Chapter 12

Linear Programming

This chapter describes the dense simplex routine, **gen_simplex**. Section 12.2 provides references.

12.1 Dense Simplex Routine

The **gen_simplex** routine solves multidimensional minimization problems using the simplex linear programming method. The goal is to find the minimum of a linear function of multiple independent variables. In the standard formulation, the problem is to minimize the inner product $c^{\mathrm{T}}x$ subject to the conditions $Mx = b$, $0 \leq x \leq u$, where M is an $m \times n$ matrix, c is a coefficient vector, and $c^{\mathrm{T}}x$ is referred to as the *cost*. The upper bound vector u may be infinity in one or more components.

12.1.1 Geometrical Description of the Algorithm

Geometrically, the algorithm can be described in terms of the polytope in n-space defined by the plane $Mx = b$ and the bounds $0 \leq x \leq u$. The problem is to find a vertex of the polytope that lies as far as possible in the direction $-c$. If the solution consists of more than one point, **gen_simplex** chooses one vertex within the solution as its result.

The simplex method involves the following steps:

1. Find any vertex of the polytope.
2. Find a neighboring vertex that results in a lower cost.
3. Repeat Step 2 until no neighboring vertex results in a lower cost.

Copyright © 1994 Thinking Machines Corporation

12.1.2 Vertices and Bases

To find a vertex, **gen_simplex** selects $n-m$ variables and fixes them to their upper bounds or to 0. These variables are selected so that the columns of M corresponding to the remaining m variables are linearly independent. Solving the resulting m-dimensional system yields a vertex. The *basis* defined by the m linearly independent columns is said to be *feasible* if the solution satisfies the bounds $0 \leq x \leq u$.

The process of moving from one basis to another is called a *pivot* or *iteration*. The input argument *max_iter* is the upper bound on the number of iterations **gen_simplex** will perform without finding a solution.

If you have any starting information about feasible bases (for example, you know or guess that certain columns belong in a feasible or optimal basis), you can supply this information to **gen_simplex** in the ***bcol*** argument, which is described in the man page following this section. Upon return, ***bcol*** contains information about the last basis **gen_simplex** found.

12.1.3 Input Array Format

When you call **gen_simplex**, you must supply a two-dimensional zero-based CM array, ***A***, whose upper left-hand corner of dimensions $(m+1)$ x $(n+1)$ contains the standard input items for simplex problems (see Figure 38):

- an offset, ω, from which $c^{T}x$ is subtracted (element (0,0)). Typically, $\omega=0$.
- the cost coefficient vector, c (row 0)
- the right-hand-side vector, b (column 0)
- the matrix M (rows 1 through m, columns 1 through n)

Copyright © 1994 Thinking Machines Corporation

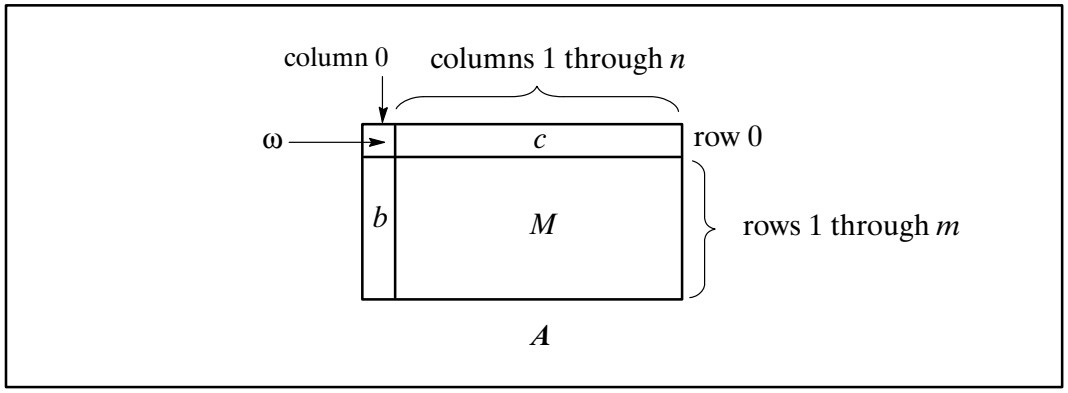

Figure 38. Format of *A*.

When **gen_simplex** transforms *M* to find a vertex (basis), row 0 also changes and becomes the *reduced cost row*. This row carries information about how close the current basis is to being a solution to the problem. Theoretically, when every element of the reduced cost row is greater than 0, the basis is optimal (the vertex is the optimal solution). In practice, when every element of the reduced cost row is greater than $-epsilon$ (where *epsilon* is an input tolerance argument), the basis is considered to be optimal.

The working version of ***A*** during **gen_simplex** processing is referred to as the *tableau*.

NOTE

The **gen_simplex** routine may place zeros in rows of ***A*** beyond row *m* and columns beyond column *n*.

12.1.4 Reinversion

In some cases, the status code returned by **gen_simplex** indicates that you should *reinvert*: that is, call the routine again, restoring the original values of ***A*** (including row and column 0) and supplying the ***bcol*** values returned in the

Copyright © 1994 Thinking Machines Corporation

last call. Reinverting allows **gen_simplex** to clear any numerical errors that have accumulated and start again from a more numerically accurate version of the last basis obtained in the prior call. The input argument *reinvert_freq* is an upper bound on the number of iterations **gen_simplex** will perform before exiting with a status that requests reinversion.

12.1.5 Degeneracy

A problem is *degenerate* if more than one basis represents the same vertex. If **gen_simplex** encounters degeneracy, it may perform multiple iterations without much improvement in cost. The input arguments *degen_iter* and *degen_tol* determine how long **gen_simplex** will continue in a degenerate case: *degen_iter* is an upper bound on the number of iterations **gen_simplex** will perform with cost improvement of less than *degen_tol*. We use a slight modification of the EXPAND anti-cycling procedure (see reference 2 in Section 12.2).

12.1.6 Implementation

The **gen_simplex** routine takes the tableau A and creates or equivalences it with a four-dimensional tableau in the CM, distributed in such a way as to minimize communications overhead. Using special temporary arrays for pivot selection and the rank 1 update, **gen_simplex** proceeds either to solve the linear problem until the reduced cost row contains elements greater than $-epsilon$ and the solution is feasible, or to exit with an appropriate *status* code.

12.1.7 Example

The following example assumes a knowledge of linear programming. Consider the linear programming problem

Copyright © 1994 Thinking Machines Corporation

$$
\begin{array}{lllll}
\text{Minimize} & -3x_1 & -5x_2 & & \\
\text{subject to} & x_1 & +x_2 & = & 11 \\
 & x_1 & -x_2 & \geq & -4 \\
 & -2x_1 & +x_2 & \leq & 2 \\
 & x_1 & +x_2 & \geq & 2
\end{array}
$$

with $x_1 \geq 1, x_2 \geq 0$. The problem may be transformed to

$$
\begin{array}{lllllll}
\text{Minimize} & -3x_1 & -5x_2 & & & & -3 \\
\text{subject to} & x_1 & +x_2 & & & & = 10 \\
 & x_1 & -x_2 & -x_3 & & & = -5 \\
 & -2x_1 & +x_2 & & +x_4 & & = 4 \\
 & x_1 & +x_2 & & & -x_5 & = 1 \\
 & x_1, & x_2, & x_3, & x_4, & x_5 & \geq 0
\end{array}
$$

by adding the slack variable x_4 and the surplus variables x_3 and x_5, and transforming the variable x_1 to be bounded below by zero. To compensate for the change in the definition of x_1, we now include a cost offset (-3). We supply **gen_simplex** with partial knowledge about a possible starting basis by setting ***bcol***(1)=0 (since we have no knowledge about a basic variable in row 1) and setting ***bcol***(2)=3, ***bcol***(3)=4, and ***bcol***(4)=5 to incorporate knowledge about slack and surplus variables. Since the variables have no upper bounds, we set ***bounds***(i) = infinity for all i. (To do this, use the built-in infinity function **d_infinity( )**; see the on-line example.) We set the input tableau argument ***A*** as follows:

$$
A = \begin{pmatrix}
3 & -3 & -5 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-5 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
4 & -2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

In this example, ***A*** contains some unused rows and columns; its active dimensions are $m+1=5$ and $n+1=6$. Elements of ***bcol*** corresponding to the unused rows are set to 0; elements of ***bounds*** corresponding to the unused columns are set to infinity. The array ***flip*** is set to **.false.**.

Copyright © 1994 Thinking Machines Corporation

Dense Simplex

The **gen_simplex** routine solves multidimensional minimization problems using the simplex linear programming method. In the standard formulation, the problem is to minimize the inner product $c^{T}x$ subject to the conditions $Mx = b$, $0 \leq x \leq u$, where M is an m x n matrix, c is a coefficient vector, and $c^{T}x$ is referred to as the *cost*. The upper bound vector u may be infinity in one or more components.

SYNTAX

gen_simplex (***A, x, flip, bounds, bcol****, statarray, constraint_axis, variable_axis, x_flip_bounds_axis, bcol_axis, m, n, iter_count, max_iter, reinvert_freq, epsilon, degen_iter, degen_tol, ier*)

ARGUMENTS

A Zero-based real CM array (single- or double-precision) with rank 2 and NEWS-ordered axes. Must have dimensions at least $(m+1)$ x $(n+1)$. On input, contains

- an offset ω from which $c^{T}x$ is subtracted (element (0,0)).
- the cost coefficient vector, c (row 0).
- the right-hand-side vector, b (column 0).
- the matrix M (rows 1 through m, columns 1 through n).

On exit, ***A*** contains the optimal tableau; that is, the basic columns of M are the optimal basis. Element $A(0,0)$ contains the optimal value $\omega - c^{T}x$, where ω is the original offset passed in and $c^{T}x$ is the optimal cost.

x One-based real CM array with rank 1, the same precision as ***A***, and length n. Must be NEWS-ordered. When **gen_simplex** exits with *status* = 0 (or *status* = −1, if *reinv_freq* = 0), ***x*** contains the solution to the linear problem.

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

flip One-based logical CM array with rank 1 and length n. Must be NEWS-ordered.The first time you call **gen_simplex**, set ***flip***(i) = **.false.** for all i. On return, element ***flip***(i) is set to **.true.** if the ith variable has been set to its upper bound. When you reinvert, supply the ***flip*** values returned by the last call.

bounds One-based real CM array with rank 1, the same precision as ***A***, and length n. Must be NEWS-ordered. The element ***bounds***(i) contains the upper bound of the ith variable of the problem. Each bound must be ≥ 0, but may be infinite.

bcol One-based integer CM array with rank 1 and length m. Must be NEWS-ordered.

On input to the first call to **gen_simplex**, if you have no information about a starting basis, supply ***bcol***(i) = 0 for all i. Alternatively, you may supply information about a possible starting basis, as follows:

- Set ***bcol***(i) to k to suggest that column k be the basic column for row i. The routine attempts to pivot on element (i, k).
- Set ***bcol***(i) to 0 if you have no information about the basic column for row i.
- Set ***bcol***(i) to a negative number if row i is redundant.

The information you supply is used as a suggestion; if it does not work, **gen_simplex** proceeds to find a full-rank feasible basis.

On exit, ***bcol***(i) contains the number of the basic column corresponding to row i of the problem. A negative entry indicates that row i appears to be numerically redundant.

When you reinvert (restore the values of ***A*** and call **gen_simplex** again), do not change the values of ***bcol***; supply the values returned by the last call. The routine uses these values to determine the new starting basis.

statarray Ignored in the current release. Supply the scalar integer constant 0.

Copyright © 1994 Thinking Machines Corporation

constraint_axis Reserved for future releases. Supply the scalar integer constant 1.

variable_axis Reserved for future releases. Supply the scalar integer constant 2.

x_flip_bounds_axis Reserved for future releases. Supply the scalar integer constant 1.

bcol_axis Reserved for future releases. Supply the scalar integer constant 1.

m Scalar integer variable. The number of rows in the matrix *M*. (The array ***A*** must have at least $m + 1$ rows, since it also includes the cost row as row 0.)

n Scalar integer variable. The number of columns in the matrix *M*. (The array ***A*** must have at least $n + 1$ columns, since it also includes the right-hand-side vector, *b*, as column 0.)

iter_count Scalar integer variable. On return, contains the total number of iterations performed. The **gen_simplex** routine does not reset this counter; if you want to reset it, you must do so explicitly. Set *iter_count* to 0 before the first call to **gen_simplex**.

max_iter Scalar integer variable. An upper bound on the number of iterations **gen_simplex** can perform without reaching an optimal solution. Suggested value: $40n$.

reinvert_freq Scalar integer variable. If you supply a value $p > 0$, **gen_simplex** will perform at most p iterations before exiting with a status that requests reinversion. (Suggested value: 10000.)

If you set *reinvert_freq* to 0, **gen_simplex** proceeds until one of the following conditions occurs: **gen_simplex** finds a solution; exceeds the value of *max_iter*; determines that the problem is infeasible, unbounded, or degenerate; or encounters another fatal error. Numerical errors do not cause the routine to exit. However, if numerical errors have accumulated when **gen_simplex** has finished, it returns a status of -1. In this case, if you reinvert,

Copyright © 1994 Thinking Machines Corporation

gen_simplex clears the numerical errors and re-checks the solution.

epsilon Double-precision real scalar variable. A tolerance for the tableau elements, introduced for numerical stability. A reduced cost or pivot element whose absolute value is less than *epsilon* is ignored in the pivot selection. These elements are not necessarily zeroed, however, and may enter the calculations at a future time. When the argument ***A*** is double-precision, suggested values are between 10^{-10} and 10^{-6}, typically 10^{-8}.

degen_iter Scalar integer variable. A bound on the number of iterations **gen_simplex** will perform with cost improvement less than *degen_tol*. Suggested value: 2000.

degen_tol Double-precision real scalar variable. A lower bound on the cost improvement after *degen_iter* iterations. Suggested value: 1e−6.

ier Scalar integer variable. If you supply the value 2 or 16 on input, **gen_simplex** assumes that you are reinverting (that is, that you have not modified ***bcol*** since the last call). If you supply any other value, **gen_simplex** assumes that you are passing it a new problem.

On return, *ier* contains one of the following status codes:

1 (Optimal)
Successful termination. An optimal solution has been found.

2 (Reinvert)
Numerical errors have accumulated, or (if *reinvert_freq* is > 0) the number of iterations has exceeded the value of *reinvert_freq*. Restore the original values of ***A*** (including row and column 0) and call **gen_simplex** again without modifying ***bcol*** or ***flip***. (This process is known as *reinversion*.)

Copyright © 1994 Thinking Machines Corporation

4 (Infeasible)
: The problem appears to be infeasible. However, the apparent infeasibility may be due to numerical ill-conditioning, in which case increasing the value of *epsilon* and calling **gen_simplex** again may clear the error.

8 (Unbounded)
: Either the cost minimizing direction extends to infinity, or bad scaling or numerical errors cause the problem to appear unbounded. If the problem is known to be bounded, restore the original values of ***A*** (including row and column 0), increase the value of *epsilon*, and call **gen_simplex** again.

16 (Degenerate)
: The number of iterations has exceeded the value of *degen_iter* and the cost has improved by less then *degen_tol*. Restore the original values of ***A*** (including row and column 0), increase *reinvert_freq* and *epsilon*, and call **gen_simplex** again without modifying ***bcol*** or ***flip***.

32 (Suboptimal)
: An optimal solution has not yet been found, but the number of iterations has exceeded *max_iter*.

−1 (Bad data type)
: ***A*** has a data type other than real (single- or double-precision), or ***A***, ***x***, and ***bounds*** do not match in data type and precision.

−2 (Axis specification error)
: You supplied an invalid value for one of the axis arguments.

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

−128 (Internal/Other)
An internal error has occurred.

DESCRIPTION

The simplex method involves the following steps:

1. Find any vertex of the polytope defined by the plane $Mx = b$ and the bounds $0 \leq x \leq u$.
2. Find a neighboring vertex that results in a lower cost.
3. Repeat Step 2 until no neighboring vertex results in a lower cost.

To find a vertex, **gen_simplex** selects $n-m$ variables and fixes them to their upper bounds or to 0. These variables are selected so that the columns of M corresponding to the remaining m variables are linearly independent. Solving the resulting m-dimensional system yields a vertex. The *basis* defined by the m linearly independent columns is said to be *feasible* if the solution satisfies the bounds $0 \leq x \leq u$. The process of moving from one basis to another is called a *pivot* or *iteration*.

When **gen_simplex** transforms M to find a vertex (basis), row 0 of $\boldsymbol{A}$ also changes and becomes the *reduced cost row*. This row carries information about how close the current basis is to being a solution to the problem. Theoretically, when every element of the reduced cost row is greater than 0, the basis is optimal (the vertex is the optimal solution). In practice, when every element of the reduced cost row is greater than $-epsilon$, the basis is considered to be optimal. The solution vertex is returned in $\boldsymbol{x}$, and the basic columns are returned in $\boldsymbol{A}$.

NOTES

Inactive Elements May be Zeroed. The **gen_simplex** routine may place zeros in rows of $\boldsymbol{A}$ beyond row m and columns beyond column n.

Performance. For best performance, follow these guidelines:

- Lay out $\boldsymbol{A}$ so that the product of the *physical axis extents* equals the total number of processing elements in the partition. The number of processing elements is the number returned by **CMF_number_of_processors( )**. The physical extent of an axis is the number of processing elements along the axis.

Copyright © 1994 Thinking Machines Corporation

- Lay out ***A*** so that the subgrid size *in each dimension* is a multiple of the vector length. Beginning with CM Fortran Version 2.1, the vector length is
  - 1 for a CM-5 without vector units
  - 1 for a CM-5 with vector units, if the main program was compiled with **-nopadding**
  - 8 for a CM-5 with vector units, if the main program was compiled with **-padding**
- Lay out ***A*** so that the subgrid size of the first axis is not 1.
- Compile the main program with **-axisreorder**.

If you do not satisfy these conditions, **gen_simplex** must make a copy of ***A***, with a resulting cost in both time and memory.

If a copy is not required (that is, the requirements listed above are satisfied), then for best performance, make the subgrid size approximately the same in each dimension. (If **gen_simplex** makes a copy of ***A***, this guideline does not apply.)

Precision. For best results, double precision is recommended.

EXAMPLES

Sample CM Fortran code that uses the **gen_simplex** routine can be found on-line in the subdirectory

```
simplex/cmf
```

of a CMSSL examples directory whose location is site-specific.

Copyright © 1994 Thinking Machines Corporation

12.2 References

For more information about the simplex method, see the following references:

1. Luenberger, D. G. *Linear and Nonlinear Programming*. Reading, MA: Addison-Wesley, 1984 (or any other introductory text on linear programming).

2. Gill, P. E., W. Murray, M. A. Saunders, and M. H. Wright. A practical anti-cycling procedure for linearly constrained problems. *Mathematical Programming* **45** (1989): 437−74.

3. Eckstein, J., R. Qi, V. I. Ragulin, and S. A. Zenios. *Data-Parallel Implementation of Dense Linear Programming Algorithms*. Thinking Machines Corporation Technical Report TMC-230, 1992.

Copyright © 1994 Thinking Machines Corporation