means that the preprocessing has reduced the gap between the relaxed linear problem and the mixed-integer program to zero.

---

## An MPS Format to Sparse Format Conversion Example

If your model input is in [MPS input format](#), you can convert it to the sparse input format of PROC LP using the SAS macro function SASMPXS. For example, if you have an MPS file called MODEL.MPS and it is stored in the directory C:\OR on a PC, the following program can help you to convert the file and solve the problem.

```
%sasmpxs(mpsfile="c:\or\model.mps",lpdata=lp);

data;
  set lp;
  retain i 1;
  if _type_="FREE" and i=1 then
    do;
      _type_="MIN";
      i=0;
    end;
run;

proc lp sparsedata;
run;
```

In the MPS input format, all objective functions, price change rows, and free rows have the type ‘N’. The SASMPXS macro marks them as ‘FREE’ rows. After the conversion, you must run a DATA step to identify the objective rows and price change rows. In this example, assume that the problem is one of minimization and the first ‘FREE’ row is an objective row.

---

## Syntax: LP Procedure

Below are statements used in PROC LP, listed in alphabetical order as they appear in the text that follows.

```

PROC LP options ;
  COEF variables ;
  COL variable ;
  ID variable(s) ;
  IPIVOT ; ;
  PIVOT ; ;
  PRINT options ;
  QUIT options ;
  RANGE variable ;
  RESET options ;
  RHS variables ;
  RHSEN variables ;
  ROW variable(s) ;
  RUN ; ;
  SHOW options ;
  TYPE variable ;
  VAR variables ;

```

The TYPE, ID (or ROW), VAR, RHS, RHSEN, and RANGE statements are used for identifying variables in the problem data set when the model is in the dense input format. In the dense input format, a model's variables appear as variables in the problem data set. The TYPE, ID (or ROW), and RHS statements can be omitted if the input data set contains variables \_TYPE\_, \_ID\_ (or \_ROW\_), and \_RHS\_; otherwise, they must be used. The VAR statement is optional. When it is omitted, PROC LP treats all numeric variables that are not explicitly or implicitly included in RHS, RHSEN, and RANGE statements as structural variables. The RHSEN and RANGE statements are optional statements for sensitivity and range analyses. They can be omitted if the input data set contains the \_RHSEN\_ and \_RANGE\_ variables.

The TYPE, COL, ROW (or ID), COEF, RHS, RHSEN, and RANGE statements are used for identifying variables in the problem data set when the model is in the sparse input format. In the sparse input format, a model's rows and columns appear as observations in the problem data set. The TYPE, COL, ROW (or ID), and COEF statements can be omitted if the input data set contains the \_TYPE\_ and \_COL\_ variables, as well as variables beginning with the prefixes \_ROW\_ (or \_ID\_) and \_COEF\_. Otherwise, they must be used. The RHS, RHSEN, and RANGE statements identify the corresponding columns in the model. These statements can be omitted if there are observations that contain the RHS, RHSEN, and RANGE types or the \_RHS\_, \_RHSEN\_, and \_RANGE\_ column values.

The SHOW, RESET, PRINT, QUIT, PIVOT, IPIVOT, and RUN statements are especially useful when executing PROC LP interactively. However, they can also be used in batch mode.

## Functional Summary

The statements and options available with PROC LP are summarized by purpose in the following table.

**Table 5.1** Functional Summary

| Description                                                                                                       | Statement | Option      |
|-------------------------------------------------------------------------------------------------------------------|-----------|-------------|
| <b>Interactive Statements:</b>                                                                                    |           |             |
| Perform one integer pivot and pause                                                                               | IPIVOT    |             |
| Perform one simplex pivot and pause                                                                               | PIVOT     |             |
| Display information at current iteration                                                                          | PRINT     |             |
| Terminate processing immediately                                                                                  | QUIT      |             |
| Reset options specified                                                                                           | RESET     |             |
| Start or resume optimization                                                                                      | RUN       |             |
| Show settings of options                                                                                          | SHOW      |             |
| <b>Variable Lists:</b>                                                                                            |           |             |
| Variables that contain coefficients (sparse)                                                                      | COEF      |             |
| Variable that contains column names (sparse)                                                                      | COL       |             |
| Alias for the ROW statement                                                                                       | ID        |             |
| Variable (column) that contains the range constant for the dense (sparse) format                                  | RANGE     |             |
| Variables (columns) that contains RHS constants for the dense (sparse) format                                     | RHS       |             |
| Variables (columns) that define RHS change vectors for the dense (sparse) format                                  | RHSEN     |             |
| Variable that contains names of constraints and objective functions (names of rows) for the dense (sparse) format | ROW       |             |
| Variable that contains the type of each observation                                                               | TYPE      |             |
| Structural variables (dense)                                                                                      | VAR       |             |
| <b>Data Set Options:</b>                                                                                          |           |             |
| Active nodes input data set                                                                                       | PROC LP   | ACTIVEIN=   |
| Active nodes output data set                                                                                      | PROC LP   | ACTIVEOUT=  |
| Input data set                                                                                                    | PROC LP   | DATA=       |
| Dual output data set                                                                                              | PROC LP   | DUALOUT=    |
| Primal input data set                                                                                             | PROC LP   | PRIMALIN=   |
| Primal output data set                                                                                            | PROC LP   | PRIMALOUT=  |
| Sparse format data input flag                                                                                     | PROC LP   | SPARSEDATA  |
| Tableau output data set                                                                                           | PROC LP   | TABLEAUOUT= |
| Convert sparse or dense format input data set into MPS-format output data set                                     | PROC LP   | MPSOUT=     |

| Description                                      | Statement | Option          |
|--------------------------------------------------|-----------|-----------------|
| <b>Display Control Options:</b>                  |           |                 |
| Display iteration log                            | PROC LP   | FLOW            |
| Nonzero tolerance displaying                     | PROC LP   | FUZZ=           |
| Inverse of FLOW option                           | PROC LP   | NOFLOW          |
| Inverse of PARAPRINT option                      | PROC LP   | NOPARAPRINT     |
| Omit some displaying                             | PROC LP   | NOPRINT         |
| Inverse of TABLEAUPRINT                          | PROC LP   | NOTABLEAUPRINT  |
| Parametric programming displaying                | PROC LP   | PARAPRINT       |
| Inverse of NOPRINT                               | PROC LP   | PRINT           |
| Iteration frequency of display                   | PROC LP   | PRINTFREQ=      |
| Level of display desired                         | PROC LP   | PRINTLEVEL=     |
| Display the final tableau                        | PROC LP   | TABLEAUPRINT    |
| <b>Interactive Control Options:</b>              |           |                 |
| Pause before displaying the solution             | PROC LP   | ENDPAUSE        |
| Pause after first feasible solution              | PROC LP   | FEASIBLEPAUSE   |
| Pause frequency of integer solutions             | PROC LP   | IFEASIBLEPAUSE= |
| Pause frequency of integer iterations            | PROC LP   | IPAUSE=         |
| Inverse of ENDPAUSE                              | PROC LP   | NOENDPAUSE      |
| Inverse of FEASIBLEPAUSE                         | PROC LP   | NOFEASIBLEPAUSE |
| Pause frequency of iterations                    | PROC LP   | PAUSE=          |
| Pause if within specified proximity              | PROC LP   | PROXIMITYPAUSE= |
| Pause after data are read                        | PROC LP   | READPAUSE       |
| <b>Preprocessing Control Options:</b>            |           |                 |
| Do not perform preprocessing                     | PROC LP   | NOPREPROCESS    |
| Preprocessing error tolerance                    | PROC LP   | PEPSILON=       |
| Limit preprocessing iterations                   | PROC LP   | PMAXIT=         |
| Perform preprocessing techniques                 | PROC LP   | PREPROCESS      |
| <b>Branch-and-Bound (BB) Control Options:</b>    |           |                 |
| Perform automatic node selection technique       | PROC LP   | AUTO            |
| Backtrack strategy to be used                    | PROC LP   | BACKTRACK=      |
| Branch on binary variables first                 | PROC LP   | BINFST          |
| Active node selection strategy                   | PROC LP   | CANSELECT=      |
| Comprehensive node selection control parameter   | PROC LP   | CONTROL=        |
| Backtrack related technique                      | PROC LP   | DELTAIT=        |
| Measure for pruning BB tree                      | PROC LP   | DOBJECTIVE=     |
| Integer tolerance                                | PROC LP   | IEPSILON=       |
| Limit integer iterations                         | PROC LP   | IMAXIT=         |
| Measure for pruning BB tree                      | PROC LP   | IOBJECTIVE=     |
| Order of two branched nodes in adding to BB tree | PROC LP   | LIFOTYPE=       |
| Inverse of AUTO                                  | PROC LP   | NOAUTO          |
| Inverse of BINFST                                | PROC LP   | NOBINFST        |
| Inverse of POSTPROCESS                           | PROC LP   | NOPOSTPROCESS   |

| Description                                                                             | Statement | Option           |
|-----------------------------------------------------------------------------------------|-----------|------------------|
| Limit number of branching variables                                                     | PROC LP   | PENALTYDEPTH=    |
| Measure for pruning BB tree                                                             | PROC LP   | POBJECTIVE=      |
| Perform variables fixing technique                                                      | PROC LP   | POSTPROCESS      |
| Percentage used in updating WOBJECTIVE                                                  | PROC LP   | PWOBJECTIVE=     |
| Compression algorithm for storing active nodes                                          | PROC LP   | TREETYPE=        |
| Branching variable selection strategy                                                   | PROC LP   | VARSELECT=       |
| Delay examination of some active nodes                                                  | PROC LP   | WOBJECTIVE=      |
| <b>Sensitivity/Parametric/Ranging Control Options:</b>                                  |           |                  |
| Inverse of RANGEPRICE                                                                   | PROC LP   | NORANGEPRICE     |
| Inverse of RANGERHS                                                                     | PROC LP   | NORANGERHS       |
| Limit perturbation of the price vector                                                  | PROC LP   | PRICEPHI=        |
| Range analysis on the price coefficients                                                | PROC LP   | RANGEPRICE       |
| Range analysis on the RHS vector                                                        | PROC LP   | RANGERHS         |
| Limit perturbation of the RHS vector                                                    | PROC LP   | RHSPHI=          |
| <b>Simplex Algorithm Control Options:</b>                                               |           |                  |
| Use devex method                                                                        | PROC LP   | DEVEX            |
| General error tolerance                                                                 | PROC LP   | EPSILON=         |
| Perform goal programming                                                                | PROC LP   | GOALPROGRAM      |
| Largest number used in computation                                                      | PROC LP   | INFINITY=        |
| Reinversion frequency                                                                   | PROC LP   | INVREQ=          |
| Reinversion tolerance                                                                   | PROC LP   | INVTOL=          |
| Simultaneously set MAXIT1, MAXIT2, MAXIT3 and IMAXIT values                             | PROC LP   | MAXIT=           |
| Limit phase 1 iterations                                                                | PROC LP   | MAXIT1=          |
| Limit phase 2 iterations                                                                | PROC LP   | MAXIT2=          |
| Limit phase 3 iterations                                                                | PROC LP   | MAXIT3=          |
| Inverse of devex                                                                        | PROC LP   | NODEVEX          |
| Restore basis after parametric programming                                              | PROC LP   | PARARESTORE      |
| Weight of the phase 2 objective function in phase 1                                     | PROC LP   | PHASEMIX=        |
| Multiple pricing strategy                                                               | PROC LP   | PRICETYPE=       |
| Number of columns to subset in multiple pricing                                         | PROC LP   | PRICE=           |
| Limit the number of iterations randomly selecting each entering variable during phase 1 | PROC LP   | RANDOMPRICEMULT= |
| Zero tolerance in ratio test                                                            | PROC LP   | REPSILON=        |
| Scaling type to be performed                                                            | PROC LP   | SCALE=           |
| Zero tolerance in LU decomposition                                                      | PROC LP   | SMALL=           |
| Time pause limit                                                                        | PROC LP   | TIME=            |
| Control pivoting during LU decomposition                                                | PROC LP   | U=               |

**RESET Statement Options:**

The RESET statement supports the same options as the PROC LP statement except for the DATA=, PRIMALIN=, and ACTIVEIN= options, and supports the following additional options:

| Description                                                                   | Statement | Option               |
|-------------------------------------------------------------------------------|-----------|----------------------|
| New variable lower bound during phase 3                                       | RESET     | LOWER=               |
| New variable upper bound during phase 3                                       | RESET     | UPPER=               |
| <b>PRINT Statement Options:</b>                                               |           |                      |
| Display the best integer solution                                             | PRINT     | BEST                 |
| Display variable summary for specified columns                                | PRINT     | COLUMN               |
| Display variable summary and price sensitivity analysis for specified columns | PRINT     | COLUMN / SENSITIVITY |
| Display variable summary for integer variables                                | PRINT     | INTEGER              |
| Display variable summary for nonzero integer variables                        | PRINT     | INTEGER_NONZEROS     |
| Display variable summary for integer variables with zero activity             | PRINT     | INTEGER_ZEROS        |
| Display submatrix for specified rows and columns                              | PRINT     | MATRIX               |
| Display formatted submatrix for specified rows and columns                    | PRINT     | MATRIX / PICTURE     |
| Display variable summary for continuous variables                             | PRINT     | NONINTEGER           |
| Display variable summary for nonzero continuous variables                     | PRINT     | NONINTEGER_NONZEROS  |
| Display variable summary for variables with nonzero activity                  | PRINT     | NONZEROS             |
| Display price sensitivity analysis or price parametric programming            | PRINT     | PRICESEN             |
| Display price range analysis                                                  | PRINT     | RANGEPRICE           |
| Display RHS range analysis                                                    | PRINT     | RANGERHS             |
| Display RHS sensitivity analysis or RHS parametric programming                | PRINT     | RHSEN                |
| Display constraint summary for specified rows                                 | PRINT     | ROW                  |
| Display constraint summary and RHS sensitivity analysis for specified rows    | PRINT     | ROW / SENSITIVITY    |
| Display solution, variable, and constraint summaries                          | PRINT     | SOLUTION             |
| Display current tableau                                                       | PRINT     | TABLEAU              |
| Display variables with zero activity                                          | PRINT     | ZEROS                |
| <b>SHOW Statement Options:</b>                                                |           |                      |
| Display options applied                                                       | SHOW      | OPTIONS              |
| Display status of the current solution                                        | SHOW      | STATUS               |
| <b>QUIT Statement Option:</b>                                                 |           |                      |
| Save the defined output data sets and then terminate PROC LP                  | QUIT      | / SAVE               |



---

## PROC LP Statement

**PROC LP** *options* ;

This statement invokes the procedure. The following options can appear in the PROC LP statement.

### Data Set Options

**ACTIVEIN=SAS-data-set**

names the SAS data set containing the active nodes in a branch-and-bound tree that is to be used to restart an integer program.

**ACTIVEOUT=SAS-data-set**

names the SAS data set in which to save the current branch-and-bound tree of active nodes.

**DATA=SAS-data-set**

names the SAS data set containing the problem data. If the DATA= option is not specified, PROC LP uses the most recently created SAS data set.

**DUALOUT=SAS-data-set**

names the SAS data set that contains the current dual solution (shadow prices) on termination of PROC LP. This data set contains the current dual solution only if PROC LP terminates successfully.

**MPSOUT=SAS-data-set**

names the SAS data set that contains converted sparse or dense format input data in MPS format. Invoking this option directs the LP procedure to halt before attempting optimization. For more information about the MPS-format SAS data set, see Chapter 17, “The MPS-Format SAS Data Set” (*SAS/OR User’s Guide: Mathematical Programming*).

**PRIMALIN=SAS-data-set**

names the SAS data set that contains a feasible solution to the problem defined by the DATA= data set. The data set specified in the PRIMALIN= option should have the same format as a data set saved using the PRIMALOUT= option. Specifying the PRIMALIN= option is particularly useful for continuing iteration on a problem previously attempted. It is also useful for performing sensitivity analysis on a previously solved problem.

**PRIMALOUT=SAS-data-set**

names the SAS data set that contains the current primal solution when PROC LP terminates.

**SPARSEDATA**

tells PROC LP that the data are in the sparse input format. If this option is not specified, PROC LP assumes that the data are in the dense input format. See the section “[Sparse Data Input Format](#)” on page 201 for information about the sparse input format.

**TABLEAUOUT=SAS-data-set**

names the SAS data set in which to save the final tableau.

## Display Control Options

### FLOW

requests that a journal of pivot information (the Iteration Log) be displayed after every **PRINTFREQ=** iterations. This includes the names of the variables entering and leaving the basis, the reduced cost of the entering variable, and the current objective value.

### FUZZ=*e*

displays all numbers within *e* of zero as zeros. The default value is 1.0E–10.

### NOFLOW

is the inverse of the **FLOW** option.

### NOPARAPRINT

is the inverse of the **PARAPRINT** option.

### NOPRINT

suppresses the display of the Variable, Constraint, and Sensitivity Analysis summaries. This option is equivalent to the **PRINTLEVEL=0** option.

### NOTABLEAUPRINT

is the inverse of the **TABLEAUPRINT** option.

### PARAPRINT

indicates that the solution be displayed at each pivot when performing parametric programming.

### PRINT

is the inverse of the **NOPRINT** option.

### PRINTFREQ=*m*

indicates that after every *m*th iteration, a line in the (Integer) Iteration Log be displayed. The default value is 1.

### PRINTLEVEL=*i*

indicates the amount of displaying that the procedure should perform.

**PRINTLEVEL=-2**      only messages to the SAS log are displayed

**PRINTLEVEL=-1**      is equivalent to **NOPRINT** unless the problem is infeasible. If it is infeasible, the infeasible rows are displayed in the Constraint Summary along with the Infeasible Information Summary.

**PRINTLEVEL=0**      is identical to **NOPRINT**

**PRINTLEVEL=1**      all output is displayed

The default value is 1.

### TABLEAUPRINT

indicates that the final tableau be displayed.

## Interactive Control Options

### ENDPAUSE

requests that PROC LP pause before displaying the solution. When this pause occurs, you can enter the **RESET**, **SHOW**, **PRINT**, **RUN**, and **QUIT** statements.

### FEASIBLEPAUSE

requests that PROC LP pause after a feasible (not necessarily integer feasible) solution has been found. At a pause, you can enter the **RESET**, **SHOW**, **PRINT**, **PIVOT**, **RUN**, and **QUIT** statements.

### IFEASIBLEPAUSE=*n*

requests that PROC LP pause after every *n* integer feasible solutions. At a pause, you can enter the **RESET**, **SHOW**, **PRINT**, **IPIVOT**, **PIVOT**, **RUN**, and **QUIT** statements. The default value is 99999999.

### IPAUSE=*n*

requests that PROC LP pause after every *n* integer iterations. At a pause, you can enter **RESET**, **SHOW**, **PRINT**, **IPIVOT**, **PIVOT**, **RUN**, and **QUIT** statements. The default value is 99999999.

### NOENDPAUSE

is the inverse of the **ENDPAUSE** option.

### NOFEASIBLEPAUSE

is the inverse of the **FEASIBLEPAUSE** option.

### PAUSE=*n*

requests that PROC LP pause after every *n* iterations. At a pause, you can enter the **RESET**, **SHOW**, **PRINT**, **IPIVOT**, **PIVOT**, **RUN**, and **QUIT** statements. The default value is 99999999.

### PROXIMITYPAUSE=*r*

causes the procedure to pause if at least one integer feasible solution has been found and the objective value of the current best integer solution can be determined to be within *r* units of the optimal integer solution. This distance, called proximity, is also displayed on the Integer Iteration Log. Note that the proximity is calculated using the minimum (maximum if the problem is maximization) objective value among the nodes that remain to be explored in the branch-and-bound tree as a bound on the value of the optimal integer solution. Following the first **PROXIMITYPAUSE=** pause, in order to avoid a pause at every iteration thereafter, it is recommended that you reduce this measure through the use of a **RESET** statement. Otherwise, if any other option or statement that causes the procedure to pause is used while the **PROXIMITYPAUSE=** option is in effect, pause interferences may occur. When this pause occurs, you can enter the **RESET**, **SHOW**, **PRINT**, **IPIVOT**, **PIVOT**, **RUN**, and **QUIT** statements. The default value is 0.

### READPAUSE

requests that PROC LP pause after the data have been read and the initial basis inverted. When this pause occurs, you can enter the **RESET**, **SHOW**, **PRINT**, **IPIVOT**, **PIVOT**, **RUN**, and **QUIT** statements.

## Preprocessing Control Options

### NOPREPROCESS

is the inverse of the **PREPROCESS** option.

**PEPSILON=*e***

specifies a positive number close to zero. This value is an error tolerance in the preprocessing. If the value is too small, any marginal changes may cause the preprocessing to repeat itself. However, if the value is too large, it may alter the optimal solution or falsely claim that the problem is infeasible. The default value is 1.0E–8.

**PMAIT=*n***

performs at most *n* preprocessings. Preprocessing repeats itself if it improves some bounds or fixes some variables. However when a problem is large and dense, each preprocessing may take a significant amount of CPU time. This option limits the number of preprocessings PROC LP performs. It can also reduce the build-up of round-off errors. The default value is 100.

**PREPROCESS**

performs preprocessing techniques. See the section “[Preprocessing](#)” on page 209 for further discussion.

## Branch-and-Bound Algorithm Control Options

**AUTO, AUTO(*m,n*)**

automatically sets and adjusts the value of the **CONTROL=** option. Initially, it sets **CONTROL=0.70**, concentrating on finding an integer feasible solution or an upper bound. When an upper bound is found, it sets **CONTROL=0.5**, concentrating on efficiency and lower bound improvement. When the number of active problems exceeds *m*, it starts to gradually increase the value of the **CONTROL=** option to keep the size of active problems under control. When total active problems exceed *n*, **CONTROL=1** will keep the active problems from growing further. You can alter the automatic process by resetting the value of the **CONTROL=** option interactively.

The default values of *m* and *n* are 20000 and 250000, respectively. You can change the two values according to your computer’s space and memory capacities.

**BACKTRACK=*rule***

specifies the rule used to choose the next active problem when backtracking is required. One of the following can be specified:

- BACKTRACK=*LIFO*
- BACKTRACK=*FIFO*
- BACKTRACK=*OBJ*
- BACKTRACK=*PROJECT*
- BACKTRACK=*PSEUDOC*
- BACKTRACK=*ERROR*

The default value is *OBJ*. See the section “[Integer Programming](#)” on page 210 for further discussion.

**BINFST**

requests that PROC LP branch on binary variables first when integer and binary variables are present. The reasoning behind this is that a subproblem will usually be fathomed or found integer feasible after less than 20% of its variables have been fixed. Considering binary variables first attempts to reduce the size of the branch-and-bound tree. It is a heuristic technique.

**CANSELECT=rule**

specifies the rule used to choose the next active problem when backtracking is not required or used. One of the following can be specified:

- CANSELECT=LIFO
- CANSELECT=FIFO
- CANSELECT=OBJ
- CANSELECT=PROJECT
- CANSELECT=PSEUDOC
- CANSELECT=ERROR

The default value is *LIFO*. See the section “[Integer Programming](#)” on page 210 for further discussion.

**CONTROL=r**

specifies a number between 0 and 1. This option combines [CANSELECT=](#) and other rules to choose the next active problem. It takes into consideration three factors: efficiency, improving lower bounds, and improving upper bounds. When  $r$  is close to 0, PROC LP concentrates on improving lower bounds (upper bounds for maximization). However, the efficiency per integer iteration is usually the worst. When  $r$  is close to 1, PROC LP concentrates on improving upper bounds (lower bounds for maximization). In addition, the growth of active problems will be controlled and stopped at  $r = 1$ . When its value is around 0.5, PROC LP will be in the most efficient state in terms of CPU time and integer number of iterations. The **CONTROL=** option will be automatically adjusted when the [AUTO](#) option is applied.

**DELTAIT=r**

is used to modify the exploration of the branch-and-bound tree. If more than  $r$  integer iterations have occurred since the last integer solution was found, then the procedure uses the backtrack strategy in choosing the next node to be explored. The default value is 3 times the number of integer variables.

**DOBJECTIVE=r**

specifies that PROC LP should discard active nodes that cannot lead to an integer solution with the objective at least as small (or as large for maximizations) as the objective of the relaxed problem plus (minus)  $r$ . The default value is  $+\infty$ .

**IEPSILON=e**

requests that PROC LP consider an integer variable as having an integer value if its value is within  $e$  units of an integer. The default value is  $1.0\text{E}-7$ .

**IMAXIT=n**

performs at most  $n$  integer iterations. The default value is 100.

**IOBJECTIVE=r**

specifies that PROC LP should discard active nodes unless the node could lead to an integer solution with the objective smaller (or larger for maximizations) than  $r$ . The default value is  $+\infty$  for minimization ( $-\infty$  for maximization).

**LIFOTYPE=*c***

specifies the order in which to add the two newly branched active nodes to the LIFO list.

|            |                                                                                  |
|------------|----------------------------------------------------------------------------------|
| LIFOTYPE=0 | add the node with minimum penalty first                                          |
| LIFOTYPE=1 | add the node with maximum penalty first                                          |
| LIFOTYPE=2 | add the node resulting from adding $x_i \geq \lceil x^{opt}(k)_i \rceil$ first   |
| LIFOTYPE=3 | add the node resulting from adding $x_i \leq \lfloor x^{opt}(k)_i \rfloor$ first |

The default value is 0.

**NOAUTO**

is the inverse of the [AUTO](#) option.

**NOBINFST**

is the inverse of the [BINFST](#) option.

**NOPOSTPROCESS**

is the inverse of the [POSTPROCESS](#) option.

**PENALTYDEPTH=*m***

requests that PROC LP examine  $m$  variables as branching candidates when [VARSELECT=PENALTY](#). If the [PENALTYDEPTH=](#) option is not specified when [VARSELECT=PENALTY](#), then all of the variables are considered branching candidates. The default value is the number of integer variables. See the section “[Integer Programming](#)” on page 210 for further discussion.

**POBJECTIVE=*r***

specifies that PROC LP should discard active nodes that cannot lead to an integer solution with objective at least as small as  $o + |o| \times r$  (at least as large as  $o - |o| \times r$  for maximizations) where  $o$  is the objective of the relaxed noninteger constrained problem. The default value is  $+\infty$ .

**POSTPROCESS**

attempts to fix binary variables globally based on the relationships among the reduced cost and objective value of the relaxed problem and the objective value of the current best integer feasible solution.

**PWOBJECTIVE=*r***

specifies a percentage for use in the automatic update of the [WOBJECTIVE=](#) option. If the [WOBJECTIVE=](#) option is not specified in PROC LP, then when an integer feasible solution is found, the value of the option is updated to be  $b + q \times r$  where  $b$  is the best bound on the value of the optimal integer solution and  $q$  is the current proximity. Note that for maximizations,  $b - q \times r$  is used. The default value is 0.95.

**TREETYPE=*i***

specifies a data compression algorithm.

|            |                                                 |
|------------|-------------------------------------------------|
| TREETYPE=0 | no data compression                             |
| TREETYPE=1 | Huffman coding compression routines             |
| TREETYPE=2 | adaptive Huffman coding compression routines    |
| TREETYPE=3 | adaptive arithmetic coding compression routines |

For IP or MIP problems, the basis and bounds information of each active node is saved to a utility file. When the number of active nodes increases, the size of the utility file becomes larger and larger. If PROC LP runs into a disk problem, like “disk full ...” or “writing failure ...”, you can use this option to compress the utility file. For more information on the data compression routines, refer to Nelson (1992). The default value is 0.

**VARSELECT=rule**

specifies the rule used to choose the branching variable on an integer iteration.

- VARSELECT=*CLOSE*
- VARSELECT=*PRIOR*
- VARSELECT=*PSEUDOC*
- VARSELECT=*FAR*
- VARSELECT=*PRICE*
- VARSELECT=*PENALTY*

The default value is FAR. See the section “[Integer Programming](#)” on page 210 for further discussion.

**WOBJECTIVE=r**

specifies that PROC LP should delay examination of active nodes that cannot lead to an integer solution with objective at least as small (as large for maximizations) as  $r$ , until all other active nodes have been explored. The default value is  $+\infty$  for minimization ( $-\infty$  for maximization).

## Sensitivity/Parametric/Ranging Control Options

**NORANGEPRICE**

is the inverse of the [RANGEPRICE](#) option.

**NORANGERHS**

is the inverse of the [RANGERHS](#) option.

**PRICEPHI= $\Phi$**

specifies the limit for parametric programming when perturbing the price vector. See the section “[Parametric Programming](#)” on page 220 for further discussion. See [Example 5.5](#) for an illustration of this option.

**RANGEPRICE**

indicates that range analysis is to be performed on the price coefficients. See the section “[Range Analysis](#)” on page 219 for further discussion.

**RANGERHS**

indicates that range analysis is to be performed on the right-hand-side vector. See the section “[Range Analysis](#)” on page 219 for further discussion.

**RHSPHI= $\Phi$**

specifies the limit for parametric programming when perturbing the right-hand-side vector. See the section “[Parametric Programming](#)” on page 220 for further discussion.

## Simplex Algorithm Control Options

### DEVEX

indicates that the devex method of weighting the reduced costs be used in pricing (Harris 1975).

### EPSILON=*e*

specifies a positive number close to zero. It is used in the following instances:

During phase 1, if the sum of the basic artificial variables is within *e* of zero, the current solution is considered feasible. If this sum is not exactly zero, then there are artificial variables within *e* of zero in the current solution. In this case, a note is displayed on the SAS log.

During phase 1, if all reduced costs are  $\leq e$  for nonbasic variables at their lower bounds and  $\geq e$  for nonbasic variables at their upper bounds and the sum of infeasibilities is greater than *e*, then the problem is considered infeasible. If the maximum reduced cost is within *e* of zero, a note is displayed on the SAS log.

During phase 2, if all reduced costs are  $\leq e$  for nonbasic variables at their lower bounds and  $\geq e$  for nonbasic variables at their upper bounds, then the current solution is considered optimal.

During phases 1, 2, and 3, the EPSILON= option is also used to test if the denominator is different from zero before performing the ratio test to determine which basic variable should leave the basis.

The default value is 1.0E-8.

### GOALPROGRAM

specifies that multiple objectives in the input data set are to be treated as sequential objectives in a goal-programming model. The value of the right-hand-side variable in the objective row gives the priority of the objective. Lower numbers have higher priority.

### INFINITY=*r*

specifies the largest number PROC LP uses in computation. The INFINITY= option is used to determine when a problem has an unbounded variable value. The default value is the largest double precision number.<sup>1</sup>

### INVREQ=*m*

reinverts the current basis matrix after *m* major and minor iterations. The default value is 100.

### INTOL=*r*

reinverts the current basis matrix if the largest element in absolute value in the decomposed basis matrix is greater than *r*. If after reinversion this condition still holds, then the value of the INTOL= option is increased by a factor of 10 and a note indicating this modification is displayed on the SAS log. When *r* is frequently exceeded, this may be an indication of a numerically unstable problem. The default value is 1000.

### MAXIT=*n*

simultaneously sets the values of the MAXIT1=, MAXIT2=, MAXIT3=, and IMAXIT= options.

### MAXIT1=*n*

performs at most  $n \geq 0$  phase 1 iterations. The default value is 100.

---

<sup>1</sup>This value is system dependent.



**MAXIT2= $n$** 

performs at most  $n \geq 0$  phase 2 iterations. If MAXIT2=0, then only phase 1 is entered so that on successful termination PROC LP will have found a feasible, but not necessarily optimal, solution. The default value is 100.

**MAXIT3= $n$** 

performs at most  $n \geq 0$  phase 3 iterations. All dual pivots are counted as phase 3 pivots. The default value is 99999999.

**NODEVEX**

is the inverse of the [DEVEX](#) option.

**PARARESTORE**

indicates that following a parametric programming analysis, PROC LP should restore the basis.

**PHASEMIX= $r$** 

specifies a number between 0 and 1. When the number is positive, PROC LP tries to improve the objective function of phase 2 during phase 1. The PHASEMIX= option is a weight factor of the phase 2 objective function in phase 1. The default value is 0.

**PRICE= $m$** 

specifies the number of columns to subset when multiple pricing is used in selecting the column to enter the basis (Greenberg 1978). The type of suboptimization used is determined by the [PRICETYPE=](#) option. See the section “[Pricing](#)” on page 208 for a description of this process.

**PRICETYPE=*pricetype***

specifies the type of multiple pricing to be performed. If this option is specified and the [PRICE=](#) option is not specified, then [PRICE=](#) is assumed to be 10. Valid values for the PRICETYPE= option are

- [PRICETYPE=COMPLETE](#)
- [PRICETYPE=DYNAMIC](#)
- [PRICETYPE=NONE](#)
- [PRICETYPE=PARTIAL](#)

The default value is PARTIAL. See the section “[Pricing](#)” on page 208 for a description of this process.

**RANDOMPRICEMULT= $r$** 

specifies a number between 0 and 1. This option sets a limit, in phase 1, on the number of iterations when PROC LP will randomly pick the entering variables. The limit equals  $r$  times the number of nonbasic variables, or the number of basic variables, whichever is smaller. The default value of the RANDOMPRICEMULT= option is 0.01.

**REPSILON= $e$** 

specifies a positive number close to zero. The REPSILON= option is used in the ratio test to determine which basic variable is to leave the basis. The default value is 1.0E–10.

**SCALE=scale**

specifies the type of scaling to be used. Valid values for the SCALE= option are

- SCALE=BOTH
- SCALE=COLUMN
- SCALE=NONE
- SCALE=ROW

The default value is BOTH. See the section “[Scaling](#)” on page 209 for further discussion.

**SMALL=e**

specifies a positive number close to zero. Any element in a matrix with a value less than *e* is set to zero. The default value is machine dependent.

**TIME=t**

checks at each iteration to see if *t* seconds have elapsed since PROC LP began. If more than *t* seconds have elapsed, the procedure pauses and displays the current solution. The default value is 120 seconds.

**U=r**

enables PROC LP to control the choice of pivots during LU decomposition and updating the basis matrix. The variable *r* should take values between EPSILON and 1.0 because small values of *r* bias the algorithm toward maintaining sparsity at the expense of numerical stability and vice versa. The more sparse the decomposed basis is, the less time each iteration takes. The default value is 0.1.

---

## COEF Statement

**COEF** *variables* ;

For the sparse input format, the COEF statement specifies the numeric variables in the problem data set that contain the coefficients in the model. The value of the coefficient variable in a given observation is the value of the coefficient in the column and row specified in the COLUMN and ROW variables in that observation. For multiple ROW variables, the LP procedure maps the ROW variables to the COEF variables on the basis of their order in the COEF and ROW statements. There must be the same number of COEF variables as ROW variables. If the COEF statement is omitted, the procedure looks for the default variable names that have the prefix \_COEF.

---

## COL Statement

**COL** *variable* ;

For the sparse input format, the COL statement specifies a character variable in the problem data set that contains the names of the columns in the model. Columns in the model are either structural variables, right-hand-side vectors, right-hand-side change vectors, or a range vector. The COL variable must be a character variable. If the COL statement is omitted, the LP procedure looks for the default variable name \_COL\_.

---

## ID Statement

**ID** *variable(s)* ;

For the dense input format, the ID statement specifies a character variable in the problem data set that contains a name for each constraint coefficients row, objective coefficients row, and variable definition row. If the ID statement is omitted, the LP procedure looks for the default variable name, `_ID_`. If this variable is not in the problem data set, the procedure assigns the default name `_OBSxx_` to each row, where `xx` specifies the observation number in the problem data set.

For the sparse input format, the ID statement specifies the character variables in the problem data set that contain the names of the rows in the model. Rows in the model are one of the following types: constraints, objective functions, bounding rows, or variable describing rows. The ID variables must be character variables. There must be the same number of ID variables as variables specified in the [COEF](#) statement. If the ID statement is omitted, the LP procedure looks for the default variable names having the prefix `_ID`.

**NOTE:** The ID statement is an alias for the [ROW](#) statement.

---

## IPIVOT Statement

**IPIVOT** ;

The IPIVOT statement causes the LP procedure to execute one integer branch-and-bound pivot and pause. If you use the IPIVOT statement while the [PROXIMITYPAUSE=](#) option is in effect, pause interferences may occur. To avoid such interferences, you must either reset the [PROXIMITYPAUSE](#) value or submit IPIVOT; RUN; instead of IPIVOT;.

---

## PIVOT Statement

**PIVOT** ;

The PIVOT statement causes the LP procedure to execute one simplex pivot and pause.

---

## PRINT Statement

**PRINT** *options* ;

The PRINT statement is useful for displaying part of a solution summary, examining intermediate tableaus, performing sensitivity analysis, and using parametric programming. In the options, the `colnames` and `rownames` lists can be empty, in which case the LP procedure displays tables with all columns or rows, or both. If a column name or a row name has spaces or other special characters in it, the name must be enclosed in single or double quotes when it appears in the argument.

The options that can be used with this statement are as follows.

### **BEST**

displays a Solution, Variable, and Constraint Summary for the best integer solution found.

### **COLUMN(*colnames*) / SENSITIVITY**

displays a Variable Summary containing the logical and structural variables listed in the *colnames* list. If the / SENSITIVITY option is included, then sensitivity analysis is performed on the price coefficients for the listed *colnames* structural variables.

### **INTEGER**

displays a Variable Summary containing only the integer variables.

### **INTEGER\_NONZEROS**

displays a Variable Summary containing only the integer variables with nonzero activity.

### **INTEGER\_ZEROS**

displays a Variable Summary containing only the integer variables with zero activity.

### **MATRIX(*rownames,colnames*) / PICTURE**

displays the submatrix of the matrix of constraint coefficients defined by the *rownames* and *colnames* lists. If the / PICTURE option is included, then the formatted submatrix is displayed. The format used is summarized in [Table 5.2](#).

**Table 5.2** Format Summary

| Condition on the Coefficient x |        |                   |         | Symbols Printed |     |
|--------------------------------|--------|-------------------|---------|-----------------|-----|
|                                |        | $\text{abs}(x) =$ | 0       |                 | “ ” |
| 0                              | <      | $\text{abs}(x) <$ | .000001 | $\text{sgn}(x)$ | “Z” |
| .000001                        | $\leq$ | $\text{abs}(x) <$ | .00001  | $\text{sgn}(x)$ | “Y” |
| .00001                         | $\leq$ | $\text{abs}(x) <$ | .0001   | $\text{sgn}(x)$ | “X” |
| .0001                          | $\leq$ | $\text{abs}(x) <$ | .001    | $\text{sgn}(x)$ | “W” |
| .001                           | $\leq$ | $\text{abs}(x) <$ | .01     | $\text{sgn}(x)$ | “V” |
| .01                            | $\leq$ | $\text{abs}(x) <$ | .1      | $\text{sgn}(x)$ | “U” |
| .1                             | $\leq$ | $\text{abs}(x) <$ | 1       | $\text{sgn}(x)$ | “T” |
|                                |        | $\text{abs}(x) =$ | 1       | $\text{sgn}(x)$ | “1” |
| 1                              | <      | $\text{abs}(x) <$ | 10      | $\text{sgn}(x)$ | “A” |
| 10                             | $\leq$ | $\text{abs}(x) <$ | 100     | $\text{sgn}(x)$ | “B” |
| 100                            | $\leq$ | $\text{abs}(x) <$ | 1000    | $\text{sgn}(x)$ | “C” |
| 1000                           | $\leq$ | $\text{abs}(x) <$ | 10000   | $\text{sgn}(x)$ | “D” |
| 10000                          | $\leq$ | $\text{abs}(x) <$ | 100000  | $\text{sgn}(x)$ | “E” |
| 100000                         | $\leq$ | $\text{abs}(x) <$ | 1.0E06  | $\text{sgn}(x)$ | “F” |

### **NONINTEGER**

displays a Variable Summary containing only the continuous variables.

### **NONINTEGER\_NONZEROS**

displays a Variable Summary containing only the continuous variables with nonzero activity.

**NONZEROS**

displays a Variable Summary containing only the variables with nonzero activity.

**PRICESEN**

displays the results of parametric programming for the current value of the **PRICEPHI=** option, the price coefficients, and all of the price change vectors.

**RANGEPRICE**

performs range analysis on the price coefficients.

**RANGERHS**

performs range analysis on the right-hand-side vector.

**RHSEN**

displays the results of parametric programming for the current value of the **RHSPHI=** option, the right-hand-side coefficients, and all of the right-hand-side change vectors.

**ROW(rownames) / SENSITIVITY**

displays a constraint summary containing the rows listed in the rowname list. If the / SENSITIVITY option is included, then sensitivity analysis is performed on the right-hand-side coefficients for the listed rownames.

**SOLUTION**

displays the Solution Summary, including the Variable Summary and the Constraint Summary.

**TABLEAU**

displays the current tableau.

**ZEROS**

displays a Variable Summary containing only the variables with zero activity. This may be useful in the analysis of ON/OFF, ZERO/ONE, scheduling, and assignment applications.

---

## QUIT Statement

**QUIT** *options* ;

The QUIT statement causes the LP procedure to terminate processing immediately. No further displaying is performed and no output data sets are created.

The QUIT/SAVE statement causes the LP procedure to save the output data sets, defined in the **PROC LP** statement or in the **RESET** statement, and then terminate the procedure.

---

## RANGE Statement

**RANGE** *variable* ;

For the dense input format, the RANGE statement identifies the variable in the problem data set that contains the range coefficients. These coefficients enable you to specify the feasible range of a row. For example, if the *i*th row is

$$a^T x \leq b_i$$

and the range coefficient for this row is  $r_i > 0$ , then all values of  $x$  that satisfy

$$b_i - r_i \leq a^T x \leq b_i$$

are feasible for this row. Table 5.3 shows the bounds on a row as a function of the row type and the sign on a nonmissing range coefficient  $r$ .

**Table 5.3** Interpretation of the Range Coefficient

| $r$      | _TYPE_ | Bounds    |           |
|----------|--------|-----------|-----------|
|          |        | Lower     | Upper     |
| $\neq 0$ | LE     | $b -  r $ | $b$       |
| $\neq 0$ | GE     | $b$       | $b +  r $ |
| $> 0$    | EQ     | $b$       | $b + r$   |
| $< 0$    | EQ     | $b + r$   | $b$       |

If you include a range variable in the model and have a missing value or zero for it in a constraint row, then that constraint is treated as if no range variable had been included.

If the RANGE statement is omitted, the LP procedure assumes that the variable named `_RANGE_` contains the range coefficients.

For the sparse input format, the RANGE statement gives the name of a column in the problem data set that contains the range constants. If the RANGE statement is omitted, then the LP procedure assumes that the column named `_RANGE_` or the column with the 'RANGE' keyword in the problem data set contains the range constants.

---

## RESET Statement

**RESET** *options* ;

The RESET statement is used to change options after the LP procedure has started execution.

All of the options that can be set in the PROC LP statement can also be reset with the RESET statement, except for the `DATA=`, the `PRIMALIN=`, and the `ACTIVEIN=` options. In addition to the options available with the PROC LP statement, the following two options can be used.

**LOWER(colnames)=n;**

During phase 3, this sets the lower bound on all of the structural variables listed in the `colnames` list to an integer value  $n$ . This may contaminate the branch-and-bound tree. All nodes that descend from the current problem have lower bounds that may be different from those input in the problem data set.

**UPPER(colnames)=n;**

During phase 3, this sets the upper bound on all of the structural variables listed in the `colnames` list to an integer value  $n$ . This may contaminate the branch-and-bound tree. All nodes that descend from the current problem have upper bounds that may be different from those input in the problem data set.

Note that the `LOWER=` and `UPPER=` options only apply to phase 3 for integer problems. Therefore, they should only be applied once the integer iterations have started; if they are applied before then, they will be ignored.

---

## RHS Statement

**RHS** *variables* ;

For the dense input format, the RHS statement identifies variables in the problem data set that contain the right-hand-side constants of the linear program. Only numeric variables can be specified. If more than one variable is included in the RHS statement, the LP procedure assumes that problems for several linear programs are defined in the problem data set. A new linear program is defined for each variable in the RHS list. If the RHS statement is omitted, the procedure assumes that a variable named `_RHS_` contains the right-hand-side constants.

For the sparse input format, the RHS statement gives the names of one or more columns in the problem data set that are to be considered as right-hand-side constants. If the RHS statement is omitted, then the LP procedure assumes that the column named `_RHS_` or columns with the 'RHS' keyword in the problem data set contain the right-hand-side constants. See the section “[Sparse Data Input Format](#)” on page 201 for further discussion.

As default, the LP procedure assumes that the RHS constant is a zero vector for the dense and sparse input formats.

---

## RHSSEN Statement

**RHSSEN** *variables* ;

For the dense input format, the RHSSEN statement identifies variables in the problem data set that define change vectors for examining the sensitivity of the optimal solution to changes in the RHS constants. If the RHSSEN statement is omitted, then the LP procedure assumes that a variable named `_RHSSEN_` contains a right-hand-side change vector.

For the sparse input format, the RHSSEN statement gives the names of one or more columns in the problem data set that are to be considered as change vectors. If the RHSSEN statement is omitted, then the LP procedure assumes that the column named `_RHSSEN_` or columns with the 'RHSSEN' keyword in the problem data set contain the right-hand-side change vectors. For further information, see the section “[Sparse Data Input Format](#)” on page 201, the section “[Right-Hand-Side Sensitivity Analysis](#)” on page 217, and the section “[Right-Hand-Side Parametric Programming](#)” on page 220.

---

## ROW Statement

**ROW** *variable(s)* ;

For the dense input format, the ROW statement specifies a character variable in the problem data set that contains a name for each row of constraint coefficients, each row of objective coefficients and each variable describing row. If the ROW statement is omitted, the LP procedure looks for the default variable name, `_ROW_`. If there is no such variable in the problem data set, the procedure assigns the default name `_OBSxx_` to each row, where *xx* specifies the observation number in the problem data set.

For the sparse input format, the ROW statement specifies the character variables in the problem data set that contain the names of the rows in the model. Rows in the model are one of the following types: constraints,

objective functions, bounding rows, or variable describing rows. The ROW variables must be character variables. There must be the same number of ROW variables as variables specified in the **COEF** statement. If the ROW statement is omitted, the LP procedure looks for the default variable names having the prefix `_ROW`.

---

## RUN Statement

**RUN ;**

The RUN statement causes optimization to be started or resumed.

The TITLE or OPTIONS statement should not appear between PROC LP and RUN statements.

---

## SHOW Statement

**SHOW options ;**

The SHOW statement specifies that the LP procedure display either the *current options* or the *current solution status* on the SAS log.

### OPTIONS

requests that the current options be displayed on the SAS log.

### STATUS

requests that the status of the current solution be displayed on the SAS log.

---

## TYPE Statement

**TYPE variable ;**

The TYPE statement specifies a character variable in the problem data set that contains the type identifier for each observation. This variable has keyword values that specify how the LP procedure should interpret the observation. If the TYPE statement is omitted, the procedure assumes that a variable named `_TYPE_` contains the type keywords.

For the dense input format, the type variable identifies the constraint and objective rows and rows that contain information about the variables. The type variable should have nonmissing values in all observations.

For the sparse input format, the type variable identifies a model's rows and columns. In an observation, a nonmissing type is associated with either a row or a column. If there are many columns sharing the same type, you can define a row of that type. Then, any nonmissing values in that row set the types of the corresponding columns.

The following are valid values for the TYPE variable in an observation:

|     |                                                                                                              |
|-----|--------------------------------------------------------------------------------------------------------------|
| MIN | contains the price coefficients of an objective row, for example, $c$ in the problem (MIP), to be minimized. |
| MAX | contains the price coefficients of an objective row, for example, $c$ , to be maximized.                     |



|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EQ (=)              | contains coefficients of an equality constrained row.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| LE ( $\leq$ )       | contains coefficients of an inequality, less than or equal to, constrained row.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| GE ( $\geq$ )       | contains coefficients of an inequality, greater than or equal to, constrained row.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| SOSEQ               | identifies the row as specifying a special ordered set. The variables flagged in this row are members of a set <i>exactly one</i> of which must be above its lower bound in the optimal solution. Note that variables in this type of special ordered set must be <i>integer</i> .                                                                                                                                                                                                                                                                                                                                                        |
| SOSLE               | identifies the row as specifying a special ordered set. The variables flagged in this row are members of a set in which only one can be above its lower bound in the optimal solution.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| UNRSTRT<br>UNRSTRCT | identifies those structural variables to be considered as unrestricted variables. These are variables for which $\ell_i = -\infty$ and $u_i = +\infty$ . Any variable that has a 1 in this observation is considered an unrestricted variable.                                                                                                                                                                                                                                                                                                                                                                                            |
| LOWERBD             | identifies lower bounds on the structural variables. If all structural variables are to be nonnegative, that is, $\ell_i = 0$ , then you do not need to include an observation with the 'LOWERBD' keyword in a variable specified in the TYPE statement. Missing values for variables in a lower-bound row indicate that the variable has lower bound equal to zero.<br><b>NOTE:</b> A variable with lower or upper bounds cannot be identified as unrestricted.                                                                                                                                                                          |
| UPPERBD             | identifies upper bounds $u_i$ on the structural variables. For each structural variable that is to have an upper bound $u_i = +\infty$ , the observation must contain a missing value or the current value of INFINITY. All other values are interpreted as upper bounds, including 0.                                                                                                                                                                                                                                                                                                                                                    |
| FIXED               | identifies variables that have fixed values. A nonmissing value in a row with 'FIXED' type keyword gives the constant value of that variable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| INTEGER             | identifies variables that are integer-constrained. In a feasible solution, these variables must have integer values. A missing value in a row with 'INTEGER' type keyword indicates that the variable is not integer-constrained. The value of variables in the 'INTEGER' row gives an ordering to the integer-constrained variables that is used when the <code>VARSELECT=</code> option equals PRIOR.<br><b>NOTE:</b> Every integer-constrained variable must have an upper bound defined in a row with type 'UPPERBD'. See the section " <a href="#">Controlling the Branch-and-Bound Search</a> " on page 213 for further discussion. |

|          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BINARY   | identifies variables that are constrained to be either 0 or 1. This is equivalent to specifying that the variable is an integer variable and has a lower bound of 0 and an upper bound of 1. A missing value in a row with 'BINARY' type keyword indicates that the variable is not constrained to be 0 or 1. The value of variables in the 'BINARY' row gives an ordering to the integer-constrained variables that is used when the <code>VARSELECT=</code> option equals <code>PRIOR</code> . See the section “ <a href="#">Controlling the Branch-and-Bound Search</a> ” on page 213 for further discussion. |
| BASIC    | identifies variables that form an initial basic feasible solution. A missing value in a row with 'BASIC' type indicates that the variable is not basic.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| PRICESEN | identifies a vector that is used to evaluate the sensitivity of the optimal solution to changes in the objective function. See the section “ <a href="#">Price Sensitivity Analysis</a> ” on page 218 and the section “ <a href="#">Price Parametric Programming</a> ” on page 221 for further discussion.                                                                                                                                                                                                                                                                                                       |
| FREE     | identifies a nonbinding constraint. Any number of FREE constraints can appear in a problem data set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| RHS      | identifies a right-hand-side column in the sparse input format. This replaces the <code>RHS</code> statement. It is useful when converting the MPS format into the sparse format of PROC LP. See the section “ <a href="#">Converting Standard MPS Format to Sparse Format</a> ” on page 204 for more information.                                                                                                                                                                                                                                                                                               |
| RHSEN    | identifies a right-hand-side sensitivity analysis vector in the sparse input format. This replaces the <code>RHSEN</code> statement. It is useful when converting the MPS format into the sparse format of PROC LP. See the section “ <a href="#">Converting Standard MPS Format to Sparse Format</a> ” on page 204 for more information.                                                                                                                                                                                                                                                                        |
| RANGE    | identifies a range vector in the sparse input format. This replaces the <code>RANGE</code> statement. It is useful when converting the MPS format into the sparse format of PROC LP. See the section “ <a href="#">Converting Standard MPS Format to Sparse Format</a> ” on page 204 for more information.                                                                                                                                                                                                                                                                                                       |

---

## VAR Statement

**VAR** *variables* ;

For the dense input format, the VAR statement identifies variables in the problem data set that are to be interpreted as structural variables in the linear program. Only numeric variables can be specified. If no VAR statement is specified, the LP procedure uses all numeric variables not included in an `RHS` or `RHSEN` statement as structural variables.

---

## Details: LP Procedure

---

### Missing Values

The LP procedure treats missing values as missing in all rows except those that identify either upper or lower bounds on structural variables. If the row is an upper-bound row, then the type identifier is 'UPPERBD' and the LP procedure treats missing values as  $+\infty$ . If the row is a lower-bound row, then the type identifier is 'LOWERBD' and the LP procedure treats missing values as 0, except for the variables that are identified as 'UNRSTRT'.

---

### Dense Data Input Format

In the dense format, a model is expressed in a similar way as it is formulated. Each SAS variable corresponds to a model's column and each SAS observation corresponds to a model's row. A SAS variable in the input data set is one of the following:

- a **type** variable
- an **id** variable
- a **structural** variable
- a **right-hand-side** variable
- a **right-hand-side sensitivity analysis** variable
- a **range** variable

The type variable tells PROC LP how to interpret the observation as a part of the mathematical programming problem. It identifies and classifies objectives, constraints, and the rows that contain information of variables like types, bounds, and so on. PROC LP recognizes the following keywords as values for the type variable: MIN, MAX, EQ, LE, GE, SOSEQ, SOSLE, UNRSTRT, LOWERBD, UPPERBD, FIXED, INTEGER, BINARY, BASIC, PRICESEN, and FREE. The values of the id variable are the names of the rows in the model. The other variables identify and classify the columns with numerical values.

The TYPE, ID (or ROW), and RHS statements can be omitted if the input data set contains variables `_TYPE_`, `_ID_` (or `_ROW_`), and `_RHS_`; otherwise, they must be used. The VAR statement is optional. When it is not specified, PROC LP uses as structural variables all numeric variables not explicitly or implicitly included in statement lists. The RHSEN and RANGE statements are optional statements for sensitivity and range analyses. They can be omitted if the input data set contains the `_RHSEN_` and `_RANGE_` variables.

---

### Sparse Data Input Format

The sparse format to PROC LP is designed to enable you to specify only the nonzero coefficients in the description of linear programs, integer programs, and mixed-integer programs. The SAS data set that describes the sparse model must contain at least four SAS variables:

- a **type** variable
- a **column** variable
- a **row** variable
- a **coefficient** variable

Each observation in the data set associates a type with a row or column, and defines a coefficient or numerical value in the model. The value of the type variable is a keyword that tells PROC LP how to interpret the observation. In addition to the keywords in the dense format, PROC LP also recognizes the keywords RHS, RHSEN, and RANGE as values of the type variable. [Table 5.5](#) shows the keywords that are recognized by PROC LP and in which variables can appear in the problem data set.

The values of the row and column variables are the names of the rows and columns in the model. The values of the coefficient variables define basic coefficients and lower and upper bounds, and identify model variables with types BASIC, FIXED, BINARY, and INTEGER. All character values in the sparse data input format are case insensitive.

The SAS data set can contain multiple pairs of rows and coefficient variables. In this way, more information about the model can be specified in each observation in the data set. See [Example 5.2](#) for details.

**Table 5.5** Variable Keywords Used in the Problem Data Set

| TYPE (_TYPE_) | COL (_COL_) |
|---------------|-------------|
| MIN           |             |
| MAX           |             |
| EQ            |             |
| LE            |             |
| GE            |             |
| SOSEQ         |             |
| SOSLE         |             |
| UNRSTR        |             |
| LOWERBD       |             |
| UPPERBD       |             |
| FIXED         |             |
| INTEGER       |             |
| BINARY        |             |
| BASIC         |             |
| PRICESEN      |             |
| FREE          |             |
| RHS           | _RHS_       |
| RHSEN         | _RHSEN_     |
| RANGE         | _RANGE_     |
| *****         |             |

Follow these rules for sparse data input:

- The order of the observations is unimportant.

- Each unique column name appearing in the **COL** variable defines a unique column in the model.
- Each unique row name appearing in the **ROW** variable defines a unique row in the model.
- The type of the row is identified when an observation in which the row name appears (in a **ROW** variable) has type MIN, MAX, LE, GE, EQ, SOSLE, SOSEQ, LOWERBD, UPPERBD, UNRSTRT, FIXED, BINARY, INTEGER, BASIC, FREE, or PRICESEN.
- The type of each row must be identified at least once. If a row is given a type more than once, the multiple definitions must be identical.
- When there are multiple rows named in an observation (that is, when there are multiple **ROW** variables), the **TYPE** variable applies to each row named in the observation.
- The type of a column is identified when an observation in which the column name but no row name appears has the type LOWERBD, UPPERBD, UNRSTRT, FIXED, BINARY, INTEGER, BASIC, RHS, RHSEN, or RANGE. A column type can also be identified in an observation in which both column and row names appear and the row name has one of the preceding types.
- Each column is assumed to be a structural column in the model unless the column is identified as a right-hand-side vector, a right-hand-side change vector, or a range vector. A column can be identified as one of these types using either the keywords RHS, RHSEN, or RANGE in the **TYPE** variable, the special column names **\_RHS\_**, **\_RHSEN\_**, or **\_RANGE\_**, or the **RHS**, **RHSEN**, or **RANGE** statements following the PROC LP statement.
- A **TYPE** variable beginning with the character \* causes the observation to be interpreted as a comment.

When the column names appear in the Variable Summary in the PROC LP output, they are listed in alphabetical order. The row names appear in the order in which they appear in the problem data set.

---

## Converting Any PROC LP Format to an MPS-Format SAS Data Set

The **MPSOUT=** option enables you to convert an input data set for the LP procedure into an MPS-format SAS data set. The converted data set is readable by the OPTLP and OPTMILP procedures.

The conversion can handle both linear and mixed integer linear programs. The **\_TYPE\_** values for sensitivity analysis (PRICESEN), parametric programming (RHSEN), and input basis (BASIS) are dropped. When multiple objective rows are present, only the first row is marked as the objective row. The remaining rows are marked as free rows. When multiple right-hand side (RHS) columns are present, only the first RHS column is processed. Constraints with a **\_TYPE\_** value of SOSEQ or SOSLE are ignored. The **MPSOUT=** option does not output branching priorities specified for the **VARSELECT=PRIOR** option to a **BRANCH** section in the MPS-format SAS data set.

For information about how the contents of the MPS-format SAS data set are interpreted, see Chapter 17, “The MPS-Format SAS Data Set” (*SAS/OR User’s Guide: Mathematical Programming*). For examples that demonstrate the use of the **MPSOUT=** option and migration to the OPTMODEL procedure, see the section “Examples: LP Procedure” on page 232.

## Converting Standard MPS Format to Sparse Format

The MPS input format was introduced by IBM as a way of specifying data for linear and integer programs. Before you can solve a linear program specified in the MPS input format by using the LP procedure, the data must be converted to the sparse format of the LP procedure. If you want to solve a linear program specified in the sparse LP format by using the OPTLP procedure, you must convert the data into an MPS-format SAS data set. This section describes how to perform both conversions.

SASMPXS is a SAS macro function that converts the standard MPS format to the sparse format of the LP procedure. The following is an example of the MPS format:

```

NAME          EXAMPLE
* THIS IS DATA FOR THE PRODUCT MIX PROBLEM.
ROWS
  N  PROFIT
  L  STAMP
  L  ASSEMB
  L  FINISH
  N  CHNROW
  N  PRICE
COLUMNS
  DESK      STAMP      3.00000  ASSEMB      10.00000
  DESK      FINISH     10.00000  PROFIT      95.00000
  DESK      PRICE     175.00000
  CHAIR     STAMP      1.50000  ASSEMB       6.00000
  CHAIR     FINISH     8.00000  PROFIT     41.00000
  CHAIR     PRICE     95.00000
  CABINET   STAMP      2.00000  ASSEMB       8.00000
  CABINET   FINISH     8.00000  PROFIT     84.00000
  CABINET   PRICE    145.00000
  BOOKCSE   STAMP      2.00000  ASSEMB       7.00000
  BOOKCSE   FINISH     7.00000  PROFIT     76.00000
  BOOKCSE   PRICE    130.00000  CHNROW       1.00000
RHS
  TIME      STAMP      800.00000  ASSEMB     1200.0000
  TIME      FINISH     800.00000
RANGES
  T1        ASSEMB     900.00000
BOUNDS
  UP        CHAIR      75.00000
  LO        BOOKCSE    50.00000
ENDATA

```

In this example, the company tries to find an optimal product mix of four items: a DESK, a CHAIR, a CABINET, and a BOOKCASE. Each item is processed in a stamping department (STAMP), an assembly department (ASSEMB), and a finishing department (FINISH). The time each item requires in each department is given in the input data. Because of resource limitations, each department has an upper limit on the time available for processing. Furthermore, because of labor constraints, the assembly department must work at least 300 hours. Finally, marketing tells you not to make more than 75 chairs, to make at least 50 bookcases, and to find the range over which the selling price of a bookcase can vary without changing the optimal product mix.

The SASMPSEX macro function uses MPSFILE='FILENAME' as an argument to read an MPS input file. It then converts the file and saves the conversion to a default SAS data set, PROB. The FILENAME should include the path.

Running the following statements on the preceding example

```
%sasmpsex(mpsfile='filename');

proc print data=prob;
run;
```

produces the sparse input form of the LP procedure:

| OBS | _TYPE_   | _COL_   | _ROW1_ | _COEF1_ | _ROW2_ | _COEF2_ |
|-----|----------|---------|--------|---------|--------|---------|
| 1   | *OW      |         |        | .       |        | .       |
| 2   | FREE     |         | PROFIT | .       |        | .       |
| 3   | LE       |         | STAMP  | .       |        | .       |
| 4   | LE       |         | ASSEMB | .       |        | .       |
| 5   | LE       |         | FINISH | .       |        | .       |
| 6   | FREE     |         | CHNROW | .       |        | .       |
| 7   | FREE     |         | PRICE  | .       |        | .       |
| 8   | *OL      | MNS     |        | .       |        | .       |
| 9   |          | DESK    | STAMP  | 3.0     | ASSEMB | 10      |
| 10  |          | DESK    | FINISH | 10.0    | PROFIT | 95      |
| 11  |          | DESK    | PRICE  | 175.0   |        | .       |
| 12  |          | CHAIR   | STAMP  | 1.5     | ASSEMB | 6       |
| 13  |          | CHAIR   | FINISH | 8.0     | PROFIT | 41      |
| 14  |          | CHAIR   | PRICE  | 95.0    |        | .       |
| 15  |          | CABINET | STAMP  | 2.0     | ASSEMB | 8       |
| 16  |          | CABINET | FINISH | 8.0     | PROFIT | 84      |
| 17  |          | CABINET | PRICE  | 145.0   |        | .       |
| 18  |          | BOOKCSE | STAMP  | 2       | ASSEMB | 7       |
| 19  |          | BOOKCSE | FINISH | 7       | PROFIT | 76      |
| 20  |          | BOOKCSE | PRICE  | 130     | CHNROW | 1       |
| 21  | *HS      |         |        | .       |        | .       |
| 22  | RHS      | TIME    | STAMP  | 800     | ASSEMB | 1200    |
| 23  | RHS      | TIME    | FINISH | 800     |        | .       |
| 24  | *AN      | ES      |        | .       |        | .       |
| 25  | RANGE    | T1      | ASSEMB | 900     |        | .       |
| 26  | *OU      | DS      |        | .       |        | .       |
| 27  | UPPERBDD | CHAIR   | UP     | 75      |        | .       |
| 28  | LOWERBDD | BOOKCSE | LO     | 50      |        | .       |

SASMPSEX recognizes four MPS row types: E, L, G, and N. It converts them into types EQ, LE, GE, and FREE. Since objective rows, price change rows and free rows all share the same type N in the MPS format, you need a DATA step to assign proper types to the objective rows and price change rows.

```

data;
  set prob;
  if _type_='free' and _row1_='profit' then _type_='max';
  if _type_='free' and _row1_='chnrow' then _type_='pricesen';
run;

proc lp sparsedata;
run;

```

In the MPS format, the variable types include LO, UP, FX, FR, MI, and BV. The SASMPXS macro converts them into types LOWERBD, UPPERBD, FIXED, UNRESTRICTED, -INFINITY, and BINARY, respectively. Occasionally, you may need to define your own variable types, in which case, you must add corresponding type handling entries in the SASMPXS.SAS program and use the SAS %INCLUDE macro to include the file at the beginning of your program. The SASMPXS macro function can be found in the SAS sample library. Information on the MPS format can be obtained from Murtagh (1981).

SASMPXS can take no arguments, or it can take one or two arguments. If no arguments are present, SASMPXS assumes that the MPS input file has been saved to a SAS data set named RAW. The macro then takes information from that data set and converts it into the sparse form of the LP procedure. The RAW data set should have the following six variables:

```

data RAW;
  infile ...;
  input field1 $ 2-3   field2 $ 5-12
        field3 $ 15-22 field4   25-36
        field5 $ 40-47 field6   50-61;
  ...
run;

```

If the preceding MPS input data set has a name other than RAW, you can use MPSDATA=*SAS-data-set* as an argument in the SASMPXS macro function. If you want the converted sparse form data set to have a name other than PROB, you can use LPDATA=*SAS-data-set* as an argument. The order of the arguments in the SASMPXS macro function is not important.

---

## The Reduced Costs, Dual Activities, and Current Tableau

The evaluation of reduced costs and the dual activities is independent of problem structure. For a basic solution, let  $B$  be the matrix composed of the basic columns of  $A$  and let  $N$  be the matrix composed of the nonbasic columns of  $A$ . The reduced cost associated with the  $i$ th variable is

$$(c^T - c_B^T B^{-1} A)_i$$

and the dual activity of the  $j$ th row is

$$(c_B^T B^{-1})_j$$



The Current Tableau is a section displayed when you specify either the **TABLEAUPRINT** option in the PROC LP statement or the **TABLEAU** option in the **PRINT** statement. The output contains a row for each basic variable and a column for each nonbasic variable. In addition, there is a row for the reduced costs and a column for the product

$$B^{-1}b$$

This column is labeled INV(B)\*R. The body of the tableau contains the matrix

$$B^{-1}N$$

---

## Macro Variable \_ORLP\_

The LP procedure defines a macro variable named \_ORLP\_. This variable contains a character string that indicates the status of the procedure. It is set whenever the user gets control, at breakpoints, and at procedure termination. The form of the \_ORLP\_ character string is STATUS= PHASE= OBJECTIVE= P\_FEAS= D\_FEAS= INT\_ITER= INT\_FEAS= ACTIVE= INT\_BEST= PHASE1\_ITER= PHASE2\_ITER= PHASE3\_ITER=. The terms are interpreted as follows:

|              |                                                                 |
|--------------|-----------------------------------------------------------------|
| STATUS=      | the status of the current solution                              |
| PHASE=       | the phase the procedure is in (1, 2, or 3)                      |
| OBJECTIVE=   | the current objective value                                     |
| P_FEAS=      | whether the current solution is primal feasible                 |
| D_FEAS=      | whether the current solution is dual feasible                   |
| INT_ITER=    | the number of integer iterations performed                      |
| INT_FEAS=    | the number of integer feasible solutions found                  |
| ACTIVE=      | the number of active nodes in the current branch-and-bound tree |
| INT_BEST=    | the best integer objective value found                          |
| PHASE1_ITER= | the number of iterations performed in phase 1                   |
| PHASE2_ITER= | the number of iterations performed in phase 2                   |
| PHASE3_ITER= | the number of iterations performed in phase 3                   |

Table 5.7 shows the possible values for the nonnumeric terms in the string.

**Table 5.7** Possible Values for Nonnumeric Terms

| STATUS=        | P_FEAS= | D_FEAS= |
|----------------|---------|---------|
| SUCCESSFUL     | YES     | YES     |
| UNBOUNDED      | NO      | NO      |
| INFEASIBLE     |         |         |
| MAX_TIME       |         |         |
| MAX_ITER       |         |         |
| PIVOT          |         |         |
| BREAK          |         |         |
| INT_FEASIBLE   |         |         |
| INT_INFEASIBLE |         |         |
| INT_MAX_ITER   |         |         |
| PAUSE          |         |         |
| FEASIBLEPAUSE  |         |         |
| IPAUSE         |         |         |
| PROXIMITYPAUSE |         |         |
| ACTIVE         |         |         |
| RELAXED        |         |         |
| FATHOMED       |         |         |
| IPIVOT         |         |         |
| UNSTABLE       |         |         |
| SINGULAR       |         |         |
| MEMORY_ERROR   |         |         |
| IO_ERROR       |         |         |
| SYNTAX_ERROR   |         |         |
| SEMANTIC_ERROR |         |         |
| BADDATA_ERROR  |         |         |
| UNKNOWN_ERROR  |         |         |

This information can be used when PROC LP is one step in a larger program that needs to identify how the LP procedure terminated. Because `_ORLP_` is a standard SAS macro variable, it can be used in the ways that all macro variables can be used (see the *SAS Guide to Macro Processing*).

## Pricing

PROC LP performs multiple pricing when determining which variable will enter the basis at each pivot (Greenberg 1978). This heuristic can shorten execution time in many problems. The specifics of the multiple pricing algorithm depend on the value of the `PRICETYPE=` option. However, in general, when some form of multiple pricing is used, during the first iteration PROC LP places the `PRICE=` nonbasic columns yielding the greatest marginal improvement to the objective function in a candidate list. This list identifies a subproblem of the original. On subsequent iterations, only the reduced costs for the nonbasic variables in the candidate list are calculated. This accounts for the potential time savings. When either the candidate list is empty or the subproblem is optimal, a new candidate list must be identified and the process repeats. Because identification

of the subproblem requires pricing the complete problem, an iteration in which this occurs is called a *major iteration*. A *minor iteration* is an iteration in which only the subproblem is to be priced.

The value of the **PRICETYPE=** option determines the type of multiple pricing that is to be used. The types of multiple pricing include partial suboptimization (**PRICETYPE=PARTIAL**), complete suboptimization (**PRICETYPE=COMPLETE**), and complete suboptimization with dynamically varying the value of the **PRICE=** option (**PRICETYPE=DYNAMIC**).

When partial suboptimization is used, in each minor iteration the nonbasic column in the subproblem yielding the greatest marginal improvement to the objective is brought into the basis and removed from the candidate list. The candidate list now has one less entry. At each subsequent iteration, another column from the subproblem is brought into the basis and removed from the candidate list. When there are either no remaining candidates or the remaining candidates do not improve the objective, the subproblem is abandoned and a major iteration is performed. If the objective cannot be improved on a major iteration, the current solution is optimal and PROC LP terminates.

Complete suboptimization is identical to partial suboptimization with one exception. When a nonbasic column from the subproblem is brought into the basis, it is replaced in the candidate list by the basic column that is leaving the basis. As a result, the candidate list does not diminish at each iteration.

When **PRICETYPE=DYNAMIC**, complete suboptimization is performed, but the value of the **PRICE=** option changes so that the ratio of minor to major iterations is within two units of the **PRICE=** option.

These heuristics can shorten execution time for small values of the **PRICE=** option. Care should be exercised in choosing a value from the **PRICE=** option because too large a value can use more time than if pricing were not used.

---

## Scaling

Based on the **SCALE=** option specified, the procedure scales the coefficients of both the constraints and objective rows before iterating. This technique can improve the numerical stability of an ill-conditioned problem. If you want to modify the default matrix scaling used, which is **SCALE=BOTH**, use the **SCALE=COLUMN**, **SCALE=ROW**, or **SCALE=NONE** option in the PROC LP statement. If **SCALE=BOTH**, the matrix coefficients are scaled so that the largest element in absolute value in each row or column equals 1. They are scaled by columns first and then by rows. If **SCALE=COLUMN** (**ROW**), the matrix coefficients are scaled so that the largest element in absolute value in each column (row) equals 1. If **SCALE=NONE**, no scaling is performed.

---

## Preprocessing

With the preprocessing option, you can identify redundant and infeasible constraints, improve lower and upper bounds of variables, fix variable values and improve coefficients and RHS values before solving a problem. Preprocessing can be applied to LP, IP and MIP problems. For an LP problem, it may significantly reduce the problem size. For an IP or MIP problem, it can often reduce the gap between the optimal solution and the solution of the relaxed problem, which could lead to a smaller search tree in the branch-and-bound algorithm. As a result, the CPU time may be reduced on many problems. Although there is no guarantee that preprocessing will always yield a faster solution, it does provide a highly effective approach to solving large and difficult problems.

Preprocessing is especially useful when the original problem causes numerical difficulties to PROC LP. Since preprocessing could identify redundant constraints and tighten lower and upper bounds of variables, the reformulated problem may eliminate the numerical difficulties in practice.

When a constraint is identified as redundant, its type is marked as ‘FREE’ in the Constraint Summary. If a variable is fixed, its type is marked as ‘FIXED’ in the Variables Summary. If a constraint is identified as infeasible, PROC LP stops immediately and displays the constraint name in the SAS log file. This capability sometimes gives valuable insight into the model or the formulation and helps establish if the model is reasonable and the formulation is correct.

For a large and dense problem, preprocessing may take a longer time for each iteration. To limit the number of preprocessings, use the `PMAXIT=` option. To stop any further preprocessings during the preprocessing stage, press the CTRL-BREAK key. PROC LP will enter phase 1 at the end of the current iteration.

---

## Integer Programming

Formulations of mathematical programs often require that some of the decision variables take only integer values. Consider the formulation

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \{ \geq, =, \leq \} b \\ & \ell \leq x \leq u \\ & x_i \text{ is integer, } i \in \mathcal{S} \end{array}$$

The set of indices  $\mathcal{S}$  identifies those variables that must take only integer values. When  $\mathcal{S}$  does not contain all of the integers between 1 and  $n$ , inclusive, this problem is called a mixed-integer program (MIP). Otherwise, it is known as an integer program. Let  $x^{opt}(\text{MIP})$  denote an optimal solution to (MIP). An integer variable with bounds between 0 and 1 is also called a binary variable.

## Specifying the Problem

An integer or mixed-integer problem can be solved with PROC LP. To solve this problem, you must identify the integer variables. You can do this with a row in the input data set that has the keyword ‘INTEGER’ for the type variable. Any variable that has a nonmissing and nonzero value for this row is interpreted as an integer variable. It is important to note that integer variables must have upper bounds explicitly defined using the ‘UPPERBD’ keyword. The values in the ‘INTEGER’ row not only identify those variables that must be integers, but they also give an ordering to the integer variables that can be used in the solution technique.

You can follow the same steps to identify binary variables. For the binary variables, there is no need to supply any upper bounds.

Following the rules of sparse data input format, you can also identify individual integer or binary variables.

## The Branch-and-Bound Technique

The branch-and-bound approach is used to solve integer and mixed-integer problems. The following discussion outlines the approach and explains how to use several options to control the procedure.

The branch-and-bound technique solves an integer program by solving a sequence of linear programs. The sequence can be represented by a tree, with each node in the tree being identified with a linear program that is derived from the problems on the path leading to the root of the tree. The root of the tree is identified with a linear program that is identical to (MIP), except that  $\mathcal{S}$  is empty. This relaxed version of (MIP), called (LP(0)), can be written as

$$\begin{aligned} x^{opt}(0) = & \min c^T x \\ \text{subject to } & Ax \{ \geq, =, \leq \} b \\ & \ell \leq x \leq u \end{aligned}$$

The branch-and-bound approach generates linear programs along the nodes of the tree using the following scheme. Consider  $x^{opt}(0)$ , the optimal solution to (LP(0)). If  $x^{opt}(0)_i$  is integer for all  $i \in \mathcal{S}$ , then  $x^{opt}(0)$  is optimal in (MIP). Suppose for some  $i \in \mathcal{S}$ ,  $x^{opt}(0)_i$  is nonintegral. In that case, define two new problems (LP(1)) and (LP(2)), descendants of the parent problem (LP(0)). The problem (LP(1)) is identical to (LP(0)) except for the additional constraint

$$x_i \leq \lfloor x^{opt}(0)_i \rfloor$$

and the problem (LP(2)) is identical to (LP(0)) except for the additional constraint

$$x_i \geq \lceil x^{opt}(0)_i \rceil$$

The notation  $\lceil y \rceil$  means the smallest integer greater than or equal to  $y$ , and the notation  $\lfloor y \rfloor$  means the largest integer less than or equal to  $y$ . Note that the two new problems do not have  $x^{opt}(0)$  as a feasible solution, but because the solution to (MIP) must satisfy one of the preceding constraints,  $x_i^{opt}(\text{MIP})$  must satisfy one of the new constraints. The two problems thus defined are called active nodes in the branch-and-bound tree, and the variable  $x_i$  is called the branching variable.

Next, the algorithm chooses one of the problems associated with an active node and attempts to solve it using the dual simplex algorithm. The problem may be infeasible, in which case the problem is dropped. If it can be solved, and it in turn does not have an integer solution (that is, a solution for which  $x_i$  is integer for all  $i \in \mathcal{S}$ ), then it defines two new problems. These new problems each contain all of the constraints of the parent problems plus the appropriate additional one.

Branching continues in this manner until either there are no active nodes or an integer solution is found. When an integer solution is found, its objective value provides a bound for the objective of (MIP). In particular, if  $z$  is the objective value of the current best integer solution, then any active problems whose parent problem has objective value  $\geq z$  can be discarded (assuming that the problem is a minimization). This can be done because all problems that descend from this parent will also have objective value  $\geq z$ . This technique is known as *fathoming*. When there are no active nodes remaining to be solved, the current integer solution is optimal in (MIP). If no integer solution has been found, then (MIP) is (integer) infeasible.

It is important to realize that integer programs are NP-complete. Roughly speaking, this means that the effort required to solve them grows exponentially with the size of the problem. For example, a problem with 10 binary variables can, in the worst case, generate  $2^{10} = 1024$  nodes in the branch-and-bound tree. A problem with 20 binary variables can, in the worst case, generate  $2^{20} = 1,048,576$  nodes in the branch-and-bound tree. Although the algorithm is unlikely to have to generate every single possible node, the need to explore even a small fraction of the potential number of nodes for a large problem can be resource intensive.

## The Integer Iteration Log

To help monitor the growth of the branch-and-bound tree, the LP procedure reports on the status of each problem that is solved. The report, displayed in the Integer Iteration Log, can be used to reconstruct the branch-and-bound tree. Each row in the report describes the results of the attempted solution of the linear program at a node in the tree. In the following discussion, a problem on a given line in the log is called the current problem. The following columns are displayed in the report:

|           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Iter      | identifies the number of the branch-and-bound iteration.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Problem   | identifies how the current problem fits in the branch-and-bound tree.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Condition | reports the result of the attempted solution of the current problem. Values for Condition are: <ul style="list-style-type: none"> <li>• ACTIVE: The current problem was solved successfully.</li> <li>• INFEASIBLE: The current problem is infeasible.</li> <li>• FATHOMED: The current problem cannot lead to an improved integer solution and therefore it is dropped.</li> <li>• SINGULAR: A singular basis was encountered in attempting to solve the current problem. Solution of this relaxed problem is suspended and will be attempted later if necessary.</li> <li>• SUBOPTIMAL: The current problem has an integer feasible solution.</li> </ul> |
| Objective | reports the objective value of the current problem.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Branched  | names the variable that is branched in subtrees defined by the descendants of this problem.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Value     | gives the current value of the variable named in the column labeled Branched.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Sinfeas   | gives the sum of the integer infeasibilities in the optimal solution to the current problem.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Active    | reports the total number of nodes currently active in the branch-and-bound tree.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Proximity | reports the gap between the best integer solution and the current lower (upper for maximizations) bound of all active nodes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

To reconstruct the branch-and-bound tree from this report, consider the interpretation of iteration  $j$ . If Iter= $j$  and Problem= $k$ , then the problem solved on iteration  $j$  is identical to the problem solved on iteration  $|k|$  with an additional constraint. If  $k > 0$ , then the constraint is an upper bound on the variable named in the Branched column on iteration  $|k|$ . If  $k < 0$ , then the constraint is a lower bound on that variable. The value

of the bound can be obtained from the value of Value in iteration  $|k|$  as described in the previous section.

Example 5.8 in the section “Examples: LP Procedure” on page 232 shows an Integer Iteration Log in its output.

### Controlling the Branch-and-Bound Search

There are several options you can use to control branching. This is accomplished by controlling the program’s choice of the branching variable and of the next active node. In the discussion that follows, let

$$f_i(k) = x^{opt}(k)_i - \lfloor x^{opt}(k)_i \rfloor$$

where  $x^{opt}(k)$  is the optimal solution to the problem solved in iteration  $k$ .

The **CANSELECT=** option directs the choice of the next active node. Valid keywords for this option include LIFO, FIFO, OBJ, PROJECT, PSEUDOC, and ERROR. The following list describes the action that each of these causes when the procedure must choose for solution a problem from the list of active nodes.

|         |                                                                                                                                                                                                                                                                  |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LIFO    | chooses the last problem added to the tree of active nodes. This search has the effect of a depth-first search of the branch-and-bound tree.                                                                                                                     |
| FIFO    | chooses the first node added to the tree of active nodes. This search has the effect of a breadth-first search of the branch-and-bound tree.                                                                                                                     |
| OBJ     | chooses the problem whose parent has the smallest (largest if the problem is a maximization) objective value.                                                                                                                                                    |
| PROJECT | chooses the problem with the largest (smallest if the problem is a maximization) projected objective value. The projected objective value is evaluated using the sum of integer infeasibilities, $s(k)$ , associated with an active node (LP( $k$ )), defined by |

$$s(k) = \sum_{i \in S} \min\{f_i(k), 1 - f_i(k)\}$$

An empirical measure of the rate of increase (decrease) in the objective value is defined as

$$\lambda = (z^* - z(0))/s(0)$$

where

- $z(k)$  is the optimal objective value for (LP( $k$ ))
- $z^*$  is the objective value of the current best integer solution

The projected objective value for problems (LP( $k + 1$ )) and (LP( $k + 2$ )) is defined as

$$z(k) + \lambda s(k)$$

**PSEUDOC** chooses the problem with the largest (least if the problem is a maximization) projected pseudocost) The projected pseudocost is evaluated using the weighted sum of infeasibilities  $s_w(k)$  associated with an active problem (LP( $k$ )), defined by

$$s_w(k) = \sum_{i \in S} \min\{d_i(k)f_i(k), u_i(k)(1 - f_i(k))\}$$

The weights  $u_i$  and  $d_i$  are initially equal to the absolute value of the  $i$ th objective coefficient and are updated at each integer iteration. They are modified by examining the empirical marginal change in the objective as additional constraints are placed on the variables in  $S$  along the path from (LP(0)) to a node associated with an integer feasible solution. In particular, if the definition of problems (LP( $k + 1$ )) and (LP( $k + 2$ )) from parent (LP( $k$ )) involve the addition of constraints  $x_i \leq \lfloor x^{opt}(k)_i \rfloor$  and  $x_i \geq \lceil x^{opt}(k)_i \rceil$ , respectively, and one of them is on a path to an integer feasible solution, then only one of the following is true:

$$d_i(k) = (z(k + 1) - z(k))/f_i(k)$$

$$u_i(k) = (z(k + 2) - z(k))/(1 - f_i(k))$$

Note the similarity between  $s_w(k)$  and  $s(k)$ . The weighted quantity  $s_w(k)$  accounts to some extent for the influence of the objective function. The projected pseudocost for problems (LP( $k + 1$ )) and (LP( $k + 2$ )) is defined as

$$z_w(k) \equiv z(k) + s_w(k)$$

**ERROR** chooses the problem with the largest error. The error associated with problems (LP( $k + 1$ )) and (LP( $k + 2$ )) is defined as

$$(z^* - z_w(k))/(z^* - z(k))$$

The **BACKTRACK=** option controls the search for the next problem. This option can take the same values as the **CANSELECT=** option. In addition to the case outlined under the **DELTAIT=** option, backtracking is required as follows based on the **CANSELECT=** option in effect:

- If **CANSELECT=LIFO** and there is no active node in the portion of the active tree currently under exploration with a bound better than the value of **WOBJECTIVE=**, then the procedure must backtrack.
- If **CANSELECT=FIFO, PROJECT, PSEUDOC, or ERROR** and the bound corresponding to the node under consideration is not better than the value of **WOBJECTIVE=**, then the procedure must backtrack.

The default value is **OBJ**.

The **VARSELECT=** option directs the choice of branching variable. Valid keywords for this option include **CLOSE, FAR, PRIOR, PSEUDOC, PRICE, and PENALTY**. The following list describes the action that each of these causes when  $x^{opt}(k)$ , an optimal solution of problem (LP( $k$ )), is used to define active problems (LP( $k + 1$ )) and (LP( $k + 2$ )).



|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CLOSE   | chooses as branching variable the variable $x_i$ such that $i$ minimizes<br>$\{\min\{f_i(k), 1 - f_i(k)\} \mid i \in \mathcal{S} \text{ and } \text{IEPSILON} \leq f_i(k) \leq 1 - \text{IEPSILON}\}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| FAR     | chooses as branching variable the variable $x_i$ such that $i$ maximizes<br>$\{\min\{f_i(k), 1 - f_i(k)\} \mid i \in \mathcal{S} \text{ and } \text{IEPSILON} \leq f_i(k) \leq 1 - \text{IEPSILON}\}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| PRIOR   | chooses as branching variable $x_i$ such that $i \in \mathcal{S}$ , $x^{opt}(k)_i$ is nonintegral, and variable $x_i$ has the minimum value in the INTEGER row in the input data set. This choice for the <b>VARSELECT=</b> option is recommended when you have enough insight into the model to identify those integer variables that have the most significant effect on the objective value.                                                                                                                                                                                                                                                                                                                                             |
| PENALTY | chooses as branching variable $x_i$ such that $i \in \mathcal{S}$ and a bound on the increase in the objective of (LP( $k$ )) (penalty) resulting from adding the constraint<br>$x_i \leq \lfloor x^{opt}(k)_i \rfloor \quad \text{or} \quad x_i \geq \lceil x^{opt}(k)_i \rceil$<br><p>is maximized. The bound is calculated without pivoting using techniques of sensitivity analysis (Garfinkel and Nemhauser 1972). Because the cost of calculating the maximum penalty can be large if <math>\mathcal{S}</math> is large, you may want to limit the number of variables in <math>\mathcal{S}</math> for which the penalty is calculated. The penalty is calculated for <b>PENALTYDEPTH=</b> variables in <math>\mathcal{S}</math>.</p> |
| PRICE   | chooses as branching variable $x_i$ such that $i \in \mathcal{S}$ , $x^{opt}(k)_i$ is nonintegral, and variable $x_i$ has the minimum price coefficient (maximum for maximization).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| PSEUDOC | chooses as branching variable the variable $x_i$ such that $i$ maximizes<br>$\{\min\{d_i f_i(k), u_i(1 - f_i(k))\} \mid i \in \mathcal{S} \text{ and } \text{IEPSILON} \leq f_i(k) \leq 1 - \text{IEPSILON}\}$<br><p>The weights <math>u_i</math> and <math>d_i</math> are initially equal to the absolute value of the <math>i</math>th objective coefficient and are updated whenever an integer feasible solution is encountered. See the discussion on the <b>CANSELECT=</b> option for details on the method of updating the weights.</p>                                                                                                                                                                                              |

## Customizing Search Heuristics

Often a good heuristic for searching the branch-and-bound tree of a problem can be found. You are tempted to continue using this heuristic when the problem data changes but the problem structure remains constant. The ability to reset procedure options interactively enables you to experiment with search techniques in an attempt to identify approaches that perform well. Then you can easily reapply these techniques to subsequent problems.

For example, the PIP branch-and-bound strategy (Crowder, Johnson, and Padberg 1983) describes one such heuristic. The following program uses a similar strategy. Here, the OBJ rule (choose the active node with least parent objective function in the case of a minimization problem) is used for selecting the next active node to be solved until an integer feasible solution is found. Once such a solution is found, the search procedure is changed to the LIFO rule: choose the problem most recently placed in the list of active nodes.

```
proc lp canselect=obj ifeasiblepause=1;
run;
    reset canselect=lifo ifeasiblepause=9999999;
run;
```

### Further Discussion on AUTO and CONTROL= options

Consider a minimization problem. At each integer iteration, PROC LP will select a node to solve from a pool of active nodes. The best bound strategy ( **CANSELECT=OBJ** ) will pick the node with the smallest projected objective value. This strategy improves the lower bound of the integer program and usually takes fewer integer iterations. One disadvantage is that PROC LP must recalculate the inverse of the basis matrix at almost every integer iteration; such recalculation is relatively expensive. Another disadvantage is that this strategy does not pay attention to improving the upper bound of the integer program. Thus the number of active nodes tends to grow rapidly if PROC LP cannot quickly find an optimal integer solution.

On the other hand, the LIFO strategy is very efficient and does not need to calculate the inverse of the basis matrix unless the previous node is fathomed. It is a depth-first strategy so it tends to find an integer feasible solution quickly. However, this strategy will pick nodes locally and usually will take longer integer iterations than the best bound strategy.

There is another strategy that is often overlooked. Here it is called the **best upper bound** strategy. With this strategy, each time you select an active node, instead of picking the node with the smallest projected objective value, you select the one with the largest projected objective value. This strategy is as efficient as the LIFO strategy. Moreover, it selects active nodes globally. This strategy tries to improve the upper bound of the integer program by searching for new integer feasible solutions. It also fathoms active nodes quickly and keeps the total number of active nodes below the current level. A disadvantage is that this strategy may evaluate more nodes that do not have any potential in finding an optimal integer solution.

The best bound strategy has the advantage of improving the lower bound. The LIFO strategy has the advantages of efficiency and finding a local integer feasible solution. The best upper bound strategy has the advantages of keeping the size of active nodes under control and at the same time trying to identify any potential integer feasible solution globally.

Although the best bound strategy is generally preferred, in some instances other strategies may be more effective. For example, if you have found an integer optimal solution but you do not know it, you still have to enumerate all possible active nodes. Then the three strategies will basically take the same number of integer iterations after an optimal solution is found but not yet identified. Since the LIFO and best upper bound strategies are very efficient per integer iteration, both will outperform the best bound strategy.

Since no one strategy suits all situations, a hybrid strategy has been developed to increase applicability. The **CONTROL=** option combines the above three strategies naturally and provides a simple control parameter in [0, 1] dealing with different integer programming problems and different solution situations. The **AUTO** option automatically sets and adjusts the **CONTROL=** parameter so that you do not need to know any problem structure or decide a node selection strategy in advance.

Since the LIFO strategy is less costly, you should use it as much as possible in the combinations. The following process is called a **diving process**. Starting from an active node, apply the LIFO strategy as much as you can until the current node becomes feasible or is fathomed, or exceeds a preset limit. During this process, there is no inverse matrix calculation involved except for the first node. When the diving process is over, apply one of the three strategies to select the next starting node. One set of combinations is called a cycle.

The control parameter  $r$  controls the frequency of the three strategies being applied and the depth of the diving process in a cycle. It starts with a pure best bound strategy at  $r=0$ , and then gradually increases the frequency of the diving processes and their depths as  $r$  increases. At  $r=0.5$ , one cycle contains a best bound strategy plus a full diving process. After  $r=0.5$ , the number of the diving processes will gradually increase in a cycle. In addition, the best upper bound strategy will join the cycle. As  $r$  continues to increase, the frequency of the best upper bound strategy will increase. At  $r=1$ , it becomes a pure best upper bound strategy.

The **AUTO** option will automatically adjust the value of the **CONTROL=** option. At the start, it sets **CONTROL=0.7**, which emphasizes finding an upper bound. After an integer feasible solution is found, it sets **CONTROL=0.5**, which emphasizes efficiency and lower bound improvement. When the number of active nodes grows over the default or user defined limit  $m$ , the number indicates that a better upper bound is needed. The **AUTO** option will start to increase the value of **CONTROL=** from 0.5. If the size of the active nodes continues to grow, so will the value of the **CONTROL=** option. When the size of active nodes reaches to the default or user-defined limit  $n$ , **CONTROL=** will be set to 1. At this moment, the growth of active nodes is stopped. When the size of active nodes reduces, **AUTO** will decrease the value of **CONTROL=** option.

You can use other strategies to improve the lower bound by setting **CANSELECT=** to other options.

### Saving and Restoring the List of Active Nodes

The list of active nodes can be saved in a SAS data set for use at a subsequent invocation of PROC LP. The **ACTIVEOUT=** option in the PROC LP statement names the data set into which the current list of active nodes is saved when the procedure terminates due to an error termination condition. Examples of such conditions are time limit exceeded, integer iterations exceeded, and phase 3 iterations exceeded. The **ACTIVEIN=** option in the PROC LP statement names a data set that can be used to initialize the list of active nodes. To achieve the greatest benefit when restarting PROC LP, use the **PRIMALOUT=** and **PRIMALIN=** options in conjunction with the **ACTIVEOUT=** and **ACTIVEIN=** options. See Example 5.10 in the section “Examples: LP Procedure” on page 232 for an illustration.

---

## Sensitivity Analysis

Sensitivity analysis is a technique for examining the effects of changes in model parameters on the optimal solution. The analysis enables you to examine the size of a perturbation to the right-hand-side or objective vector by an arbitrary change vector for which the basis of the current optimal solution remains optimal.

**NOTE:** When sensitivity analysis is performed on integer-constrained problems, the integer variables are fixed at the value they obtained in the integer optimal solution. Therefore, care must be used when interpreting the results of such analyses. Care must also be taken when preprocessing is enabled, because preprocessing usually alters the original formulation.

### Right-Hand-Side Sensitivity Analysis

Consider the problem ( $lpr(\phi)$ ):

$$\begin{aligned} x^{opt}(\phi) = & \min c^T x \\ \text{subject to} \quad & Ax \{ \geq, =, \leq \} b + \phi r \\ & \ell \leq x \leq u \end{aligned}$$

where  $r$  is a right-hand-side change vector.

Let  $x^{opt}(\phi)$  denote an optimal basic feasible solution to  $(lpr(\phi))$ . PROC LP can be used to examine the effects of changes in  $\phi$  on the solution  $x^{opt}(0)$  of problem  $(lpr(0))$ . For the basic solution  $x^{opt}(0)$ , let  $B$  be the matrix composed of the basic columns of  $A$  and let  $N$  be the matrix composed of the nonbasic columns of  $A$ . For the basis matrix  $B$ , the basic components of  $x^{opt}(0)$ , written as  $x^{opt}(0)_B$ , can be expressed as

$$x^{opt}(0)_B = B^{-1}(b - Nx^{opt}(0)_N)$$

Furthermore, because  $x^{opt}(0)$  is feasible,

$$\ell_B \leq B^{-1}(b - Nx^{opt}(0)_N) \leq u_B$$

where  $\ell_B$  is a column vector of the lower bounds on the structural basic variables, and  $u_B$  is a column vector of the upper bounds on the structural basic variables. For each right-hand-side change vector  $r$  identified in the RHSEN statement, PROC LP finds an interval  $[\phi_{min}, \phi_{max}]$  such that

$$\ell_B \leq B^{-1}(b + \phi r - Nx^{opt}(0)_N) \leq u_B$$

for  $\phi \in [\phi_{min}, \phi_{max}]$ . Furthermore, because changes in the right-hand side do not affect the reduced costs, for  $\phi \in [\phi_{min}, \phi_{max}]$ ,

$$x^{opt}(\phi)^T = ((B^{-1}(b + \phi r - Nx^{opt}(0)_N))^T, x^{opt}(0)_N^T)$$

is optimal in  $(lpr(\phi))$ .

For  $\phi = \phi_{min}$  and  $\phi = \phi_{max}$ , PROC LP reports the following:

- the names of the leaving variables
- the value of the optimal objective in the modified problems
- the RHS values in the modified problems
- the solution status, reduced costs and activities in the modified problems

The leaving variable identifies the basic variable  $x_i$  that first reaches either the lower bound  $\ell_i$  or the upper bound  $u_i$  as  $\phi$  reaches  $\phi_{min}$  or  $\phi_{max}$ . This is the basic variable that would leave the basis to maintain primal feasibility. Multiple RHSEN variables can appear in a problem data set.

## Price Sensitivity Analysis

Consider the problem  $(lpp(\phi))$ :

$$\begin{aligned} x^{opt}(\phi) = & \min(c + \phi r)^T x \\ \text{subject to } & Ax \{ \geq, =, \leq \} b \\ & \ell \leq x \leq u \end{aligned}$$

where  $r$  is a price change vector.

Let  $x^{opt}(\phi)$  denote an optimal basic feasible solution to  $(lpp(\phi))$ . PROC LP can be used to examine the effects of changes in  $\phi$  on the solution  $x^{opt}(0)$  of problem  $(lpp(0))$ . For the basic solution  $x^{opt}(0)$ , let  $B$  be

the matrix composed of the basic columns of  $A$  and let  $N$  be the matrix composed of the nonbasic columns of  $A$ . For basis matrix  $B$ , the reduced cost associated with the  $i$ th variable can be written as

$$rc_i(\phi) = ((c + \phi r)_N^T - (c + \phi r)_B^T B^{-1} N)_i$$

where  $(c + \phi r)_N$  and  $(c + \phi r)_B$  is a partition of the vector of price coefficients into nonbasic and basic components. Because  $x^{opt}(0)$  is optimal in  $(lpp(0))$ , the reduced costs satisfy

$$rc_i(\phi) \geq 0$$

if the nonbasic variable in column  $i$  is at its lower bound, and

$$rc_i(\phi) \leq 0$$

if the nonbasic variable in column  $i$  is at its upper bound.

For each price coefficient change vector  $r$  identified with the keyword PRICESEN in the **TYPE** variable, PROC LP finds an interval  $[\phi_{min}, \phi_{max}]$  such that for  $\phi \in [\phi_{min}, \phi_{max}]$ ,

$$rc_i(\phi) \geq 0$$

if the nonbasic variable in column  $i$  is at its lower bound, and

$$rc_i(\phi) \leq 0$$

if the nonbasic variable in column  $i$  is at its upper bound. Because changes in the price coefficients do not affect feasibility, for  $\phi \in [\phi_{min}, \phi_{max}]$ ,  $x^{opt}(\phi)$  is optimal in  $(lpp(\phi))$ . For  $\phi = \phi_{min}$  and  $\phi = \phi_{max}$ , PROC LP reports the following:

- the names of entering variables
- the value of the optimal objective in the modified problems
- the price coefficients in the modified problems
- the solution status, reduced costs, and activities in the modified problems

The entering variable identifies the variable whose reduced cost first goes to zero as  $\phi$  reaches  $\phi_{min}$  or  $\phi_{max}$ . This is the nonbasic variable that would enter the basis to maintain optimality (dual feasibility). Multiple PRICESEN variables may appear in a problem data set.

---

## Range Analysis

Range analysis is sensitivity analysis for specific change vectors. As with the sensitivity analysis case, care must be used in interpreting the results of range analysis when the problem has integers or the preprocessing option is enabled.

## Right-Hand-Side Range Analysis

The effects on the optimal solution of changes in each right-hand-side value can be studied using the **RANGERHS** option in the **PROC LP** or **RESET** statement. This option results in sensitivity analysis for the  $m$  right-hand-side change vectors specified by the columns of the  $m \times m$  identity matrix.

## Price Range Analysis

The effects on the optimal solution of changes in each price coefficient can be studied using the **RANGEPRICE** option in the **PROC LP** or **RESET** statement. This option results in sensitivity analysis for the  $n$  price change vectors specified by the rows of the  $n \times n$  identity matrix.

---

## Parametric Programming

Sensitivity analysis and range analysis examine how the optimal solution behaves with respect to perturbations of model parameter values. These approaches assume that the basis at optimality is not allowed to change. When greater flexibility is desired and a change of basis is acceptable, parametric programming can be used.

As with the sensitivity analysis case, care must be used in interpreting the results of parametric programming when the problem has integers or the preprocessing option is enabled.

## Right-Hand-Side Parametric Programming

As discussed in the section “**Right-Hand-Side Sensitivity Analysis**” on page 217, for each right-hand-side change vector  $r$ , **PROC LP** finds an interval  $[\phi_{min}, \phi_{max}]$  such that for  $\phi \in [\phi_{min}, \phi_{max}]$ ,

$$x^{opt}(\phi)^T = ((B^{-1}(b + \phi r - Nx^{opt}(0)_N))^T, x^{opt}(0)_N^T)$$

is optimal in  $(lpr(\phi))$  for the fixed basis  $B$ . Leaving variables that inhibit further changes in  $\phi$  without a change in the basis  $B$  are associated with the quantities  $\phi_{min}$  and  $\phi_{max}$ . By specifying **RHSPHI**= $\Phi$  in either the **PROC LP** statement or the **RESET** statement, you can examine the solution  $x^{opt}(\phi)$  as  $\phi$  increases or decreases from 0 to  $\Phi$ .

When **RHSPHI**= $\Phi$  is specified, the procedure first finds the interval  $[\phi_{min}, \phi_{max}]$  as described previously. Then, if  $\Phi \in [\phi_{min}, \phi_{max}]$ , no further investigation is needed. However, if  $\Phi > \phi_{max}$  or  $\Phi < \phi_{min}$ , then the procedure attempts to solve the new problem  $(lpr(\Phi))$ . To accomplish this, it pivots the leaving variable out of the basis while maintaining dual feasibility. If this new solution is primal feasible in  $(lpr(\Phi))$ , no further investigation is needed; otherwise, the procedure identifies the new leaving variable and pivots it out of the basis, again maintaining dual feasibility. Dual pivoting continues in this manner until a solution that is primal feasible in  $(lpr(\Phi))$  is identified. Because dual feasibility is maintained at each pivot, the  $(lpr(\Phi))$  primal feasible solution is optimal.

At each pivot, the procedure reports on the variables that enter and leave the basis, the current range of  $\phi$ , and the objective value. When  $x^{opt}(\Phi)$  is found, it is displayed. If you want the solution  $x^{opt}(\phi)$  at each pivot, then specify the **PARAPRINT** option in either the **PROC LP** or the **RESET** statement.

## Price Parametric Programming

As discussed in the section “[Price Sensitivity Analysis](#)” on page 218, for each price change vector  $r$ , PROC LP finds an interval  $[\phi_{min}, \phi_{max}]$  such that for each  $\phi \in [\phi_{min}, \phi_{max}]$ ,

$$rc_i(\phi) = ((c + \phi r)_N^T - (c + \phi r)_B^T B^{-1} N)_i$$

satisfies the conditions for optimality in  $(lpp(\phi))$  for the fixed basis  $B$ . Entering variables that inhibit further changes in  $\phi$  without a change in the basis  $B$  are associated with the quantities  $\phi_{min}$  and  $\phi_{max}$ . By specifying [PRICEPHI](#)= $\Phi$  in either the PROC LP statement or the [RESET](#) statement, you can examine the solution  $x^{opt}(\phi)$  as  $\phi$  increases or decreases from 0 to  $\Phi$ .

When [PRICEPHI](#)= $\Phi$  is specified, the procedure first finds the interval  $[\phi_{min}, \phi_{max}]$ , as described previously. Then, if  $\Phi \in [\phi_{min}, \phi_{max}]$ , no further investigation is needed. However, if  $\Phi > \phi_{max}$  or  $\Phi < \phi_{min}$ , the procedure attempts to solve the new problem  $(lpp(\Phi))$ . To accomplish this, it pivots the entering variable into the basis while maintaining primal feasibility. If this new solution is dual feasible in  $(lpp(\Phi))$ , no further investigation is needed; otherwise, the procedure identifies the new entering variable and pivots it into the basis, again maintaining primal feasibility. Pivoting continues in this manner until a solution that is dual feasible in  $(lpp(\Phi))$  is identified. Because primal feasibility is maintained at each pivot, the  $(lpp(\Phi))$  dual feasible solution is optimal.

At each pivot, the procedure reports on the variables that enter and leave the basis, the current range of  $\phi$ , and the objective value. When  $x^{opt}(\Phi)$  is found, it is displayed. If you want the solution  $x^{opt}(\phi)$  at each pivot, then specify the [PARAPRINT](#) option in either the [PROC LP](#) or the [RESET](#) statement.

---

## Interactive Facilities

The interactive features of the LP procedure enable you to examine intermediate results, perform sensitivity analysis, parametric programming, and range analysis, and control the solution process.

### Controlling Interactive Features

You can gain control of the LP procedure for interactive processing by setting a breakpoint or pressing the CTRL-BREAK key combination, or when certain error conditions are encountered:

- when a feasible solution is found
- at each pivot of the simplex algorithm
- when an integer feasible solution is found
- at each integer pivot of the branch-and-bound algorithm
- after the data are read but before iteration begins
- after at least one integer feasible solution has been found which is within desirable proximity of optimality
- after the problem has been solved but before results are displayed



When the LP procedure pauses, you can enter any of the interactive statements **RESET**, **PIVOT**, **IPIVOT**, **PRINT**, **SHOW**, **QUIT**, and **RUN**.

Breakpoints are set using the **FEASIBLEPAUSE**, **PAUSE=**, **IFEASIBLEPAUSE=**, **IPAUSE=**, **PROXIMITYPAUSE=**, **READPAUSE**, and **ENDPAUSE** options. The LP procedure displays a message on the SAS log when it gives you control because of encountering one of these breakpoints.

During phase 1, 2, or 3, the CTRL-BREAK key pauses the LP procedure and releases the control at the beginning of the next iteration.

The error conditions, which usually cause the LP procedure to pause, include time limit exceeded, phase 1 iterations exceeded, phase 2 iterations exceeded, phase 3 iterations exceeded, and integer iterations exceeded. You can use the **RESET** statement to reset the option that caused the error condition.

The **PIVOT** and **IPIVOT** statements result in control being returned to you after a single simplex algorithm pivot and an integer pivot. The **PRINT** and **SHOW** statements display current solution information and return control to you. On the other hand, the **QUIT** statement requests that you leave the LP procedure immediately. If you want to quit but save output data sets, then type **QUIT/SAVE**. The **RUN** statement requests the LP procedure to continue its execution immediately.

## Displaying Intermediate Results

Once you have control of the procedure, you can examine the current values of the options and the status of the problem being solved using the **SHOW** statement. All displaying done by the **SHOW** statement goes to the SAS log.

Details about the current status of the solution are obtained using the **PRINT** statement. The various display options enable you to examine parts of the variable and constraint summaries, display the current tableau, perform sensitivity analysis on the current solution, and perform range analysis.

## Interactive Facilities in Batch Mode

All of the interactive statements can be used when processing in batch mode. This is particularly convenient when the interactive facilities are used to combine different search strategies in solving integer problems.

## Sensitivity Analysis

Two features that enhance the ability to perform sensitivity analysis need further explanation. When you specify **/SENSITIVITY** in a **PRINT COLUMN(colnames)** statement, the LP procedure defines a new change row to use in sensitivity analysis and parametric programming. This new change row has a +1 entry for each variable listed in the **PRINT** statement. This enables you to define new change rows interactively.

When you specify **/SENSITIVITY** in a **PRINT ROW(rownames)** statement, the LP procedure defines a new change column to use in sensitivity analysis and parametric programming. This new change column has a +1 entry for each right-hand-side coefficient listed in the **PRINT** statement. This enables you to define new change columns interactively.

In addition, you can interactively change the **RHSPHI=** and **PRICEPHI=** options using the **RESET** statement. This enables you to perform parametric programming interactively.



## Memory Management

There are no restrictions on the problem size in the LP procedure. The number of constraints and variables in a problem that PROC LP can solve depends on the host platform, the available memory, and the available disk space for utility data sets.

Memory usage is affected by a great many factors including the density of the technological coefficient matrix, the model structure, and the density of the decomposed basis matrix. The algorithm requires that the decomposed basis fit completely in memory. Any additional memory is used for nonbasic columns. The partition between the decomposed basis and the nonbasic columns is dynamic so that as the inverse grows, which typically happens as iterations proceed, more memory is available to it and less is available for the nonbasic columns.

The LP procedure determines the initial size of the decomposed basis matrix. If the area used is too small, PROC LP must spend time compressing this matrix, which degrades performance. If PROC LP must compress the decomposed basis matrix on the average more than 15 times per iteration, then the size of the memory devoted to the basis is increased. If the work area cannot be made large enough to invert the basis, an error return occurs. On the other hand, if PROC LP compresses the decomposed basis matrix on the average once every other iteration, then memory devoted to the decomposed basis is decreased, freeing memory for the nonbasic columns.

For many models, memory constraints are not a problem because both the decomposed basis and all the nonbasic columns will have no problem fitting. However, when the models become large relative to the available memory, the algorithm tries to adjust memory distribution in order to solve the problem. In the worst cases, only one nonbasic column fits in memory with the decomposed basis matrix.

Problems involving memory use can occur when solving mixed-integer problems. Data associated with each node in the branch-and-bound tree must be kept in memory. As the tree grows, competition for memory by the decomposed basis, the nonbasic columns, and the branch-and-bound tree may become critical. If the situation becomes critical, the procedure automatically switches to branching strategies that use less memory. However, it is possible to reach a point where no further processing is possible. In this case, PROC LP terminates on a memory error.

## Output Data Sets

The LP procedure can optionally produce five output data sets. These are the **ACTIVEOUT=**, **PRIMALOUT=**, **DUALOUT=**, **TABLEAUOUT=**, and **MPSOUT=** data sets. Each contains two variables that identify the particular problem in the input data set. These variables are

|                 |                                          |
|-----------------|------------------------------------------|
| <b>_OBJ_ID_</b> | identifies the objective function ID.    |
| <b>_RHS_ID_</b> | identifies the right-hand-side variable. |

Additionally, each data set contains other variables, which are discussed below.

## ACTIVEOUT= Data Set

The **ACTIVEOUT=** data set contains a representation of the current active branch-and-bound tree. You can use this data set to initialize the branch-and-bound tree to continue iterations on an incompletely solved problem. Each active node in the tree generates two observations in this data set. The first is a 'LOWERBD' observation that is used to reconstruct the lower-bound constraints on the currently described active node. The second is an 'UPPERBD' observation that is used to reconstruct the upper-bound constraints on the currently described active node. In addition to these, an observation that describes the current best integer solution is included. The data set contains the following variables:

|                 |                                                                                                                                 |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>_STATUS_</b> | contains the keywords LOWERBD, UPPERBD, and INTBEST for identifying the type of observation.                                    |
| <b>_PROB_</b>   | contains the problem number for the current observation.                                                                        |
| <b>_OBJECT_</b> | contains the objective value of the parent problem that generated the current observation's problem.                            |
| <b>_SINFEA_</b> | contains the sum of the integer infeasibilities of the current observation's problem.                                           |
| <b>_PROJEC_</b> | contains the data needed for <b>CANSELECT=PROJECT</b> when the branch-and-bound tree is read using the <b>ACTIVEIN=</b> option. |
| <b>_PSEUDO_</b> | contains the data needed for <b>CANSELECT=PSEUDOC</b> when the branch-and-bound tree is read using the <b>ACTIVEIN=</b> option. |

**INTEGER VARIABLES** Integer-constrained structural variables are also included in the **ACTIVEOUT=** data set. For each observation, these variables contain values for defining the active node in the branch-and-bound tree.

## PRIMALOUT= Data Set

The **PRIMALOUT=** data set contains the current primal solution. If the problem has integer-constrained variables, the **PRIMALOUT=** data set contains the current best integer feasible solution. If none have been found, the **PRIMALOUT=** data set contains the relaxed solution. In addition to **\_OBJ\_ID\_** and **\_RHS\_ID\_**, the data set contains the following variables:

|                 |                                                                                                                                                                                                                                                                                                                  |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>_VAR_</b>    | identifies the variable name.                                                                                                                                                                                                                                                                                    |
| <b>_TYPE_</b>   | identifies the type of the variable as specified in the input data set. Artificial variables are labeled as type 'ARTIFCL'.                                                                                                                                                                                      |
| <b>_STATUS_</b> | identifies whether the variable is basic, nonbasic, or at an upper bound in the current solution.                                                                                                                                                                                                                |
| <b>_LBOUND_</b> | contains the input lower bound on the variable unless the variable is integer-constrained and an integer solution is given. In this case, <b>_LBOUND_</b> contains the lower bound on the variable needed to realize the integer solution on subsequent calls to PROC LP when using the <b>PRIMALIN=</b> option. |
| <b>_VALUE_</b>  | identifies the value of the variable in the current solution or the current best integer feasible solution.                                                                                                                                                                                                      |
| <b>_UBOUND_</b> | contains the input upper bound on the variable unless the variable is integer-constrained and an integer solution is given. In this case, <b>_UBOUND_</b> contains the upper bound on                                                                                                                            |

the variable needed to realize the integer solution on subsequent calls to PROC LP when using the **PRIMALIN=** option.

|                 |                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>_PRICE_</b>  | contains the input price coefficient of the variable.                                                                                                                                                                                                                                                                                                                          |
| <b>_R_COST_</b> | identifies the value of the reduced cost in the current solution. <a href="#">Example 5.3</a> in the section “ <a href="#">Examples: LP Procedure</a> ” on page 232 shows a typical <b>PRIMALOUT=</b> data set. Note that it is necessary to include the information on objective function and right-hand side in order to distinguish problems in multiple problem data sets. |

### DUALOUT= Data Set

The **DUALOUT=** data set contains the dual solution for the current solution. If the problem has integer-constrained variables, the **DUALOUT=** data set contains the dual for the current best integer solution, if any. Otherwise it contains the dual for the relaxed solution. In addition to **\_OBJ\_ID\_** and **\_RHS\_ID\_**, it contains the following variables:

|                 |                                                                                                                                                                       |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>_ROW_ID_</b> | identifies the row or constraint name.                                                                                                                                |
| <b>_TYPE_</b>   | identifies the type of the row as specified in the input data set.                                                                                                    |
| <b>_RHS_</b>    | gives the value of the right-hand side on input.                                                                                                                      |
| <b>_L_RHS_</b>  | gives the lower bound for the row evaluated from the input right-hand-side value, the <b>TYPE</b> of the row, and the value of the <b>RANGE</b> variable for the row. |
| <b>_VALUE_</b>  | gives the value of the row, at optimality, excluding logical variables.                                                                                               |
| <b>_U_RHS_</b>  | gives the upper bound for the row evaluated from the input right-hand-side value, the <b>TYPE</b> of the row, and the value of the <b>RANGE</b> variable for the row. |
| <b>_DUAL_</b>   | gives the value of the dual variable associated with the row.                                                                                                         |

### TABLEAUOUT= Data Set

The **TABLEAUOUT=** data set contains the current tableau. Each observation, except for the first, corresponds to a basic variable in the solution. The observation labeled **R\_COSTS** contains the reduced costs  $c_N^T - c_B^T B^{-1} N$ . In addition to **\_OBJ\_ID\_** and **\_RHS\_ID\_**, it contains the following variables:

|                             |                                                                          |
|-----------------------------|--------------------------------------------------------------------------|
| <b>_BASIC_</b>              | gives the names of the basic variables in the solution.                  |
| <b>INVB_R</b>               | gives the values of $B^{-1}r$ , where $r$ is the right-hand-side vector. |
| <b>STRUCTURAL VARIABLES</b> | give the values in the tableau, namely $B^{-1}A$ .                       |

### MPSOUT= Data Set

The **MPSOUT=** data set contains problem data converted from a PROC LP format into an MPS-format SAS data set. The six fields, **FIELD1** to **FIELD6**, in the **MPSOUT=** data set correspond to the six columns in MPS standard. For more information about the MPS-format SAS data set, see Chapter 17, “The MPS-Format SAS Data Set” (*SAS/OR User’s Guide: Mathematical Programming*).

---

## Input Data Sets

In addition to the **DATA=** input data set, PROC LP recognizes the **ACTIVEIN=** and the **PRIMALIN=** data sets.

### ACTIVEIN= Data Set

The **ACTIVEIN=** data set contains a representation of the current active tree. The format is identical to that of the **ACTIVEOUT=** data set.

### PRIMALIN= Data Set

The format of the **PRIMALIN=** data set is identical to the **PRIMALOUT=** data set. PROC LP uses the **PRIMALIN=** data set to identify variables at their upper bounds in the current solution and variables that are basic in the current solution.

You can add observations to the end of the problem data set if they define cost (right-hand-side) sensitivity change vectors and have **PRICESEN** (**RHSEN**) types. This enables you to solve a problem, save the solution in a SAS data set, and perform sensitivity analysis later. You can also use the **PRIMALIN=** data set to restart problems that have not been completely solved or to which new variables have been added.

---

## Displayed Output

The output from the LP procedure is discussed in the following six sections:

- Problem Summary
- Solution Summary including a Variable Summary and a Constraint Summary
- Infeasible Information Summary
- RHS Sensitivity Analysis Summary (the RHS Range Analysis Summary is not discussed)
- Price Sensitivity Analysis Summary (the Price Range Analysis Summary is not discussed)
- Iteration Log

For integer-constrained problems, the procedure also displays an **Integer Iteration Log**. The description of this Log can be found in the section “**Integer Programming**” on page 210. When you request that the tableau be displayed, the procedure displays the Current Tableau. The description of this can be found in the section “**The Reduced Costs, Dual Activities, and Current Tableau**” on page 206.

A problem data set can contain a set of constraints with several right-hand sides and several objective functions. PROC LP considers each combination of right-hand side and objective function as defining a new linear programming problem and solves each, performing all specified sensitivity analysis on each problem. For each problem defined, PROC LP displays a new sequence of output sections. **Example 5.1** in the section “**Examples: LP Procedure**” on page 232 discusses each of these elements.

The LP procedure produces the following displayed output by default.

## The Problem Summary

The problem summary includes the

- type of optimization and the name of the objective row (as identified by the **ID** or **ROW** variable)
- name of the SAS variable that contains the right-hand-side constants
- name of the SAS variable that contains the type keywords
- density of the coefficient matrix (the ratio of the number of nonzero elements to the number of total elements) after the slack and surplus variables have been appended
- number of each type of variable in the mathematical program
- number of each type of constraint in the mathematical program

## The Solution Summary

The solution summary includes the

- termination status of the procedure
- objective value of the current solution
- number of phase 1 iterations that were completed
- number of phase 2 iterations that were completed
- number of phase 3 iterations that were completed
- number of integer iterations that were completed
- number of integer feasible solutions that were found
- number of initial basic feasible variables identified
- time used in solving the problem excluding reading the data and displaying the solution
- number of inversions of the basis matrix
- current value of several of the options

## The Variable Summary

The variable summary includes the

- column number associated with each structural or logical variable in the problem
- name of each structural or logical variable in the problem. (PROC LP gives the logical variables the name of the constraint **ID**. If no **ID** variable is specified, the procedure names the logical variable `_OBS $n$ _`, where  $n$  is the observation that describes the constraint.)

- variable's status in the current solution. The status can be BASIC, DEGEN, ALTER, blank, LOWBD, or UPPBD, depending upon whether the variable is a basic variable, a degenerate variable (that is, a basic variable whose activity is at its input lower bound), a nonbasic variable that can be brought into the basis to define an alternate optimal solution, a nonbasic variable at its default lower bound 0, a nonbasic variable at its lower bound, or a nonbasic variable at its upper bound.
- type of variable (whether it is logical or structural, and, if structural, its bound type, or other value restriction). See [Example 5.1](#) for a list of possible types in the variable summary.
- value of the objective coefficient associated with each variable
- activity of the variable in the current solution
- variable's reduced cost in the current solution

### The Constraint Summary

The constraint summary includes the

- constraint row number and its [ID](#)
- kind of constraint (whether it is an OBJECTIVE, LE, EQ, GE, RANGELE, RANGEEQ, RANGEGE, or FREE row)
- number of the slack or surplus variable associated with the constraint row
- value of the right-hand-side constant associated with the constraint row
- current activity of the row (excluding logical variables)
- current activity of the dual variable (shadow price) associated with the constraint row

### The Infeasible Information Summary

The infeasible information summary includes the

- name of the infeasible row or variable
- current activity for the row or variable
- type of the row or variable
- value of right-hand-side constant
- name of each nonzero and nonmissing variable in the row
- activity and upper and lower bounds for the variable

## The RHS Sensitivity Analysis Summary

The RHS sensitivity analysis summary includes the

- value of  $\phi_{min}$
- leaving variable when  $\phi = \phi_{min}$
- objective value when  $\phi = \phi_{min}$
- value of  $\phi_{max}$
- leaving variable when  $\phi = \phi_{max}$
- objective value when  $\phi = \phi_{max}$
- column number and name of each logical and structural variable
- variable's status when  $\phi \in [\phi_{min}, \phi_{max}]$
- variable's reduced cost when  $\phi \in [\phi_{min}, \phi_{max}]$
- value of right-hand-side constant when  $\phi = \phi_{min}$
- activity of the variable when  $\phi = \phi_{min}$
- value of right-hand-side constant when  $\phi = \phi_{max}$
- activity of the variable when  $\phi = \phi_{max}$

## The Price Sensitivity Analysis Summary

The price sensitivity analysis summary includes the

- value of  $\phi_{min}$
- entering variable when  $\phi = \phi_{min}$
- objective value when  $\phi = \phi_{min}$
- value of  $\phi_{max}$
- entering variable when  $\phi = \phi_{max}$
- objective value when  $\phi = \phi_{max}$
- column number and name of each logical and structural variable
- variable's status when  $\phi \in [\phi_{min}, \phi_{max}]$
- activity of the variable when  $\phi \in [\phi_{min}, \phi_{max}]$
- price of the variable when  $\phi = \phi_{min}$
- variable's reduced cost when  $\phi = \phi_{min}$
- price of the variable when  $\phi = \phi_{max}$
- variable's reduced cost when  $\phi = \phi_{max}$

## The Iteration Log

The iteration log includes the

- phase number
- iteration number in each phase
- name of the leaving variable
- name of the entering variable
- variable's reduced cost
- objective value

## ODS Table and Variable Names

PROC LP assigns a name to each table it creates. You can use these names to select output tables when using the Output Delivery System (ODS).

**Table 5.9** ODS Tables Produced in PROC LP

| Table Name              | Description                                       | Statement/Option                                     |
|-------------------------|---------------------------------------------------|------------------------------------------------------|
| ProblemSummary          | Problem summary                                   | Default                                              |
| SolutionSummary         | Solution summary                                  | Default                                              |
| VariableSummary         | Variable summary                                  | Default                                              |
| ConstraintSummary       | Constraint summary                                | Default                                              |
| IterationLog            | Iteration log                                     | FLOW                                                 |
| IntegerIterationLog     | Integer iteration log                             | Default                                              |
| PriceSensitivitySummary | Price sensitivity analysis summary                | Default, PRINT PRICESEN, or PRINT COLUMN/SENSITIVITY |
| PriceActivities         | Price activities at $\phi_{min}$ and $\phi_{max}$ | Default, PRINT PRICESEN, or PRINT COLUMN/SENSITIVITY |
| PriceActivity           | Price activity at $\phi_{min}$ or $\phi_{max}$    | PRICEPHI= and PARAPRINT                              |
| PriceParametricLog      | Price parametric programming log                  | PRICEPHI=                                            |
| PriceRangeSummary       | Price range analysis                              | RANGEPRICE or PRINT RANGEPRICE                       |
| RhsSensitivitySummary   | RHS sensitivity analysis summary                  | Default, PRINT RHSEN, or PRINT ROW/SENSITIVITY       |
| RhsActivities           | RHS activities at $\phi_{min}$ and $\phi_{max}$   | Default, PRINT RHSEN, or PRINT ROW/SENSITIVITY       |
| RhsActivity             | RHS activity at $\phi_{min}$ or $\phi_{max}$      | RHSPHI= and PARAPRINT                                |
| RhsParametricLog        | RHS parametric programming log                    | RHSPHI=                                              |
| RhsRangeSummary         | RHS range analysis                                | RANGERHS or PRINT RANGERHS                           |
| InfeasibilitySummary    | Infeasible row or variable summary                | Default                                              |



**Table 5.9** (continued)

| Table Name            | Description                            | Statement/Option              |
|-----------------------|----------------------------------------|-------------------------------|
| InfeasibilityActivity | Variable activity in an infeasible row | Default                       |
| CurrentTableau        | Current tableau                        | TABLEAUPRINT or PRINT TABLEAU |
| Matrix                | Technological matrix                   | PRINT MATRIX                  |
| MatrixPicture         | Technological matrix picture           | PRINT MATRIX/PICTURE          |
| MatrixPictureLegend   | Technological matrix picture legend    | PRINT MATRIX/PICTURE          |

The following table lists the variable names of the preceding tables used in the ODS template of the LP procedure.

**Table 5.10** Variable Names for the ODS Tables Produced in PROC LP

| Table Name            | Variables                                                                               |
|-----------------------|-----------------------------------------------------------------------------------------|
| VariableSummary       | VarName, Status, Type, Price, Activity, ReducedCost                                     |
| ConstraintSummary     | Row, RowName, Type, SSCol, Rhs, Activity, Dual                                          |
| IterationLog          | Phase, Iteration, EnterVar, EnterCol, LeaveVar, LeaveCol, ReducedCost, ObjValue         |
| IntegerIterationLog   | Iteration, Problem, Condition, Objective, Branch, Value, SumOfInfeas, Active, Proximity |
| PriceActivities       | Col, VarName, Status, Activity, MinPrice, MinReducedCost, MaxPrice, MaxReducedCost      |
| PriceActivity         | Col, VarName, Status, Activity, Price, ReducedCost                                      |
| PriceParametricLog    | LeaveVar, LeaveCol, EnterVar, EnterCol, ObjValue, CurrentPhi                            |
| PriceRangeSummary     | Col, VarName, MinPrice, MinEnterVar, MinObj, MaxPrice, MaxEnterVar, MaxObj              |
| RhsActivities         | Col, VarName, Status, ReducedCost, MinRhs, MinActivity, MaxRhs, MaxActivity             |
| RhsActivity           | Col, VarName, Status, ReducedCost, Rhs, Activity,                                       |
| RhsParametricLog      | LeaveVar, LeaveCol, EnterVar, EnterCol, ObjValue, CurrentPhi                            |
| RhsRangeSummary       | RowName, MinRhs, MinLeaveVar, MinObj, MaxRhs, MaxLeaveVar, MaxObj                       |
| InfeasibilityActivity | VarName, Coefficient, Activity, Lower, Upper                                            |

## Memory Limit

The system option MEMSIZE sets a limit on the amount of memory used by the SAS System. If you do not specify a value for this option, then the SAS System sets a default memory limit. Your operating environment determines the actual size of the default memory limit, which is sufficient for many applications. However, to solve most realistic optimization problems, the LP procedure might require more memory. Increasing the memory limit can reduce the chance of an out-of-memory condition.

**NOTE:** The MEMSIZE system option is not available in some operating environments. See the documentation for your operating environment for more information.

You can specify -MEMSIZE 0 to indicate all available memory should be used, but this setting should be used with caution. In most operating environments, it is better to specify an adequate amount of memory than to specify -MEMSIZE 0. For example, if you are running PROC OPTLP to solve LP problems with only a few hundred thousand variables and constraints, -MEMSIZE 500M might be sufficient to allow the procedure to run without an out-of-memory condition. When problems have millions of variables, -MEMSIZE 1000M or higher might be needed. These are “rules of thumb”—problems with atypical structure, density, or other characteristics can increase the optimizer’s memory requirements.

The MEMSIZE option can be specified at system invocation, on the SAS command line, or in a configuration file. The syntax is described in the *SAS Companion* book for your operating system.

To report a procedure’s memory consumption, you can use the FULLSTIMER option. The syntax is described in the *SAS Companion* book for your operating system.

---

## Examples: LP Procedure

The following examples illustrate some of the capabilities of PROC LP. These examples, together with the other SAS/OR examples, can be found in the SAS sample library. A description of the features of PROC LP as shown in the examples are

- [Example 5.1](#) shows dense input format.
- [Example 5.2](#) shows sparse input format.
- [Example 5.3](#) uses the [RANGEPRICE](#) option to show you the range over which each objective coefficient can vary without changing the variables in the basis.
- [Example 5.4](#) shows more sensitivity analysis and restarting a problem.
- [Example 5.5](#) shows parametric programming.
- [Example 5.6](#) shows special ordered sets.
- [Example 5.7](#) shows goal programming.
- [Example 5.8](#) shows integer programming.
- [Example 5.9](#) shows an infeasible problem.
- [Example 5.10](#) shows restarting integer programs.
- [Example 5.11](#) controls the search of the branch-and-bound tree.
- [Example 5.12](#) shows matrix generation and report writing for an assignment problem.
- [Example 5.13](#) shows matrix generation and report writing for a scheduling problem.
- [Example 5.14](#) shows a multicommodity transshipment problem.

- Example 5.15 shows migration to PROC OPTLP via the `MPSOUT=` option.
- Example 5.16 shows migration to PROC OPTMODEL.
- Example 5.17 shows migration to PROC OPTMODEL.

## Example 5.1: An Oil Blending Problem

The blending problem presented in the introduction is a good example for demonstrating some of the features of the LP procedure. Recall that a step in refining crude oil into finished oil products involves a distillation process that splits crude into various streams. Suppose that there are three types of crude available: Arabian light, Arabian heavy, and Brega. These are distilled into light naphtha, intermediate naphtha, and heating oil. Using one of two recipes, these in turn are blended into jet fuel.

Assume that you can sell as much fuel as is produced. What production strategy maximizes the profit from jet fuel sales? The following SAS code demonstrates a way of answering this question using linear programming. The SAS data set is a representation of the formulation for this model given in the introductory section.

```
data;
  input _row_ $17.
        a_light a_heavy brega naphthal naphthai heatingo jet_1
        jet_2 _type_ $ _rhs_;
  datalines;
profit          -175 -165 -205   0  0  0  300  300  max      .
naphtha_1_conv   .035 .030 .045  -1  0  0   0   0   eq      0
naphtha_i_conv   .100 .075 .135   0 -1  0   0   0   eq      0
heating_o_conv   .390 .300 .430   0  0 -1   0   0   eq      0
recipe_1         0    0    0    0 .3 .7  -1   0   eq      0
recipe_2         0    0    0    .2 0 .8   0  -1   eq      0
available        110  165   80   .  .  .   .   . upperbd .
;
```

The `_ROW_` variable contains the names of the rows in the model; the variables `A_LIGHT` to `JET_2` are the names of the structural variables in the model; the `_TYPE_` variable contains the keywords that tell the LP procedure how to interpret each row in the model; and the `_RHS_` variable gives the value of the right-hand-side constants.

The structural variables are interpreted as the quantity of each type of constituent or finished product. For example, the value of `A_HEAVY` in the solution is the amount of Arabian heavy crude to buy while the value of `JET_1` in the solution is the amount of recipe 1 jet fuel that is produced. As discussed previously, the values given in the model data set are the technological coefficients whose interpretation depends on the model. In this example, the coefficient -175 in the `PROFIT` row for the variable `A_LIGHT` gives a cost coefficient (because the row with `_ROW_=PROFIT` has `_TYPE_=MAX`) for the structural variable `A_LIGHT`. This means that for each unit of Arabian heavy crude purchased, a cost of 175 units is incurred.

The coefficients 0.035, 0.100, and 0.390 for the `A_LIGHT` variable give the percentages of each unit of Arabian light crude that is distilled into the light naphtha, intermediate naphtha, and heating oil components. The 110 value in the row `_ROW_=AVAILABLE` gives the quantity of Arabian light that is available.

PROC LP produces the following [Problem Summary](#) output. Included in the summary is an identification of the objective, defined by the first observation of the problem data set; the right-hand-side variable, defined by the variable `_RHS_`; and the type identifier, defined by the variable `_TYPE_`. See [Output 5.1.1](#).

**Output 5.1.1** Problem Summary for the Oil Blending Problem

| The LP Procedure    |            |
|---------------------|------------|
| Problem Summary     |            |
| Objective Function  | Max profit |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 45.00      |
|                     |            |
| Variables           | Number     |
| Non-negative        | 5          |
| Upper Bounded       | 3          |
| Total               | 8          |
|                     |            |
| Constraints         | Number     |
| EQ                  | 5          |
| Objective           | 1          |
| Total               | 6          |

The next section of output ([Output 5.1.2](#)) contains the [Solution Summary](#), which indicates whether or not an optimal solution was found. In this example, the procedure terminates successfully (with an optimal solution), with 1544 as the value of the objective function. Also included in this section of output is the number of phase 1 and phase 2 iterations, the number of variables used in the initial basic feasible solution, and the time used to solve the problem. For several options specified in the PROC LP statement, the current option values are also displayed.

**Output 5.1.2** Solution Summary for the Oil Blending Problem**The LP Procedure**

| Solution Summary                 |              |
|----------------------------------|--------------|
| Terminated Successfully          |              |
| Objective Value                  | 1544         |
| Phase 1 Iterations               | 0            |
| Phase 2 Iterations               | 4            |
| Phase 3 Iterations               | 0            |
| Integer Iterations               | 0            |
| Integer Solutions                | 0            |
| Initial Basic Feasible Variables | 5            |
| Time Used (seconds)              | 0            |
| Number of Inversions             | 3            |
| Epsilon                          | 1E-8         |
| Infinity                         | 1.797693E308 |
| Maximum Phase 1 Iterations       | 100          |
| Maximum Phase 2 Iterations       | 100          |
| Maximum Phase 3 Iterations       | 99999999     |
| Maximum Integer Iterations       | 100          |
| Time Limit (seconds)             | 120          |

The next section of output ([Output 5.1.3](#)) contains the [Variable Summary](#). A line is displayed for each variable in the mathematical program with the variable name, the status of the variable in the solution, the type of variable, the variable's price coefficient, the activity of the variable in the solution, and the reduced cost for the variable. The status of a variable can be

|       |                                                                                 |
|-------|---------------------------------------------------------------------------------|
| BASIC | if the variable is a basic variable in the solution.                            |
| DEGEN | if the variable is a basic variable whose activity is at its input lower bound. |
| ALTER | if the variable is nonbasic and is basic in an alternate optimal solution.      |
| LOWBD | if the variable is nonbasic and is at its lower bound.                          |
| UPPBD | if the variable is nonbasic and is at its upper bound.                          |

The **TYPE** column shows how PROC LP interprets the variable in the problem data set. Types include the following:

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NON-NEG | if the variable is a nonnegative variable with lower bound 0 and upper bound $+\infty$ .                                                                                                                                                                                                                                                                                                                                           |
| LOWERBD | if the variable has a lower bound specified in a LOWERBD observation and upper bound $+\infty$ .                                                                                                                                                                                                                                                                                                                                   |
| UPPERBD | if the variable has an upper bound that is less than $+\infty$ and lower bound 0. This upper bound is specified in an UPPERBD observation.                                                                                                                                                                                                                                                                                         |
| UPLOWBD | if the variable has a lower bound specified in a LOWERBD observation and an upper bound specified in an UPPERBD observation.                                                                                                                                                                                                                                                                                                       |
| INTEGER | if the variable is constrained to take integer values. If this is the case, then it must also be upper and lower bounded.                                                                                                                                                                                                                                                                                                          |
| BINARY  | if the variable is constrained to take value 0 or 1.                                                                                                                                                                                                                                                                                                                                                                               |
| UNRSTR  | if the variable is an unrestricted variable having bounds of $-\infty$ and $+\infty$ .                                                                                                                                                                                                                                                                                                                                             |
| SLACK   | if the variable is a slack variable that PROC LP has appended to a LE constraint. For variables of this type, the variable name is the same as the name of the constraint (given in the <b>ROW</b> variable) for which this variable is the slack. A nonzero slack variable indicates that the constraint is not tight. The slack is the amount by which the right-hand side of the constraint exceeds the left-hand side.         |
| SURPLUS | if the variable is a surplus variable that PROC LP has appended to a GE constraint. For variables of this type, the variable name is the same as the name of the constraint (given in the <b>ROW</b> variable) for which this variable is the surplus. A nonzero surplus variable indicates that the constraint is not tight. The surplus is the amount by which the left-hand side of the constraint exceeds the right-hand side. |

The **Variable Summary** gives the value of the structural variables at optimality. In this example, it tells you how to produce the jet fuel to maximize your profit. You should buy 110 units of A\_LIGHT and 80 units of BREGA. These are used to make 7.45 units of NAPHTHAL, 21.8 units of NAPHTHAL, and 77.3 units of HEATINGO. These in turn are used to make 60.65 units of JET\_1 using recipe 1 and 63.33 units of JET\_2 using recipe 2.

**Output 5.1.3** Variable Summary for the Oil Blending Problem**The LP Procedure**

| Variable Summary |               |        |         |       |          |              |
|------------------|---------------|--------|---------|-------|----------|--------------|
| Col              | Variable Name | Status | Type    | Price | Activity | Reduced Cost |
| 1                | a_light       | UPPBD  | UPPERBD | -175  | 110      | 11.6         |
| 2                | a_heavy       |        | UPPERBD | -165  | 0        | -21.45       |
| 3                | brega         | UPPBD  | UPPERBD | -205  | 80       | 3.35         |
| 4                | naphthal      | BASIC  | NON-NEG | 0     | 7.45     | 0            |
| 5                | naphthai      | BASIC  | NON-NEG | 0     | 21.8     | 0            |
| 6                | heatingo      | BASIC  | NON-NEG | 0     | 77.3     | 0            |
| 7                | jet_1         | BASIC  | NON-NEG | 300   | 60.65    | 0            |
| 8                | jet_2         | BASIC  | NON-NEG | 300   | 63.33    | 0            |

The reduced cost associated with each nonbasic variable is the marginal value of that variable if it is brought into the basis. In other words, the objective function value would (assuming no constraints were violated) increase by the reduced cost of a nonbasic variable if that variable's value increased by one. Similarly, the objective function value would (assuming no constraints were violated) decrease by the reduced cost of a nonbasic variable if that variable's value were decreased by one. Basic variables always have a zero reduced cost. At optimality, for a maximization problem, nonbasic variables that are not at an upper bound have nonpositive reduced costs (for example, A\_HEAVY has a reduced cost of -21.45). The objective would decrease if they were to increase beyond their optimal values. Nonbasic variables at upper bounds have nonnegative reduced costs, showing that increasing the upper bound (if the reduced cost is not zero) does not decrease the objective. For a nonbasic variable at its upper bound, the reduced cost is the marginal value of increasing its upper bound, often called its shadow price.

For minimization problems, the definition of reduced costs remains the same but the conditions for optimality change. For example, at optimality the reduced costs of all non-upper-bounded variables are nonnegative, and the reduced costs of upper-bounded variables at their upper bound are nonpositive.

The next section of output ([Output 5.1.4](#)) contains the [Constraint Summary](#). For each constraint row, free row, and objective row, a line is displayed in the Constraint Summary. Included on the line are the constraint name, the row type, the slack or surplus variable associated with the row, the right-hand-side constant associated with the row, the activity of the row (not including the activity of the slack and surplus variables), and the dual activity (shadow prices).

A dual variable is associated with each constraint row. At optimality, the value of this variable, the dual activity, tells you the marginal value of the right-hand-side constant. For each unit increase in the right-hand-side constant, the objective changes by this amount. This quantity is also known as the shadow price. For example, the marginal value for the right-hand-side constant of constraint HEATING\_O\_CONV is -450.

**Output 5.1.4** Constraint Summary for the Oil Blending Problem**The LP Procedure**

| Constraint Summary |                 |          |            |     |          |                  |
|--------------------|-----------------|----------|------------|-----|----------|------------------|
| Row                | Constraint Name | Type     | S/S<br>Col | Rhs | Activity | Dual<br>Activity |
| 1                  | profit          | OBJECTVE | .          | 0   | 1544     | .                |
| 2                  | naphtha_l_conv  | EQ       | .          | 0   | 0        | -60              |
| 3                  | naphtha_i_conv  | EQ       | .          | 0   | 0        | -90              |
| 4                  | heating_o_conv  | EQ       | .          | 0   | 0        | -450             |
| 5                  | recipe_1        | EQ       | .          | 0   | 0        | -300             |
| 6                  | recipe_2        | EQ       | .          | 0   | 0        | -300             |

**Example 5.2: A Sparse View of the Oil Blending Problem**

Typically, mathematical programming models are very sparse. This means that only a small percentage of the coefficients are nonzero. The sparse problem input is ideal for these models. The oil blending problem in the section “[An Introductory Example](#)” on page 171 has a sparse form. This example shows the same problem in a sparse form with the data given in a different order. In addition to representing the problem in a concise form, the sparse format

- allows long column names
- enables easy matrix generation (see [Example 5.12](#), [Example 5.13](#), and [Example 5.14](#))
- is compatible with [MPS](#) sparse format

The model in the sparse format is solved by invoking PROC LP with the [SPARSEDATA](#) option as follows.

```
data oil;
    format _type_ $8. _col_ $14. _row_ $16. ;
    input _type_ $ _col_ $ _row_ $ _coef_ ;
datalines;
max      .          profit          .
      .  arabian_light profit      -175
      .  arabian_heavy profit      -165
      .  brega       profit      -205
      .  jet_1       profit       300
      .  jet_2       profit       300
eq       .          napha_l_conv     .
      .  arabian_light napha_l_conv .035
      .  arabian_heavy napha_l_conv .030
      .  brega       napha_l_conv .045
      .  naphtha_light napha_l_conv -1
eq       .          napha_i_conv     .
      .  arabian_light napha_i_conv .100
      .  arabian_heavy napha_i_conv .075
      .  brega       napha_i_conv .135
      .  naphtha_inter napha_i_conv -1
eq       .          heating_oil_conv .
```



```

.      arabian_light heating_oil_conv      .390
.      arabian_heavy heating_oil_conv      .300
.      brega        heating_oil_conv      .430
.      heating_oil   heating_oil_conv      -1
eq      .            recipe_1              .
.      naphtha_inter recipe_1              .3
.      heating_oil   recipe_1              .7
eq      .            recipe_2              .
.      jet_1         recipe_1              -1
.      naphtha_light recipe_2              .2
.      heating_oil   recipe_2              .8
.      jet_2         recipe_2              -1
.      _rhs_         profit                0
upperbd .            available              .
.      arabian_light available              110
.      arabian_heavy available              165
.      brega         available              80
;

proc lp SPARSEDATA;
run;

```

The output from PROC LP follows.

#### Output 5.2.1 Output for the Sparse Oil Blending Problem

##### The LP Procedure

| Problem Summary     |            |
|---------------------|------------|
| Objective Function  | Max profit |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 45.00      |
| Variables           |            |
| Number              |            |
| Non-negative        | 5          |
| Upper Bounded       | 3          |
| Total               | 8          |
| Constraints         |            |
| Number              |            |
| EQ                  | 5          |
| Objective           | 1          |
| Total               | 6          |

### The LP Procedure

| Solution Summary                 |              |
|----------------------------------|--------------|
| Terminated Successfully          |              |
| Objective Value                  | 1544         |
| Phase 1 Iterations               | 0            |
| Phase 2 Iterations               | 5            |
| Phase 3 Iterations               | 0            |
| Integer Iterations               | 0            |
| Integer Solutions                | 0            |
| Initial Basic Feasible Variables | 5            |
| Time Used (seconds)              | 0            |
| Number of Inversions             | 3            |
| Epsilon                          | 1E-8         |
| Infinity                         | 1.797693E308 |
| Maximum Phase 1 Iterations       | 100          |
| Maximum Phase 2 Iterations       | 100          |
| Maximum Phase 3 Iterations       | 99999999     |
| Maximum Integer Iterations       | 100          |
| Time Limit (seconds)             | 120          |

### The LP Procedure

| Variable Summary |               |        |         |       |          |              |
|------------------|---------------|--------|---------|-------|----------|--------------|
| Col              | Variable Name | Status | Type    | Price | Activity | Reduced Cost |
| 1                | arabian_heavy |        | UPPERBD | -165  | 0        | -21.45       |
| 2                | arabian_light | UPPBD  | UPPERBD | -175  | 110      | 11.6         |
| 3                | brega         | UPPBD  | UPPERBD | -205  | 80       | 3.35         |
| 4                | heating_oil   | BASIC  | NON-NEG | 0     | 77.3     | 0            |
| 5                | jet_1         | BASIC  | NON-NEG | 300   | 60.65    | 0            |
| 6                | jet_2         | BASIC  | NON-NEG | 300   | 63.33    | 0            |
| 7                | naphtha_inter | BASIC  | NON-NEG | 0     | 21.8     | 0            |
| 8                | naphtha_light | BASIC  | NON-NEG | 0     | 7.45     | 0            |

### The LP Procedure

| Constraint Summary |                  |           |         |     |          |               |
|--------------------|------------------|-----------|---------|-----|----------|---------------|
| Row                | Constraint Name  | Type      | S/S Col | Rhs | Activity | Dual Activity |
| 1                  | profit           | OBJECTIVE | .       | 0   | 1544     | .             |
| 2                  | napha_i_conv     | EQ        | .       | 0   | 0        | -60           |
| 3                  | napha_i_conv     | EQ        | .       | 0   | 0        | -90           |
| 4                  | heating_oil_conv | EQ        | .       | 0   | 0        | -450          |
| 5                  | recipe_1         | EQ        | .       | 0   | 0        | -300          |
| 6                  | recipe_2         | EQ        | .       | 0   | 0        | -300          |

## Example 5.3: Sensitivity Analysis: Changes in Objective Coefficients

Simple solution of a linear program is often not enough. A manager needs to evaluate how sensitive the solution is to changing assumptions. The LP procedure provides several tools that are useful for “what if,” or sensitivity, analysis. One tool studies the effects of changes in the objective coefficients.

For example, in the oil blending problem, the cost of crude and the selling price of jet fuel can be highly variable. If you want to know the range over which each objective coefficient can vary without changing the variables in the basis, you can use the [RANGEPRICE](#) option in the PROC LP statement.

```
proc lp data=oil sparsedata
    rangeprice primalout=solution;
run;
```

In addition to the Problem and Solution summaries, the LP procedure produces a Price Range Summary, shown in [Output 5.3.1](#).

For each structural variable, the upper and lower ranges of the price (objective function coefficient) and the objective value are shown. The blocking variables, those variables that would enter the basis if the objective coefficient were perturbed further, are also given. For example, the output shows that if the cost of ARABIAN\_LIGHT crude were to increase from 175 to 186.6 per unit (remember that you are maximizing profit so the ARABIAN\_LIGHT objective coefficient would decrease from -175 to -186.6), then it would become optimal to use less of this crude for any fractional increase in its cost. Increasing the unit cost to 186.6 would drive its reduced cost to zero. Any additional increase would drive its reduced cost negative and would destroy the optimality conditions; thus, you would want to use less of it in your processing. The output shows that, at the point where the reduced cost is zero, you would only be realizing a profit of  $268 = 1544 - (110 \times 11.6)$  and that ARABIAN\_LIGHT enters the basis, that is, leaves its upper bound. On the other hand, if the cost of ARABIAN\_HEAVY were to decrease to 143.55, you would want to stop using the formulation of 110 units of ARABIAN\_LIGHT and 80 units of BREGA and switch to a production scheme that included ARABIAN\_HEAVY, in which case the profit would increase from the 1544 level.

**Output 5.3.1** Price Range Summary for the Oil Blending Problem**The LP Procedure**

| Problem Summary     |            |
|---------------------|------------|
| Objective Function  | Max profit |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 45.00      |

| Variables     | Number |
|---------------|--------|
| Non-negative  | 5      |
| Upper Bounded | 3      |
| Total         | 8      |

| Constraints | Number |
|-------------|--------|
| EQ          | 5      |
| Objective   | 1      |
| Total       | 6      |

| Solution Summary                 |              |
|----------------------------------|--------------|
| Terminated Successfully          |              |
| Objective Value                  | 1544         |
| Phase 1 Iterations               | 0            |
| Phase 2 Iterations               | 5            |
| Phase 3 Iterations               | 0            |
| Integer Iterations               | 0            |
| Integer Solutions                | 0            |
| Initial Basic Feasible Variables | 5            |
| Time Used (seconds)              | 0            |
| Number of Inversions             | 3            |
| Epsilon                          | 1E-8         |
| Infinity                         | 1.797693E308 |
| Maximum Phase 1 Iterations       | 100          |
| Maximum Phase 2 Iterations       | 100          |
| Maximum Phase 3 Iterations       | 99999999     |
| Maximum Integer Iterations       | 100          |
| Time Limit (seconds)             | 120          |

**Output 5.3.1** *continued*

| Variable Summary |               |        |         |       |          |              |
|------------------|---------------|--------|---------|-------|----------|--------------|
| Col              | Variable Name | Status | Type    | Price | Activity | Reduced Cost |
| 1                | arabian_heavy |        | UPPERBD | -165  | 0        | -21.45       |
| 2                | arabian_light | UPPBD  | UPPERBD | -175  | 110      | 11.6         |
| 3                | brega         | UPPBD  | UPPERBD | -205  | 80       | 3.35         |
| 4                | heating_oil   | BASIC  | NON-NEG | 0     | 77.3     | 0            |
| 5                | jet_1         | BASIC  | NON-NEG | 300   | 60.65    | 0            |
| 6                | jet_2         | BASIC  | NON-NEG | 300   | 63.33    | 0            |
| 7                | naphtha_inter | BASIC  | NON-NEG | 0     | 21.8     | 0            |
| 8                | naphtha_light | BASIC  | NON-NEG | 0     | 7.45     | 0            |

| Constraint Summary |                  |           |         |     |          |               |
|--------------------|------------------|-----------|---------|-----|----------|---------------|
| Row                | Constraint Name  | Type      | S/S Col | Rhs | Activity | Dual Activity |
| 1                  | profit           | OBJECTIVE | .       | 0   | 1544     | .             |
| 2                  | naphtha_i_conv   | EQ        | .       | 0   | 0        | -60           |
| 3                  | naphtha_i_conv   | EQ        | .       | 0   | 0        | -90           |
| 4                  | heating_oil_conv | EQ        | .       | 0   | 0        | -450          |
| 5                  | recipe_1         | EQ        | .       | 0   | 0        | -300          |
| 6                  | recipe_2         | EQ        | .       | 0   | 0        | -300          |

| Price Range Analysis |               |           |               |             |           |               |           |
|----------------------|---------------|-----------|---------------|-------------|-----------|---------------|-----------|
| Minimum Phi          |               |           |               | Maximum Phi |           |               |           |
| Col                  | Variable Name | Price     | Entering      | Objective   | Price     | Entering      | Objective |
| 1                    | arabian_heavy | -INFINITY | .             | 1544        | -143.55   | arabian_heavy | 1544      |
| 2                    | arabian_light | -186.6    | arabian_light | 268         | INFINITY  | .             | INFINITY  |
| 3                    | brega         | -208.35   | brega         | 1276        | INFINITY  | .             | INFINITY  |
| 4                    | heating_oil   | -7.790698 | brega         | 941.77907   | 71.5      | arabian_heavy | 7070.95   |
| 5                    | jet_1         | 290.19034 | brega         | 949.04392   | 392.25806 | arabian_heavy | 7139.4516 |
| 6                    | jet_2         | 290.50992 | brega         | 942.99292   | 387.19512 | arabian_heavy | 7066.0671 |
| 7                    | naphtha_inter | -24.81481 | brega         | 1003.037    | 286       | arabian_heavy | 7778.8    |
| 8                    | naphtha_light | -74.44444 | brega         | 989.38889   | 715       | arabian_heavy | 6870.75   |

Note that in the PROC LP statement, the **PRIMALOUT= SOLUTION** option was given. This caused the procedure to save the optimal solution in a SAS data set named SOLUTION. This data set can be used to perform further analysis on the problem without having to restart the solution process. [Example 5.4](#) shows how this is done. A display of the data follows in [Output 5.3.2](#).

**Output 5.3.2** The PRIMALOUT= Data Set for the Oil Blending Problem

| Obs | _OBJ_ID_ | _RHS_ID_ | _VAR_            | _TYPE_  | _STATUS_ | _LBOUND_ | _VALUE_ |
|-----|----------|----------|------------------|---------|----------|----------|---------|
| 1   | profit   | _rhs_    | arabian_heavy    | UPPERBD |          | 0        | 0.00    |
| 2   | profit   | _rhs_    | arabian_light    | UPPERBD | _UPPER_  | 0        | 110.00  |
| 3   | profit   | _rhs_    | brega            | UPPERBD | _UPPER_  | 0        | 80.00   |
| 4   | profit   | _rhs_    | heating_oil      | NON-NEG | _BASIC_  | 0        | 77.30   |
| 5   | profit   | _rhs_    | jet_1            | NON-NEG | _BASIC_  | 0        | 60.65   |
| 6   | profit   | _rhs_    | jet_2            | NON-NEG | _BASIC_  | 0        | 63.33   |
| 7   | profit   | _rhs_    | naphtha_inter    | NON-NEG | _BASIC_  | 0        | 21.80   |
| 8   | profit   | _rhs_    | naphtha_light    | NON-NEG | _BASIC_  | 0        | 7.45    |
| 9   | profit   | _rhs_    | PHASE_1_OBJECTIV | OBJECT  | _DEGEN_  | 0        | 0.00    |
| 10  | profit   | _rhs_    | profit           | OBJECT  | _BASIC_  | 0        | 1544.00 |

| Obs | _UBOUND_   | _PRICE_ | _R_COST_ |
|-----|------------|---------|----------|
| 1   | 165        | -165    | -21.45   |
| 2   | 110        | -175    | 11.60    |
| 3   | 80         | -205    | 3.35     |
| 4   | 1.7977E308 | 0       | 0.00     |
| 5   | 1.7977E308 | 300     | 0.00     |
| 6   | 1.7977E308 | 300     | 0.00     |
| 7   | 1.7977E308 | 0       | -0.00    |
| 8   | 1.7977E308 | 0       | 0.00     |
| 9   | 0          | 0       | 0.00     |
| 10  | 1.7977E308 | 0       | 0.00     |

### Example 5.4: Additional Sensitivity Analysis

The objective coefficient ranging analysis, discussed in the last example, is useful for assessing the effects of changing costs and returns on the optimal solution if each objective function coefficient is modified in isolation. However, this is often not the case.

Suppose you anticipate that the cost of crude will be increasing and you want to examine how that will affect your optimal production plans. Furthermore, you estimate that if the price of ARABIAN\_LIGHT goes up by 1 unit, then the price of ARABIAN\_HEAVY will rise by 1.2 units and the price of BREGA will increase by 1.5 units. However, you plan on passing some of your increased overhead on to your jet fuel customers, and you decide to increase the price of jet fuel 1 unit for each unit of increased cost of ARABIAN\_LIGHT.

An examination of the solution sensitivity to changes in the cost of crude is a two-step process. First, add the information on the proportional rates of change in the crude costs and the jet fuel price to the problem data set. Then, invoke the LP procedure. The following program accomplishes this. First, it adds a new row, named CHANGE, to the model. It gives this row a type of PRICESEN. That tells PROC LP to perform objective function coefficient sensitivity analysis using the given rates of change. The program then invokes PROC LP to perform the analysis. Notice that the **PRIMALIN= SOLUTION** option is used in the PROC LP statement. This tells the LP procedure to use the saved solution. Although it is not necessary to do this, it will eliminate the need for PROC LP to re-solve the problem and can save computing time.

```

data sen;
    format _type_ $8. _col_ $14. _row_ $6.;
    input _type_ $ _col_ $ _row_ $ _coef_;
    datalines;
pricesen .          change    .
.        arabian_light change    1
.        arabian_heavy change  1.2
.        brega        change  1.5
.        jet_1         change   -1
.        jet_2         change   -1
;

data;
    set oil sen;
run;

proc lp sparsedata primalin=solution;
run;

```

Output 5.4.1 shows the range over which the current basic solution remains optimal so that the current production plan need not change. The objective coefficients are modified by adding  $\phi$  times the change vector given in the SEN data set, where  $\phi$  ranges from a minimum of -4.15891 to a maximum of 29.72973. At the minimum value of  $\phi$ , the profit decreases to 1103.073. This value of  $\phi$  corresponds to an increase in the cost of ARABIAN\_HEAVY to 169.99 (namely,  $-175 + \phi \times 1.2$ ), ARABIAN\_LIGHT to 179.16 ( $= -175 + \phi \times 1$ ), and BREGA to 211.24 ( $= -205 + \phi \times 1.5$ ), and corresponds to an increase in the price of JET\_1 and JET\_2 to 304.16 ( $= 300 + \phi \times (-1)$ ). These values can be found in the Price column under the section labeled Minimum Phi.

**Output 5.4.1** The Price Sensitivity Analysis Summary for the Oil Blending Problem

| The LP Procedure    |            |
|---------------------|------------|
| Problem Summary     |            |
| Objective Function  | Max profit |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 45.00      |
| Variables           |            |
| Number              |            |
| Non-negative        | 5          |
| Upper Bounded       | 3          |
| Total               | 8          |
| Constraints         |            |
| Number              |            |
| EQ                  | 5          |
| Objective           | 1          |
| Total               | 6          |

**Output 5.4.1** *continued*


---

**Solution Summary**  
**Terminated Successfully**

---

**Objective Value** 1544

**Phase 1 Iterations** 0  
**Phase 2 Iterations** 0  
**Phase 3 Iterations** 0  
**Integer Iterations** 0  
**Integer Solutions** 0  
**Initial Basic Feasible Variables** 7  
**Time Used (seconds)** 0  
**Number of Inversions** 2

**Epsilon** 1E-8  
**Infinity** 1.797693E308  
**Maximum Phase 1 Iterations** 100  
**Maximum Phase 2 Iterations** 100  
**Maximum Phase 3 Iterations** 99999999  
**Maximum Integer Iterations** 100  
**Time Limit (seconds)** 120

---



---

**Variable Summary**

---

| Col | Variable Name | Status | Type    | Price | Activity | Reduced Cost |
|-----|---------------|--------|---------|-------|----------|--------------|
| 1   | arabian_heavy |        | UPPERBD | -165  | 0        | -21.45       |
| 2   | arabian_light | UPPBD  | UPPERBD | -175  | 110      | 11.6         |
| 3   | brega         | UPPBD  | UPPERBD | -205  | 80       | 3.35         |
| 4   | heating_oil   | BASIC  | NON-NEG | 0     | 77.3     | 0            |
| 5   | jet_1         | BASIC  | NON-NEG | 300   | 60.65    | 0            |
| 6   | jet_2         | BASIC  | NON-NEG | 300   | 63.33    | 0            |
| 7   | naphtha_inter | BASIC  | NON-NEG | 0     | 21.8     | 0            |
| 8   | naphtha_light | BASIC  | NON-NEG | 0     | 7.45     | 0            |

---



---

**Constraint Summary**

---

| Row | Constraint Name  | Type      | S/S Col | Rhs | Activity | Dual Activity |
|-----|------------------|-----------|---------|-----|----------|---------------|
| 1   | profit           | OBJECTIVE | .       | 0   | 1544     | .             |
| 2   | napha_l_conv     | EQ        | .       | 0   | 0        | -60           |
| 3   | napha_i_conv     | EQ        | .       | 0   | 0        | -90           |
| 4   | heating_oil_conv | EQ        | .       | 0   | 0        | -450          |
| 5   | recipe_1         | EQ        | .       | 0   | 0        | -300          |
| 6   | recipe_2         | EQ        | .       | 0   | 0        | -300          |

---



**Output 5.4.1** *continued***The LP Procedure**

| Price Sensitivity Analysis<br>Summary |               |
|---------------------------------------|---------------|
| Sensitivity Vector change             |               |
| Minimum Phi                           | -4.158907511  |
| Entering Variable                     | brega         |
| Optimal Objective                     | 1103.0726257  |
|                                       |               |
| Maximum Phi                           | 29.72972973   |
| Entering Variable                     | arabian_heavy |
| Optimal Objective                     | 4695.9459459  |

| Col | Variable Name | Status | Activity | Minimum Phi |              | Maximum Phi |              |
|-----|---------------|--------|----------|-------------|--------------|-------------|--------------|
|     |               |        |          | Price       | Reduced Cost | Price       | Reduced Cost |
| 1   | arabian_heavy |        | 0        | -169.9907   | -24.45065    | -129.3243   | 0            |
| 2   | arabian_light | UPPBD  | 110      | -179.1589   | 10.027933    | -145.2703   | 22.837838    |
| 3   | brega         | UPPBD  | 80       | -211.2384   | 0            | -160.4054   | 27.297297    |
| 4   | heating_oil   | BASIC  | 77.3     | 0           | 0            | 0           | 0            |
| 5   | jet_1         | BASIC  | 60.65    | 304.15891   | 0            | 270.27027   | 0            |
| 6   | jet_2         | BASIC  | 63.33    | 304.15891   | 0            | 270.27027   | 0            |
| 7   | naphtha_inter | BASIC  | 21.8     | 0           | 0            | 0           | 0            |
| 8   | naphtha_light | BASIC  | 7.45     | 0           | 0            | 0           | 0            |

The [Price Sensitivity Analysis Summary](#) also shows the effects of lowering the cost of crude and lowering the price of jet fuel. In particular, at the maximum  $\phi$  of 29.72973, the current optimal production plan yields a profit of 4695.95. Any increase or decrease in  $\phi$  beyond the limits given results in a change in the production plan. More precisely, the columns that constitute the basis change.

## Example 5.5: Price Parametric Programming for the Oil Blending Problem

This example continues to examine the effects of a change in the cost of crude and the selling price of jet fuel. Suppose that you know the cost of ARABIAN\_LIGHT crude is likely to increase 30 units, with the effects on oil and fuel prices as described in [Example 5.4](#). The analysis in the last example only accounted for an increase of a little over 4 units (because the minimum  $\phi$  was -4.15891). Because an increase in the cost of ARABIAN\_LIGHT beyond 4.15891 units requires a change in the optimal basis, it may also require a change in the optimal production strategy. This type of analysis, where you want to find how the solution changes with changes in the objective function coefficients or right-hand-side vector, is called parametric programming.

You can answer this question by using the `PRICEPHI=` option in the `PROC LP` statement. The following program instructs `PROC LP` to continually increase the cost of the crudes and the return from jet fuel using the ratios given previously, until the cost of `ARABIAN_LIGHT` increases at least 30 units.

```
proc lp sparsedata primalin=solution pricephi=-30;
run;
```

The `PRICEPHI=` option in the `PROC LP` statement tells `PROC LP` to perform parametric programming on any price change vectors specified in the problem data set. The value of the `PRICEPHI=` option tells `PROC LP` how far to change the value of  $\phi$  and in what direction. A specification of `PRICEPHI=-30` tells `PROC LP` to continue pivoting until the problem has objective function equal to (original objective function value)  $- 30 \times$  (change vector).

**Output 5.5.1** shows the result of this analysis. The first page is the **Price Sensitivity Analysis Summary**, as discussed in **Example 5.4**. The next page is an accounting for the change in basis as a result of decreasing  $\phi$  beyond -4.1589. It shows that `BREGA` left the basis at an upper bound and entered the basis at a lower bound. The interpretation of these basis changes can be difficult (Hadley 1962; Dantzig 1963).

The last page of output shows the optimal solution at the displayed value of  $\phi$ , namely -30.6878. At an increase of 30.6878 units in the cost of `ARABIAN_LIGHT` and the related changes to the other crudes and the jet fuel, it is optimal to modify the production of jet fuel as shown in the activity column. Although this plan is optimal, it results in a profit of 0. This may suggest that the ratio of a unit increase in the price of jet fuel per unit increase in the cost of `ARABIAN_LIGHT` is lower than desirable.

**Output 5.5.1** Price Parametric Programming for the Oil Blending Problem

| The LP Procedure    |            |
|---------------------|------------|
| Problem Summary     |            |
| Objective Function  | Max profit |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 45.00      |
|                     |            |
| Variables           | Number     |
| Non-negative        | 5          |
| Upper Bounded       | 3          |
| Total               | 8          |
|                     |            |
| Constraints         | Number     |
| EQ                  | 5          |
| Objective           | 1          |
| Total               | 6          |

**Output 5.5.1** *continued*

---

**Solution Summary**  
**Terminated Successfully**

---

|                                         |              |
|-----------------------------------------|--------------|
| <b>Objective Value</b>                  | 1544         |
| <b>Phase 1 Iterations</b>               | 0            |
| <b>Phase 2 Iterations</b>               | 0            |
| <b>Phase 3 Iterations</b>               | 0            |
| <b>Integer Iterations</b>               | 0            |
| <b>Integer Solutions</b>                | 0            |
| <b>Initial Basic Feasible Variables</b> | 7            |
| <b>Time Used (seconds)</b>              | 0            |
| <b>Number of Inversions</b>             | 2            |
| <b>Epsilon</b>                          | 1E-8         |
| <b>Infinity</b>                         | 1.797693E308 |
| <b>Maximum Phase 1 Iterations</b>       | 100          |
| <b>Maximum Phase 2 Iterations</b>       | 100          |
| <b>Maximum Phase 3 Iterations</b>       | 99999999     |
| <b>Maximum Integer Iterations</b>       | 100          |
| <b>Time Limit (seconds)</b>             | 120          |

---

**Variable Summary**

| Col | Variable Name | Status | Type    | Price | Activity | Reduced Cost |
|-----|---------------|--------|---------|-------|----------|--------------|
| 1   | arabian_heavy |        | UPPERBD | -165  | 0        | -21.45       |
| 2   | arabian_light | UPPBD  | UPPERBD | -175  | 110      | 11.6         |
| 3   | brega         | UPPBD  | UPPERBD | -205  | 80       | 3.35         |
| 4   | heating_oil   | BASIC  | NON-NEG | 0     | 77.3     | 0            |
| 5   | jet_1         | BASIC  | NON-NEG | 300   | 60.65    | 0            |
| 6   | jet_2         | BASIC  | NON-NEG | 300   | 63.33    | 0            |
| 7   | naphtha_inter | BASIC  | NON-NEG | 0     | 21.8     | 0            |
| 8   | naphtha_light | BASIC  | NON-NEG | 0     | 7.45     | 0            |

---

**Constraint Summary**

| Row | Constraint Name  | Type      | S/S Col | Rhs | Activity | Dual Activity |
|-----|------------------|-----------|---------|-----|----------|---------------|
| 1   | profit           | OBJECTIVE | .       | 0   | 1544     | .             |
| 2   | napha_l_conv     | EQ        | .       | 0   | 0        | -60           |
| 3   | napha_i_conv     | EQ        | .       | 0   | 0        | -90           |
| 4   | heating_oil_conv | EQ        | .       | 0   | 0        | -450          |
| 5   | recipe_1         | EQ        | .       | 0   | 0        | -300          |
| 6   | recipe_2         | EQ        | .       | 0   | 0        | -300          |

---

**Output 5.5.1** *continued***The LP Procedure****Price Sensitivity Analysis  
Summary****Sensitivity Vector change**

Minimum Phi      -4.158907511  
 Entering Variable      brega  
 Optimal Objective    1103.0726257

Maximum Phi      29.72972973  
 Entering Variable    arabian\_heavy  
 Optimal Objective    4695.9459459

| Col | Variable Name | Status | Activity | Minimum Phi |              | Maximum Phi |              |
|-----|---------------|--------|----------|-------------|--------------|-------------|--------------|
|     |               |        |          | Price       | Reduced Cost | Price       | Reduced Cost |
| 1   | arabian_heavy |        | 0        | -169.9907   | -24.45065    | -129.3243   | 0            |
| 2   | arabian_light | UPPBD  | 110      | -179.1589   | 10.027933    | -145.2703   | 22.837838    |
| 3   | brega         | UPPBD  | 80       | -211.2384   | 0            | -160.4054   | 27.297297    |
| 4   | heating_oil   | BASIC  | 77.3     | 0           | 0            | 0           | 0            |
| 5   | jet_1         | BASIC  | 60.65    | 304.15891   | 0            | 270.27027   | 0            |
| 6   | jet_2         | BASIC  | 63.33    | 304.15891   | 0            | 270.27027   | 0            |
| 7   | naphtha_inter | BASIC  | 21.8     | 0           | 0            | 0           | 0            |
| 8   | naphtha_light | BASIC  | 7.45     | 0           | 0            | 0           | 0            |

**The LP Procedure****Price Parametric Programming Log****Sensitivity Vector change**

| Leaving Variable | Entering Variable | Objective | Current Phi |
|------------------|-------------------|-----------|-------------|
| brega            | brega             | 1103.0726 | -4.158908   |

**The LP Procedure****Price Sensitivity Analysis  
Summary****Sensitivity Vector change**

Minimum Phi      -30.68783069  
 Entering Variable    arabian\_light  
 Optimal Objective      0

**Output 5.5.1** *continued*

| Col | Variable Name | Status | Activity | Minimum Phi |              |
|-----|---------------|--------|----------|-------------|--------------|
|     |               |        |          | Price       | Reduced Cost |
| 1   | arabian_heavy |        | 0        | -201.8254   | -43.59127    |
| 2   | arabian_light | ALTER  | 110      | -205.6878   | 0            |
| 3   | brega         |        | 0        | -251.0317   | -21.36905    |
| 4   | heating_oil   | BASIC  | 42.9     | 0           | 0            |
| 5   | jet_1         | BASIC  | 33.33    | 330.68783   | 0            |
| 6   | jet_2         | BASIC  | 35.09    | 330.68783   | 0            |
| 7   | naphtha_inter | BASIC  | 11       | 0           | 0            |
| 8   | naphtha_light | BASIC  | 3.85     | 0           | 0            |

What is the optimal return if  $\phi$  is exactly -30? Because the change in the objective is linear as a function of  $\phi$ , you can calculate the objective for any value of  $\phi$  between those given by linear interpolation. For example, for any  $\phi$  between -4.1589 and -30.6878, the optimal objective value is

$$\phi \times (1103.0726 - 0) / (-4.1589 - 30.6878) + b$$

where

$$b = 30.6878 \times (1103.0726 - 0) / (-4.1589 - 30.6878)$$

For  $\phi = -30$ , this is 28.5988.

---

## Example 5.6: Special Ordered Sets and the Oil Blending Problem

Often managers want to evaluate the cost of making a choice among alternatives. In particular, they want to make the most profitable choice. Suppose that only one oil crude can be used in the production process. This identifies a set of variables of which only one can be above its lower bound. This additional restriction could be included in the model by adding a binary integer variable for each of the three crudes. Constraints would be needed that would drive the appropriate binary variable to 1 whenever the corresponding crude is used in the production process. Then a constraint limiting the total of these variables to only one would be added. A similar formulation for a fixed charge problem is shown in [Example 5.8](#).

The SOSLE type implicitly does this. The following DATA step adds a row to the model that identifies which variables are in the set. The SOSLE type tells the LP procedure that only one of the variables in this set can be above its lower bound. If you use the SOSEQ type, it tells PROC LP that exactly one of the variables in the set must be above its lower bound. Only integer variables can be in an SOSEQ set.

```
data special;
  format _type_ $6. _col_ $14. _row_ $8. ;
  input _type_ $ _col_ $ _row_ $ _coef_;
  datalines;
SOSLE .          special .
```

```

.      arabian_light special 1
.      arabian_heavy special 1
.      brega          special 1
;

data;
  set oil special;
run;

proc lp sparsedata;
run;

```

Output 5.6.1 includes an Integer Iteration Log. This log shows the progress that PROC LP is making in solving the problem. This is discussed in some detail in [Example 5.8](#).

**Output 5.6.1** The Oil Blending Problem with a Special Ordered Set

### The LP Procedure

| Problem Summary     |            |
|---------------------|------------|
| Objective Function  | Max profit |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 45.00      |
| Variables           |            |
| Number              |            |
| Non-negative        | 5          |
| Upper Bounded       | 3          |
| Total               | 8          |
| Constraints         |            |
| Number              |            |
| EQ                  | 5          |
| Objective           | 1          |
| Total               | 6          |

| Integer Iteration Log |         |            |           |               |       |         |        |           |
|-----------------------|---------|------------|-----------|---------------|-------|---------|--------|-----------|
| Iter                  | Problem | Condition  | Objective | Branched      | Value | Sinfeas | Active | Proximity |
| 1                     | 0       | ACTIVE     | 1544      | arabian_light | 110   | 0       | 2      | .         |
| 2                     | -1      | SUBOPTIMAL | 1276      | .             | .     | .       | 1      | 268       |
| 3                     | 1       | FATHOMED   | 268       | .             | .     | .       | 0      | .         |

**Output 5.6.1** *continued*

| Solution Summary                 |  |              |  |  |  |  |
|----------------------------------|--|--------------|--|--|--|--|
| Integer Optimal Solution         |  |              |  |  |  |  |
| Objective Value                  |  | 1276         |  |  |  |  |
| Phase 1 Iterations               |  | 0            |  |  |  |  |
| Phase 2 Iterations               |  | 5            |  |  |  |  |
| Phase 3 Iterations               |  | 0            |  |  |  |  |
| Integer Iterations               |  | 3            |  |  |  |  |
| Integer Solutions                |  | 1            |  |  |  |  |
| Initial Basic Feasible Variables |  | 5            |  |  |  |  |
| Time Used (seconds)              |  | 0            |  |  |  |  |
| Number of Inversions             |  | 5            |  |  |  |  |
| Epsilon                          |  | 1E-8         |  |  |  |  |
| Infinity                         |  | 1.797693E308 |  |  |  |  |
| Maximum Phase 1 Iterations       |  | 100          |  |  |  |  |
| Maximum Phase 2 Iterations       |  | 100          |  |  |  |  |
| Maximum Phase 3 Iterations       |  | 99999999     |  |  |  |  |
| Maximum Integer Iterations       |  | 100          |  |  |  |  |
| Time Limit (seconds)             |  | 120          |  |  |  |  |

| Variable Summary |               |        |         |       |          |              |
|------------------|---------------|--------|---------|-------|----------|--------------|
| Col              | Variable Name | Status | Type    | Price | Activity | Reduced Cost |
| 1                | arabian_heavy |        | UPPERBD | -165  | 0        | -21.45       |
| 2                | arabian_light | UPPBD  | UPPERBD | -175  | 110      | 11.6         |
| 3                | brega         |        | UPPERBD | -205  | 0        | 3.35         |
| 4                | heating_oil   | BASIC  | NON-NEG | 0     | 42.9     | 0            |
| 5                | jet_1         | BASIC  | NON-NEG | 300   | 33.33    | 0            |
| 6                | jet_2         | BASIC  | NON-NEG | 300   | 35.09    | 0            |
| 7                | naphtha_inter | BASIC  | NON-NEG | 0     | 11       | 0            |
| 8                | naphtha_light | BASIC  | NON-NEG | 0     | 3.85     | 0            |

| Constraint Summary |                  |           |         |     |          |               |
|--------------------|------------------|-----------|---------|-----|----------|---------------|
| Row                | Constraint Name  | Type      | S/S Col | Rhs | Activity | Dual Activity |
| 1                  | profit           | OBJECTIVE | .       | 0   | 1276     | .             |
| 2                  | napha_l_conv     | EQ        | .       | 0   | 0        | -60           |
| 3                  | napha_i_conv     | EQ        | .       | 0   | 0        | -90           |
| 4                  | heating_oil_conv | EQ        | .       | 0   | 0        | -450          |
| 5                  | recipe_1         | EQ        | .       | 0   | 0        | -300          |
| 6                  | recipe_2         | EQ        | .       | 0   | 0        | -300          |

The solution shows that only the ARABIAN\_LIGHT crude is purchased. The requirement that only one crude be used in the production is met, and the profit is 1276. This tells you that the value of purchasing crude from an additional source, namely BREGA, is worth  $1544 - 1276 = 268$ .

## Example 5.7: Goal-Programming a Product Mix Problem

This example shows how to use PROC LP to solve a linear goal-programming problem. PROC LP has the ability to solve a series of linear programs, each with a new objective function. These objective functions are ordered by priority. The first step is to solve a linear program with the highest priority objective function constrained only by the formal constraints in the model. Then, the problem with the next highest priority objective function is solved, constrained by the formal constraints in the model and by the value that the highest priority objective function realized. That is, the second problem optimizes the second highest priority objective function among the alternate optimal solutions to the first optimization problem. The process continues until a linear program is solved for each of the objectives.

This technique is useful for differentiating among alternate optimal solutions to a linear program. It also fits into the formal paradigm presented in goal programming. In goal programming, the objective functions typically take on the role of driving a linear function of the structural variables to meet a target level as closely as possible. The details of this can be found in many books on the subject, including Ignizio (1976).

Consider the following problem taken from Ignizio (1976). A small paint company manufactures two types of paint, latex and enamel. In production, the company uses 10 hours of labor to produce 100 gallons of latex and 15 hours of labor to produce 100 gallons of enamel. Without hiring outside help or requiring overtime, the company has 40 hours of labor available each week. Furthermore, each paint generates a profit at the rate of \$1.00 per gallon. The company has the following objectives listed in decreasing priority:

- avoid the use of overtime
- achieve a weekly profit of \$1000
- produce at least 700 gallons of enamel paint each week

The program to solve this problem follows.

```
data object;
  input _row_ $ latex enamel n1 n2 n3 p1 p2 p3 _type_ $ _rhs_;
  datalines;
overtime . . . . 1 . . min 1
profit . . . 1 . . . min 2
enamel . . . . 1 . . min 3
overtime 10 15 1 . . -1 . . eq 40
profit 100 100 . 1 . . -1 . eq 1000
enamel . 1 . . 1 . . -1 eq 7
;

proc lp data=object goalprogram;
run;
```

The data set called OBJECT contains the model. Its first three observations are the objective rows, and the next three observations are the constraints. The values in the right-hand-side variable `_RHS_` in the objective rows give the priority of the objectives. The objective in the first observation with `_ROW_='OVERTIME'` has the highest priority, the objective named PROFIT has the next highest, and the objective named ENAMEL has the lowest. Note that the value of the right-hand-side variable determines the priority, not the order, in the data set.



Because this example is set in the formal goal-programming scheme, the model has structural variables representing negative (n1, n2, and n3) and positive (p1, p2, and p3) deviations from target levels. For example, n1+p1 is the deviation from the objective of avoiding the use of overtime and underusing the normal work time, namely using exactly 40 work hours. The other objectives are handled similarly.

Notice that the PROC LP statement includes the **GOALPROGRAM** option. Without this option, the procedure would solve three separate problems: one for each of the three objective functions. In that case, however, the procedure would not constrain the second and third programs using the results of the preceding programs; also, the values 1, 2, and 3 for **\_RHS\_** in the objective rows would have no effect.

**Output 5.7.1** shows the solution of the goal program, apparently as three linear program outputs. However, examination of the constraint summaries in the second and third problems shows that the constraints labeled by the objectives **OVERTIME** and **PROFIT** have type **FIXEDOBJ**. This indicates that these objective rows have become constraints in the subsequent problems.

### **Output 5.7.1** Goal Programming

#### **The LP Procedure**

| <b>Problem Summary</b>     |              |
|----------------------------|--------------|
| <b>Objective Function</b>  | Min overtime |
| <b>Rhs Variable</b>        | _rhs_        |
| <b>Type Variable</b>       | _type_       |
| <b>Problem Density (%)</b> | 45.83        |
|                            |              |
| <b>Variables</b>           | Number       |
| <b>Non-negative</b>        | 8            |
| <b>Total</b>               | 8            |
|                            |              |
| <b>Constraints</b>         | Number       |
| <b>EQ</b>                  | 3            |
| <b>Objective</b>           | 3            |
| <b>Total</b>               | 6            |

**Output 5.7.1** *continued*

| Solution Summary                 |  |              |  |  |  |  |
|----------------------------------|--|--------------|--|--|--|--|
| Terminated Successfully          |  |              |  |  |  |  |
| Objective Value                  |  | 0            |  |  |  |  |
| Phase 1 Iterations               |  | 2            |  |  |  |  |
| Phase 2 Iterations               |  | 0            |  |  |  |  |
| Phase 3 Iterations               |  | 0            |  |  |  |  |
| Integer Iterations               |  | 0            |  |  |  |  |
| Integer Solutions                |  | 0            |  |  |  |  |
| Initial Basic Feasible Variables |  | 7            |  |  |  |  |
| Time Used (seconds)              |  | 0            |  |  |  |  |
| Number of Inversions             |  | 2            |  |  |  |  |
| Epsilon                          |  | 1E-8         |  |  |  |  |
| Infinity                         |  | 1.797693E308 |  |  |  |  |
| Maximum Phase 1 Iterations       |  | 100          |  |  |  |  |
| Maximum Phase 2 Iterations       |  | 100          |  |  |  |  |
| Maximum Phase 3 Iterations       |  | 99999999     |  |  |  |  |
| Maximum Integer Iterations       |  | 100          |  |  |  |  |
| Time Limit (seconds)             |  | 120          |  |  |  |  |

| Variable Summary |        |        |         |       |          |         |
|------------------|--------|--------|---------|-------|----------|---------|
| Variable         |        | Status | Type    | Price | Activity | Reduced |
| Col              | Name   |        |         |       |          | Cost    |
| 1                | latex  | ALTER  | NON-NEG | 0     | 0        | 0       |
| 2                | enamel | ALTER  | NON-NEG | 0     | 0        | 0       |
| 3                | n1     | BASIC  | NON-NEG | 0     | 40       | 0       |
| 4                | n2     | BASIC  | NON-NEG | 0     | 1000     | 0       |
| 5                | n3     | BASIC  | NON-NEG | 0     | 7        | 0       |
| 6                | p1     |        | NON-NEG | 1     | 0        | 1       |
| 7                | p2     | ALTER  | NON-NEG | 0     | 0        | 0       |
| 8                | p3     | ALTER  | NON-NEG | 0     | 0        | 0       |

| Constraint Summary |            |          |     |      |      |   |
|--------------------|------------|----------|-----|------|------|---|
| Row                | Constraint |          | S/S |      | Dual |   |
|                    | Name       | Type     | Col | Rhs  |      |   |
| 1                  | overtime   | OBJECTVE | .   | 0    | 0    | . |
| 2                  | profit     | FREE_OBJ | .   | 0    | 1000 | . |
| 3                  | enamel     | FREE_OBJ | .   | 0    | 7    | . |
| 4                  | overtime   | EQ       | .   | 40   | 40   | 0 |
| 5                  | profit     | EQ       | .   | 1000 | 1000 | 0 |
| 6                  | enamel     | EQ       | .   | 7    | 7    | 0 |

**Output 5.7.1** *continued*

| Problem Summary     |            |
|---------------------|------------|
| Objective Function  | Min profit |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 45.83      |
|                     |            |
| Variables           | Number     |
| Non-negative        | 8          |
| Total               | 8          |
|                     |            |
| Constraints         | Number     |
| EQ                  | 3          |
| Objective           | 3          |
| Total               | 6          |

| Solution Summary                 |              |
|----------------------------------|--------------|
| Terminated Successfully          |              |
| Objective Value                  | 600          |
|                                  |              |
| Phase 1 Iterations               | 0            |
| Phase 2 Iterations               | 3            |
| Phase 3 Iterations               | 0            |
| Integer Iterations               | 0            |
| Integer Solutions                | 0            |
| Initial Basic Feasible Variables | 7            |
| Time Used (seconds)              | 0            |
| Number of Inversions             | 5            |
|                                  |              |
| Epsilon                          | 1E-8         |
| Infinity                         | 1.797693E308 |
| Maximum Phase 1 Iterations       | 100          |
| Maximum Phase 2 Iterations       | 100          |
| Maximum Phase 3 Iterations       | 99999999     |
| Maximum Integer Iterations       | 100          |
| Time Limit (seconds)             | 120          |

**Output 5.7.1** *continued*

| Variable Summary |        |        |         |       |          |      |
|------------------|--------|--------|---------|-------|----------|------|
| Variable         |        |        | Reduced |       |          |      |
| Col              | Name   | Status | Type    | Price | Activity | Cost |
| 1                | latex  | BASIC  | NON-NEG | 0     | 4        | 0    |
| 2                | enamel |        | NON-NEG | 0     | 0        | 50   |
| 3                | n1     |        | NON-NEG | 0     | 0        | 10   |
| 4                | n2     | BASIC  | NON-NEG | 1     | 600      | 0    |
| 5                | n3     | BASIC  | NON-NEG | 0     | 7        | 0    |
| 6                | p1     | DEGEN  | NON-NEG | 0     | 0        | 0    |
| 7                | p2     |        | NON-NEG | 0     | 0        | 1    |
| 8                | p3     | ALTER  | NON-NEG | 0     | 0        | 0    |

| Constraint Summary |          |          |     |      |          |          |
|--------------------|----------|----------|-----|------|----------|----------|
| Constraint         |          | S/S      |     | Dual |          |          |
| Row                | Name     | Type     | Col | Rhs  | Activity | Activity |
| 1                  | overtime | FIXEDOBJ | .   | 0    | 0        | .        |
| 2                  | profit   | OBJECTVE | .   | 0    | 600      | .        |
| 3                  | enamel   | FREE_OBJ | .   | 0    | 7        | .        |
| 4                  | overtime | EQ       | .   | 40   | 40       | -10      |
| 5                  | profit   | EQ       | .   | 1000 | 1000     | 1        |
| 6                  | enamel   | EQ       | .   | 7    | 7        | 0        |

| Problem Summary     |            |
|---------------------|------------|
| Objective Function  | Min enamel |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 45.83      |
| Variables           |            |
| Number              |            |
| Non-negative        | 8          |
| Total               | 8          |
| Constraints         |            |
| Number              |            |
| EQ                  | 3          |
| Objective           | 3          |
| Total               | 6          |

**Output 5.7.1** *continued*

| Solution Summary                 |  |              |  |  |  |  |
|----------------------------------|--|--------------|--|--|--|--|
| Terminated Successfully          |  |              |  |  |  |  |
| Objective Value                  |  | 7            |  |  |  |  |
| Phase 1 Iterations               |  | 0            |  |  |  |  |
| Phase 2 Iterations               |  | 1            |  |  |  |  |
| Phase 3 Iterations               |  | 0            |  |  |  |  |
| Integer Iterations               |  | 0            |  |  |  |  |
| Integer Solutions                |  | 0            |  |  |  |  |
| Initial Basic Feasible Variables |  | 7            |  |  |  |  |
| Time Used (seconds)              |  | 0            |  |  |  |  |
| Number of Inversions             |  | 8            |  |  |  |  |
| Epsilon                          |  | 1E-8         |  |  |  |  |
| Infinity                         |  | 1.797693E308 |  |  |  |  |
| Maximum Phase 1 Iterations       |  | 100          |  |  |  |  |
| Maximum Phase 2 Iterations       |  | 100          |  |  |  |  |
| Maximum Phase 3 Iterations       |  | 99999999     |  |  |  |  |
| Maximum Integer Iterations       |  | 100          |  |  |  |  |
| Time Limit (seconds)             |  | 120          |  |  |  |  |

| Variable Summary |        |         |         |       |          |      |
|------------------|--------|---------|---------|-------|----------|------|
| Variable         |        | Reduced |         |       |          |      |
| Col              | Name   | Status  | Type    | Price | Activity | Cost |
| 1                | latex  | BASIC   | NON-NEG | 0     | 4        | 0    |
| 2                | enamel | DEGEN   | NON-NEG | 0     | 0        | 0    |
| 3                | n1     |         | NON-NEG | 0     | 0        | 0.2  |
| 4                | n2     | BASIC   | NON-NEG | 0     | 600      | 0    |
| 5                | n3     | BASIC   | NON-NEG | 1     | 7        | 0    |
| 6                | p1     | DEGEN   | NON-NEG | 0     | 0        | 0    |
| 7                | p2     |         | NON-NEG | 0     | 0        | 0.02 |
| 8                | p3     |         | NON-NEG | 0     | 0        | 1    |

| Constraint Summary |          |          |     |      |          |          |
|--------------------|----------|----------|-----|------|----------|----------|
| Constraint         |          | S/S      |     | Dual |          |          |
| Row                | Name     | Type     | Col | Rhs  | Activity | Activity |
| 1                  | overtime | FIXEDOBJ | .   | 0    | 0        | .        |
| 2                  | profit   | FIXEDOBJ | .   | 0    | 600      | .        |
| 3                  | enamel   | OBJECTVE | .   | 0    | 7        | .        |
| 4                  | overtime | EQ       | .   | 40   | 40       | -0.2     |
| 5                  | profit   | EQ       | .   | 1000 | 1000     | 0.02     |
| 6                  | enamel   | EQ       | .   | 7    | 7        | 1        |

The solution to the last linear program shows a value of 4 for the variable LATEX and a value of 0 for the variable ENAMEL. This tells you that the solution to the linear goal program is to produce 400 gallons of latex and no enamel paint.

The values of the objective functions in the three linear programs tell you whether you can achieve the three objectives. The activities of the constraints labeled OVERTIME, PROFIT, and ENAMEL tell you values of

the three linear program objectives. Because the first linear programming objective OVERTIME is 0, the highest priority objective, which is to avoid using additional labor, is accomplished. However, because the second and third objectives are nonzero, the second and third priority objectives are not satisfied completely. The PROFIT objective is 600. Because the PROFIT objective is to minimize the negative deviation from the profit constraint, this means that a profit of only  $400 = 1000 - 600$  is realized. Similarly, the ENAMEL objective is 7, indicating that there is a negative deviation from the ENAMEL target of 7 units.

### Example 5.8: A Simple Integer Program

Recall the linear programming problem presented in Chapter 3, “Introduction to Optimization” (*SAS/OR User’s Guide: Mathematical Programming*). In that problem, a firm produces two products, chocolates and gumdrops, that are processed by four processes: cooking, color/flavor, condiments, and packaging. The objective is to determine the product mix that maximizes the profit to the firm while not exceeding manufacturing capacities. The problem is extended to demonstrate a use of integer-constrained variables.

Suppose that you must manufacture only one of the two products. In addition, there is a setup cost of 100 if you make the chocolates and 75 if you make the gumdrops. To identify which product will maximize profit, you define two zero-one integer variables, ICHOCO and IGUMDR, and you also define two new constraints, CHOCOLATE and GUM. The constraint labeled CHOCOLATE forces ICHOCO to equal one when chocolates are manufactured. Similarly, the constraint labeled GUM forces IGUMDR to equal 1 when gumdrops are manufactured. Also, you should include a constraint labeled ONLY\_ONE that requires the sum of ICHOCO and IGUMDR to equal 1. (Note that this could be accomplished more simply by including ICHOCO and IGUMDR in a SOSEQ set.) Since ICHOCO and IGUMDR are integer variables, this constraint eliminates the possibility of both products being manufactured. Notice the coefficients -10000, which are used to force ICHOCO or IGUMDR to 1 whenever CHOCO and GUMDR are nonzero. This technique, which is often used in integer programming, can cause severe numerical problems. If this driving coefficient is too large, then arithmetic overflows and underflow may result. If the driving coefficient is too small, then the integer variable may not be driven to 1 as desired by the modeler.

The objective coefficients of the integer variables ICHOCO and IGUMDR are the negatives of the setup costs for the two products. The following is the data set that describes this problem and the call to PROC LP to solve it:

```
data;
  format _row_ $10. ;
  input _row_ $ choco gumdr ichoco igumdr _type_ $ _rhs_;
  datalines;
object          .25      .75      -100      -75 max          .
cooking          15       40        0        0 1e          27000
color            0      56.25        0        0 1e          27000
package        18.75        0        0        0 1e          27000
condiments       12       50        0        0 1e          27000
chocolate        1        0 -10000        0 1e            0
gum              0        1        0 -10000 1e            0
only_one         0        0        1        1 eq            1
binary           .        .        1        2 binary        .
;
```

```
proc lp;
run;
```

The solution shows that gumdrops are produced. See [Output 5.8.1](#).

**Output 5.8.1** Summaries and an Integer Programming Iteration Log

**The LP Procedure**

| Problem Summary     |            |
|---------------------|------------|
| Objective Function  | Max object |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 25.71      |
|                     |            |
| Variables           | Number     |
|                     |            |
| Non-negative        | 2          |
| Binary              | 2          |
| Slack               | 6          |
|                     |            |
| Total               | 10         |
|                     |            |
| Constraints         | Number     |
|                     |            |
| LE                  | 6          |
| EQ                  | 1          |
| Objective           | 1          |
|                     |            |
| Total               | 8          |

| Integer Iteration Log |         |            |           |                |         |        |           |
|-----------------------|---------|------------|-----------|----------------|---------|--------|-----------|
| Iter                  | Problem | Condition  | Objective | Branched Value | Sinfeas | Active | Proximity |
| 1                     | 0       | ACTIVE     | 397.5     | ichoco         | 0.1     | 0.2    | 2         |
| 2                     | -1      | SUBOPTIMAL | 260       | .              | .       | .      | 1         |
| 3                     | 1       | SUBOPTIMAL | 285       | .              | .       | .      | 0         |

**Output 5.8.1** *continued*

| Solution Summary                 |  |              |  |  |  |  |
|----------------------------------|--|--------------|--|--|--|--|
| Integer Optimal Solution         |  |              |  |  |  |  |
| Objective Value                  |  | 285          |  |  |  |  |
| Phase 1 Iterations               |  | 0            |  |  |  |  |
| Phase 2 Iterations               |  | 5            |  |  |  |  |
| Phase 3 Iterations               |  | 5            |  |  |  |  |
| Integer Iterations               |  | 3            |  |  |  |  |
| Integer Solutions                |  | 2            |  |  |  |  |
| Initial Basic Feasible Variables |  | 9            |  |  |  |  |
| Time Used (seconds)              |  | 0            |  |  |  |  |
| Number of Inversions             |  | 5            |  |  |  |  |
| Epsilon                          |  | 1E-8         |  |  |  |  |
| Infinity                         |  | 1.797693E308 |  |  |  |  |
| Maximum Phase 1 Iterations       |  | 100          |  |  |  |  |
| Maximum Phase 2 Iterations       |  | 100          |  |  |  |  |
| Maximum Phase 3 Iterations       |  | 99999999     |  |  |  |  |
| Maximum Integer Iterations       |  | 100          |  |  |  |  |
| Time Limit (seconds)             |  | 120          |  |  |  |  |

| Variable Summary |               |        |         |       |          |              |
|------------------|---------------|--------|---------|-------|----------|--------------|
| Col              | Variable Name | Status | Type    | Price | Activity | Reduced Cost |
| 1                | choco         | DEGEN  | NON-NEG | 0.25  | 0        | 0            |
| 2                | gumdr         | BASIC  | NON-NEG | 0.75  | 480      | 0            |
| 3                | ichoco        |        | BINARY  | -100  | 0        | 2475         |
| 4                | igumdr        | BASIC  | BINARY  | -75   | 1        | 0            |
| 5                | cooking       | BASIC  | SLACK   | 0     | 7800     | 0            |
| 6                | color         |        | SLACK   | 0     | 0        | -0.013333    |
| 7                | package       | BASIC  | SLACK   | 0     | 27000    | 0            |
| 8                | condiments    | BASIC  | SLACK   | 0     | 3000     | 0            |
| 9                | chocolate     |        | SLACK   | 0     | 0        | -0.25        |
| 10               | gum           | BASIC  | SLACK   | 0     | 9520     | 0            |

| Constraint Summary |                 |          |         |       |          |               |
|--------------------|-----------------|----------|---------|-------|----------|---------------|
| Row                | Constraint Name | Type     | S/S Col | Rhs   | Activity | Dual Activity |
| 1                  | object          | OBJECTVE | .       | 0     | 285      | .             |
| 2                  | cooking         | LE       | 5       | 27000 | 19200    | 0             |
| 3                  | color           | LE       | 6       | 27000 | 27000    | 0.0133333     |
| 4                  | package         | LE       | 7       | 27000 | 0        | 0             |
| 5                  | condiments      | LE       | 8       | 27000 | 24000    | 0             |
| 6                  | chocolate       | LE       | 9       | 0     | 0        | 0.25          |
| 7                  | gum             | LE       | 10      | 0     | -9520    | 0             |
| 8                  | only one        | EQ       | .       | 1     | 1        | -75           |



The branch-and-bound tree can be reconstructed from the information contained in the integer iteration log. The column labeled *Iter* numbers the integer iterations. The column labeled *Problem* identifies the *Iter* number of the parent problem from which the current problem is defined. For example, *Iter*=2 has *Problem*=-1. This means that problem 2 is a direct descendant of problem 1. Furthermore, because problem 1 branched on *ICHOCO*, you know that problem 2 is identical to problem 1 with an additional constraint on variable *ICHOCO*. The minus sign in the *Problem*=-1 in *Iter*=2 tells you that the new constraint on variable *ICHOCO* is a lower bound. Moreover, because *Value*=0.1 in *Iter*=1, you know that *ICHOCO*=0.1 in *Iter*=1 so that the added constraint in *Iter*=2 is  $\text{ICHOCO} \geq \lceil 0.1 \rceil$ . In this way, the information in the log can be used to reconstruct the branch-and-bound tree. In fact, when you save an **ACTIVEOUT=** data set, it contains information in this format that is used to reconstruct the tree when you restart a problem using the **ACTIVEIN=** data set. See [Example 5.10](#).

Note that if you defined a **SOSEQ** special ordered set containing the variables *CHOCO* and *GUMDR*, the integer variables *ICHOCO* and *IGUMDR* and the three associated constraints would not have been needed.

---

## Example 5.9: An Infeasible Problem

This is an example of the **Infeasible Information Summary** that is displayed when an infeasible problem is encountered. Consider the following problem:

$$\begin{array}{ll}
 \max & x + y + z + w \\
 \text{subject to} & x + 3y + 2z + 4w \leq 5 \\
 & 3x + y + 2z + w \leq 4 \\
 & 5x + 3y + 3z + 3w = 9 \\
 & x, y, z, w \geq 0
 \end{array}$$

Examination of this problem reveals that it is unsolvable. Consequently, PROC LP identifies it as infeasible. The following program attempts to solve it.

```

data infeas;
    format _id_ $6.;
    input _id_ $ x1-x4 _type_ $ _rhs_;
    datalines;
profit    1    1    1    1    max    .
const1    1    3    2    4    le     5
const2    3    1    2    1    le     4
const3    5    3    3    3    eq     9
;

```

The results are shown in [Output 5.9.1](#).

**Output 5.9.1** The Solution of an Infeasible Problem

**The LP Procedure**

| Problem Summary     |            |
|---------------------|------------|
| Objective Function  | Max profit |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 77.78      |
|                     |            |
| Variables           | Number     |
| Non-negative        | 4          |
| Slack               | 2          |
| Total               | 6          |
|                     |            |
| Constraints         | Number     |
| LE                  | 2          |
| EQ                  | 1          |
| Objective           | 1          |
| Total               | 4          |

**ERROR:** Infeasible problem. Note the constraints in the constraint summary that are identified as infeasible. If none of the constraints are flagged then check the implicit bounds on the variables.

**The LP Procedure**

| Solution Summary                 |              |
|----------------------------------|--------------|
| Infeasible Problem               |              |
| Objective Value                  | 2.5          |
| Phase 1 Iterations               | 2            |
| Phase 2 Iterations               | 0            |
| Phase 3 Iterations               | 0            |
| Integer Iterations               | 0            |
| Integer Solutions                | 0            |
| Initial Basic Feasible Variables | 5            |
| Time Used (seconds)              | 0            |
| Number of Inversions             | 2            |
| Epsilon                          | 1E-8         |
| Infinity                         | 1.797693E308 |
| Maximum Phase 1 Iterations       | 100          |
| Maximum Phase 2 Iterations       | 100          |
| Maximum Phase 3 Iterations       | 99999999     |
| Maximum Integer Iterations       | 100          |
| Time Limit (seconds)             | 120          |

**The LP Procedure**

| Variable Summary |        |        |         |       |          |      |
|------------------|--------|--------|---------|-------|----------|------|
| Variable         |        |        | Reduced |       |          |      |
| Col              | Name   | Status | Type    | Price | Activity | Cost |
| 1                | x1     | BASIC  | NON-NEG | 1     | 0.75     | 0    |
| 2                | x2     | BASIC  | NON-NEG | 1     | 1.75     | 0    |
| 3                | x3     |        | NON-NEG | 1     | 0        | 0.5  |
| 4                | x4     |        | NON-NEG | 1     | 0        | 0    |
| *INF*            | const1 | BASIC  | SLACK   | 0     | -1       | 0    |
| 6                | const2 |        | SLACK   | 0     | 0        | 0.5  |

**The LP Procedure**

| Constraint Summary |        |           |     |      |          |          |
|--------------------|--------|-----------|-----|------|----------|----------|
| Constraint         |        | S/S       |     | Dual |          |          |
| Row                | Name   | Type      | Col | Rhs  | Activity | Activity |
| 1                  | profit | OBJECTIVE | .   | 0    | 2.5      | .        |
| *INF*              | const1 | LE        | 5   | 5    | 6        | 0        |
| 3                  | const2 | LE        | 6   | 4    | 4        | -0.5     |
| 4                  | const3 | EQ        | .   | 9    | 9        | 0.5      |

### The LP Procedure

| Infeasible Information Summary |        |  |  |  |
|--------------------------------|--------|--|--|--|
| Infeasible Row                 | const1 |  |  |  |
| Constraint Activity            | 6      |  |  |  |
| Row Type                       | LE     |  |  |  |
| Rhs Data                       | 5      |  |  |  |

| Variable | Coefficient | Activity | Lower Bound | Upper Bound |
|----------|-------------|----------|-------------|-------------|
| x1       | 1           | 0.75     | 0           | INFINITY    |
| x2       | 3           | 1.75     | 0           | INFINITY    |
| x3       | 2           | 0        | 0           | INFINITY    |
| x4       | 4           | 0        | 0           | INFINITY    |

Note the information given in the Infeasible Information Summary for the infeasible row CONST1. It shows that the inequality row CONST1 with right-hand side 5 was found to be infeasible with activity 6. The summary also shows each variable that has a nonzero coefficient in that row and its activity level at the infeasibility. Examination of these model parameters might give you a clue as to the cause of infeasibility, such as an incorrectly entered coefficient or right-hand-side value.

### Example 5.10: Restarting an Integer Program

The following example is attributed to Haldi (Garfinkel and Nemhauser 1972) and is used in the literature as a test problem. Notice that the `ACTIVEOUT=` and the `PRIMALOUT=` options are used when invoking PROC LP. These cause the LP procedure to save the primal solution in the data set named P and the active tree in the data set named A. If the procedure fails to find an optimal integer solution on the initial call, it can be called later using the A and P data sets as starting information.

```
data haldi10;
  input x1-x12 _type_ $ _rhs_;
  datalines;
    0  0  0  0  0  0  1  1  1  1  1  1  MAX  .
    9  7 16  8 24  5  3  7  8  4  6  5  LE 110
   12  6  6  2 20  8  4  6  3  1  5  8  LE  95
   15  5 12  4  4  5  5  5  6  2  1  5  LE  80
   18  4  4 18 28  1  6  4  2  9  7  1  LE 100
  -12  0  0  0  0  0  1  0  0  0  0  0  LE  0
    0 -15  0  0  0  0  0  1  0  0  0  0  LE  0
    0  0 -12  0  0  0  0  0  1  0  0  0  LE  0
    0  0  0 -10  0  0  0  0  0  1  0  0  LE  0
    0  0  0  0 -11  0  0  0  0  0  1  0  LE  0
    0  0  0  0  0 -11  0  0  0  0  0  1  LE  0
    1  1  1  1  1  1 1000 1000 1000 1000 1000 1000 UPPERBD .
    1  2  3  4  5  6  7  8  9 10 11 12  INTEGER .
  ;
```

The **ACTIVEOUT=** data set contains a representation of the current active problems in the branch-and-bound tree. The **PRIMALOUT=** data set contains a representation of the solution to the current problem. These two can be used to restore the procedure to an equivalent state to the one it was in when it stopped.

The results from the call to PROC LP is shown in [Output 5.10.1](#). Notice that the procedure performed 100 iterations and then terminated on maximum integer iterations. This is because, by default, **IMAXIT=100**. The procedure reports the current best integer solution.

#### **Output 5.10.1** Output from the HALDI10 Problem

##### **The LP Procedure**

| Problem Summary     |            |
|---------------------|------------|
| Objective Function  | Max _OBS1_ |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 31.82      |
| Variables           |            |
| Number              |            |
| Integer             | 6          |
| Binary              | 6          |
| Slack               | 10         |
| Total               | 22         |
| Constraints         |            |
| Number              |            |
| LE                  | 10         |
| Objective           | 1          |
| Total               | 11         |

## The LP Procedure

| Integer Iteration Log |         |            |           |          |       |         |        |           |
|-----------------------|---------|------------|-----------|----------|-------|---------|--------|-----------|
| Iter                  | Problem | Condition  | Objective | Branched | Value | Sinfeas | Active | Proximity |
| 1                     | 0       | ACTIVE     | 18.709524 | x9       | 1.543 | 1.11905 | 2      | .         |
| 2                     | 1       | ACTIVE     | 18.467723 | x12      | 9.371 | 0.88948 | 3      | .         |
| 3                     | 2       | ACTIVE     | 18.460133 | x8       | 0.539 | 1.04883 | 4      | .         |
| 4                     | -3      | ACTIVE     | 18.453638 | x12      | 8.683 | 1.12993 | 5      | .         |
| 5                     | 4       | ACTIVE     | 18.439678 | x10      | 7.448 | 1.20125 | 6      | .         |
| 6                     | 5       | ACTIVE     | 18.403728 | x6       | 0.645 | 1.3643  | 7      | .         |
| 7                     | -6      | ACTIVE     | 18.048289 | x4       | 0.7   | 1.18395 | 8      | .         |
| 8                     | -7      | ACTIVE     | 17.679087 | x8       | 1.833 | 0.52644 | 9      | .         |
| 9                     | 8       | ACTIVE     | 17.52     | x10      | 6.667 | 0.70111 | 10     | .         |
| 10                    | 9       | ACTIVE     | 17.190085 | x12      | 7.551 | 1.37615 | 11     | .         |
| 11                    | -10     | ACTIVE     | 17.02     | x1       | 0.085 | 0.255   | 12     | .         |
| 12                    | 11      | ACTIVE     | 16.748    | x11      | 0.748 | 0.47    | 13     | .         |
| 13                    | -12     | ACTIVE     | 16.509091 | x9       | 0.509 | 0.69091 | 14     | .         |
| 14                    | 13      | ACTIVE     | 16.261333 | x11      | 1.261 | 0.44267 | 15     | .         |
| 15                    | 14      | ACTIVE     | 16        | x3       | 0.297 | 0.45455 | 16     | .         |
| 16                    | 15      | ACTIVE     | 16        | x5       | 0.091 | 0.15758 | 16     | .         |
| 17                    | -16     | INFEASIBLE | -0.4      | .        | .     | .       | 15     | .         |
| 18                    | -15     | ACTIVE     | 11.781818 | x10      | 1.782 | 0.37576 | 15     | .         |
| 19                    | 18      | ACTIVE     | 11        | x5       | 0.091 | 0.15758 | 15     | .         |
| 20                    | -19     | INFEASIBLE | -6.4      | .        | .     | .       | 14     | .         |
| 21                    | -14     | ACTIVE     | 11.963636 | x5       | 0.182 | 0.28485 | 14     | .         |
| 22                    | -21     | INFEASIBLE | -4.4      | .        | .     | .       | 13     | .         |
| 23                    | -13     | ACTIVE     | 15.281818 | x10      | 4.282 | 0.52273 | 13     | .         |
| 24                    | 23      | ACTIVE     | 15.041333 | x5       | 0.095 | 0.286   | 14     | .         |
| 25                    | -24     | INFEASIBLE | -2.9      | .        | .     | .       | 13     | .         |
| 26                    | 24      | INFEASIBLE | 14        | .        | .     | .       | 12     | .         |
| 27                    | 12      | ACTIVE     | 16        | x3       | 0.083 | 0.15    | 13     | .         |
| 28                    | -27     | ACTIVE     | 15.277778 | x9       | 0.278 | 0.34444 | 14     | .         |
| 29                    | -28     | ACTIVE     | 13.833333 | x10      | 3.833 | 0.23333 | 14     | .         |
| 30                    | 29      | ACTIVE     | 13        | x2       | 0.4   | 0.4     | 15     | .         |
| 31                    | 30      | INFEASIBLE | 12        | .        | .     | .       | 14     | .         |
| 32                    | -30     | SUBOPTIMAL | 10        | .        | .     | .       | 13     | 8         |
| 33                    | 28      | ACTIVE     | 15        | x2       | 0.067 | 0.06667 | 13     | 8         |
| 34                    | -33     | SUBOPTIMAL | 12        | .        | .     | .       | 12     | 6         |
| 35                    | 27      | ACTIVE     | 15        | x2       | 0.067 | 0.06667 | 12     | 6         |
| 36                    | -35     | SUBOPTIMAL | 15        | .        | .     | .       | 11     | 3         |
| 37                    | -11     | FATHOMED   | 14.275    | .        | .     | .       | 10     | 3         |
| 38                    | 10      | ACTIVE     | 16.804848 | x1       | 0.158 | 0.50313 | 11     | 3         |
| 39                    | -38     | FATHOMED   | 14.784    | .        | .     | .       | 10     | 3         |
| 40                    | 38      | ACTIVE     | 16.40381  | x11      | 1.404 | 0.68143 | 11     | 3         |
| 41                    | -40     | ACTIVE     | 16.367677 | x10      | 5.368 | 0.69949 | 12     | 3         |
| 42                    | 41      | ACTIVE     | 16.113203 | x11      | 2.374 | 1.00059 | 12     | 3         |
| 43                    | 42      | ACTIVE     | 16        | x5       | 0.182 | 0.33182 | 12     | 3         |
| 44                    | -43     | FATHOMED   | 13.822222 | .        | .     | .       | 11     | 3         |
| 45                    | -41     | FATHOMED   | 12.642424 | .        | .     | .       | 10     | 3         |
| 46                    | 40      | ACTIVE     | 16        | x5       | 0.229 | 0.37857 | 10     | 3         |

## The LP Procedure

| Integer Iteration Log |         |            |           |          |       |         |        |           |
|-----------------------|---------|------------|-----------|----------|-------|---------|--------|-----------|
| Iter                  | Problem | Condition  | Objective | Branched | Value | Sinfeas | Active | Proximity |
| 47                    | 46      | FATHOMED   | 15 .      | .        | .     | .       | 9      | 3         |
| 48                    | -9      | ACTIVE     | 17.453333 | x7       | 0.453 | 0.64111 | 10     | 3         |
| 49                    | 48      | ACTIVE     | 17.35619  | x11      | 0.356 | 0.53857 | 11     | 3         |
| 50                    | 49      | ACTIVE     | 17        | x5       | 0.121 | 0.27143 | 12     | 3         |
| 51                    | 50      | ACTIVE     | 17        | x3       | 0.083 | 0.15    | 13     | 3         |
| 52                    | -51     | FATHOMED   | 15.933333 | .        | .     | .       | 12     | 3         |
| 53                    | 51      | ACTIVE     | 16        | x2       | 0.067 | 0.06667 | 12     | 3         |
| 54                    | -53     | SUBOPTIMAL | 16 .      | .        | .     | .       | 8      | 2         |
| 55                    | -8      | ACTIVE     | 17.655399 | x12      | 7.721 | 0.56127 | 9      | 2         |
| 56                    | 55      | ACTIVE     | 17.519375 | x10      | 6.56  | 0.76125 | 10     | 2         |
| 57                    | 56      | ACTIVE     | 17.256874 | x2       | 0.265 | 0.67388 | 11     | 2         |
| 58                    | 57      | INFEASIBLE | 17.167622 | .        | .     | .       | 10     | 2         |
| 59                    | -57     | FATHOMED   | 16.521755 | .        | .     | .       | 9      | 2         |
| 60                    | -56     | FATHOMED   | 17.03125  | .        | .     | .       | 8      | 2         |
| 61                    | -55     | ACTIVE     | 17.342857 | x9       | 0.343 | 0.50476 | 8      | 2         |
| 62                    | 61      | ACTIVE     | 17.2225   | x7       | 0.16  | 0.37333 | 9      | 2         |
| 63                    | 62      | ACTIVE     | 17.1875   | x8       | 2.188 | 0.33333 | 9      | 2         |
| 64                    | 63      | ACTIVE     | 17.153651 | x11      | 0.154 | 0.30095 | 10     | 2         |
| 65                    | -64     | FATHOMED   | 12.381818 | .        | .     | .       | 9      | 2         |
| 66                    | 64      | ACTIVE     | 17        | x2       | 0.133 | 0.18571 | 9      | 2         |
| 67                    | -66     | FATHOMED   | 13 .      | .        | .     | .       | 8      | 2         |
| 68                    | -62     | FATHOMED   | 14.2 .    | .        | .     | .       | 7      | 2         |
| 69                    | 7       | FATHOMED   | 15.428583 | .        | .     | .       | 6      | 2         |
| 70                    | 6       | FATHOMED   | 16.75599  | .        | .     | .       | 5      | 2         |
| 71                    | -5      | ACTIVE     | 17.25974  | x6       | 0.727 | 0.82078 | 5      | 2         |
| 72                    | -71     | FATHOMED   | 17.142857 | .        | .     | .       | 4      | 2         |
| 73                    | -4      | ACTIVE     | 18.078095 | x4       | 0.792 | 0.70511 | 5      | 2         |
| 74                    | -73     | ACTIVE     | 17.662338 | x10      | 7.505 | 0.91299 | 5      | 2         |
| 75                    | 74      | ACTIVE     | 17.301299 | x9       | 0.301 | 0.57489 | 5      | 2         |
| 76                    | 75      | ACTIVE     | 17.210909 | x7       | 0.211 | 0.47697 | 5      | 2         |
| 77                    | 76      | FATHOMED   | 17.164773 | .        | .     | .       | 4      | 2         |
| 78                    | 73      | FATHOMED   | 12.872727 | .        | .     | .       | 3      | 2         |
| 79                    | 3       | ACTIVE     | 18.368316 | x10      | 7.602 | 1.20052 | 4      | 2         |
| 80                    | 79      | ACTIVE     | 18.198323 | x7       | 1.506 | 1.85351 | 5      | 2         |
| 81                    | 80      | ACTIVE     | 18.069847 | x12      | 8.517 | 1.67277 | 6      | 2         |
| 82                    | -81     | ACTIVE     | 17.910909 | x4       | 0.7   | 0.73015 | 7      | 2         |
| 83                    | -82     | ACTIVE     | 17.790909 | x7       | 0.791 | 0.54015 | 8      | 2         |
| 84                    | -83     | ACTIVE     | 17.701299 | x9       | 0.701 | 0.62229 | 8      | 2         |
| 85                    | 84      | ACTIVE     | 17.17619  | x6       | 0.818 | 0.45736 | 8      | 2         |
| 86                    | -85     | ACTIVE     | 17.146667 | x11      | 0.147 | 0.24333 | 8      | 2         |
| 87                    | 86      | ACTIVE     | 17        | x1       | 0.167 | 0.16667 | 8      | 2         |
| 88                    | 87      | INFEASIBLE | 16 .      | .        | .     | .       | 7      | 2         |
| 89                    | 83      | ACTIVE     | 17.58     | x11      | 0.58  | 0.73788 | 8      | 2         |
| 90                    | -89     | FATHOMED   | 17.114286 | .        | .     | .       | 7      | 2         |
| 91                    | -80     | ACTIVE     | 18.044048 | x12      | 8.542 | 1.71158 | 8      | 2         |
| 92                    | 91      | ACTIVE     | 17.954536 | x11      | 0.477 | 1.90457 | 9      | 2         |

## The LP Procedure

| Integer Iteration Log |         |           |           |          |       |         |        |           |
|-----------------------|---------|-----------|-----------|----------|-------|---------|--------|-----------|
| Iter                  | Problem | Condition | Objective | Branched | Value | Sinfeas | Active | Proximity |
| 93                    | 92      | ACTIVE    | 17.875084 | x4       | 0.678 | 1.16624 | 10     | 2         |
| 94                    | 93      | FATHOMED  | 13.818182 | .        | .     | .       | 9      | 2         |
| 95                    | -93     | ACTIVE    | 17.231221 | x6       | 0.727 | 0.76182 | 9      | 2         |
| 96                    | -95     | FATHOMED  | 17.085714 | .        | .     | .       | 8      | 2         |
| 97                    | -92     | FATHOMED  | 17.723058 | .        | .     | .       | 7      | 2         |
| 98                    | -91     | FATHOMED  | 16.378788 | .        | .     | .       | 6      | 2         |
| 99                    | 89      | ACTIVE    | 17        | x6       | 0.818 | 0.26515 | 6      | 2         |
| 100                   | -99     | ACTIVE    | 17        | x3       | 0.083 | 0.08333 | 6      | 2         |

**WARNING:** The maximum number of integer iterations has been exceeded. Increase this limit with the 'IMAXIT=' option on the RESET statement.

## The LP Procedure

| Solution Summary                         |              |
|------------------------------------------|--------------|
| Terminated on Maximum Integer Iterations |              |
| Integer Feasible Solution                |              |
| Objective Value                          | 16           |
| Phase 1 Iterations                       | 0            |
| Phase 2 Iterations                       | 13           |
| Phase 3 Iterations                       | 161          |
| Integer Iterations                       | 100          |
| Integer Solutions                        | 4            |
| Initial Basic Feasible Variables         | 12           |
| Time Used (seconds)                      | 0            |
| Number of Inversions                     | 37           |
| Epsilon                                  | 1E-8         |
| Infinity                                 | 1.797693E308 |
| Maximum Phase 1 Iterations               | 100          |
| Maximum Phase 2 Iterations               | 100          |
| Maximum Phase 3 Iterations               | 99999999     |
| Maximum Integer Iterations               | 100          |
| Time Limit (seconds)                     | 120          |



### The LP Procedure

| Variable Summary |               |        |         |       |          |              |
|------------------|---------------|--------|---------|-------|----------|--------------|
| Col              | Variable Name | Status | Type    | Price | Activity | Reduced Cost |
| 1                | x1            | DEGEN  | BINARY  | 0     | 0        | 0            |
| 2                | x2            | ALTER  | BINARY  | 0     | 1        | 0            |
| 3                | x3            |        | BINARY  | 0     | 0        | 12           |
| 4                | x4            | ALTER  | BINARY  | 0     | 1        | 0            |
| 5                | x5            | ALTER  | BINARY  | 0     | 0        | 0            |
| 6                | x6            | ALTER  | BINARY  | 0     | 1        | 0            |
| 7                | x7            |        | INTEGER | 1     | 0        | 1            |
| 8                | x8            |        | INTEGER | 1     | 1        | 1            |
| 9                | x9            | DEGEN  | INTEGER | 1     | 0        | 0            |
| 10               | x10           |        | INTEGER | 1     | 7        | 1            |
| 11               | x11           |        | INTEGER | 1     | 0        | 1            |
| 12               | x12           |        | INTEGER | 1     | 8        | 1            |
| 13               | _OBS2_        | BASIC  | SLACK   | 0     | 15       | 0            |
| 14               | _OBS3_        | BASIC  | SLACK   | 0     | 2        | 0            |
| 15               | _OBS4_        | BASIC  | SLACK   | 0     | 7        | 0            |
| 16               | _OBS5_        | BASIC  | SLACK   | 0     | 2        | 0            |
| 17               | _OBS6_        | ALTER  | SLACK   | 0     | 0        | 0            |
| 18               | _OBS7_        | BASIC  | SLACK   | 0     | 14       | 0            |
| 19               | _OBS8_        |        | SLACK   | 0     | 0        | -1           |
| 20               | _OBS9_        | BASIC  | SLACK   | 0     | 3        | 0            |
| 21               | _OBS10_       | DEGEN  | SLACK   | 0     | 0        | 0            |
| 22               | _OBS11_       | BASIC  | SLACK   | 0     | 3        | 0            |

### The LP Procedure

| Constraint Summary |                 |           |         |     |          |               |
|--------------------|-----------------|-----------|---------|-----|----------|---------------|
| Row                | Constraint Name | Type      | S/S Col | Rhs | Activity | Dual Activity |
| 1                  | _OBS1_          | OBJECTIVE | .       | 0   | 16       | .             |
| 2                  | _OBS2_          | LE        | 13      | 110 | 95       | 0             |
| 3                  | _OBS3_          | LE        | 14      | 95  | 93       | 0             |
| 4                  | _OBS4_          | LE        | 15      | 80  | 73       | 0             |
| 5                  | _OBS5_          | LE        | 16      | 100 | 98       | 0             |
| 6                  | _OBS6_          | LE        | 17      | 0   | 0        | 0             |
| 7                  | _OBS7_          | LE        | 18      | 0   | -14      | 0             |
| 8                  | _OBS8_          | LE        | 19      | 0   | 0        | 1             |
| 9                  | _OBS9_          | LE        | 20      | 0   | -3       | 0             |
| 10                 | _OBS10_         | LE        | 21      | 0   | 0        | 0             |
| 11                 | _OBS11_         | LE        | 22      | 0   | -3       | 0             |

To continue with the solution of this problem, invoke PROC LP with the **ACTIVEIN=** and **PRIMALIN=** options and reset the **IMAXIT=** option. This restores the branch-and-bound tree and simplifies calculating a basic feasible solution from which to start processing.

```
proc lp data=haldi10 activein=a primalin=p imaxit=250;
run;
```

The procedure picks up iterating from a equivalent state to where it left off. The problem will still not be solved when **IMAXIT=250** occurs.

---

### Example 5.11: Alternative Search of the Branch-and-Bound Tree

In this example, the HALDI10 problem from [Example 5.10](#) is solved. However, here the default strategy for searching the branch-and-bound tree is modified. By default, the search strategy has **VARSELECT=FAR**. This means that when searching for an integer variable on which to branch, the procedure uses the one that has a value farthest from an integer value. An alternative strategy has **VARSELECT=PENALTY**. This strategy causes PROC LP to look at the cost, in terms of the objective function, of branching on an integer variable. The procedure looks at **PENALTYDEPTH=** integer variables before choosing the one with the largest cost. This is a much more expensive strategy (in terms of execution time) than the **VARSELECT=FAR** strategy, but it can be beneficial if fewer integer iterations must be done to find an optimal solution.

```
proc lp data=haldi10 varselect=penalty;
run;
```

Compare the number of integer iterations needed to solve the problem using this heuristic with the default strategy used in [Example 5.10](#). In this example, the difference is profound; in general, solution times can vary significantly with the search technique. See [Output 5.11.1](#).

**Output 5.11.1** Summaries and an Integer Programming Iteration Log: Using VARSELECT=PENALTY**The LP Procedure**

| Problem Summary     |            |
|---------------------|------------|
| Objective Function  | Max _OBS1_ |
| Rhs Variable        | _rhs_      |
| Type Variable       | _type_     |
| Problem Density (%) | 31.82      |
|                     |            |
| Variables           | Number     |
|                     |            |
| Integer             | 6          |
| Binary              | 6          |
| Slack               | 10         |
|                     |            |
| Total               | 22         |
|                     |            |
| Constraints         | Number     |
|                     |            |
| LE                  | 10         |
| Objective           | 1          |
|                     |            |
| Total               | 11         |

## Output 5.11.1 continued

| Integer Iteration Log |         |            |           |          |       |         |        |           |
|-----------------------|---------|------------|-----------|----------|-------|---------|--------|-----------|
| Iter                  | Problem | Condition  | Objective | Branched | Value | Sinfeas | Active | Proximity |
| 1                     | 0       | ACTIVE     | 18.709524 | x4       | 0.8   | 1.11905 | 2      | .         |
| 2                     | 1       | ACTIVE     | 16.585187 | x1       | 0.447 | 2.33824 | 3      | .         |
| 3                     | 2       | ACTIVE     | 14.86157  | x5       | 0.221 | 2.09584 | 4      | .         |
| 4                     | 3       | ACTIVE     | 14.807195 | x2       | 0.897 | 1.31729 | 5      | .         |
| 5                     | -4      | ACTIVE     | 14.753205 | x8       | 14.58 | 0.61538 | 6      | .         |
| 6                     | 5       | ACTIVE     | 14.730078 | x6       | 0.043 | 0.79446 | 7      | .         |
| 7                     | -6      | ACTIVE     | 13.755102 | x3       | 0.051 | 0.58163 | 8      | .         |
| 8                     | -7      | ACTIVE     | 11.6      | x8       | 11.6  | 0.4     | 9      | .         |
| 9                     | 8       | ACTIVE     | 11.6      | x12      | 0.6   | 0.4     | 10     | .         |
| 10                    | -9      | ACTIVE     | 11.6      | x8       | 10.6  | 0.4     | 11     | .         |
| 11                    | 10      | ACTIVE     | 11.6      | x12      | 1.6   | 0.4     | 12     | .         |
| 12                    | -11     | ACTIVE     | 11.6      | x8       | 9.6   | 0.4     | 13     | .         |
| 13                    | 12      | ACTIVE     | 11.6      | x12      | 2.6   | 0.4     | 14     | .         |
| 14                    | -13     | ACTIVE     | 11.571429 | x9       | 0.143 | 0.57143 | 15     | .         |
| 15                    | 14      | ACTIVE     | 11.5      | x8       | 8.5   | 0.5     | 16     | .         |
| 16                    | -15     | INFEASIBLE | 9         | .        | .     | .       | 15     | .         |
| 17                    | 15      | ACTIVE     | 11.375    | x12      | 3.375 | 0.375   | 16     | .         |
| 18                    | -17     | ACTIVE     | 11.166667 | x8       | 7.167 | 0.16667 | 17     | .         |
| 19                    | 18      | ACTIVE     | 11.125    | x12      | 4.125 | 0.125   | 18     | .         |
| 20                    | 19      | SUBOPTIMAL | 11        | .        | .     | .       | 7      | 7         |
| 21                    | 7       | ACTIVE     | 13.5      | x8       | 13.5  | 0.5     | 8      | 7         |
| 22                    | -21     | INFEASIBLE | 11        | .        | .     | .       | 7      | 7         |
| 23                    | 21      | ACTIVE     | 13.375    | x12      | 0.375 | 0.375   | 8      | 7         |
| 24                    | -23     | ACTIVE     | 13.166667 | x8       | 12.17 | 0.16667 | 9      | 7         |
| 25                    | 24      | ACTIVE     | 13.125    | x12      | 1.125 | 0.125   | 10     | 7         |
| 26                    | 25      | SUBOPTIMAL | 13        | .        | .     | .       | 4      | 5         |
| 27                    | 6       | ACTIVE     | 14.535714 | x3       | 0.045 | 0.50893 | 5      | 5         |
| 28                    | -27     | FATHOMED   | 12.625    | .        | .     | .       | 4      | 5         |
| 29                    | 27      | SUBOPTIMAL | 14        | .        | .     | .       | 1      | 4         |
| 30                    | -1      | ACTIVE     | 18.309524 | x3       | 0.129 | 1.31905 | 2      | 4         |
| 31                    | 30      | ACTIVE     | 17.67723  | x6       | 0.886 | 0.43662 | 3      | 4         |
| 32                    | 31      | ACTIVE     | 15.485156 | x2       | 0.777 | 1.50833 | 4      | 4         |
| 33                    | -32     | ACTIVE     | 15.2625   | x1       | 0.121 | 1.38333 | 4      | 4         |
| 34                    | 33      | ACTIVE     | 15.085106 | x10      | 3.532 | 0.91489 | 4      | 4         |
| 35                    | 34      | FATHOMED   | 14.857143 | .        | .     | .       | 3      | 4         |
| 36                    | 32      | FATHOMED   | 11.212121 | .        | .     | .       | 2      | 4         |
| 37                    | -31     | ACTIVE     | 17.56338  | x10      | 7.93  | 0.43662 | 3      | 4         |
| 38                    | 37      | ACTIVE     | 17.225962 | x8       | 2.38  | 0.69231 | 4      | 4         |
| 39                    | 38      | ACTIVE     | 17.221818 | x1       | 0.016 | 0.37111 | 5      | 4         |
| 40                    | -39     | FATHOMED   | 14.43662  | .        | .     | .       | 4      | 4         |
| 41                    | 39      | ACTIVE     | 17.172375 | x2       | 0.133 | 0.31948 | 5      | 4         |
| 42                    | 41      | ACTIVE     | 16.890196 | x5       | 0.086 | 0.19608 | 6      | 4         |
| 43                    | 42      | ACTIVE     | 16.75     | x12      | 9.75  | 0.25    | 7      | 4         |
| 44                    | -43     | SUBOPTIMAL | 15        | .        | .     | .       | 6      | 3         |
| 45                    | 43      | SUBOPTIMAL | 16        | .        | .     | .       | 3      | 2         |
| 46                    | -38     | FATHOMED   | 17.138028 | .        | .     | .       | 2      | 2         |

**Output 5.11.1** *continued*

| Integer Iteration Log |         |            |           |          |       |         |                  |
|-----------------------|---------|------------|-----------|----------|-------|---------|------------------|
| Iter                  | Problem | Condition  | Objective | Branched | Value | Sinfeas | Active Proximity |
| 47                    | -37     | SUBOPTIMAL | 17        | .        | .     | .       | 1 1              |
| 48                    | -30     | FATHOMED   | 16.566667 | .        | .     | .       | 0 .              |

| Solution Summary                 |              |
|----------------------------------|--------------|
| Integer Optimal Solution         |              |
| Objective Value                  | 17           |
| Phase 1 Iterations               | 0            |
| Phase 2 Iterations               | 13           |
| Phase 3 Iterations               | 79           |
| Integer Iterations               | 48           |
| Integer Solutions                | 6            |
| Initial Basic Feasible Variables | 12           |
| Time Used (seconds)              | 0            |
| Number of Inversions             | 17           |
| Epsilon                          | 1E-8         |
| Infinity                         | 1.797693E308 |
| Maximum Phase 1 Iterations       | 100          |
| Maximum Phase 2 Iterations       | 100          |
| Maximum Phase 3 Iterations       | 99999999     |
| Maximum Integer Iterations       | 100          |
| Time Limit (seconds)             | 120          |

Output 5.11.1 *continued*

| Variable Summary |         |        |         |       |          |           |
|------------------|---------|--------|---------|-------|----------|-----------|
| Variable         |         | Status | Type    | Price | Activity | Reduced   |
| Col              | Name    |        |         |       |          | Cost      |
| 1                | x1      | DEGEN  | BINARY  | 0     | 0        | 0         |
| 2                | x2      |        | BINARY  | 0     | 0        | -4        |
| 3                | x3      |        | BINARY  | 0     | 0        | -4        |
| 4                | x4      |        | BINARY  | 0     | 1        | -18       |
| 5                | x5      | DEGEN  | BINARY  | 0     | 0        | 0         |
| 6                | x6      |        | BINARY  | 0     | 1        | -1        |
| 7                | x7      |        | INTEGER | 1     | 0        | -6.5      |
| 8                | x8      |        | INTEGER | 1     | 0        | -3        |
| 9                | x9      |        | INTEGER | 1     | 0        | -1        |
| 10               | x10     |        | INTEGER | 1     | 8        | -8        |
| 11               | x11     |        | INTEGER | 1     | 0        | -8.545455 |
| 12               | x12     | BASIC  | INTEGER | 1     | 9        | 0         |
| 13               | _OBS2_  | BASIC  | SLACK   | 0     | 20       | 0         |
| 14               | _OBS3_  | BASIC  | SLACK   | 0     | 5        | 0         |
| 15               | _OBS4_  | BASIC  | SLACK   | 0     | 10       | 0         |
| 16               | _OBS5_  |        | SLACK   | 0     | 0        | -1        |
| 17               | _OBS6_  |        | SLACK   | 0     | 0        | -1.5      |
| 18               | _OBS7_  | DEGEN  | SLACK   | 0     | 0        | 0         |
| 19               | _OBS8_  | DEGEN  | SLACK   | 0     | 0        | 0         |
| 20               | _OBS9_  | BASIC  | SLACK   | 0     | 2        | 0         |
| 21               | _OBS10_ |        | SLACK   | 0     | 0        | -2.545455 |
| 22               | _OBS11_ | BASIC  | SLACK   | 0     | 2        | 0         |

| Constraint Summary |         |           |     |     |      |          |
|--------------------|---------|-----------|-----|-----|------|----------|
| Constraint         |         | Type      | S/S |     | Dual | Activity |
| Row                | Name    |           | Col | Rhs |      |          |
| 1                  | _OBS1_  | OBJECTIVE | .   | 0   | 17   | .        |
| 2                  | _OBS2_  | LE        | 13  | 110 | 90   | 0        |
| 3                  | _OBS3_  | LE        | 14  | 95  | 90   | 0        |
| 4                  | _OBS4_  | LE        | 15  | 80  | 70   | 0        |
| 5                  | _OBS5_  | LE        | 16  | 100 | 100  | 1        |
| 6                  | _OBS6_  | LE        | 17  | 0   | 0    | 1.5      |
| 7                  | _OBS7_  | LE        | 18  | 0   | 0    | 0        |
| 8                  | _OBS8_  | LE        | 19  | 0   | 0    | 0        |
| 9                  | _OBS9_  | LE        | 20  | 0   | -2   | 0        |
| 10                 | _OBS10_ | LE        | 21  | 0   | 0    | 2.545455 |
| 11                 | _OBS11_ | LE        | 22  | 0   | -2   | 0        |

Although the `VARSELECT=PENALTY` strategy works well in this example, there is no guarantee that it will work well with your model. Experimentation with various strategies is necessary to find the one that works well with your model and data, particularly if a model is solved repeatedly with few changes to either the structure or the data.

## Example 5.12: An Assignment Problem

This example departs somewhat from the emphasis of previous ones. Typically, linear programming models are large, have considerable structure, and are solved with some regularity. Some form of automatic model building, or matrix generation as it is commonly called, is a useful aid. The sparse input format provides a great deal of flexibility in model specification so that, in many cases, the DATA step can be used to generate the matrix.

The following assignment problem illustrates some techniques in matrix generation. In this example, you have four machines that can produce any of six grades of cloth, and you have five customers that demand various amounts of each grade of cloth. The return from supplying a customer with a demanded grade depends on the machine on which the cloth was made. In addition, the machine capacity depends both upon the specific machine used and the grade of cloth made.

To formulate this problem, let  $i$  denote customer,  $j$  denote grade, and  $k$  denote machine. Then let  $x_{ijk}$  denote the amount of cloth of grade  $j$  made on machine  $k$  for customer  $i$ ; let  $r_{ijk}$  denote the return from selling one unit of grade  $j$  cloth made on machine  $k$  to customer  $i$ ; let  $d_{ij}$  denote the demand for grade  $j$  cloth by customer  $i$ ; let  $c_{jk}$  denote the number of units of machine  $k$  required to produce one unit of grade  $j$  cloth; and let  $a_k$  denote the number of units of machine  $k$  available. Then, you get

$$\begin{array}{ll} \max & \sum_{ijk} r_{ijk} x_{ijk} \\ \text{subject to} & \sum_k x_{ijk} = d_{ij} \quad \text{for all } i \text{ and } j \\ & \sum_{ij} c_{jk} x_{ijk} \leq a_k \quad \text{for all } k \\ & x_{ijk} \geq 0 \quad \text{for all } i, j \text{ and } k \end{array}$$

The data are saved in three data sets. The OBJECT data set contains the returns for satisfying demand, the DEMAND data set contains the amounts demanded, and the RESOURCE data set contains the conversion factors for each grade and the total amounts of machine resources available.

```

title 'An Assignment Problem';

data object;
  input machine customer
         grade1 grade2 grade3 grade4 grade5 grade6;
  datalines;
1 1 102 140 105 105 125 148
1 2 115 133 118 118 143 166
1 3 70 108 83 83 88 86
1 4 79 117 87 87 107 105
1 5 77 115 90 90 105 148
2 1 123 150 125 124 154 .
2 2 130 157 132 131 166 .
2 3 103 130 115 114 129 .
2 4 101 128 108 107 137 .
2 5 118 145 130 129 154 .
3 1 83 . . 97 122 147
3 2 119 . . 133 163 180
3 3 67 . . 91 101 101
3 4 85 . . 104 129 129
3 5 90 . . 114 134 179
4 1 108 121 79 . 112 132
4 2 121 132 92 . 130 150

```

```

4 3 78 91 59 . 77 72
4 4 100 113 76 . 109 104
4 5 96 109 77 . 105 145
;

data demand;
  input customer
        grade1 grade2 grade3 grade4 grade5 grade6;
  datalines;
1 100 100 150 150 175 250
2 300 125 300 275 310 325
3 400 0 400 500 340 0
4 250 0 750 750 0 0
5 0 600 300 0 210 360
;

data resource;
  input machine
        grade1 grade2 grade3 grade4 grade5 grade6 avail;
  datalines;
1 .250 .275 .300 .350 .310 .295 744
2 .300 .300 .305 .315 .320 . 244
3 .350 . . .320 .315 .300 790
4 .280 .275 .260 . .250 .295 672
;

```

The linear program is built using the DATA step. The model is saved in a SAS data set in the sparse input format for PROC LP. Each section of the following DATA step generates a piece of the linear program. The first section generates the objective function; the next section generates the demand constraints; and the last section generates the machine resource availability constraints.

```

/* build the linear programming model */

data model;
  array grade{6} grade1-grade6;
  length _type_ $ 8 _row_ $ 8 _col_ $ 8;
  keep _type_ _row_ _col_ _coef_;

  ncust=5;
  nmach=4;
  ngrade=6;

/* generate the objective function */

  _type_='MAX';
  _row_='OBJ';
  do k=1 to nmach;
    do i=1 to ncust;
      link readobj;      /* read the objective coefficient data */
      do j=1 to ngrade;
        if grade{j}^=. then do;
          _col_='X' || put(i,1.) || put(j,1.) || put(k,1.);
          _coef_=grade{j};
          output;
        end;
      end;
    end;
  end;

```



```

        end;
    end;
end;
end;

/* generate the demand constraints */

do i=1 to ncust;
    link readdmd;          /* read the demand data */
    do j=1 to ngrade;
        if grade{j}^=. then do;
            _type_='EQ';
            _row_='DEMAND' || put(i,1.) || put(j,1.);
            _col_='_RHS_';
            _coef_=grade{j};
            output;
            _type_=' ';
            do k=1 to nmach;
                _col_='X' || put(i,1.) || put(j,1.) || put(k,1.);
                _coef_=1.0;
                output;
            end;
        end;
    end;
end;

/* generate the machine constraints */

do k=1 to nmach;
    link readres;          /* read the machine data */
    _type_='LE';
    _row_='MACHINE' || put(k,1.);
    _col_='_RHS_';
    _coef_=avail;
    output;
    _type_=' ';
    do i=1 to ncust;
        do j=1 to ngrade;
            if grade{j}^=. then do;
                _col_='X' || put(i,1.) || put(j,1.) || put(k,1.);
                _coef_=grade{j};
                output;
            end;
        end;
    end;
end;

readobj: set object;
return;
readdmd: set demand;
return;
readres: set resource;
return;
run;

```

With the model built and saved in a data set, it is ready for solution using PROC LP. The following program solves the model and saves the solution in the data set called PRIMAL:

```
/* solve the linear program */

proc lp data=model sparsedata noprint primalout=primal;
run;
```

The following output is produced by PROC LP.

### Output 5.12.1 An Assignment Problem

#### An Assignment Problem

##### The LP Procedure

| Problem Summary     |         |
|---------------------|---------|
| Objective Function  | Max OBJ |
| Rhs Variable        | _RHS_   |
| Type Variable       | _type_  |
| Problem Density (%) | 5.31    |
| Variables           |         |
| Number              |         |
| Non-negative        | 120     |
| Slack               | 4       |
| Total               | 124     |
| Constraints         |         |
| Number              |         |
| LE                  | 4       |
| EQ                  | 30      |
| Objective           | 1       |
| Total               | 35      |

**Output 5.12.1** *continued*

| Solution Summary                 |              |
|----------------------------------|--------------|
| Terminated Successfully          |              |
| Objective Value                  | 871426.03763 |
| Phase 1 Iterations               | 0            |
| Phase 2 Iterations               | 40           |
| Phase 3 Iterations               | 0            |
| Integer Iterations               | 0            |
| Integer Solutions                | 0            |
| Initial Basic Feasible Variables | 36           |
| Time Used (seconds)              | 0            |
| Number of Inversions             | 3            |
| Epsilon                          | 1E-8         |
| Infinity                         | 1.797693E308 |
| Maximum Phase 1 Iterations       | 100          |
| Maximum Phase 2 Iterations       | 100          |
| Maximum Phase 3 Iterations       | 99999999     |
| Maximum Integer Iterations       | 100          |
| Time Limit (seconds)             | 120          |

The solution is prepared for reporting using the DATA step, and a report is written using PROC TABULATE.

```

/* report the solution */

data solution;
  set primal;
  keep customer grade machine amount;
  if substr(_var_,1,1)='X' then do;
    if _value_ ^=0 then do;
      customer = substr(_var_,2,1);
      grade    = substr(_var_,3,1);
      machine  = substr(_var_,4,1);
      amount   = _value_;
      output;
    end;
  end;
run;

proc tabulate data=solution;
  class customer grade machine;
  var amount;
  table (machine*customer), (grade*amount);
run;

```

The report shown in [Output 5.12.2](#) gives the assignment of customer, grade of cloth, and machine that maximizes the return and does not violate the machine resource availability.

**Output 5.12.2** An Assignment Problem

**An Assignment Problem**

|         |          | grade  |        |        |        |        |        |
|---------|----------|--------|--------|--------|--------|--------|--------|
|         |          | 1      | 2      | 3      | 4      | 5      | 6      |
|         |          | amount | amount | amount | amount | amount | amount |
|         |          | Sum    | Sum    | Sum    | Sum    | Sum    | Sum    |
| machine | customer |        |        |        |        |        |        |
| 1       | 1        | .      | 100.00 | 150.00 | 150.00 | 175.00 | 250.00 |
|         | 2        | .      | .      | 300.00 | .      | .      | .      |
|         | 3        | .      | .      | 256.72 | 210.31 | .      | .      |
|         | 4        | .      | .      | 750.00 | .      | .      | .      |
|         | 5        | .      | 92.27  | .      | .      | .      | .      |
| 2       | 3        | .      | .      | 143.28 | .      | 340.00 | .      |
|         | 5        | .      | .      | 300.00 | .      | .      | .      |
| 3       | 2        | .      | .      | .      | 275.00 | 310.00 | 325.00 |
|         | 3        | .      | .      | .      | 289.69 | .      | .      |
|         | 4        | .      | .      | .      | 750.00 | .      | .      |
|         | 5        | .      | .      | .      | .      | 210.00 | 360.00 |
| 4       | 1        | 100.00 | .      | .      | .      | .      | .      |
|         | 2        | 300.00 | 125.00 | .      | .      | .      | .      |
|         | 3        | 400.00 | .      | .      | .      | .      | .      |
|         | 4        | 250.00 | .      | .      | .      | .      | .      |
|         | 5        | .      | 507.73 | .      | .      | .      | .      |

### Example 5.13: A Scheduling Problem

Scheduling is an application area where techniques in model generation can be valuable. Problems involving scheduling are often solved with integer programming and are similar to assignment problems. In this example, you have eight one-hour time slots in each of five days. You have to assign four people to these time slots so that each slot is covered on every day. You allow the people to specify preference data for each slot on each day. In addition, there are constraints that must be satisfied:

- Each person has some slots for which they are unavailable.
- Each person must have either slot 4 or 5 off for lunch.
- Each person can work only two time slots in a row.
- Each person can work only a specified number of hours in the week.

To formulate this problem, let  $i$  denote person,  $j$  denote time slot, and  $k$  denote day. Then, let  $x_{ijk} = 1$  if person  $i$  is assigned to time slot  $j$  on day  $k$ , and 0 otherwise; let  $p_{ijk}$  denote the preference of person  $i$  for slot

$j$  on day  $k$ ; and let  $h_i$  denote the number of hours in a week that person  $i$  will work. Then, you get

$$\begin{array}{ll}
 \max & \sum_{ijk} p_{ijk} x_{ijk} \\
 \text{subject to} & \sum_i x_{ijk} = 1 \quad \text{for all } j \text{ and } k \\
 & x_{i4k} + x_{i5k} \leq 1 \quad \text{for all } i \text{ and } k \\
 & x_{i,\ell,k} + x_{i,\ell+1,k} + x_{i,\ell+2,k} \leq 2 \quad \text{for all } i \text{ and } k, \text{ and } \ell = 1, \dots, 6 \\
 & \sum_{jk} x_{ijk} \leq h_i \quad \text{for all } i \\
 & x_{ijk} = 0 \text{ or } 1 \quad \text{for all } i \text{ and } k \text{ such that } p_{ijk} > 0, \\
 & \quad \text{otherwise } x_{ijk} = 0
 \end{array}$$

To solve this problem, create a data set that has the hours and preference data for each individual, time slot, and day. A 10 represents the most desirable time slot, and a 1 represents the least desirable time slot. In addition, a 0 indicates that the time slot is not available.

```

data raw;
  input name $ hour slot mon tue wed thu fri;
  datalines;
marc 20 1 10 10 10 10 10
marc 20 2 9 9 9 9 9
marc 20 3 8 8 8 8 8
marc 20 4 1 1 1 1 1
marc 20 5 1 1 1 1 1
marc 20 6 1 1 1 1 1
marc 20 7 1 1 1 1 1
marc 20 8 1 1 1 1 1
mike 20 1 10 9 8 7 6
mike 20 2 10 9 8 7 6
mike 20 3 10 9 8 7 6
mike 20 4 10 3 3 3 3
mike 20 5 1 1 1 1 1
mike 20 6 1 2 3 4 5
mike 20 7 1 2 3 4 5
mike 20 8 1 2 3 4 5
bill 20 1 10 10 10 10 10
bill 20 2 9 9 9 9 9
bill 20 3 8 8 8 8 8
bill 20 4 0 0 0 0 0
bill 20 5 1 1 1 1 1
bill 20 6 1 1 1 1 1
bill 20 7 1 1 1 1 1
bill 20 8 1 1 1 1 1
bob 20 1 10 9 8 7 6
bob 20 2 10 9 8 7 6
bob 20 3 10 9 8 7 6
bob 20 4 10 3 3 3 3
bob 20 5 1 1 1 1 1
bob 20 6 1 2 3 4 5
bob 20 7 1 2 3 4 5
bob 20 8 1 2 3 4 5
;

```

These data are read by the following DATA step, and an integer program is built to solve the problem. The model is saved in the data set named MODEL. First, the objective function is built using the data saved in the RAW data set. Then, the constraints requiring a person to be working in each time slot are built. Next, the constraints allowing each person time for lunch are added. Then, the constraints restricting people to only two consecutive hours are added. Next, the constraints limiting the time that any one person works in a week are added. Finally, the constraints allowing a person to be assigned only to a time slot for which he is available are added. The code to build each of these constraints follows the formulation closely.

```
data model;
  array workweek{5} mon tue wed thu fri;
  array hours{4} hours1 hours2 hours3 hours4;
  retain hours1-hours4;

  set raw end=eof;

  length _row_ $ 8 _col_ $ 8 _type_ $ 8;
  keep _type_ _col_ _row_ _coef_;

  if      name='marc' then i=1;
  else if name='mike' then i=2;
  else if name='bill' then i=3;
  else if name='bob'  then i=4;

  hours{i}=hour;

/* build the objective function */

do k=1 to 5;
  _col_='x' || put(i,1.) || put(slot,1.) || put(k,1.);

  _row_='object';
  _coef_=workweek{k} * 1000;
  output;
  _row_='upper';
  if workweek{k}^=0 then _coef_=1;
  output;
  _row_='integer';
  _coef_=1;
  output;
end;

/* build the rest of the model */

if eof then do;
  _coef_=.;
  _col_=' ';
  _type_='upper';
  _row_='upper';
  output;
  _type_='max';
  _row_='object';
  output;
  _type_='int';
```

```

_row_='integer';
output;

/* every hour 1 person working */

do j=1 to 8;
  do k=1 to 5;
    _row_='work' || put(j,1.) || put(k,1.);
    _type_='eq';
    _col_='_RHS_';
    _coef_=1;
    output;
    _coef_=1;
    _type_=' ';
    do i=1 to 4;
      _col_='x' || put(i,1.) || put(j,1.) || put(k,1.);
      output;
    end;
  end;
end;

/* each person has a lunch */

do i=1 to 4;
  do k=1 to 5;
    _row_='lunch' || put(i,1.) || put(k,1.);
    _type_='le';
    _col_='_RHS_';
    _coef_=1;
    output;
    _coef_=1;
    _type_=' ';
    _col_='x' || put(i,1.) || '4' || put(k,1.);
    output;
    _col_='x' || put(i,1.) || '5' || put(k,1.);
    output;
  end;
end;

/* work at most 2 slots in a row */

do i=1 to 4;
  do k=1 to 5;
    do l=1 to 6;
      _row_='seq' || put(i,1.) || put(k,1.) || put(l,1.);
      _type_='le';
      _col_='_RHS_';
      _coef_=2;
      output;
      _coef_=1;
      _type_=' ';
      do j=0 to 2;
        _col_='x' || put(i,1.) || put(l+j,1.) || put(k,1.);
        output;
      end;
    end;
  end;
end;

```

```

        end;
    end;
end;
end;

/* work at most n hours in a week */

do i=1 to 4;
    _row_='capacit' || put(i,1.);
    _type_='le';
    _col_='_RHS_';
    _coef_=hours{i};
    output;
    _coef_=1;
    _type_=' ';
    do j=1 to 8;
        do k=1 to 5;
            _col_='x' || put(i,1.) || put(j,1.) || put(k,1.);
            output;
        end;
    end;
end;
end;
end;
run;

```

The model saved in the data set named MODEL is in the sparse format. The constraint that requires one person to work in time slot 1 on day 2 is named WORK12; it is  $\sum_i x_{i12} = 1$ .

The following model is saved in the MODEL data set (which has 1387 observations).

| <u>_TYPE_</u> | <u>_COL_</u> | <u>_ROW_</u> | <u>_COEF_</u> |
|---------------|--------------|--------------|---------------|
| eq            | <u>_RHS_</u> | work12       | 1             |
|               | x112         | work12       | 1             |
|               | x212         | work12       | 1             |
|               | x312         | work12       | 1             |
|               | x412         | work12       | 1             |

The model is solved using the LP procedure. The option **PRIMALOUT=SOLUTION** causes PROC LP to save the primal solution in the data set named SOLUTION.

```

/* solve the linear program */

proc lp sparsedata noprint primalout=solution
    time=1000 maxit1=1000 maxit2=1000;
run;

```



The following DATA step takes the solution data set SOLUTION and generates a report data set named REPORT. It translates the variable names  $x_{ijk}$  so that a more meaningful report can be written. Then, the PROC TABULATE procedure is used to display a schedule showing how the eight time slots are covered for the week.

```

/* report the solution */
title 'Reported Solution';

data report;
  set solution;
  keep name slot mon tue wed thu fri;
  if substr(_var_,1,1)='x' then do;
    if _value_>0 then do;
      n=substr(_var_,2,1);
      slot=substr(_var_,3,1);
      d=substr(_var_,4,1);
      if      n='1' then name='marc';
      else if n='2' then name='mike';
      else if n='3' then name='bill';
      else      name='bob';
      if      d='1' then mon=1;
      else if d='2' then tue=1;
      else if d='3' then wed=1;
      else if d='4' then thu=1;
      else      fri=1;
      output;
    end;
  end;
run;

proc format;
  value xfmt 1='  xxx  ';
run;

proc tabulate data=report;
  class name slot;
  var mon--fri;
  table (slot * name), (mon tue wed thu fri)*sum=' '*f=xfmt.
    /misstext=' ';
run;

```

Output 5.13.1 from PROC TABULATE summarizes the schedule. Notice that the constraint requiring that a person be assigned to each possible time slot on each day is satisfied.

Output 5.13.1 A Scheduling Problem

|      |      | mon | tue | wed | thu | fri |
|------|------|-----|-----|-----|-----|-----|
| slot | name |     |     |     |     |     |
| 1    | bill | xxx | xxx | xxx | xxx | xxx |
| 2    | bob  | xxx |     |     |     |     |
|      | marc |     | xxx | xxx | xxx | xxx |
| 3    | bob  |     | xxx |     |     |     |
|      | marc |     |     | xxx | xxx | xxx |
|      | mike | xxx |     |     |     |     |
| 4    | mike | xxx | xxx | xxx | xxx | xxx |
| 5    | bob  | xxx | xxx | xxx | xxx | xxx |
| 6    | bob  |     | xxx |     | xxx |     |
|      | marc | xxx |     |     |     |     |
|      | mike |     |     | xxx |     | xxx |
| 7    | bill | xxx |     |     |     |     |
|      | bob  |     |     | xxx |     |     |
|      | mike |     | xxx |     | xxx | xxx |
| 8    | bill | xxx |     |     |     |     |
|      | bob  |     |     |     |     | xxx |
|      | mike |     | xxx | xxx | xxx |     |

Recall that PROC LP puts a character string in the macro variable `_ORLP_` that describes the characteristics of the solution on termination. This string can be parsed using macro functions and the information obtained can be used in report writing. The variable can be written to the log with the command

```
%put &_orlp_;
```

which produces Figure 5.1.

Figure 5.1 `_ORLP_` Macro Variable

```
STATUS=SUCCESSFUL PHASE=3 OBJECTIVE=211000 P_FEAS=YES D_FEAS=YES
INT_ITER=0 INT_FEAS=1 ACTIVE=0 INT_BEST=211000 PHASE1_ITER=34
PHASE2_ITER=51 PHASE3_ITER=0
```

From this you learn, for example, that at termination the solution is integer optimal and has an objective value of 211000.

## Example 5.14: A Multicommodity Transshipment Problem with Fixed Charges

The following example illustrates a DATA step program for generating a linear program to solve a multicommodity network flow model that has fixed charges. Consider a network consisting of the following nodes: farm-a, farm-b, farm-c, Chicago, St. Louis, and New York. You can ship four commodities from each farm to Chicago or St. Louis and from Chicago or St. Louis to New York. The following table shows the unit shipping cost for each of the four commodities across each of the arcs. The table also shows the supply (positive numbers) at each of the from nodes and the demand (negative numbers) at each of the to nodes. The fixed charge is a fixed cost for shipping any nonzero amount across an arc. For example, if any amount of any of the four commodities is sent from farm-c to St. Louis, then a fixed charge of 75 units is added to the shipping cost.

**Table 5.13** Farms to Cities Network Problem

| From<br>Node | To<br>Node | Unit Shipping<br>Cost |    |    |    | Supply and Demand |      |     |     | Fixed<br>Charge |
|--------------|------------|-----------------------|----|----|----|-------------------|------|-----|-----|-----------------|
|              |            | 1                     | 2  | 3  | 4  | 1                 | 2    | 3   | 4   |                 |
| farm-a       | Chicago    | 20                    | 15 | 17 | 22 | 100               | 100  | 40  | .   | 100             |
| farm-b       | Chicago    | 15                    | 15 | 15 | 30 | 100               | 200  | 50  | 50  | 75              |
| farm-c       | Chicago    | 30                    | 30 | 10 | 10 | 40                | 100  | 75  | 100 | 100             |
| farm-a       | StLouis    | 30                    | 25 | 27 | 22 | .                 | .    | .   | .   | 150             |
| farm-c       | StLouis    | 10                    | 9  | 11 | 10 | .                 | .    | .   | .   | 75              |
| Chicago      | NY         | 75                    | 75 | 75 | 75 | -150              | -200 | -50 | -75 | 200             |
| StLouis      | NY         | 80                    | 80 | 80 | 80 | .                 | .    | .   | .   | 200             |

The following program is designed to take the data in the form given in the preceding table. It builds the node arc incidence matrix for a network given in this form and adds integer variables to capture the fixed charge using the type of constraints discussed in [Example 5.8](#). The program solves the model using PROC LP, saves the solution in the `PRIMALOUT=` data set named SOLUTION, and displays the solution. The DATA step can be easily modified to handle larger problems with similar structure.

```

title 'Multi-commodity Transshipment Problem with Fixed-Charges';
%macro dooversd;
  _coef_=sd{_i_};
  if sd{_i_}>0 then do;                                /* the node is a supply node */
    _row_=from||' commodity'||put(_i_,2.);
    if from^=' ' then output;
  end;
  else if sd{_i_}<0 then do;                            /* the node is a demand node */
    _row_=to||' commodity'||put(_i_,2.);
    if to^=' ' then output;
  end;
  else if from^=' ' & to^=' ' then do; /* a transshipment node */
    _coef_=0;
    _row_=from||' commodity'||put(_i_,2.); output;
    _row_=to ||' commodity'||put(_i_,2.); output;
  end;
%mend dooversd;

```

```

%macro dooverc;
  _col_=arc||' commodity' ||put(_i_,2.);
  if from^=' ' & to^=' ' then do; /* add node arc incidence matrix */
    _type_='le'; _row_=from||' commodity' ||put(_i_,2.);
    _coef_=1; output;
    _row_=to ||' commodity' ||put(_i_,2.);
    _coef_=-1; output;
    _type_=''; _row_='obj';
    _coef_=c{_i_}; output;
    /* add fixed charge variables */
    _type_='le'; _row_=arc;
    _coef_=1; output;
    _col_='_rhs_';
    _type_='';
    _coef_=0; output;
    _col_=arc||'fx';
    _coef_=-M; output;
    _row_='int';
    _coef_=1; output;
    _row_='obj';
    _coef_=fx; output;
    _row_='upper';
    _coef_=1; output;
  end;
%mend dooverc;

data network;
retain M 1.0e6;
length _col_ $ 22 _row_ $ 22;
keep _type_ _col_ _row_ _coef_ ;
array sd sd1-sd4;
array c c1-c4;

input arc $10. from $ to $ c1 c2 c3 c4 sd1 sd2 sd3 sd4 fx;

/* for the first observation define some of the rows */

if _n_=1 then do;
  _type_='upperbd'; _row_='upper'; output;
  _type_='lowerbd'; _row_='lower'; output;
  _type_='min'; _row_='obj'; output;
  _type_='integer'; _row_='int'; output;
end;

_col_='_rhs_'; _type_='le';

do _i_ = 1 to dim(sd);
  %dooversd;
end;
do _i_ = 1 to dim(c);
  %dooverc;
end;

```

```
datalines;
a-Chicago  farm-a  Chicago  20 15 17 22      100 100  40   .  100
b-Chicago  farm-b  Chicago  15 15 15 30      100 200  50  50   75
c-Chicago  farm-c  Chicago  30 30 10 10       40 100  75 100  100
a-StLouis   farm-a  StLouis  30 25 27 22       .   .   .   .  150
c-StLouis   farm-c  StLouis  10  9 11 10       .   .   .   .   75
Chicago-NY  Chicago NY      75 75 75 75      -150 -200 -50 -75 200
StLous-NY   StLouis NY      80 80 80 80       .   .   .   .  200
;

/* solve the model */

proc lp sparsedata pout=solution noprint;
run;

/* print the solution */

data;
  set solution;
  rename _var_=arc _value_=amount;
  if _value_>=0 & _type_='NON-NEG';
run;

proc print;
  id arc;
  var amount;
run;
```

The results from this example are shown in [Output 5.14.1](#). The **NOPRINT** option in the PROC LP statement suppresses the Variable and Constraint Summary sections. This is useful when solving large models for which a report program is available. Here, the solution is saved in data set SOLUTION and reported using PROC PRINT. The solution shows the amount that is shipped over each arc.

**Output 5.14.1** Multicommodity Transshipment Problem with Fixed Charges  
**Multi-commodity Transshipment Problem with Fixed-Charges**

| arc                    | amount |
|------------------------|--------|
| a-Chicago commodity 1  | 10     |
| b-Chicago commodity 1  | 100    |
| b-Chicago commodity 2  | 100    |
| c-Chicago commodity 3  | 50     |
| c-Chicago commodity 4  | 75     |
| c-StLouis commodity 1  | 40     |
| c-StLouis commodity 2  | 100    |
| Chicago-NY commodity 1 | 110    |
| Chicago-NY commodity 2 | 100    |
| Chicago-NY commodity 3 | 50     |
| Chicago-NY commodity 4 | 75     |
| StLous-NY commodity 1  | 40     |
| StLous-NY commodity 2  | 100    |

### Example 5.15: Converting to an MPS-Format SAS Data Set

This example demonstrates the use of the `MPSOUT=` option to convert problem data in PROC LP input format into an MPS-format SAS data set for use with the OPTLP procedure.

Consider the oil blending problem introduced in the section “An Introductory Example” on page 171. Suppose you have saved the problem data in dense format by using the following DATA step:

```
data exdata;
  input _id_ $17. a_light a_heavy brega naphthal naphthai
           heatingo jet_1 jet_2 _type_ $ _rhs_;
datalines;
profit          -175 -165 -205  0  0  0 300 300 max      .
naphtha_1_conv   .035 .030 .045 -1  0  0  0  0 eq      0
naphtha_i_conv   .100 .075 .135  0 -1  0  0  0 eq      0
heating_o_conv   .390 .300 .430  0  0 -1  0  0 eq      0
recipe_1         0    0    0  0 .3 .7 -1  0 eq      0
recipe_2         0    0    0 .2  0 .8  0 -1 eq      0
available        110  165  80  .  .  .  .  . upperbd .
;
```

If you decide to solve the problem by using the OPTLP procedure, you will need to convert the data set `exdata` from dense format to MPS format. You can accomplish this by using the following statements:

```
proc lp data=exdata mpsout=mpsdata;
run;
```

The MPS-format SAS data set mpsdata is shown in [Output 5.15.1](#).

**Output 5.15.1** Data Set mpsdata

| Obs | FIELD1  | FIELD2         | FIELD3         | FIELD4   | FIELD5         | FIELD6 |
|-----|---------|----------------|----------------|----------|----------------|--------|
| 1   | NAME    |                | PROBLEM        | .        |                | .      |
| 2   | ROWS    |                |                | .        |                | .      |
| 3   | MAX     | profit         |                | .        |                | .      |
| 4   | E       | naphtha_l_conv |                | .        |                | .      |
| 5   | E       | naphtha_i_conv |                | .        |                | .      |
| 6   | E       | heating_o_conv |                | .        |                | .      |
| 7   | E       | recipe_1       |                | .        |                | .      |
| 8   | E       | recipe_2       |                | .        |                | .      |
| 9   | COLUMNS |                |                | .        |                | .      |
| 10  |         | a_light        | profit         | -175.000 | naphtha_l_conv | 0.035  |
| 11  |         | a_light        | naphtha_i_conv | 0.100    | heating_o_conv | 0.390  |
| 12  |         | a_heavy        | profit         | -165.000 | naphtha_l_conv | 0.030  |
| 13  |         | a_heavy        | naphtha_i_conv | 0.075    | heating_o_conv | 0.300  |
| 14  |         | brega          | profit         | -205.000 | naphtha_l_conv | 0.045  |
| 15  |         | brega          | naphtha_i_conv | 0.135    | heating_o_conv | 0.430  |
| 16  |         | naphthal       | naphtha_l_conv | -1.000   | recipe_2       | 0.200  |
| 17  |         | naphthai       | naphtha_i_conv | -1.000   | recipe_1       | 0.300  |
| 18  |         | heatingo       | heating_o_conv | -1.000   | recipe_1       | 0.700  |
| 19  |         | heatingo       | recipe_2       | 0.800    |                | .      |
| 20  |         | jet_1          | profit         | 300.000  | recipe_1       | -1.000 |
| 21  |         | jet_2          | profit         | 300.000  | recipe_2       | -1.000 |
| 22  | BOUNDS  |                |                | .        |                | .      |
| 23  | UP      | .BOUNDS.       | a_light        | 110.000  |                | .      |
| 24  | UP      | .BOUNDS.       | a_heavy        | 165.000  |                | .      |
| 25  | UP      | .BOUNDS.       | brega          | 80.000   |                | .      |
| 26  | ENDATA  |                |                | .        |                | .      |

Now that the problem data are in MPS format, you can solve the problem by using the OPTLP procedure. For more information, see Chapter 12, “The OPTLP Procedure” (*SAS/OR User’s Guide: Mathematical Programming*).

## Example 5.16: Migration to OPTMODEL: Assignment

The following example shows how to solve [Example 5.12](#) using PROC OPTMODEL. The OBJECT, DEMAND, and RESOURCE data sets are the same as in that example. A new data set, GRADE, is added to aid in separating the data from the model.

```
title 'An Assignment Problem';

data grade(drop=i);
  do i = 1 to 6;
    grade = 'grade' || put(i,1.);
    output;
  end;
run;
```

The following PROC OPTMODEL statements read the data sets, build the linear programming model, solve the model, and output the optimal solution to a SAS data set called SOLUTION:

```
proc optmodel;
  /* declare index sets */
  set CUSTOMERS;
  set <str> GRADES;
  set MACHINES;

  /* declare parameters */
  num return {CUSTOMERS, GRADES, MACHINES} init 0;
  num demand {CUSTOMERS, GRADES};
  num cost {GRADES, MACHINES} init 0;
  num avail {MACHINES};

  /* read the set of grades */
  read data grade into GRADES=[grade];

  /* read the set of customers and their demands */
  read data demand
    into CUSTOMERS=[customer]
    {j in GRADES} <demand[customer,j]=col(j)>;

  /* read the set of machines, costs, and availability */
  read data resource nomiss
    into MACHINES=[machine]
    {j in GRADES} <cost[j,machine]=col(j)>
    avail;

  /* read objective data */
  read data object nomiss
    into [machine customer]
    {j in GRADES} <return[customer,j,machine]=col(j)>;

  /* declare the model */
  var AmountProduced {CUSTOMERS, GRADES, MACHINES} >= 0;
  max TotalReturn = sum {i in CUSTOMERS, j in GRADES, k in MACHINES}
    return[i,j,k] * AmountProduced[i,j,k];
```



```

con req_demand {i in CUSTOMERS, j in GRADES}:
    sum {k in MACHINES} AmountProduced[i,j,k] = demand[i,j];
con req_avail {k in MACHINES}:
    sum {i in CUSTOMERS, j in GRADES}
        cost[j,k] * AmountProduced[i,j,k] <= avail[k];

/* call the solver and save the results */
solve;
create data solution
    from [customer grade machine] = {i in CUSTOMERS, j in GRADES,
        k in MACHINES: AmountProduced[i,j,k].sol ne 0}
    amount=AmountProduced;

/* print optimal solution */
print AmountProduced;
quit;

```

The statements use both numeric (NUM) and character (STR) index sets, which are populated from the corresponding data set variables in the READ DATA statements. The OPTMODEL parameters can be either single-dimensional (AVAIL) or multiple-dimensional (COST, DEMAND, RETURN). The RETURN and COST parameters are given initial values of 0, and the NOMISS option in the READ DATA statement tells OPTMODEL to read only the nonmissing values from the input data sets. The model declaration is nearly identical to the mathematical formulation. The logical condition `AmountProduced[i,j,k].sol ne 0` in the CREATE DATA statement makes sure that only the nonzero parts of the solution appear in the SOLUTION data set. In [Example 5.12](#), the creation of this data set required postprocessing of the PROC LP output data set.

The main point is that the PROC OPTMODEL statements are much easier to read and maintain than the corresponding DATA step statements required for PROC LP.

The SOLUTION data set can be processed by PROC TABULATE as follows to create a compact representation of the solution:

```

proc tabulate data=solution;
    class customer grade machine;
    var    amount;
    table (machine*customer), (grade*amount=' '*sum=' ');
run;

```

The output is the same as [Output 5.12.2](#). The log is displayed in [Output 5.16.1](#).

### Output 5.16.1 OPTMODEL Log

---

```
NOTE: There were 6 observations read from the data set WORK.GRADE.
NOTE: There were 5 observations read from the data set WORK.DEMAND.
NOTE: There were 4 observations read from the data set WORK.RESOURCE.
NOTE: There were 20 observations read from the data set WORK.OBJECT.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 120 variables (0 free, 0 fixed).
NOTE: The problem has 34 linear constraints (4 LE, 30 EQ, 0 GE, 0 range).
NOTE: The problem has 220 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The OPTMODEL presolver is disabled for linear problems.
NOTE: The LP presolver value AUTOMATIC is applied.
NOTE: The LP presolver removed 43 variables and 7 constraints.
NOTE: The LP presolver removed 66 constraint coefficients.
NOTE: The presolved problem has 77 variables, 27 constraints, and 154
      constraint coefficients.
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         | 2.753237E+06 | 0    |
| D 2   | 55        | 8.714553E+05 | 0    |
| D 2   | 59        | 8.714260E+05 | 0    |

```
NOTE: Optimal.
NOTE: Objective = 871426.03763.
NOTE: The Dual Simplex solve time is 0.00 seconds.
NOTE: The data set WORK.SOLUTION has 26 observations and 4 variables.
```

---

## Example 5.17: Migration to OPTMODEL: Multicommodity Transshipment

The following example shows how to solve [Example 5.14](#) using PROC OPTMODEL. Three data sets contain the input data used in that example.

```
title 'Multicommodity Transshipment Problem with Fixed Charges';

data commodity_data;
  do c = 1 to 4;
    output;
  end;
run;

data arc_data;
  input from $ to $ c1 c2 c3 c4 fx;
  datalines;
farm-a Chicago 20 15 17 22 100
farm-b Chicago 15 15 15 30 75
```

```

farm-c  Chicago 30 30 10 10 100
farm-a  StLouis 30 25 27 22 150
farm-c  StLouis 10 9 11 10 75
Chicago NY      75 75 75 75 200
StLouis NY      80 80 80 80 200
;
run;

data supply_data;
  input node $ sd1 sd2 sd3 sd4;
  datalines;
farm-a  100 100 40 .
farm-b  100 200 50 50
farm-c   40 100 75 100
NY      -150 -200 -50 -75
;
run;

```

The following PROC OPTMODEL statements read the data sets, print the input parameters, build the mixed-integer linear programming model, solve the model, and output the optimal solution to a SAS data set called SOLUTION:

```

proc optmodel;
  set COMMODITIES;
  read data commodity_data into COMMODITIES=[c];

  set <str,str> ARCS;
  num unit_cost {ARCS, COMMODITIES};
  num fixed_charge {ARCS};
  read data arc_data into ARCS=[from to] {c in COMMODITIES}
    <unit_cost[from,to,c]=col('c' || c)> fixed_charge=fx;
  print unit_cost fixed_charge;

  set <str> NODES = union {<i,j> in ARCS} {i,j};
  num supply {NODES, COMMODITIES} init 0;
  read data supply_data nomiss into [node] {c in COMMODITIES}
    <supply[node,c]=col('sd' || c)>;
  print supply;

  var AmountShipped {ARCS, c in COMMODITIES} >= 0 <= sum {i in NODES}
    max(supply[i,c],0);

  /* UseArc[i,j] = 1 if arc (i,j) is used, 0 otherwise */
  var UseArc {ARCS} binary;

  /* TotalCost = variable costs + fixed charges */
  min TotalCost = sum {<i,j> in ARCS, c in COMMODITIES}
    unit_cost[i,j,c] * AmountShipped[i,j,c]
    + sum {<i,j> in ARCS} fixed_charge[i,j] * UseArc[i,j];

  con flow_balance {i in NODES, c in COMMODITIES}:
    sum {<(i,j)> in ARCS} AmountShipped[i,j,c] -
    sum {<j,(i)> in ARCS} AmountShipped[j,i,c] <= supply[i,c];

```

```

/* if AmountShipped[i,j,c] > 0 then UseArc[i,j] = 1 */
con fixed_charge_def {<i,j> in ARCS, c in COMMODITIES}:
    AmountShipped[i,j,c] <= AmountShipped[i,j,c].ub * UseArc[i,j];

solve;

print AmountShipped;

create data solution from [from to commodity]={<i,j> in ARCS,
c in COMMODITIES: AmountShipped[i,j,c].sol ne 0} amount=AmountShipped;
quit;

```

Although [Example 5.14](#) used  $M = 1.0e6$  in the `FIXED_CHARGE_DEF` constraint that links the continuous variable to the binary variable, it is numerically preferable to use a smaller, data-dependent value. Here, the upper bound on `AmountShipped[i,j,c]` is used instead. This upper bound is calculated in the first `VAR` statement as the sum of all positive supplies for commodity  $c$ . The logical condition `AmountShipped[i,j,k].sol ne 0` in the `CREATE DATA` statement makes sure that only the nonzero parts of the solution appear in the `SOLUTION` data set.

The optimal solution is the same as in [Output 5.14.1](#). The log is displayed in [Output 5.17.1](#).

#### Output 5.17.1 OPTMODEL Log

---

```

NOTE: There were 4 observations read from the data set WORK.COMMODITY_DATA.
NOTE: There were 7 observations read from the data set WORK.ARC_DATA.
NOTE: There were 4 observations read from the data set WORK.SUPPLY_DATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 35 variables (0 free, 0 fixed).
NOTE: The problem has 7 binary and 0 integer variables.
NOTE: The problem has 52 linear constraints (52 LE, 0 EQ, 0 GE, 0 range).
NOTE: The problem has 112 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The OPTMODEL presolver is disabled for linear problems.
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 6 variables and 14 constraints.
NOTE: The MILP presolver removed 26 constraint coefficients.
NOTE: The MILP presolver modified 22 constraint coefficients.
NOTE: The presolved problem has 29 variables, 38 constraints, and 86 constraint
coefficients.
NOTE: The MILP solver is called.

```

| Node | Active | Sols | BestInteger   | BestBound     | Gap    | Time |
|------|--------|------|---------------|---------------|--------|------|
| 0    | 1      | 2    | 43180.0000000 | 26450.0000000 | 63.25% | 0    |
| 0    | 1      | 2    | 43180.0000000 | 42610.0000000 | 1.34%  | 0    |
| 0    | 1      | 3    | 42825.0000000 | 42824.9900000 | 0.00%  | 0    |

```

NOTE: The MILP solver added 8 cuts with 28 cut coefficients at the root.
NOTE: Optimal within relative gap.
NOTE: Objective = 42825.
NOTE: The data set WORK.SOLUTION has 14 observations and 4 variables.

```

---

## References

- Bartels, R. (1971), “A Stabilization of the Simplex Method,” *Numerical Mathematics*, 16, 414–434.
- Bland, R. G. (1977), “New Finite Pivoting Rules for the Simplex Method,” *Mathematics of Operations Research*, 2, 103–107.
- Breau, R. and Burdet, C. A. (1974), “Branch and Bound Experiments in Zero-One Programming,” *Mathematical Programming Study*, 2, 1–50.
- Crowder, H., Johnson, E. L., and Padberg, M. W. (1983), “Solving Large-Scale Zero-One Linear Programming Problems,” *Operations Research*, 31, 803–834.
- Dantzig, G. B. (1963), *Linear Programming and Extensions*, Princeton, NJ: Princeton University Press.
- Garfinkel, R. S. and Nemhauser, G. L. (1972), *Integer Programming*, New York: John Wiley & Sons.
- Greenberg, H. J. (1978), “Pivot Selection Tactics,” in H. J. Greenberg, ed., *Design and Implementation of Optimization Software*, 143–174, Leiden, Netherlands: Sijthoff & Noordhoff.
- Hadley, G. (1962), *Linear Programming*, Reading, MA: Addison-Wesley.
- Harris, P. M. J. (1975), “Pivot Selection Methods of the Devex LP Code,” *Mathematical Programming Study*, 4, 30–57.
- Ignizio, J. P. (1976), *Goal Programming and Extensions*, Lexington, MA: D. C. Heath.
- Murtagh, B. A. (1981), *Advanced Linear Programming: Computation and Practice*, New York: McGraw-Hill.
- Nelson, M. (1992), *The Data Compression Book*, New York: M&T Books.
- Reid, J. K. (1975), *A Sparsity-Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases*, Technical Report Harwell CSS 20, Atomic Energy Research Establishment, Harwell, UK.
- Reid, J. K. (1976), *Fortran Subroutines for Handling Sparse Linear Programming Bases*, Technical Report Harwell AERE R 8269, Atomic Energy Research Establishment, Harwell, UK.
- Savelsbergh, M. W. P. (1994), “Preprocessing and Probing Techniques for Mixed Integer Programming Problems,” *ORSA Journal on Computing*, 6, 445–454.
- Taha, H. A. (1975), *Integer Programming*, New York: Academic Press.

# Subject Index

- backtracking rules
  - LP procedure, 186, 214
- branch-and-bound, 210
  - branching variable, 214, 215
  - breadth-first search, 213
  - control options, 186, 213
  - depth-first search, 213
- coefficients
  - LP procedure, 192
- columns
  - LP procedure, 192
- complete pricing
  - LP procedure, 191
- constraint summary
  - LP procedure, 228, 237
- current tableau
  - LP procedure, 207
- data compression
  - LP procedure, 188
- dense input format
  - LP procedure, 170
- devex method, 190
- displayed output
  - LP procedure, 184, 193, 226–229
- dual activities
  - LP procedure, 206
- dynamic pricing
  - LP procedure, 191
- examples, *see* LP examples
- functional summary
  - LP procedure, 179
- goal-programming model, 190
- infeasibility
  - LP procedure, 184, 209, 228
- infeasible information summary
  - LP procedure, 228
- infinity
  - LP procedure, 190
- input data sets
  - LP procedure, 183, 226
- integer iteration log, 212
- integer programs, 168, 210
- interactive processing
  - LP procedure, 175, 185, 221, 222
- iteration log
  - integer iteration log (LP), 212
  - LP procedure, 184, 230
- linear programming problems, 168
  - Bartels-Golub update, 169
- LP examples, 232
  - assignment problem, 277
  - assignment problem (OPTMODEL), 294
  - blending problem, 171, 233
  - branch-and-bound search, 272
  - converting PROC LP format to MPS format, 292
  - fixed charges, 289, 296
  - goal programming, 254
  - infeasibilities, 263
  - integer program, 260
  - introductory example, 171
  - mixed-integer program, 176
  - MPS file conversion, 177
  - multicommodity transshipment problem, 289
  - multicommodity transshipment problem (OPTMODEL), 296
  - preprocessing, 176
  - price parametric programming, 247
  - price sensitivity analysis, 241
  - product mix problem, 254
  - range analysis, 244
  - restarting a problem, 244
  - restarting an integer program, 266
  - scheduling problem, 282
  - sparse data format, 238
  - special ordered sets, 251
- LP procedure
  - backtracking rules, 186, 214
  - branch-and-bound, 186, 210, 213
  - coefficients, 192
  - columns, 192
  - complete pricing, 191
  - constraint summary, 228, 237
  - current tableau, 207
  - customizing search heuristics, 215
  - data compression, 188
  - data set options, 183
  - dense format, 170
  - details, 201
  - devex method, 190
  - displayed output, 184, 193, 226–230

- dual activities, 206
- dynamic pricing, 191
- functional summary, 179
- infeasible information summary, 228
- input data sets, 183, 226
- integer iteration log, 212
- integer programs, 210
- interactive processing, 175, 185, 221, 222
- introductory example, 171
- iteration log, 184, 230
- memory limit, 231
- memory requirements, 223
- missing values, 201
- mixed-integer programs, 210
- MPS file conversion, 177, 204
- multiple pricing, 191, 208
- ODS table names, 230
- ODS variable names, 230, 231
- options classified by function, 179
- \_ORLP\_ macro variable, 207
- output data sets, 183, 223–225
- Output Delivery System (ODS), 230
- overview, 168
- parametric programming, 189, 220, 222
- partial pricing, 191
- pause processing, 185
- preprocessing, 176, 185, 209, 210
- price parametric programming, 221
- price sensitivity analysis, 218, 229
- pricing strategies, 208
- problem definition statements, 174
- problem input, 173
- problem summary, 227, 234
- projected objective value, 213
- projected pseudocost, 214
- range analysis, 189, 219
- range coefficient, 195
- reduced costs, 206
- reset options, 196
- right-hand-side constants, 197
- right-hand-side parametric programming, 220
- right-hand-side sensitivity analysis, 217, 229
- rows, 193, 197
- scaling input data, 192, 209
- sensitivity analysis, 189, 217, 222
- simplex algorithm control options, 190
- solution summary, 227, 234
- sparse format, 171, 183, 201
- suppress printing, 184
- syntax skeleton, 178
- table of syntax elements, 179
- terminate processing, 195
- tolerance, 184, 186, 187, 190–192
- TYPE variable, 198, 202, 236
- variables, 170, 171, 200, 224, 225, 227, 235
- macro variable
  - \_ORLP\_, 207
  - ORLP, 207
- memory requirements
  - LP procedure, 223
- migration to PROC OPTMODEL
  - from PROC LP, 294, 296
- missing values
  - LP procedure, 201
- mixed-integer programs, 168
  - form of, 168
  - LP procedure, 210
- MPS file conversion
  - LP procedure, 177, 204
- multiple pricing
  - LP procedure, 191, 208
- objective function
  - LP procedure, 169
- ODS variable names
  - LP procedure, 231
- options classified by function, *see* functional summary
- \_ORLP\_ macro variable, 207
- output data sets
  - LP procedure, 183, 223–225
- Output Delivery System (ODS)
  - LP procedure, 230
- overview
  - LP procedure, 168
- parametric control options
  - LP procedure, 189
- parametric programming, 189, 220, 222
- partial pricing
  - LP procedure, 191
- pause processing
  - LP procedure, 185
- preprocessing
  - LP procedure, 176, 185, 209, 210
- price parametric programming, 221
- price sensitivity analysis, 218, 229
- pricing strategies
  - LP procedure, 208
- problem definition statements
  - LP procedure, 174
- problem summary
  - LP procedure, 227, 234
- projected objective value
  - LP procedure, 213
- projected pseudocost
  - LP procedure, 214
- range analysis, 189, 219

- range coefficient
  - LP procedure, 195
- ranging control options
  - LP procedure, 189
- reduced costs
  - LP procedure, 206
- right-hand-side constants
  - LP procedure, 197
- right-hand-side parametric programming, 220
- right-hand-side sensitivity analysis, 217, 229
- rows
  - LP procedure, 193, 197
- scaling input data
  - LP procedure, 192, 209
- sensitivity analysis, 217, 222
- sensitivity control options
  - LP procedure, 189
- simplex algorithm control options
  - LP procedure, 190
- solution summary
  - LP procedure, 227, 234
- sparse input format
  - LP procedure, 171, 183, 201
- special ordered set, 199
- syntax skeleton
  - LP procedure, 178
- table of syntax elements, *see* functional summary
- tableau
  - display current, 207
- tolerance
  - LP procedure, 184, 186, 187, 190–192
- TYPE variable
  - LP procedure, 198, 202, 236
- variables
  - LP procedure, 200, 224, 225, 227, 235





# Syntax Index

- ACTIVEIN= option
  - PROC LP statement, [183, 217, 226](#)
- ACTIVEOUT= option
  - PROC LP statement, [183, 217, 224](#)
- AUTO option
  - PROC LP statement, [186, 216, 217](#)
- BACKTRACK= option
  - PROC LP statement, [186, 214](#)
- BASIC keyword
  - TYPE variable (LP), [200](#)
- BEST option
  - PRINT statement (LP), [194](#)
- BINARY keyword
  - TYPE variable (LP), [200](#)
- BINFST option
  - PROC LP statement, [186](#)
- BOTH keyword
  - SCALE= option (LP), [192, 209](#)
- CANSELECT= option
  - PROC LP statement, [187, 213, 216, 217](#)
- CLOSE keyword
  - VARSELECT= option (LP), [189, 215](#)
- COEF statement
  - LP procedure, [192](#)
- COL statement
  - LP procedure, [192](#)
- COLUMN keyword
  - SCALE= option (LP), [192, 209](#)
- COLUMN option
  - PRINT statement (LP), [194](#)
- COMPLETE keyword
  - PRICETYPE= option (LP), [191, 209](#)
- CONTROL= option
  - PROC LP statement, [186, 187, 216, 217](#)
- DATA= option
  - PROC LP statement, [183](#)
- DELTAIT= option
  - PROC LP statement, [187](#)
- DEVEX option
  - PROC LP statement, [190](#)
- DOBJECTIVE= option
  - PROC LP statement, [187](#)
- DUALOUT= option
  - PROC LP statement, [183, 225](#)
- DYNAMIC keyword
  - PRICETYPE= option (LP), [191, 209](#)
- ENDPAUSE option
  - PROC LP statement, [185, 222](#)
- EPSILON= option
  - PROC LP statement, [190](#)
- EQ keyword
  - TYPE variable (LP), [199](#)
- ERROR keyword
  - BACKTRACK= option (LP), [186](#)
  - CANSELECT= option (LP), [187, 214](#)
- FAR keyword
  - VARSELECT= option (LP), [189, 215](#)
- FEASIBLEPAUSE option
  - PROC LP statement, [185, 222](#)
- FIFO keyword
  - BACKTRACK= option (LP), [186](#)
  - CANSELECT= option (LP), [187, 213](#)
- FIXED keyword
  - TYPE variable (LP), [199, 210](#)
- FLOW option
  - PROC LP statement, [184](#)
- FREE keyword
  - TYPE variable (LP), [200, 210](#)
- FUZZ= option
  - PROC LP statement, [184](#)
- GE keyword
  - TYPE variable (LP), [199](#)
- GOALPROGRAM option
  - PROC LP statement, [190](#)
- ID statement
  - LP procedure, [193](#)
- IEPSILON= option
  - PROC LP statement, [187](#)
- IFEASIBLEPAUSE= option
  - PROC LP statement, [185, 222](#)
- IMAXIT= option
  - PROC LP statement, [187](#)
- INFINITY= option
  - PROC LP statement, [190](#)
- INTEGER keyword
  - TYPE variable (LP), [199, 210](#)
- INTEGER\_NONZEROS option
  - PRINT statement (LP), [194](#)
- INTEGER option
  - PRINT statement (LP), [194](#)
- INTEGER\_ZEROS option
  - PRINT statement (LP), [194](#)

- INVFREQ= option
  - PROC LP statement, 190
- INVTOL= option
  - PROC LP statement, 190
- IOBJECTIVE= option
  - PROC LP statement, 187
- IPAUSE= option
  - PROC LP statement, 185, 222
- IPIVOT statement
  - LP procedure, 193, 222
- LE keyword
  - TYPE variable (LP), 199
- LIFO keyword
  - BACKTRACK= option (LP), 186
  - CANSELECT= option (LP), 187, 213, 216
- LIFOTYPE= option
  - PROC LP statement, 188
- LOWER= option
  - RESET statement (LP), 196
- LOWERBD keyword
  - TYPE variable (LP), 199
- LP procedure, 178
  - COEF statement, 192
  - COL statement, 192
  - ID statement, 193
  - IPIVOT statement, 193
  - PIVOT statement, 193
  - PRINT statement, 193
  - PROC LP statement, 183
  - QUIT statement, 195
  - RANGE statement, 195
  - RESET statement, 196
  - RHS statement, 197
  - RHSEN statement, 197
  - ROW statement, 197
  - RUN statement, 198
  - SHOW statement, 198
  - TYPE statement, 198
  - VAR statement, 200
- MATRIX option
  - PRINT statement (LP), 194
- MAX keyword
  - TYPE variable (LP), 198
- MAXIT1= option
  - PROC LP statement, 190
- MAXIT2= option
  - PROC LP statement, 191
- MAXIT3= option
  - PROC LP statement, 191
- MAXIT= option
  - PROC LP statement, 190
- MIN keyword
  - TYPE variable (LP), 198
- MPSOUT= option
  - PROC LP statement, 183, 203, 225
- NOAUTO option
  - PROC LP statement, 188
- NOBINFAST option
  - PROC LP statement, 188
- NODEVEX option
  - PROC LP statement, 191
- NOENDPAUSE option
  - PROC LP statement, 185
- NOFEASIBLEPAUSE option
  - PROC LP statement, 185
- NOFLOW option
  - PROC LP statement, 184
- NONE keyword
  - PRICETYPE= option (LP), 191
  - SCALE= option (LP), 192, 209
- NONINTEGER\_NONZEROS option
  - PRINT statement (LP), 194
- NONINTEGER option
  - PRINT statement (LP), 194
- NONZEROS option
  - PRINT statement (LP), 195
- NOPARAPRINT option
  - PROC LP statement, 184
- NOPOSTPROCESS option
  - PROC LP statement, 188
- NOPREPROCESS option
  - PROC LP statement, 185
- NOPRINT option
  - PROC LP statement, 184
- NORANGEPRICE option
  - PROC LP statement, 189
- NORANGERHS option
  - PROC LP statement, 189
- NOTABLEAUPRINT option
  - PROC LP statement, 184
- OBJ keyword
  - BACKTRACK= option (LP), 186
  - CANSELECT= option (LP), 187, 213, 216
- OPTIONS option
  - SHOW statement (LP), 198
- PARAPRINT option
  - PROC LP statement, 184, 220, 221
- PARARESTORE option
  - PROC LP statement, 191
- PARTIAL keyword
  - PRICETYPE= option (LP), 191, 209
- PAUSE= option
  - PROC LP statement, 185, 222
- PENALTY keyword

- VARSELECT= option (LP), 188, 189, 215
- PENALTYDEPTH= option
  - PROC LP statement, 188, 215
- PEPSILON= option
  - PROC LP statement, 186
- PHASEMIX= option
  - PROC LP statement, 191
- PICTURE option
  - PRINT statement (LP), 194
- PIVOT statement
  - LP procedure, 193, 222
- PMAXIT= option
  - PROC LP statement, 186, 210
- POBJECTIVE= option
  - PROC LP statement, 188
- POSTPROCESS option
  - PROC LP statement, 188
- PREPROCESS option
  - PROC LP statement, 186
- PRICE keyword
  - VARSELECT= option (LP), 189, 215
- PRICE= option
  - PROC LP statement, 191, 208
- PRICEPHI= option
  - PROC LP statement, 189, 195, 221, 222
- PRICESEN keyword
  - TYPE variable (LP), 200, 219
- PRICESEN option
  - PRINT statement (LP), 195
- PRICETYPE= option
  - PROC LP statement, 191, 208
- PRIMALIN= option
  - PROC LP statement, 183, 217, 226
- PRIMALOUT= option
  - PROC LP statement, 183, 217, 224
- PRINT option
  - PROC LP statement, 184
- PRINT statement
  - LP procedure, 193, 222
- PRINTFREQ= option
  - PROC LP statement, 184
- PRINTLEVEL= option
  - PROC LP statement, 184
- PRIOR keyword
  - VARSELECT= option (LP), 189, 215
- PROC LP statement, *see* LP procedure
  - branch-and-bound control options, 186
  - data set options, 183
  - display control options, 184
  - interactive control options, 185
  - parametric control options, 189
  - preprocessing control options, 185
  - ranging control options, 189
  - sensitivity control options, 189
  - simplex algorithm control options, 190
- PROJECT keyword
  - BACKTRACK= option (LP), 186
  - CANSELECT= option (LP), 187, 213
- PROXIMITYPAUSE= option
  - PROC LP statement, 185, 193, 222
- PSEUDOC keyword
  - BACKTRACK= option (LP), 186
  - CANSELECT= option (LP), 187, 214
  - VARSELECT= option (LP), 189, 215
- PWOBJECTIVE= option
  - PROC LP statement, 188
- QUIT statement
  - LP procedure, 195, 222
- RANDOMPRICEMULT= option
  - PROC LP statement, 191
- RANGE keyword
  - TYPE variable (LP), 200
- RANGE statement
  - LP procedure, 195
- RANGEPRIce option
  - PRINT statement (LP), 195
  - PROC LP statement, 189, 220
- RANGERHS option
  - PRINT statement (LP), 195
  - PROC LP statement, 189, 220
- READPAUSE option
  - PROC LP statement, 185, 222
- REPSILON= option
  - PROC LP statement, 191
- RESET statement
  - LP procedure, 196, 222
- RHS keyword
  - TYPE variable (LP), 200
- RHS statement
  - LP procedure, 197
- RHSPHI= option
  - PROC LP statement, 189, 195, 220, 222
- RHSSEN keyword
  - TYPE variable (LP), 200
- RHSSEN option
  - PRINT statement (LP), 195
- RHSSEN statement
  - LP procedure, 197, 218
- ROW keyword
  - SCALE= option (LP), 192, 209
- ROW option
  - PRINT statement (LP), 195
- ROW statement
  - LP procedure, 197
- RUN statement
  - LP procedure, 198, 222

- SASMPSEX macro function, [177](#), [204](#)
- SAVE option
  - QUIT statement (LP), [195](#)
- SCALE= option
  - PROC LP statement, [192](#), [209](#)
- SENSITIVITY option
  - PRINT statement (LP), [194](#), [195](#), [222](#)
- SHOW statement
  - LP procedure, [198](#), [222](#)
- SMALL= option
  - PROC LP statement, [192](#)
- SOLUTION option
  - PRINT statement (LP), [195](#)
- SOSEQ keyword
  - TYPE variable (LP), [199](#)
- SOSLE keyword
  - TYPE variable (LP), [199](#)
- SPARSEDATA option
  - PROC LP statement, [183](#)
- STATUS option
  - SHOW statement (LP), [198](#)
- TABLEAU option
  - PRINT statement (LP), [195](#), [207](#)
- TABLEAUOUT= option
  - PROC LP statement, [183](#), [225](#)
- TABLEAUPRINT option
  - PROC LP statement, [184](#), [207](#)
- TIME= option
  - PROC LP statement, [192](#)
- TREETYPE= option
  - PROC LP statement, [188](#)
- TYPE statement
  - LP procedure, [198](#)
- U= option
  - PROC LP statement, [192](#)
- UNRSTR keyword
  - TYPE variable (LP), [199](#)
- UPPER= option
  - RESET statement (LP), [196](#)
- UPPERBD keyword
  - TYPE variable (LP), [199](#), [210](#)
- VAR statement
  - LP procedure, [200](#)
- VARSELECT= option
  - PROC LP statement, [188](#), [189](#), [214](#)
- WOBJECTIVE= option
  - PROC LP statement, [188](#), [189](#)
- ZEROS option
  - PRINT statement (LP), [195](#)