DOCUMENTATION FOR THE ADO/SDS COLLECTION OF STOCHASTIC PROGRAMMING CODES

Jonathan Edwards

Editor

January 1985
WP-85-002

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
A-2361 Laxenburg, Austria

FOREWORD

Developing methodology and tools for optimal decision making under uncertainty was always a major part of research in System Decision Sciences Area. For the last two years, the Adaptation and Optimization Project was involved in developing methods and computer implementations for one of the important parts of such methodology -- stochastic programming.

This paper is among those which describes one of the results of these efforts -- the collection of routines designed to solve stochastic programming problems. It contains concise documentation of this collection.

A.B. Kurzhanski
Chairman
Systems and Decision Sciences
Program

Preface

This paper contains most of the documentation for a collection of routines designed to solve problems in stochastic linear and nonlinear programming. The programs were contributed to the Adaptation and Optimization (ADO) project of the System and Decision Sciences program by several researchers and represent the current state-of-the-art in stochastic programming algorithms (several of the algorithms are discussed in "Numerical Techniques for Stochastic Optimization Problems," Yu. Ermoliev and R. Wets eds., whose compilation has been a part of ADO's work recently). There is much work yet to be done - this paper describes but a single brick in the foundation of stochastic programming techniques. But one brick is vastly superior to no bricks.

Documentation for the ADO/SDS Collection of Stochastic Programming Codes

Jonathan Edwards, editor

Introduction

This working paper contains most of the documentation for a collection of routines designed to solve problems in stochastic linear and nonlinear programming. The programs were contributed to the Adaptation and Optimization (ADO) project of the System and Decision Sciences program by several researchers. The codes in the collection implement several of the algorithms discussed in "Numerical Techniques for Stochastic Optimization Problems" (Yu. Ermoliev and R. Wets eds.), whose compilation has been a part of ADO's work recently.

This paper consists of the User's Manuals for eight of the nine programs on the ADO/SDS tape (the documentation for Alexei Gaivoronsky's STO routine is itself a working paper - "Stochastic Quasigradient Methods and their Implementation" (IIASA WP-84-55), by Yuri Ermoliev and Alexei Gaivoronsky - and therefore is not included). The tape itself includes the text for all the User's Manuals (including Gaivoronsky's) as well as a table of contents (the first file).

The ADO collection may be obtained from

Project Secretary
ADO/SDS
IIASA
A-2361 Laxenburg
Austria.

Persons who would like a copy should send a blank reel of 9-track computer tape to the above address and should include a note indicating their preferences for density and character set (IIASA's computer can generate unlabelled tapes at 1600 or 800 bits per inch using either the ASCII or EBCDIC character codes).

NDSP User's Manual

J. Edwards

Introduction

This program is based on the L-shaped method of Van Slyke and Wets for two-stage stochastic linear programs. The method is described in [1]. The algorithm particular to this implementation (Nested Decomposition for Stochastic Programming [NDSP]) was invented by John Birge and is described in [2]. The program was developed by John Birge at the University of Michigan. The linear programming sections were taken from the program LPM-1, developed by J.A. Tomlin at Stanford University [3]. The program is written in FORTRAN IV. The following description of the problem and the discussion of the algorithm are adapted from [4].

The Problem

The multi-stage stochastic linear program under consideration is

minimize

$$z = c_1 x_1 + E_{\xi_2}\left\{\min c_2 x_2 + \cdots + E_{\xi_T}\{\min c_T x_T\}\right\} \qquad (1)$$

subject to

$$\begin{aligned} A_1 x_1 &= b_1 \\ B_1 x_1 + A_2 x_2 &= \xi_2 \\ \cdots\quad & \quad\cdots \\ B_{T-1} x_{T-1} + A_T x_T &= \xi_T \end{aligned}$$

$x_t \geq 0,\ t=1,\ldots,T,\ \ \xi_t \in \Xi_t,\ t=2,\ldots,T$

where c_t is a known vector in R^{n_t} for $t=1,\ldots,T$, b_1 is a known vector in R^{m_1}, ξ_t is a random m_t vector defined on the probability space $(\Xi_t, \mathcal{F}_t, F_t)$ for $t=2,\ldots,T$, and A_t and B_t are appropriately dimensioned known real valued matrices. "E_{ξ_t}" denotes mathematical expectation with respect to ξ_t.

The Method

The L-shaped method of Van Slyke and Wets applies to this problem when T=2. It is an outer linearization procedure that approximates the convex objective term in the stochastic program by successively appending supporting hyperplanes. In NDSP, the supports are found by optimizing a nested sequence of linear programs. Previous methods for the multi-stage problem have tended to assume a specific structure for the problem; NDSP does not require any special structure, although there must be a finite number of random variables ξ_t and these must be discretely distributed.

NDSP is based on the observation that, given some realization ξ_t of the random vector in period t and given the solution x_{t-1} from period t-1, the decision problem at period t can be written

minimize

$$c_t x_t + Q_{t+1}(x_t) \tag{2}$$

subject to

$$\begin{aligned} A_t x_t &= \xi_t + B_{t-1}x_{t-1} \\ D_t^l x_t &\geq d_t^l \ , \quad l=1,\dots,r_t \\ x_t &\geq 0 \ , \end{aligned}$$

where $Q_{t+1}(x_t)$ is a convex function, $D_t^l \in R^{n_t}$ for all l, and $r_t \leq m_{t+1}$.

Problem (2) is solved using a *relaxed master problem*, viz.

minimize

$$c_t x_t + \vartheta_t \tag{3.0}$$

subject to

$$A_t x_t = \xi_t + B_{t-1}x_{t-1} \tag{3.1}$$

$$D_t^l x_t \geq d_t^l \ , \quad l=1,\dots,r_t \tag{3.2}$$

$$E_t^l x_t + \vartheta_t \geq e_t^l \ , \quad l=1,\dots,s_t \tag{3.3}$$

$$x_t \geq 0 \ . \tag{3.4}$$

where E_t^l and e_t^l are chosen so that $e_t^l - E_t^l x_t = Q_{t+1}(x_t)$ [i.e., (3.3) is equivalent to $\vartheta_t \geq Q_{t+1}(x_t)$]. r_t and s_t count the number of constraints (3.2) and (3.3), respectively, in period t and are initially set to zero. This problem is solved to obtain $(\bar{x}_t, \bar{\vartheta}_t)$ for t=1,...,T (for the first period, $\vartheta_1 = 0$ and (3.1) is replaced by $A_1 x_1 = b_1$). The solution $\bar{x}_{t-1}$ from period t-1 is used on the right hand side of (3.1) when solving (3) for period t.

If the problem (3) is infeasible in some period t, NDSP adds a "feasibility cut" (3.2) to (3) in period t-1, adds 1 to r_{t-1}, and solves (3) anew for all periods τ, τ=t−1,...,T. Note that when an infeasibility occurs in period t and a feasibility cut is added to (3) in period t-1, the resulting problem in period t-1 may be infeasible, requiring a feasibility cut to be added to the problem in period t-2. In this manner, infeasibility can propagate back to the first period.

Once feasible solutions have been found for (3) in all periods, NDSP calculates E_t and e_t in each period t using the solutions from period t+1 and the formulas

$$E_t = \pi_{t+1}B_{t+1} \tag{4}$$

and

$$e_t = \pi_{t+1}\xi_{t+1}$$

where π_{t+1} is the optimal dual vector in period t+1. If E_t and e_t are found such that (3.3) is not satisfied for some period t, NDSP adds an optimality cut (3.3) to the problem in period t and adds 1 to s_t. Introducing the optimality cut changes the problem in period t, so NDSP repeats the process outlined in the preceding two paragraphs to find a new feasible solution. The "forward pass" to obtain feasibility in each period and the "backward pass" to solve (2) based on the relaxation in (3) are repeated until the optimal solution has been found (i.e., until the constraints (3.3) may be satisfied in every period).

In the above discussion, the ξ_ts were fixed in the sense that at each period t one realization of ξ_t was chosen and used to calculate the optimal solution for use in the next period. To solve the problem fully, all realizations of the random variable in a particular period must be examined.

For implementation with multi-stage problems, it is assumed that there is a finite number K_t of "scenarios" in each period t. Each scenario in a given period corresponds to a problem (3) given a single realization of the random vector in that period. For every scenario j in period t, t=1,...,T-1, there is a unique "ancestor" scenario $a(j)$ in period t-1 and one or more "descendant" scenarios $d(j)$ in period t+1. It is further assumed that every scenario in a given period has the same number of descendants as every other scenario in the period. In other words, every set $d(j)$, $j=1,\ldots,K_t$ contains exactly K_{t+1}/K_t scenarios in period t+1. The first set $[d(1)]$ contains the first through the K_{t+1}/K_tth scenarios, the second set $[d(2)]$ contains the $[(K_{t+1}/K_t)+1]$th through the $2K_{t+1}/K_t$th scenarios, etc.

In the last period (T), the program uses Wets' "bunching" method [5] to examine all realizations of ξ and find those for which a given basis is optimal. This method represents an alternative to the "sifting" procedure of Garstka and Rutenberg. In order to apply this method, the algorithm assumes that the random vector in period T contains a fixed number of independent random elements, that these elements are discretely distributed, and that every scenario in period T-1 is an ancestor of every scenario in period T.

Adding multiple realizations to the original description effectively adds superscripts to problems (2) and (3) and changes equations (4), viz.

Equation (2) becomes minimize

$$c_t x_t^j + Q_{t+1}(x_t^j) \tag{2'}$$

subject to

$$A_t x_t^j = \xi_t^j + B_{t-1} x_{t-1}^{a(j)}$$
$$D_t^{l,j} x_t^j \geq d_t^{l,j} \quad , \quad l=1,\ldots,r_t^j$$
$$x_t^j \geq 0 \quad .$$

Equation (3) becomes minimize

$$c_t x_t^j + \vartheta_t^j \tag{3.0'}$$

subject to

$$A_t x_t^j = \xi_t^j + B_{t-1} x_{t-1}^{a(j)} \tag{3.1'}$$
$$D_t^{l,j} x_t^j \geq d_t^{l,j} \quad , \quad l=1,\ldots,r_t^j \tag{3.2'}$$
$$E_t^{l,j} x_t^j + \vartheta_t^j \geq e_t^{l,j} \quad , \quad l=1,\ldots,s_t^j \tag{3.3'}$$
$$x_t^j \geq 0 \quad . \tag{3.4'}$$

The equations for E_t and e_t become

$$E_t^j = \sum_k p_{t+1}^k (\pi_{t+1}^k B_{t+1}) \tag{4'}$$

and

$$e_t^j = \sum_k p_{t+1}^k \pi_{t+1}^k \xi_{t+1}^k$$

where p_{t+1}^k is the probability that the random vector in period t+1 assumes the value associated with the kth scenario. The sum is taken over every descendant of scenario j in period t (i.e., k runs from $[(j-1)(K_{t+1}/K_t)+1]$ to jK_{t+1}/K_t).

Rather than solve one problem (3) at each period t, NDSP solves (3') for all j, $j=1,\ldots,K_t$ using the solution to the appropriate ancestor scenario on the right hand side of (3.1'). Similarly, during the backward pass, NDSP checks that (3.3') holds for every scenario in each period.

Unboundedness may be handled explicitly following the procedure in [1], but in this implementation all variables are upper bounded and hence unboundedness is avoided.

Input Overview

The input format for this program follows the MPS standard for mathematical programs [6] in most respects. However, the multi-stage nature of the algorithm demands that the data be split into periods and scenarios within each period. There is also some control information that does not comply with the standard.

The program takes its data from unit 5. It is the user's responsibility to connect this unit to the appropriate file before the program is invoked.

Control Information

The first line of the input contains five integers that control the program's execution. Each is read using an I4 format and there are no blanks between the integers. The numbers provide the following information and must appear in the order specified:

- the problem number. This is used simply to identify the problem. It must not be zero.
- the row index of the objective rows. This integer identifies which of the rows specified in the ROWS sections of the file are the objective rows. If this number is zero or is omitted, a value of 1 is assumed.
- the number of iterations between matrix inversions when solving the linear program (3'). NDSP uses a revised, primal-simplex method to solve (3'). This is the number of iterations between inversions of the basis. If this number is zero or is omitted, a value of 99999 is assumed.
- the maximum number of iterations allowed to solve the linear program (3'). If this number is zero or is omitted, a value of 99999 is assumed.
- the number of periods (T). This number must appear.

The next several lines contain the number of scenarios in each period that have the same ancestor in the previous period (i.e., the values of K_t/K_{t-1}). There is one such line for each period and the values are read using an I4 format. This value should be 1 for the first and last periods, since the right hand side for the first period is deterministic and the right hand side for the last period is entered separately at the end of the data.

Data

The remainder of the input file provides the values for the c_ts, the A_ts, the B_ts, and the variables on the right hand sides (b_1 and the ξ_ts), as well as bounds for the solution. The user may also include sections that specify an initial basis in any period for any scenario.

Note that no case conversion is performed and therefore all section headers should be capitalized.

The following information must be provided for each ξ_t^j, j=1,..., K_t in a given period t, t=1,...,T-1, and once for the last period:

- the probability that the random variable assumes the realization ξ_t^j. This value is read using an F5.3 format. It should be 1.00 for the first and last periods.
- two sections (ROWS and COLUMNS) in standard MPS format containing the values of c_t and A_t. The values for c_t are taken from the entries for the objective row in the COLUMNS section. The remainder of the entries in the COLUMNS section specify the contents of A_t.
- an optional section (BASIS) which follows standard MPS format containing an initial basis for the current period. This section contains a list of column and row names indicating which variables are basic. The column name appears in the first name field (columns 5 through 12) and the row name appears in the second name field (columns 15 through 22). The program writes sections in this format to unit 7 containing the names of the basic variables in the optimal solution (see the "Basis File" section below).
- a section (RHS) in standard MPS format containing the value of ξ_t^j. For the first period, this section contains the value of b_1. For the last period, it contains the values of any nonstochastic elements of ξ_T as well as one value for each of the stochastic elements of ξ_T.
- an ENDATA card in standard MPS format (i.e., a card with the characters "ENDATA" in the first 6 columns).
- lower bounds on all variables except slacks. These values are read using a 9F8.0 format. There must be enough lines to supply a lower bound for every non-slack variable.
- upper bounds on all variables except slacks. These values are read using a 9F8.0 format. There must be enough lines to supply an upper bound for every non-slack variable.
- two sections (ROWS and COLUMNS) in standard MPS format containing the values of B_t. Since B_t is used in period t+1, these two sections do not appear in the data for the last period.
- an ENDATA card in standard MPS format.

Following the last of these specifications (which gives the values of c_t, A_t, etc., for the last period) is a section (STOCH) containing the values for the stochastic elements of ξ in the last period. This section follows standard MPS format: the row name of the element appears in the first name field (columns 5 through 12), a value for the element appears in the first numeric field (columns 25 through 36), and the probability that the element takes the associated value appears in the second numeric field (columns 50 through 61). Both numbers are read using an F12.4 format. As many as five separate values may be specified for each random element.

Output File

The program writes a log and most of its results to unit 6. It first prints the problem number, the densities of the A_t matrices, and the values and probabilities of the stochastic elements in the last period.

Basis File

The program writes the names of the variables that are basic in the optimal solution to the linear program (3') in each period and for each scenario to unit 7 (an exception is the last period, for which the program writes only the names of the variables which appear in the last basis found). The names are in the form "column name" "row name" and appear in the first and second name fields (columns 5 through 12 and 15 through 22), respectively. The names for each scenario in each period are preceded by a basis section header card. These sections may be included in the input to provide the program with starting bases.

Data Structures

Many variables used by NDSP have a distinct value in each period and for each scenario within a period (ξ is a good example). To keep these values separate yet readily available, the program uses multidimensional arrays. In general, each array contains all the values for a single variable and an array reference whose last three subscripts are (i,j,k) returns the value that the variable the array contains assumes in period i. If i is 1, j and k must also be 1 (other values of j or k reference storage that is not used). If i is 2 and there are more than two periods, the value applies in the second period to the jth scenario and k must be 1 (other values of k reference storage that is not used). If i is 3 and there are more than three periods, the value applies in the third period to the kth descendant of the jth scenario in the second period. In this case, k is *not* the index of the scenario in the third period. It is the index of the scenario within the set $d(j)$ in the second period (i.e. $1 \le k \le K_3 / K_2$). Whenever i is equal to the number of periods (T), j and k must be 1 (since all scenarios at period T share the same ancestors).

As an example, let us assume that we have a four period problem and that there are two second period scenarios and six third period scenarios (i.e., each second period scenario has three descendants). Let $m_1 = m_2 = m_3 = M$ and let XKSI have the smallest dimensions possible, i.e., M by 4 by 2 by 3. The elements of b_1 and the ξs would appear in the array XKSI as shown below, where m=1,...,M.

XKSI(m,1,1,1) - b_1

XKSI(m,1,1,2), XKSI(m,1,1,3), XKSI(m,1,2,1), XKSI(m,1,2,2), and XKSI(m,1,2,3) - unused

XKSI(m,2,1,1) - ξ_2^1

XKSI(m,2,1,2) and XKSI(m,2,1,3) - unused

XKSI(m,2,2,1) - ξ_2^2

XKSI(m,2,2,2) and XKSI(m,2,2,3) - unused

XKSI(m,3,1,1) - ξ_3^1

XKSI(m,3,1,2) - ξ_3^2

XKSI(m,3,1,3) - ξ_3^3

XKSI(m,3,2,1) - ξ_3^4

XKSI(m,3,2,2) - ξ_3^5

XKSI(m,3,2,3) - ξ_3^6

The data for the fourth period ξs appears elsewhere, but they would appear in XKSI(m,4,1,1) otherwise.

The algorithm requires several matrices for each period (e.g., A_t). These matrices tend to be rather sparse, and the program represents them in a compact fashion to save space. To represent a large, sparse, two-dimensional array, the program uses three smaller one-dimensional arrays. The first array contains the nonzero elements of the matrix. These elements are ordered by column. Each element of the second array contains the row index within the sparse matrix of the corresponding element in the first array. The third array contains the indices within the first two arrays where the entries for each column of the sparse matrix begin. The ith entry in the third array is a pointer to the beginning of the ith column.

Common Blocks and User-Accessible Parameters

Most of the major variables used by this program are commented within the program and are of little concern to the user. Of potential interest, however, are several constants in the BLOCK DATA subroutine and the major array dimensions. The constants are discussed below. An explanation of the array dimensions appears in the section entitled "Limits and Extensions."

The following tolerances and limits appear in the named common block "BLOCK" and are initialized in the BLOCK DATA subroutine:

- zero tolerance (ZTOLZE).
- pivot tolerance (ZTOLPV).
- reduced cost tolerance (ZTCOST).
- maximum number of nonzero elements in any A_t array (NEMAX).
- maximum number of rows in any right hand side element of problem (3) (i.e., b_1 or ξ_t^j) (NRMAX).
- maximum number of columns in the η vector form of the basis inverse (NTMAX).

The variables ZTOLSM and NEGINF, which also appear in the common block, are not used.

The subroutine SHIFTR is used to move blocks of data around within certain arrays. Due to the methods it uses, the arrays B, X, Y, and YTEMP *must* appear as a group in that order within the blank common block.

Limits and Extensions

The current version of this program is somewhat experimental and several limitations have been imposed during its development. This section enumerates the limits and offers instructions concerning removing or extending them. *This advice should not be regarded as gospel!*

The current version of this program permits three periods with up to three scenarios in the second period. The random vector in the last period may have up to three independent random elements which may assume one of as many as five values for a total of 125 scenarios in the last period. Each scenario problem (3') is limited to 350 rows and 600 columns. Within a scenario problem (3'), the matrix A_t may have no more than 3000 nonzero elements and the matrix B_{t-1} may have no more than 600 nonzero elements. The η vector form of the basis inverse may have no more than 1000 columns and 3000 nonzero elements.

To change these limits, the user must change the dimensions on the arrays listed below as directed. To change the maximum number of rows, the maximum number of nonzero elements in the A_t matrix in each scenario problem (3'), and the maximum number of columns in the η form of the basis inverse, the user must also update the constants NEMAX, NRMAX, and NTMAX, respectively, in the BLOCK DATA subroutine. Following is a list of arrays, the common blocks (or subroutines) in which they appear, and the dimensions which they must have. "nzero" is the maximum number of nonzero elements allowed in the A_t matrix in each scenario problem (3'), "nrows" and "ncols" are the maximum number of rows and columns, respectively, allowed in a scenario problem (3'), "neta" is the maximum number of columns allowed in the η vector form of the basis inverse, "nper" is the number of periods, "ntwo" is the number of scenarios in the second period, "nthree" is the number of descendants in the third period that belong to each scenario in the second period, "nxi" is the number of stochastic elements in the random vector in the last period, and "nrel" is the maximum number of values that each stochastic element may assume. Note that nper cannot be larger than 4 and that nxi cannot be larger than 3.

Blank Common

```
        A(nzero,nper,ntwo,nthree)
        ATMP(nzero)
        ABN(maximum number of nonzero elements in B_t,nper,ntwo,nthree)
        E(nzero)
        IA(nzero,nper,ntwo,nthree)
        IE(nzero)
        IBN(maximum number of nonzero elements in B_t,nper,ntwo,nthree)
        ITMP(nzero)
        JH(nrow,nper,ntwo,nthree)
        KBTMP(ncol+2)
        KINBAS(ncol+2,nper,ntwo,nthree)
        LA(ncol+2,nper,ntwo,nthree)
        LBN(ncol+2,nper,ntwo,nthree)
        LE(neta+2)
        LTMP(ncol+2)
        NCOL(nper,ntwo,nthree)
        NCOLP(nper,ntwo,nthree)
        NELM(nper,ntwo,nthree)
        NROW(nper,ntwo,nthree)
        NROWP(nper,ntwo,nthree)
        NTH(nper,ntwo,nthree)
        PROB(nper,ntwo,nthree)
```

XKSI(nrow,nper,ntwo,nthree)
XLB(ncol,nper,ntwo,nthree)
XLTMP(ncol+2)
XUB(ncol+2,nper,ntwo,nthree)
XUTMP(ncol+2)
YPI(nrow,nper,ntwo,nthree)
YTEMP1(ncol+22)
B(nrow,nper,ntwo,nthree)
X(nrow,nper,ntwo,nthree)
Y(nrow)
YTEMP(ncol+2)

BLOCK3
JPER(nper)
NND(nper)

BLOCK4
BND(nrow)
CBST(nxi,nrel)
IBST(nxi)
INST(nxi)
JSTCH(nrel,nrel,nrel)
NCUR(nxi)
NETND(see footnote below)
NXNF(nxi)
PRBV(nxi,nrel)
PRST(nrel,nrel,nrel)
YBX(nrow)
YPIBAR(ncol+2)
MXNST - this variable should be set to nxi.

Subroutine INPUT
ICN(ncol+2,2)
ICNAM(ncol+2,2,nper,ntwo,nthree)

Subroutine INVERT
MREG(nrow)
HREG(nrow)
VREG(nrow)

Subroutine WRAPUP
ICNAM(ncol+2,2,nper,ntwo,nthree)
XTEMP(ncol+2)

There are many loops in the subroutine INIT and several loops in the subroutine INPUT whose upper limits must be changed to match those of the dimensions of the arrays which the loops initialize.

The subroutine SHIFTR contains several constants which may need to be changed. The numbers in the equations are formed as follows (the numbers in parentheses to the right of the expressions refer to the value of IOLD and INEW when the expression applies):

NETND(i) contains the number of η vectors in the ith basis and its dimension should be large enough to accomodate as many bases as are likely to be generated.

nrmax*(JCUR − 1) + nper*nrmax*(JPER(2) − 1) (1)
+ nper*nrmax*ntwo*(JCUR(3) − 1)

nper*nrmax*ntwo plus value (1) above. (2)

2*nper*nrmax*ntwo (3)

nrmax plus value (3) above. (4)

The dimension of the array BARRAY should be at least 2*nper*nrmax*ntwo + nrmax + ncol + 2.

Subprocedure Hierarchy

See the attached figure.

References

[1] R. Van Slyke and R. Wets, "L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Linear Programs," *SIAM Journal on Applied Mathematics* v. 17, pp. 638-663, 1969

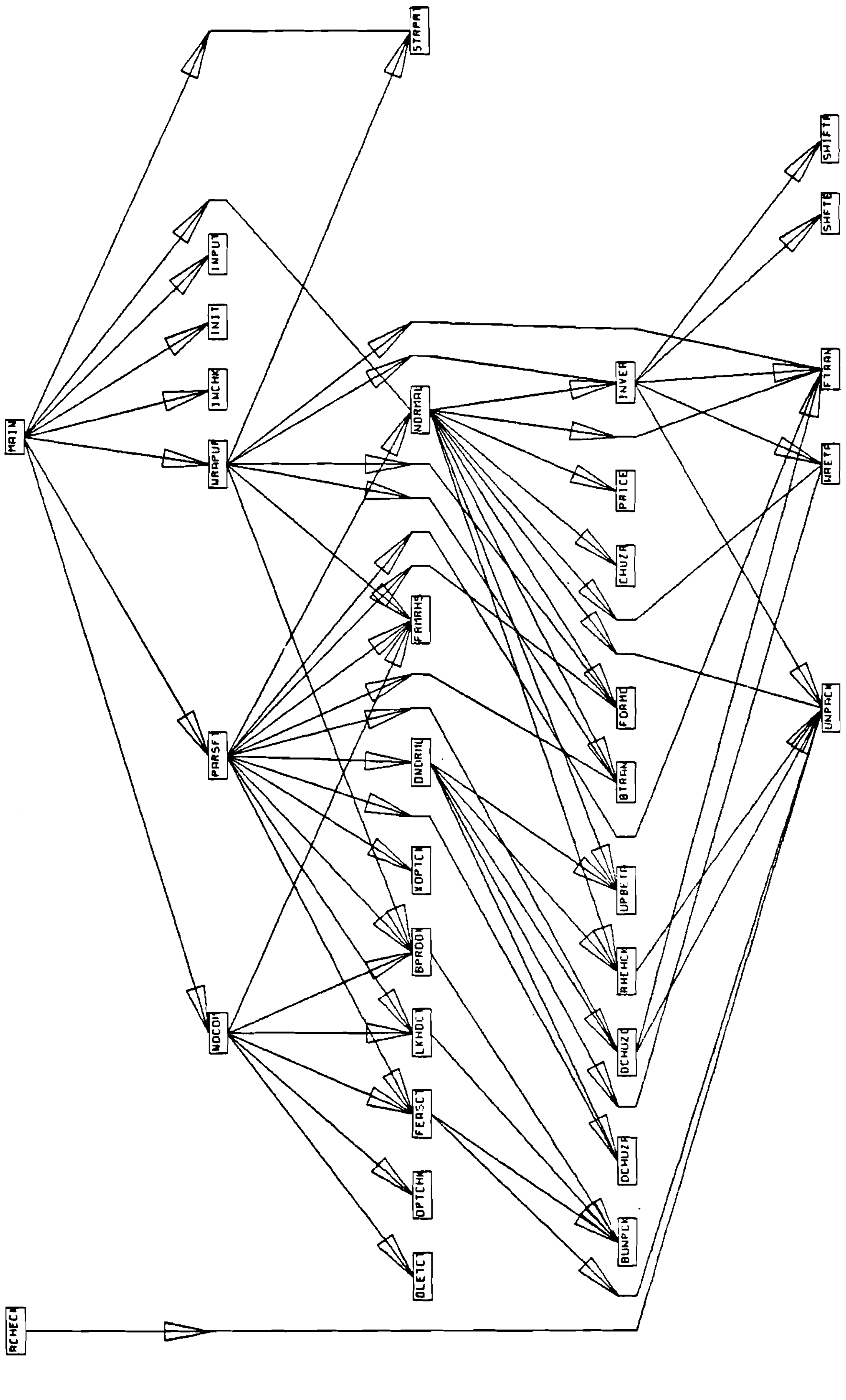
[2] J. Birge, "Decomposition and Partitioning Methods for Multi-Stage Stochastic Linear Programs," Technical Report 82-6, Department of Industrial and Operations Engineering, the University of Michigan, 1982

[3] C. E. Pfefferkorn and J. A. Tomlin, "Design of a Linear Programming System for ILIAC IV," Technical Report SOL 76-8, Systems Optimization Laboratory, Stanford University, 1976

[4] J. Birge, "An L-Shaped Method Computer Code for Multi-Stage Stochastic Linear Programs," in **Numerical Methods for Stochastic Optimization**, Yu. Ermol'ev and R. Wets (eds), to appear in 1985

[5] R. Wets, "Stochastic Programming: Solution Techniques and Approximation Schemes", in **Mathematical Programming: State-of-the-Art 1982**, Bachem, Groetschel, and Korte (eds), 1983

[6] IBM Corp., **Mathematical Programming Subsystem - Extended (MPSX) and Generalized Upper Bounding (GUB) Program Description**, document number SH20-0968-1


MAIN
RCHECK
STRPRT
INPUT
INIT
INCHK
WRAPUP
PARSFI
NOCOM
NORMAL
FRMRHS
DNORML
XOPTCK
BPROD
LKHDL
FEASCT
OPTCHK
DLETC
INVER
PRICE
CHUZR
FORMD
BTRAN
UPBETA
RCHECK
DCHUZQ
DCHUZR
BUNPCK
SHIFTA
SHFTB
FTRAN
WRETA
UNPACK

The Introduction to STOCHASTIC QUASIGRADIENT METHODS AND THEIR IMPLEMENTATION

Yuri Ermoliev and Alexei Gaivoronski

ABSTRACT

[Editor's note - What follows is an excerpt from the introduction to IIASA Working Paper WP-84-55, which serves as a user's manual for Alexei Gaivoronski's program, STO. A complete copy of that paper appears on the ADO/SDS tape.]

1. INTRODUCTION

This paper discusses various stochastic quasigradient methods (see [1,2]) and considers their computer implementation. It is based on experience gained both at the V. Glushkov Institute of Cybernetics in Kiev and at IIASA.

We are concerned here mainly with questions of implementation, such as the best way to choose step directions and step sizes, and therefore little attention will be paid to theoretical aspects such as convergence theorems and their proofs. Readers interested in the theoretical side are referred to [1,2].

The paper is divided into five sections. After introducing the main problem in Section 1, we discuss the various ways of choosing the step size and step direction in Sections 2 and 3. A detailed description of an interactive stochastic optimization package (STO) currently available at IIASA is given in Section 4. This package represents one possible implementation of the methods described in the previous sections. Finally, Section 5 deals with the solution of some test problems using this package. These problems were brought to our attention by other IIASA projects and collaborating institutions and include a facility location problem, a water resources management problem, and the problem of choosing the parameters in a closed loop control law for a stochastic dynamical system with delay.

We are mainly concerned with the problem

$$\min \{F(x) : x \in X\} \ , \quad F(x) = \mathrm{E}_\omega f(x,\omega) \ , \tag{1}$$

where x represents the variables to be chosen optimally, X is a set of constraints, and ω is a random variable belonging to some probabilistic space (Ω,B,P). Here B is a Borel field and P is a probabilistic measure.

There are currently two main approaches to this problem. In the first, we take the mathematical expectation in (1), which leads to multidimensional integration and involves the use of various approximation schemes. This reduces problem (1) to a special kind of nonlinear programming problem which allows the application of deterministic optimization techniques. In this paper we concentrate on the second approach, in which we consider a very limited number of observations of random function $f(x,\omega)$ at each iteration in order to determine the direction of the next step. The resulting errors are smoothed out until the optimization process terminates (which happens when the step size becomes sufficiently small).

We assume that set X is defined in such a way that the projection operation $x \to \pi_X(x)$ is comparatively inexpensive from a computational point of view, where $\pi_X(x) = \arg\min_{z \in X} \|x - z\|$. For instance, if X is defined by linear constraints, then projection is reduced to a quadratic programming problem which, although challenging if large scale, can nevertheless be solved in a finite number of iterations. In this case it is possible to implement a stochastic quasigradient algorithm of the following type:

$$x^{s+1} = \pi_X(x^s - \rho_s v^s) \quad . \tag{2}$$

Here x^s is the current approximation of the optimal solution, ρ_s is the step size, and v^s is a random step direction. This step direction may, for instance, be a statistical estimate of the gradient (or subgradient in the nondifferentiable case) of function $F(x)$: then $v^s \equiv \xi^s$ such that

$$\mathrm{E}(\xi^s \mid x^1, x^2, \ldots, x^s) = F_x(x^s) + a^s \quad , \tag{3}$$

where a^s decreases as the number of iterations increases, and the vector v^s is called a *stochastic quasigradient* of function $F(x)$. Usually $\rho_s \to 0$ as $s \to \infty$ and therefore $\|x^{s+1} - x^s\| \to 0$ from (2). This suggests that we should take x^s as the initial point for the solution of the projection problem at iteration number $s+1$, thus reducing considerably the computational effort needed to solve the quadratic programming problem at each step $s = 1, 2, \ldots$. Algorithm (2)–(3) can also cope with problems with more general constraints formulated in terms of mathematical expectations

$$\mathrm{E}_\omega f^i(x, \omega) \geq 0 \ , \quad i = \overline{1,m}$$

by making use of penalty functions or the Lagrangian.

The principal peculiarity of such methods is their nonmonotonicity, which may sometimes show itself in highly oscillatory behavior. In this case it is difficult to judge whether the algorithm has already approached a neighborhood of the optimal point or not, since exact values of the objective function are not available. The best way of dealing with such difficulties seems to be to use an interactive procedure to choose the step sizes and step directions, especially if it does not take much time to make one observation. More reasons for adopting an interactive approach and details of the implementation are given in the following sections.

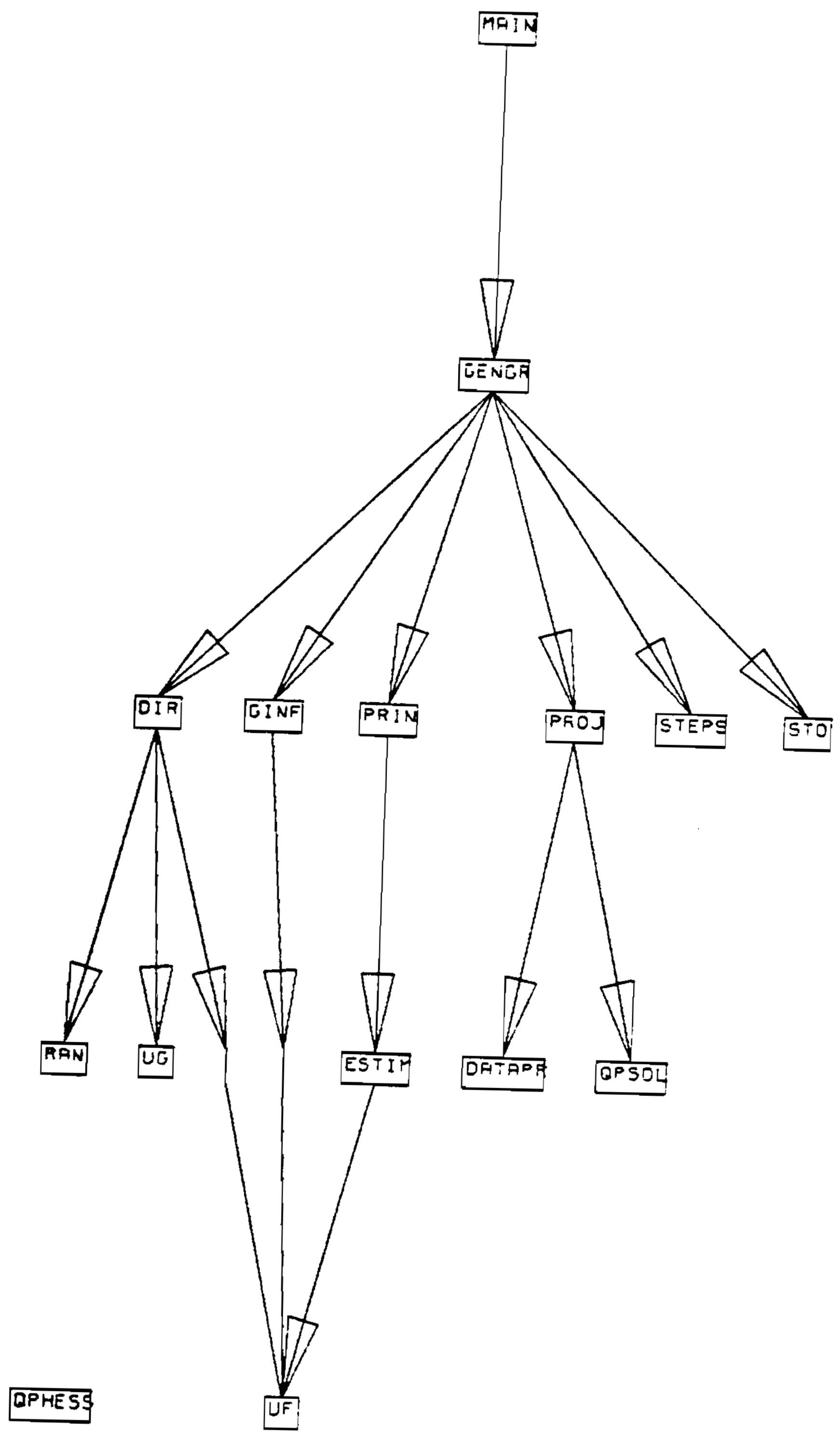
Another characteristic of the algorithms described here is their pattern of convergence. Because of the probabilistic nature of the problem, their asymptotic rate of convergence is extremely slow and may be represented by

$$\|x^* - x^s\| \sim \frac{C}{\sqrt{k}} \quad . \tag{4}$$

Here x^* is the optimal point to which sequence x^s converges and k is the number of observations of random parameters ω, which in many cases is proportional to the number of iterations. In deterministic optimization a superlinear asymptotic convergence rate is generally expected; a rate such as (4) would be considered as nonconvergence. But no algorithm can do asymptotically any better than this for stochastic problem (1) in the presence of nondegenerate random disturbances, and therefore the aim is to reach some *neighborhood* of the solution rather than to find the precise value of the solution itself. Algorithm (2)–(3) is quite good enough for this purpose.

REFERENCES

1. Yu. Ermoliev. *Methods of Stochastic Programming* (in Russian). Nauka, Moscow, 1976.
2. Yu. Ermoliev. Stochastic quasigradient methods and their applications to systems optimization. *Stochastics*, 9 (1983) 1–36.


MAIN
GENGR
DIR
GINF
PRIN
PROJ
STEPS
STO
RAN
UG
ESTIM
DATAPR
QPSOL
QPHESS
UF

LFGM User's Manual

J. Edwards

Introduction

This program implements Rockafellar and Wets' Lagrangian Finite Generation Method (LFGM) for stochastic quadratic programs with simple recourse. This technique is described in [1]. The program was developed at IIASA by Alan King. It is written in FORTRAN 77. A complete description of the implementation may be found in [2]. The following description of the problem and the discussion of the algorithm are adapted from [2], which contains further details and a discussion of possible future developments.

The Problem

The standard formulation of the stochastic quadratic program with simple recourse is to find $x \in R^n$ to *maximize*

$$\sum_{j=1,n} c_j x_j - \tfrac{1}{2} \sum_{j=1,n} d_j x_j^2 - E\left\{ \sum_{i=1,l} e_i q_i \vartheta(e_i^{-1} \underline{v}_i) \right\} \qquad \text{(SQP)}$$

subject to

$$\sum_{j=1,n} a_{ij} x_j \le b_i \quad , \quad i=1,\ldots,m$$

$$ee_j \le x_j \le ff_j \quad , \quad j=1,\ldots,n \quad ,$$

where

$$\underline{v}_i = \sum_{j=1,n} \underline{t}_{ij} x_j - \underline{h}_i \quad .$$

ϑ is a piecewise linear-quadratic function given by

$$\vartheta(\tau) = \begin{cases} 0 & \text{if } \tau \le 0 \\ \tfrac{1}{2}\tau^2 & \text{if } 0 \le \tau \le 1 \\ \tau - \tfrac{1}{2} & \text{if } \tau \ge 1 \end{cases} ,$$

the $\underline{t}_{ij}$s and $\underline{h}_i$s are square summable random variables, the other coefficients are fixed (nonstochastic) with $d_j \ge 0$, $e_i > 0$, and $q_i \ge 0$, and "E" denotes mathematical expectation.

The Method

The chief difficulty in solving (SQP) is the computation of the expectation of the recourse penalties. Achieving reasonable accuracy requires a large number of points at which to evaluate the integrals. Of course, this vastly increases the number of dimensions in the problem. The mainstay of LFGM is a special Lagrangian whose introduction yields a dual problem (DQP) involving minimization over a function space $\underline{Z}$. This problem is less tractable than (SQP), but via the finite generation technique (DQP) may be reduced to a sequence of quadratic programs with few dimensions. Each problem in this sequence can be solved by MINOS [3]. A more detailed description of the algorithm follows.

For various reasons, it is advantageous to include a strongly quadratic term in the xs. The algorithm generates a sequence of points $\{\bar{x}^{\mu},\mu=1,\ldots\}$ that converges at least linearly to the optimal solution of (SQP). The μ+1th point is obtained by solving a modification (SQP$_{\mu}$) of the original problem wherein the "proximal" term $\frac{1}{2} s^{-1}\|x - \bar{x}^{\mu}\|^2$ is added to the objective (s is a constant).

(SQP$_{\mu}$) is solved by applying the finite generation technique to its dual, (DQP$_{\mu}$). This technique replaces minimization over $\underline{Z}$ with minimization over the convex hull of a certain collection of elements, $\underline{Z}^{\nu} = \{\underline{\zeta}^1, \ldots, \underline{\zeta}^{N_{\nu}}\}$. (It is this minimization which is solved by MINOS.) The algorithm uses the information gained by solving (DQP$_{\mu}$) over co $\underline{Z}^{\nu}$ to generate a new collection $\underline{Z}^{\nu+1}$ and in this way obtains a sequence $\{\hat{\chi}^{\nu},\nu=1,\ldots\}$ [the dual variables to (DQP$_{\mu}$)] which converges at a linear rate to the optimal solution of (SQP$_{\mu}$).

The finite generation technique may be summarized as follows:

1) Find $(\hat{\chi}^{\nu},\hat{\underline{z}}^{\nu})$, the saddlepoint of $L^{\mu}(x,\underline{z})$ over $\chi \times$ co $\underline{Z}^{\nu}$.
2) Find $\underline{z}^{\nu} \in \underset{\underline{z} \in Z}{\operatorname{argmax}}\, L^{\mu}(\hat{\chi}^{\nu},\underline{z})$.
3) Find $\underline{Z}^{\nu+1} = \{\underline{\zeta}^1, \ldots, \underline{\zeta}^{N_{\nu+1}}\} \supset \{\hat{\underline{z}}^{\nu},\underline{z}^{\nu}\}$ and return to step 1) with $\nu = \nu + 1$.

L^{μ} is the Lagrangian associated with the primal-dual pair $\{$SQP$_{\mu}$,DQP$_{\mu}\}$.

The program itself is essentially two nested loops. The outer loop uses the current value of $\bar{x}^{\mu}$ to establish (SQP$_{\mu}$) and its dual. The inner loop applies the LFGM to obtain the sequence $\{\hat{\chi}^{\nu}\}$, which converges to $\bar{x}^{\mu+1}$. The inner loop calls MINOS to solve the resulting quadratic programs. An outline of the program follows:

0) Set $\mu = 0$ and initialize $\bar{x}^0$.
1) (Begin outer loop) Set SQP$_{\mu}$ = SQP + $\frac{1}{2} s^{-1}\|x - \bar{x}^{\mu}\|^2$ and establish DQP$_{\mu}$.
2) (Inner loop) Use the finite generation technique to generate $\{\hat{\chi}^{\nu}\}$, which converges to $\bar{x}^{\mu+1}$.
3) (End outer loop) Test for convergence of $\{\bar{x}^{\mu}\}$. If the sequence has not converged, set $\mu = \mu + 1$ and go to step 1).

Convergence

Due to the limited precision of the computer's internal representation of real numbers, it is not a simple matter to decide when the sequences $\{\bar{x}^{\mu}\}$ and $\{\hat{\chi}^{\nu}\}$ have converged. The user therefore must specify a number of tolerances which the program uses to determine whether the inner or outer loop has completed its task. Furthermore, the rate of convergence of the sequence $\{\bar{x}^{\mu}\}$ depends on the value of the constant s in problem (SQP$_{\mu}$), and if s is not chosen with care the sequence may not converge in a reasonable amount of time. The program will act to increase the rate of convergence if necessary but requires some guidance from the user to do so.

Three conditions cause the inner loop to terminate. They are

1) the current value of $\hat{\chi}^\nu$ represents a "good" step in the sequence $\{\bar{x}^\mu\}$ (i.e., if the substitution $\bar{x}^{\mu+1} = \hat{\chi}^\nu$ yields a linear rate of convergence for the sequence $\{\bar{x}^\mu\}$). $\hat{\chi}^\nu$ need not be precisely equal to the primal half of the saddlepoint of L^μ; it is sufficient that

$$\|\hat{\chi}^\nu - M_\mu(\bar{x}^\mu)\|^2 \le \delta^2\|\hat{\chi}^\nu - \bar{x}^\mu\|^2 \quad ,$$

where $M_\mu(\bar{x}^\mu)$ is the primal half of the true saddlepoint of L^μ and δ^2 is a nonnegative constant supplied by the user.

2) the values of successive $\hat{\chi}^\nu$s are changing very slowly or not at all. The inner loop halts and processing resumes in the outer loop if

$$\|\hat{\chi}^\nu - \hat{\chi}^{\nu-1}\| \,/\, \|\hat{\chi}^\nu\| \le \chi_\varepsilon \quad ,$$

where χ_ε is a nonnegative constant supplied by the user.

3) the sequence $\{\hat{\chi}^\nu\}$ has not converged as desired within a specified number of iterations. In this case, the current value of $\hat{\chi}^\nu$ is returned as $\bar{x}^{\mu+1}$ to the outer loop and a warning message is printed.

Similarly, three conditions cause the outer loop to terminate. They are

1) the current value of $\bar{x}^\mu$ is sufficiently close to the optimal solution of (SQP). This decision is made on the basis of the "duality gap." If the normalized difference between the values of (SQP) and its dual at $\bar{x}^\mu$ is less than a constant supplied by the user, the program prints the solution and halts.

2) the values of successive $\bar{x}^\mu$s are changing very slowly or not at all. The program prints the solution and halts if

$$|\bar{x}^\mu - \bar{x}^{\mu-1}| \,/\, |\bar{x}^\mu| \le \chi'_\varepsilon \quad ,$$

where χ'_ε is a nonnegative constant supplied by the user.

3) the sequence $\{\bar{x}^\mu\}$ has not converged as desired within a specified number of iterations. In this case, the value of s may be such that the sequence is converging too slowly. The program therefore increases the value of s and attempts to solve the problem once again. The user supplies an initial value for s, a constant, σ, used to generate new values of s according to the rule $s_{\text{new}}=\sigma s_{\text{old}}$, and a maximum permissible value for s. If the program cannot solve the problem using a value of s that is less than or equal to the maximum value, it writes a message to that effect and halts.

Distribution of the Random Variables

The presentation of the LFGM in [1] requires that the random variables $\underline{t}_{ij}$ and $\underline{h}_i$ have finite, discrete supports. The program allows the user to specify such a distribution in either of two ways.

The user may allow each component of $\underline{h}$ and of $\underline{T}$ to assume a value independently of the other components. In this case, each $\underline{t}_{ij}$ and $\underline{h}_i$ is a random variable, and the user specifies the number of points in the variable's support, the value the variable assumes at each point in its support, and the probability that the variable assumes the value associated with each point in its support. This is called an "independent distribution."

Alternatively, the user may supply a set of two sample populations, one containing observations of the vector $\underline{h}$ and one containing observations of of the matrix $\underline{T}$. In this case, each element of $\underline{h}$ and of $\underline{T}$ is assigned the expected value of the corresponding element of the observations in the appropriate sample. This is called a "Monte Carlo distribution." The user may obtain results for several sets of samples during a single invocation of the program. The user specifies the number of observations in each sample in the first set, the number of observations to add to each sample in the nth set to obtain the (n+1)st set, and a maximum sample size. The program repeats its calculations for every set generated in the manner described whose samples contain no more than the prescribed maximum number of observations.

User Supplied Routines

The user must write three subroutines, **uinput**, **smp**, and **output**, to perform various chores.

The program requires two distinct sets of input. The first set contains vital parameters and control information. This data appears in a single file, has a specific format (described in the section entitled "Control Information" below) and is read automatically by the program. The second set contains the actual data for the problem, i.e., the contents of the various matrices and vectors. After the program reads its control information, it calls the subroutine uinput to read this second set of input. The calling sequence is

```
    call uinput(a,b,c,d,e,ee,ff,qplus,qminus,
   1        x,
   2        pcexp,tcexp,pprob,tprob,nsuppp,nsuppt,
   3        npart,l,m,n).
```

The values of npart, l, m, and n are passed to the subroutine. The subroutine *must* return valid data in a, b, c, d, e, ee, ff, qplus, and qminus. The remaining variables need be assigned values only in certain cases. The parameters, their types, and their dimensions (where applicable) are listed below.

a(m,n) (real*8) upon return contains the values of the a_{ij}s.

b(m) (real*8) upon return contains the values of the b_is.

c(n) (real*8) upon return contains the values of the c_js.

d(n) (real*8) upon return contains the values of the d_js.

e(l) (real*8) upon return contains the values of the e_is.

ee(n) (real*8) upon return contains the values of the ee_js.

ff(n) (real*8) upon return contains the values of the ff_js. The program automatically adjusts the problem so that the bounds on x change from $ee_j \le x_j \le ff_j$ to $0 \le x_j \le r_j$.

qplus(l) and **qminus(l)** (both real*8) upon return contain the penalty coefficients for excess and shortage, respectively. The program automatically adjusts the problem to the form required in (SQP) (i.e., qminus $\equiv$ 0).

x(n) (real*8) upon return contains the value of $\bar{x}^0$, which is used to construct the first problem (SQP$_\mu$). The contents of this array are used only if the "initialize x vector indicator" is set accordingly (see the section on control information below).

pcexp(l,npart) (real*8) upon return contains the values that the rows of the vector $\underline{h}$ assume at each point in their respective supports. "pcexp(i,j)" contains the value of $\underline{h}_i$ at the jth point in its support. The contents of this array are used only if the "independent distribution flag" is set accordingly (see the section on control information below).

tcexp(l,n,npart) (real*8) upon return contains the values that the elements of the matrix $\underline{T}$ assume at each point in their respective supports. "tcexp(i,j,k)" contains the value of $\underline{t}_{ij}$ at the kth point in its support. The contents of this array are used only if the "independent distribution flag" is set accordingly (see the section on control information below).

pprob(l,npart) (real*8) upon return contains the probabilities that the rows of the vector $\underline{h}$ assume the values at each point in their respective supports. "pprob(i,j)" contains the probability that $\underline{h}_i$ assumes the value associated with the jth point in its support. The contents of this array are used only if the "independent distribution flag" is set accordingly (see the section on control information below).

tprob(l,n,npart) (real*8) upon return contains the probabilities that the elements of the matrix $\underline{T}$ assume the values at each point in their respective supports. "tprob(i,j,k)" contains the probability that $\underline{t}_{ij}$ assumes the value associated with the kth point in its support. The contents of this array are used only if the "independent distribution flag" is set accordingly (see the section on control information below).

nsuppp(l) (integer*2) upon return contains the number of points in the support of each row of the vector $\underline{h}$. "nsuppp(i)" contains the number of points in the support of $\underline{h}_i$. The contents of this array are used only if the "independent distribution flag" is set accordingly (see the section on control information below).

nsuppt(l,n) (integer*2) upon return contains the number of points in the support of each element of the matrix $\underline{T}$. "nsuppt(i,j)" contains the number of points in the support of $\underline{t}_{ij}$. The contents of ths array are used only if the "independent distribution flag" is set accordingly (see the section on control information below).

npart (integer) is the maximum number of points that may appear in the support of a single row of the vector $\underline{h}$ or of a single element of the matrix $\underline{T}$.

l (integer) is the number of random constraints.

m (integer) is the number of deterministic constraints.

n (integer) is the number of decision variables.

If the user has specified that a Monte Carlo distribution is to be used, the program calls the subroutine smp to obtain sample populations of the vector $\underline{h}$ and of the matrix $\underline{T}$. The subroutine must generate a specified number of vectors and matrices and must place these new observations into the appropriate samples. The calling sequence is

call smp(newsmp,numsmp,maxsmp,l,n,p,t,dseed1)

All the parameters contain values when passed to the subroutine, which must return valid data in p and t. The parameters, their types, and their dimensions (where applicable) are listed below.

newsmp (integer) is the index into the arrays p and t where the first of the new observations should be placed.

numsmp (integer) is the index into p and t where the last of the new observations should be placed. The subroutine must therefore generate numsmp - newsmp + 1 observations of the vector $\underline{h}$ and the same number of observations of the matrix $\underline{T}$.

maxsmp (integer) is the maximum number of observations that may appear in a sample.

l (integer) is the number of random constraints.

n (integer) is the number of decision variables.

p(l,maxsmp) (real*8) contains the observations of the vector $\underline{h}$ generated by previous calls to the subroutine. Upon return, "p(i,newsmp)," "p(i,newsmp+1)," ..., "p(i,numsmp)" contain the values of $\underline{h}_i$ generated by the current call.

t(l,n,maxsmp) (real*8) contains the samples of the matrix $\underline{T}$ generated by previous calls to the subroutine. Upon return, "t(i,j,newsmp)," "t(i,j,newsmp+1)," ..., "t(i,j,numsmp)" contain the values of $\underline{t}_{ij}$ generated by the current call.

dseed1 (real*8) is provided for use as a random number generator seed.

Once the solution has been found, the program calls the subroutine output to print the results in whatever format desired. The calling sequence is

```
call output(x,
        a,b,c,d,e,ee,r,q,
        l,m,n,
        discrt,
        pcexp,tcexp,pprob,tprob,nsuppp,nsuppt,
        p,t,numsmp).
```

All values are passed to the subroutine, although some do not contain valid data. Furthermore, several of the values do not match those entered because the program adjusts the problem as described in the discussion of the parameters to the subroutine uinput, above. The parameters, their types, and their dimensions (where applicable) are listed below.

x(n) (real*8) contains the optimal solution to the adjusted problem.

a(m,n) (real*8) contains the coefficients a_{ij}.

b(m) (real*8) contains the values of the b_is as modified by the program.

c(n) (real*8) contains the coefficients c_j as modified by the program.

d(n) (real*8) contains the coefficients d_j as modified by the program.

e(l) (real*8) contains the values of the e_is.

ee(n) (real*8) contains the values of the ee_js.

r(n) (real*8) contains the values of the ff_js as modified by the program.

q(l) (real*8) contains the values of qplus as modified by the program.

l (integer) is the number of random constraints.

m (integer) is the number of deterministic constraints.

n (integer) is the number of decision variables.

discrt (logical) is a flag indicating how the distribution of the random variables has been specified. If it is .TRUE., the distribution is an independent one. If it is .FALSE., the distribution is a Monte Carlo one.

pcexp(l,npart) (real*8) contains the values that the rows of the vector $\underline{h}$ assume at each point in their respective supports. This array contains valid data only if the discrt flag above is .TRUE..

tcexp(l,n,npart) (real*8) contains the values that the elements of the matrix $\underline{T}$ assume at each point in their respective supports. This array contains valid data only if the discrt flag above is .TRUE..

pprob(l,npart) (real*8) contains the probabilities that the rows of the vector $\underline{h}$ assume the values at each point in their respective supports. This array contains valid data only if the discrt flag above is .TRUE..

tprob(l,n,npart) (real*8) contains the probabilities that the elements of the matrix $\underline{T}$ assume the values at each point in their respective supports. This array contains valid data only if the discrt flag above is .TRUE..

nsuppp(l) (integer*2) contains the number of points in the support of each row of the vector $\underline{h}$. This array contains valid data only if the discrt flag above is .TRUE..

nsuppt(l,n) (integer*2) contains the number of points in the support of each element of the matrix $\underline{T}$. This array contains valid data only if the discrt flag above is .TRUE..

p(l,numsmp) (real*8) contains the observations of the vector $\underline{h}$ generated by the subroutine smp. This array contains valid data only if the discrt flag above is .FALSE..

t(l,n,numsmp) (real*8) contains the samples of the matrix $\underline{T}$ generated by the subroutine smp. This array contains valid data only if the discrt flag above is .FALSE..

numsmp (integer) contains the number of samples in the arrays p and t.

Filenames and Unit Numbers

The user must specify a filename for each of the eight files used by the program and must specify unit numbers for most of them. The files and the variables that correspond to their unit numbers are described in the next few sections. All files are opened (and their unit numbers established) in the subroutine named "input."

Unit number 8 is reserved and may not be assigned by the user.

Control Information

The user must supply the program with several constants, tolerances, and limits. This control information resides in a "specs" file. The variable "inp" contains the unit number of this file.

The specs file contains the information shown below. Each value appears on a separate line and all values begin in column 31 (the first thirty columns may be used for comments). The information must appear in the order specified below:

- **name of the problem.** This is read using an A32 format.
- **number of random constraints (l).** This value must not exceed the constant "lmax" (see the section entitled "Common Blocks and User-Acessible Parameters" below). It is read using an I5 format.
- **number of deterministic constraints (m).** This value must not exceed the constant "mmax" (see the section entitled "Common Blocks and User-Acessible Parameters" below). It is read using an I5 format.
- **number of decision variables (n).** This value must not exceed the constant "nmax" (see the section entitled "Common Blocks and User-Acessible Parameters" below). It is read using an I5 format.
- **independent distribution flag.** This is read using an L10 format. If the independent distribution flag is true, the user must provide an independent distribution for the random variables (see the section entitled "Distribution of the Random Variables" above). In this case, the Monte Carlo distribution flag (see below) must be false and the control variables dealing with Monte Carlo simulation (starting sample size, sample size increment, maximum sample size, and random number generator seed) are read but are not used.
- **independent distribution maximum number of partitions.** This is the maximum number of values that an element of the vector $\underline{h}$ or of the matrix $\underline{T}$ may assume (i.e., the maximum number of points in the support of an element). The constant "smpmax" places a limit on this value (see the section entitled "Common Blocks and User-Acessible Parameters" below). It is read using an I5 format. It is not used if the independent distribution flag is false.
- **Monte Carlo distribution flag.** This is read using an L10 format. If the Monte Carlo distribution flag is true, the user must provide a Monte Carlo distribution for the random variables (see the section entitled "Distribution of the Random Variables" above). In this case, the independent distribution flag (see above) must be false and the control variable dealing with independent distributions (the maximum number of partitions) is read but is not used.

- **Monte Carlo distribution starting sample size.** This is the number of observations in each of the two sample populations in the first set. This value must not exceed the constant "smpmax" (see the section entitled "Common Blocks and User-Acessible Parameters" below). It is read using an I5 format. It is not used if the Monte Carlo distribution flag is false.
- **Monte Carlo distribution sample size increment.** This is the number of new observations to add to each sample in the nth set to obtain the (n+1)th set. This value is read using an I5 format. It is not used if the Monte Carlo distribution flag is false.
- **Monte Carlo distribution maximum sample size.** This is the maximum number of observations that may appear in a sample. This value must not exceed the constant "smpmax" (see the section entitled "Common Blocks and User-Acessible Parameters" below). It is read using an I5 format. It is not used if the Monte Carlo distribution flag is false.
- **Monte Carlo distribution random number generator seed.** This is passed to the user subroutine smp. This value is read using an F10.4 format. It is not used if the Monte Carlo distribution flag is false.
- **maximum number of outer loop iterations.** This value is read using an I5 format.
- **maximum number of inner loop iterations.** This value is read using an I5 format.
- **maximum number of finite elements.** This is the maximum number of elements $\underline{\zeta}$ that may appear in the collection $\underline{Z}^\nu$. This value must not exceed the constant "nymax" (see the section entitled "Common Blocks and User-Acessible Parameters" below). It is read using an I5 format.
- **proximal point algorithm control.** This flag controls whether the proximal point algorithm is used. It is read using an L10 format. If this flag is true, the proximal point algorithm is used. If this flag is false, the proximal point algorithm is not used and the proximal term does not appear in (SQP_μ) (i.e., (SQP_μ) is identical to the original problem).
- **proximal point algorithm starting s-value (s).** This is the value of s used to obtain the first problem (SQP_μ). This value is read using a D10.4 format.
- **proximal point algorithm maximum s-value.** This is the maximum value of s for which the program will attempt to solve the problem. This value is read using a D10.4 format.
- **proximal point algorithm s-adjusting factor (σ).** This value must be greater than 1 and is read using an F10.4 format.
- **proximal penalty factor (δ^2).** This value is read using an F10.4 format.
- **minimum change in duals for the outer loop (χ'_ε).** This value is read using a D10.4 format.
- **minimum change in duals for the inner loop (χ_ε).** This value is read using a D10.4 format.

- **minimum duality gap.** This value is used to determine whether the first stopping criterion for the outer loop is satisfied. It is read using a D10.4 format.
- **initialize x vector indicator.** This value controls whether an initialization subroutine is called to establish the value of the first vector in the sequence $\{\bar{x}^\mu\}$. If it is 1, no initialization is performed and the value of $\bar{x}^0$ must be provided by the user subroutine uinput. This value is read using an I5 format.
- **print option.** This controls the amount of information printed by the program. This value is read using an I5 format.

MINOS Files

Five files are used by MINOS. They are listed below along with the variables which contain their unit numbers.

MINOS File	Unit Number in
specifications	specs
input data	mps
output	minpr
scratch file	(assigned to unit 8)
dump file	mindmp

Output File

This file contains the results generated by the program. The variable "out" contains the unit number of this file.

Proximal Sequence Output File

This file contains the vectors in the sequence $\{\bar{x}^\mu\}$. The variable "eval" contains the unit number of this file.

Data Structures

This program uses no particularly complicated data structures. However, it does use one (very large) array to hold the contents of every matrix and vector used by the algorithm. Several pointers provide the necessary bookkeeping information. The following list shows which values appear where within the array "zz."

Matrix or Vector	Starting Index	Ending Index
A	1	n1
b	n1+1	n2
c	n2+1	n3
d	n3+1	n4
e	n4+1	n5
ee	n5+1	n6
ff	n6+1	n7
qplus	n7+1	n8
qminus	n8+1	n9
x	n9+1	n10
pcexp	n10+1	n11
tcexp	n11+1	n12
pprob	n12+1	n13

tprob	n13+1	n14
nsuppp	n14+1	n15
nsuppt	n15+1	n16
np	n16+1	n17
p	n17+1	n18
t	n18+1	n19
piexp	n19+1	n20
tiexp	n20+1	n21

(for INIT)

v0	n21+1	n22
v1	n22+1	n23
v2	n23+1	n24

(for LOOP)

chi	n19+1	n20
oldchi	n20+1	n21
zeta	n21+1	n22
dk	n22+1	n23
ck	n23+1	n24
lam	n24+1	n25
qexp	n25+1	n26
pexp	n26+1	n27
texp	n27+1	n28
w1	n28+1	n29
w0	n29+1	n30
w2	n30+1	n31
y	n31+1	n32

Common Blocks and User-Accessible Parameters

The program uses several common blocks, most of which are contained in three include files. The names of these blocks, the include files in which they appear, and a description of the variables they contain follow.

Common Block	Include File	Contents
zzcore	incl.core	all global matrices and vectors
ziodev	incl.glob	I/O unit numbers
zdimen	incl.glob	dimensions of matrices, current sample size
zmxdms	incl.glob	various maxima
zparam	incl.glob	s, σ, δ^2, χ_ε, etc.
zseed	incl.glob	random number generator seeds
zdistr	incl.glob	distribution description flags
zpntrs	incl.pntrs	bookkeeping indices into the array "zz"

The program currently imposes several limits on the size of the problem (e.g., no more than four decision variables). To obtain results for larger problems, the user must change the constants in the include file "incl.core" as follows:

nmax must be set to the maximum number of decision variables (n).

lmax must be set to the maximum number of random constraints (l).

mmax must be set to the maximum number of deterministic constraints (m).

nymax must be set to the maximum number of elements ζ in any set $\underline{Z}^{\nu}$.

smpmax must be set to the maximum number of points in the space over which the integrals for the expected values of h and of T are calculated. For independent distributions, it is also used to allocate storage for a number of additional arrays. "smpmax" must be set to the maximum number of members in any sample if a Monte Carlo distribution of the random variables is given and to

$$\max [2\mathrm{npmax} + 1 , \mathrm{npmax}^{\mathrm{nmax}+1}],$$

where npmax is the maximum number of points in any random variable's support, if an independent distribution of the random variables is given.

None of these values may be less than 1.

Subprocedure Hierarchy

See the attached figure.

Library Routines

The program uses version 3.9 of MINOS. The MINOS subroutine "go" has been slightly modified to perform additional file assignments.

Notes

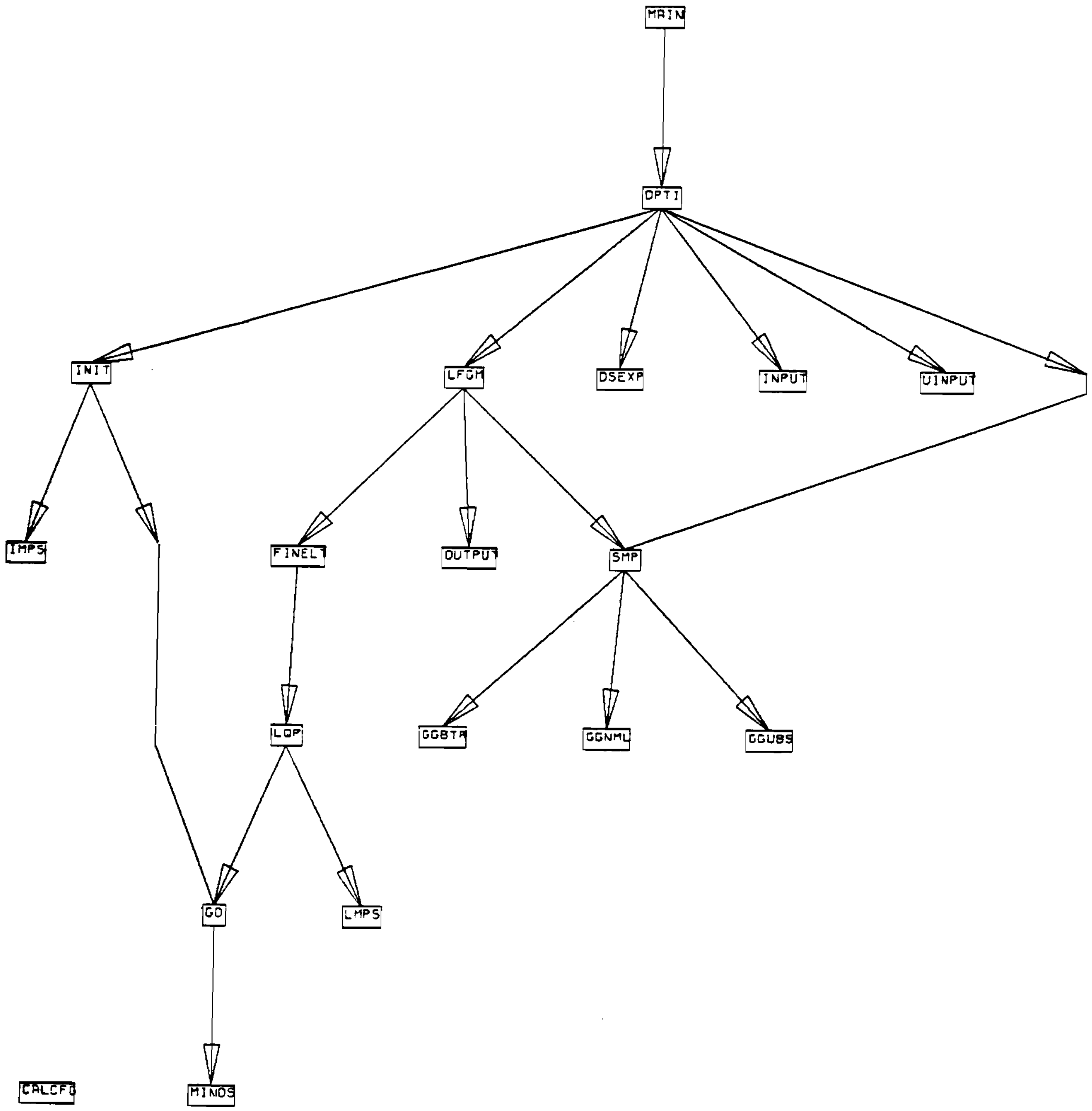
On occasion, MINOS will return an error code and the program will halt. This is usually due to insufficient space or iteration limits. The MINOS specifications file is written by the subroutine lmps, which may need to be changed in such cases.

References

[1] R.T. Rockafellar and R. Wets, "A Lagrangian Finite Generation Technique for Solving Linear-Quadratic Problems in Stochastic Programming," IIASA Working Paper WP-84-25, 1984

[2] A. King, "An Implementation of the LFGM," in **Numerical Methods for Stochastic Optimization**, Y. Ermol'ev and R. Wets (eds), to appear in 1985

[3] B. Murtagh and M. Saunders, "Large Scale Linearly Constrained Optimization," *Mathematical Programming* v. 14, pp. 41-72, 1978


MAIN
OPTI
INIT
LFGM
DSEXP
INPUT
UINPUT
IMPS
FINELT
OUTPUT
SMP
LQP
GGBTR
GGNML
GGUBS
GD
LMPS
CALCFG
MINOS

Descent Stochastic Quasigradient Methods

Kurt Marti

HSBw Muenchen, FB LRT
Werner-Heisenberg-Weg 39
D-8014 Neubiberg/Muenchen

1. Introduction

The FORTRAN code "SEMI STOCHASTIC APPROXIMATION" can be applied to solve stochastic optimization problems of the following type

$$\text{minimize } F(x) \text{ s.t. } x \in D, \tag{1}$$

where D is a closed convex subset of $\mathbb{R}^n$ and $F=F(x)$ is the convex mean value function defined by

$$F(x) = Eu[A(\omega) - b(\omega)],\ x \in \mathbb{R}^n. \tag{1.1}$$

Here $[A(\omega), b(\omega)]$ is an $m \times (n+1)$ random matrix and u is a convex loss function on $\mathbb{R}^m$ such that the mean value $F(x)$ in (1.1) is real for every $x \in \mathbb{R}^n$. We suppose that the set D^* of optimal solutions x^* of (1) is nonempty.

Problems of the form (1) arise in many different connections, e.g.
Stochastic linear programming with recourse [7],[22]
Portfolio optimization [9],[23]
Error minimization and optimal design [2],[20]
Statistical prediction [1]
Optimal decision functions [5],[10].

Since the gradient (or subgradient) ∂F of F exists under weak assumptions and is given then by the formula

$$\partial F(x) = EA(\omega)^T \partial u[A(\omega)x - b(\omega)], \tag{2}$$

where A^T is the transpose of a matrix A and ∂u denotes the subgradient of u, our basic problem (1) could be attacked in principle by a gradient (or quasigradient) procedure of the type

$$x_{k+1} = P_D(x_k - \rho_k g_k),\ k=1,2,\dots, \tag{3}$$

where $\rho_k > 0$ is a step size, $g_k \in \partial F(x_k)$ and P_D denotes the projection of $\mathbb{R}^n$ onto D.

However, in practice the computation of the gradient (subgradient) $\partial F(x_k)$ is beset by one of the following difficulties:

- Formula (2) cannot be evaluated at all because only a stochastic estimate Y_k of an element $g_k \in \partial F(x_k)$ is available [3],[21]. In this case we have only

$$Y_k = g_k + \text{noise with some } g_k \in \partial F(x_k) \tag{4.1}$$

- Although the integrand $A^T\partial u(Ax-b)$ and the probability distribution $P_{[A(\cdot),b(\cdot)]}$ of $[A(\omega),b(\omega)]$ in (2) is known, the numerical evaluation of this formula - which involves a multiple integral - is computationally infeasible. In this case $\partial F(x_k)$ may be approximated by

$$Y_k \in A(\omega_k)^T\partial u[A(\omega_k)x_k - b(\omega_k)], \tag{4.2}$$

where $[A(\omega_k),b(\omega_k)]$ is a realization of the random matrix $[A(\omega),b(\omega)]$ generated independently of x_k by means of a pseudo-random generator [11].

Consequently, in both cases (4.1) and (4.2) the gradient procedure of (3) cannot be applied in practice. It is therefore often replaced by the stochastic quasigradient method [3],[6]

$$X_{k+1} = P_D(X_k - \rho_k Y_k),\ k=1,2,..., \tag{5}$$

where the random direction $-Y_k$ is defined by (4.1) or (4.2) as appropriate.

Selecting *a priori* a sequence of step sizes $\rho_1,\rho_2,...$ such that

$$\rho_k>0,\ \sum_{k>0}\rho_k = +\infty,\ \sum_{k>0}\rho_k^2 < +\infty,$$

e.g., $\rho_k = \frac{c}{q+k}$ for some constants $c>0$ and $q\in N\cup\{0\}$, it is well known [19],[21] that the sequence of random iterates $X_1,X_2,...$ generated by (5) converges with probability one to the set D^* of optimal solutions x^* of (1), provided that the approximates Y_k of $\partial F(x_k)$ fulfill a certain uniform second order integrability condition and that D^* is a bounded set.

Unfortunately, due to their probabilistic nature, stochastic approximation procedures have a very slow asymptotic rate of convergence of the type

$$E\|X_k - x^*\|^2 = O(k^{-\lambda}),$$

where λ is a constant such that $0<\lambda\leq 1$. Moreover, the main disadvantage of stochastic quasigradient procedures such as (5) is their nonmonotonicity which sometimes may manifest itself as a highly oscillatory behavior [4]. Hence, in many cases it is not known when the algorithm has reached a certain neighborhood of an optimal solution x^*. To improve the convergence properties of (5), several methods have been suggested, including those based on the adaptive selection of the step sizes ρ_k, see [8], and on the use of second order information about F, see [18]. An additional method - which has a partial monotonicity property - is presented in the next section.

2. Semi-Stochastic Approximation

As was shown in several papers [10],[12],[14],[15],[17], for several classes U of convex loss functions u and several classes Π of distribution $P_{[A(\cdot),b(\cdot)]}$ of the random matrix $[A(\omega),b(\omega)]$, our minimization problem (1) has the following important

PROPERTY: (6)

At certain "nonefficient" or "nonstationary" points $x\in D$ there exists a deterministic (feasible) descent direction $h=h(x)$ of F which can be computed with less effort than can an element g_k of $\partial F(x_k)$. Moreover, $h(x)$ is stable with respect to variations of the loss function $u\in U$.

Consequently, if at a certain iteration point X_k property (6) holds, then clearly one can replace the stochastic direction $-Y_k$, which is a descent direction only in the mean, by the descent direction $h_k=h(X_k)$ of F.

We thus obtain the following, as already described in [11],[13]:

Descent Stochastic Quasigradient Method

$$X_{k+1} = \begin{cases} P_D(X_k + \rho_k h_k) \text{ if (6) holds at } X_k \\ P_D(X_k - \rho_k Y_k) \text{ otherwise.} \end{cases} \tag{7.1}$$

In many important applications this hybrid procedure has the important feature that property (6) is frequently satisfied, for example at every other iteration point X_k. In this case, (7.1) has the more convenient form

$$X_{k+1} = \begin{cases} P_D(X_k + \rho_k h_k),\ k \in N_1 \\ P_D(X_k - \rho_k Y_k),\ k \in N_2' \end{cases} \tag{7.2}$$

where N_1, N_2 is a known decomposition of the set of integers, N, e.g. $N_1=\{1,2,3,...\}$ and $N_2=\{2,4,6,...\}$. As was shown in [13], if the step sizes $\rho_1, \rho_2, ...$ are chosen such that

$$\rho_k > 0,\ \sum_{k \in N_2} \rho_k = +\infty,\ \sum_{k>0} \rho_k^2 < +\infty,$$

then the semi-stochastic approximation procedure (7) converges with probability one to the set D^* of optimal solutions x^* of (1). As expected, several numerical examples [11] show that the descent stochastic quasigradient method (7) has a much better rate of convergence than the pure stochastic quasigradient method. In particular, the highly oscillatory behavior of the random iterates $X_1, X_2, ...$ observed in (5) is greatly damped by the use of the deterministic descent directions h_k in (7); moreover, the approximation to the set D^* is more exact. In a recent paper [16], the rate of convergence of (7) was estimated using the following

THEOREM 2.1

Denote by $b_k = E\|X_k - x^*\|^2$ and $b_k^s = E\|X_k^s - x^*\|^2$ the mean square error of the descent stochastic quasigradient and pure stochastic quasigradient, respectively.

a) If the ratio of stochastic to deterministic steps taken in (7) is fixed, then there exist constants Q_1, Q_2 with $0<Q_1<1$ and $Q_1<Q_2$ such that

$$Q_1 \cdot b_k^s \le b_k \le Q_2 \cdot b_k^s \text{ as k approaches infinity.} \tag{8.1}$$

Furthermore, Q_1 and Q_2 are given by known formulas and $Q_2<1$ if $N/M<\gamma$, where N and M are the number of stochastic and deterministic steps, respectively, in one complete turn of iterations and γ is a constant that depends on the parameters of problem (1).

b) If the stochastic steps in (7) are made at a decreasing rate, the rate of convergence is increased from $O(1/k)$ in the pure stochastic case to $O(k^{-\lambda})$, where $1<\lambda<2$, in the semi-stochastic case.

3. Construction of Deterministic Descent Directions

Currently deterministic feasible descent directions may be constructed if the distribution $P_{[A(\cdot),b(\cdot)]}$ is

* stable [12]
* invariant [15]
* discrete [14].

Our implementation is based on the assumption that

$[A(\omega),b(\omega)]$ has an $m \times (n+1)$ normal distribution (9)

with mean $(\bar{A},\bar{b})$ (9.1)

and covariance matrix

$$Q = \begin{bmatrix} Q_{11} & Q_{12} & \cdots & Q_{1m} \\ Q_{21} & Q_{22} & \cdots & Q_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ Q_{m1} & Q_{m2} & \cdots & Q_{mm} \end{bmatrix} \tag{9.2}$$

where the $(n+1)\times(n+1)$ matrix Q_{ij} denotes the covariance matrix of the ith and jth rows of the random matrix $[A(\omega),b(\omega)]$.

In addition to (9) we suppose

The objective function F of (1) is not constant on arbitrary line segments $\overline{xy}$ of R^n. (10)

From (9) is follows that the random m-vector $A(\omega)x - b(\omega)$ has a normal distribution with mean $\bar{A}x - \bar{b}$ and covariance matrix

$$Q_x = \begin{bmatrix} \hat{x}^TQ_{11}\hat{x} & \hat{x}^TQ_{12}\hat{x} & \cdots & \hat{x}^TQ_{1m}\hat{x} \\ \hat{x}^TQ_{21}\hat{x} & \hat{x}^TQ_{22}\hat{x} & \cdots & \hat{x}^TQ_{2m}\hat{x} \\ \cdots & \cdots & \cdots & \cdots \\ \hat{x}^TQ_{m1}\hat{x} & \hat{x}^TQ_{m2}\hat{x} & \cdots & \hat{x}^TQ_{mm}\hat{x} \end{bmatrix},$$

where $\hat{x} = \begin{bmatrix} x \\ -1 \end{bmatrix}$.

The key to the construction of descent directions is now

***THEOREM* 3.1**

Suppose that assumptions (9) and (10) are justified. If the n-vectors x and $y \neq x$ are related according to

$$\bar{A}x = \bar{A}y \tag{11.1}$$

and

$$Q_x - Q_y \text{ is positive semidefinite,} \tag{11.2}$$

then $F(y) \leq F(x)$ and $h=y-x$ is a descent direction of F at x. Moreover, if $x \in D$ and in addition to (11.1) and (11.2) we have

$$y \in D, \tag{11.3}$$

then $h=y-x$ is a feasible descent direction of F at x.

NOTE

For a given x, (11.1) is a system of m linear equations for y. Relation (11.2) means that the smallest eigenvalue of $Q_x - Q_y$ is nonnegative. In the important special case m=1, (11.2) is reduced to the single quadratic constraint

$$\hat{x}^TQ_{11}\hat{x} \geq \hat{y}^TQ_{11}\hat{y}. \tag{11.2a}$$

If $[A(\omega),b(\omega)]$ has stochastically independent rows, then (11.2) is equivalent to

$$\hat{x}^TQ_{ii}\hat{x} \geq \hat{y}^TQ_{ii}\hat{y} \text{ for all } i=1,2,\ldots,m. \tag{11.2b}$$

In this case, solutions y of (11) may be obtained by solving for a given vector x the convex program

$$\text{minimze } \hat{y}^TQ_{i_0i_0}\hat{y} \tag{12}$$

subject to

$$\hat{y}^T Q_{ii}\hat{y} \le \hat{x}^T Q_{ii}\hat{x},\ i=1,2,\dots,m$$
$$\bar{A}y = \hat{A}x$$
$$y \in D,$$

where $1 \le i_0 \le m$ is a fixed integer.

In the general case one must consider the program

$$\text{maximize } \lambda(Q_x - Q_y) \tag{13}$$

subject to

$$\bar{A}y = \hat{A}x$$
$$y \in D,$$

where $\lambda(Q_x - Q_y)$ denotes the smallest eigenvalue of $Q_x - Q_y$.

4. Implementation

4.1. Representation of the random matrix $[A(\omega),b(\omega)]$

$[A(\omega),b(\omega)]$ is defined by

$$[A(\omega),b(\omega)] = [A^0,b^0] + \sum_{j=1,r} \omega^j[A^j,b^j], \tag{14}$$

where $[A^j,b^j]$, $j=0,1,\dots,r$, are $m\times(n+1)$ matrices to be selected by the user and $\omega^1,\omega^2,\dots,\omega^r$ are independent normal random variables with mean zero and variance one. A realization $[A(\omega_k),b(\omega_k)]$ of $[A(\omega),b(\omega)]$ is then given by

$$[A(\omega_k),b(\omega_k)] = [A^0,b^0] + \sum_{j=1,r} \omega_k^j[A^j,b^j],$$

where $\omega_k = (\omega_k^1,\omega_k^2,\dots,\omega_k^r)$, $k=0,1,\dots$, is a sequence of stochastically independent realizations of the random r-vector $\omega=(\omega^1,\omega^2,\dots,\omega^r)$ generated by means of a pseudo-random generator that converts uniformly distributed pseudo-random numbers into normally distributed ones based on the central limit theorem.

4.2. Computation of the search directions

We suppose that rank $\bar{A}$ = rank $A^0 = m < n$. The matrix $\bar{A} = [\bar{a}_1,\bar{a}_2,\dots,\bar{a}_m]$, where $\bar{a}_k$ is a column vector, must be partitioned by the user into a regular $m\times m$ matrix

$B=[\bar{a}_{k_1},\bar{a}_{k_2},\dots,\bar{a}_{k_m}]$

and an $m\times(n-m)$ remainder matrix

$E=[\bar{a}_{\kappa_1},\bar{a}_{\kappa_2},\dots,\bar{a}_{\kappa_{n-m}}]$.

The user must then define the index set

INDXA0=$\{k_1,k_2,\dots,k_m,\kappa_1,\kappa_2,\dots,\kappa_{n-m}\}$.

Given the last iteration point x_k, subroutine FUNCT computes a solution y_k of the relations (11.1)-(11.3). At present only the case of $D=\mathbb{R}^n$ is implemented. For the sake of generality the system of relations (11) is solved by means of the program (13). If the situation demands it, the user need only replace this subroutine with a custom procedure for solving (11).

If $y_k \ne x_k$, then $h_k=y_k-x_k$ is a feasible descent direction (see theorem 3.1) and the next iteration point x_{k+1} is given by

$$x_{k+1} = x_k + \rho_k(y_k - x_k),$$

where $\rho_k>0$ is a step size.

If $y_k=x_k$, then FUNCT fails to find a descent direction. The next iteration point is then given by

$$x_{k+1} = x_k - \rho_k Y_k,$$

where

$Y_k \in A(\omega_k)^T \partial u[A(\omega_k)x_k - b(\omega_k)]$.

4.3. Step size

At present, the step sizes ρ_k, k=0,1,..., are defined by

$$\rho_k = \frac{1}{k+1}.$$

For a deterministic step the user may also take $\rho_k=1$ or $\rho_k=0.5$.

4.4. Loss function u

The following classes of loss function are supported:

a) Quadratic loss function

$$u(z) = c + q^T z + z^T W z,\ z \in R^m,$$

where c is a fixed number, q is an m-vector, and W is a positive semidefinite m×m matrix.

b) Pclynomial loss function

$$u(z) = \sum_{j=1,m} z_j^{2s},\ z=(z_1, \ldots, z_m)^T \in R^m,$$

where s is a fixed integer.

c) Sublinear loss function

$$u(z) = \max_{1\le t\le p} f_t^T z,\ z \in R^m,$$

where $f_1, f_2, \ldots, f_p$ are fixed m-vectors.

4.5. Stopping criteria

The user must select a (small) positive number EPS>0, an integer ITMAX, and a number TMAX. The program executes until one of the following conditions is fulfilled:

$\|x_{k+1} - x_k\| \le$ EPS,

k>ITMAX (= maximum number of iterations),

or

T>TMAX (= maximum computing time)†,

where $\|\cdot\|$ denotes the Euclidean norm.

† Since system calls to determine the time and date vary from machine to machine, the code has been changed so that this test is no longer performed - Ed.

Acknowledgment
The FORTRAN code was written by A. Boehme.

References

[1] Aitchison, J., Dunsmore, I.R.: Statistical Prediction Analysis. Cambridge: University Press 1975

[2] Astroem, K.J.: Introduction to Stochastic Control Theory. New York-London: Academic Press 1970

[3] Ermoliev, Yu.: Stochastic Quasigradient Methods and their Application to System Optimization. Stochastics 9, 1-36 (1983)

[4] Ermoliev, Yu., Gaivoronsky, A.: Stochastic Quasigradient Methods and their Implementation. IIASA Working Paper, Laxenburg 1983

[5] Ferguson, T.S.: Mathematical Statistics. New York-London: Academic Press 1967

[6] Hirriart-Urruty, J.B.: Contributions a la programmation mathematique: Cas deterministe et stochastique. Thesis, University of Clermont-Ferrand II, 1977

[7] Kall, P.: Stochastic Linear Programming. Berlin-Heidelberg-New York: Springer 1976

[8] Kesten, H.: Accelerated stochastic approximation. Ann. Math. Statist. 29, 41-59 (1958)

[9] Marti, K., Riepl, R.-J.: Optimale Portefeuilles mit stabil verteilten Renditen. ZAMM 57, T337-T339 (1977)

[10] Marti, K.: Approximationen stochastischer Optimierungsprobleme. Koenigstein/Ts.: Hain 1979

[11] Marti, K.: On solutions of stochastic programming problems by descent procedures with stochastic and deterministic directions. Methods of Operations Research 33, 281-293 (1979)

[12] Marti, K.: Stochastic linear programs with stable distributed random variables. In: Optimization Techniques Part 2, J. Stoer (ed.), Lecture Notes in Control and Information Sciences 7, 76-86 (1978)

[13] Marti, K.: Solving stochastic linear programs by semi-stochastic approximation algorithms. In: Recent results in stochastic programming, P. Kall, A. Prekopa (eds.), Lecture Notes in Economics and Mathematical Systems 179, 191-213 (1980)

[14] Marti, K.: On the construction of descent directions in a stochastic program having a discrete distribution. ZAMM 64, T336-T338 (1984)

[15] Marti, K.: Computation of Descent Directions in Stochastic Optimization Problems with Invariant Distributions. To appear in ZAMM 64 (1984), Heft 11

[16] Marti, K., Fuchs, E.: Rates of convergence of semi-stochastic approximation procedures for solving stochastic optimization problems. Preprint in: Mitteilungen des Forschungsschwerpunktes Simulation und Optimierung deterministischer und stochastischer dynamischer Systeme. HSBw Muenchen, November 1984.

[17] Marti, K.: Stochastische Dominanz und stochastische lineare Programme. Methods of Operations Research 23, 141-160 (1977)

[18] Poljak, B.T., Tsypkin, Ya.Z.: Robust pseudogradient adaption algorithms. Automation and Remote Control 41, 1404-1410 (1980)

[19] Schmetterer, L.: Stochastic approximation. Proceedings 4th Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1, 587-609 (1960)

[20] Sorenson, H.W.: Parameter Estimation. New York-Basel: M. Dekker 1980

[21] Wasan, M.T.: Stochastic Approximation. Cambridge: University Press 1969

[22] Wets, R.: Stochastic Programming: Solution Techniques and Approximation Schemes. In: Mathematical Programming: The State of the Art, A. Bachem, M. Groetschel, B. Korte (eds.), 566-603, Berlin-Heidelburg-New York: Springer 1983

[23] Ziemba, W.T., Vickson, R.G. (eds.): Stochastic Optimization Models in Finance. New York-London: Academic Press 1975

SPORT User's Manual

J. Edwards

Introduction

This program, Stochastic Programming Optimizer with Recourse and Tenders (SPORT), implements Nazareth and Wets' inner linearization method for stochastic programs with recourse. It also includes an implementation of a method for solving simple recourse problems that relies on the introduction of bounded variables. The two methods are called ILSRDD (Inner Linearization, Simple Recourse, Discrete Distribution) and BVSRDD (Bounded Variables, Simple Recourse, Discrete Distribution), respectively, and are described in [1], [2], and [3]. The program was developed at the University of California at Berkeley and at IIASA by Larry Nazareth. It is written in FORTRAN IV. A description of the implementation may be found in [1]. The contents of this manual are largely taken from [1], [2], and [3].

The current version of SPORT (Version 1.0) addresses problems with simple recourse and stochastic right hand side elements with a given discrete probability distribution.

The Problem

SPORT is designed to solve two-stage stochastic linear programs with recourse, whose general form is to find $x \in R^{n_1}$ to minimize

$$cx + E_w[Q(x,w)] \tag{1}$$

subject to

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

where

$$Q(x,w) = \inf_{y \geq 0}[qy \mid Wy = h(w) - Tx] \quad , \tag{1.1}$$

$h(w)$ is a random vector with m_2 elements defined on a probability space whose events are denoted by w; x is the decision vector and contains n_1 elements; y is the optimal recourse vector given some (x,w) and contains n_2 elements; A, T, and W are fixed matrices with dimensions $m_1 \times n_1$, $m_2 \times n_1$, and $m_2 \times n_2$, respectively; b, c, and q are fixed vectors containing m_1, n_1, and n_2 elements, respectively; and 'E_w' denotes mathematical expectation with respect to w. Note that only the right-hand-side, $h(w)$, is random.

As noted above, the current version of SPORT solves the above problem only for simple recourse (i.e., $W = [I,-I]$), stochastic right-hand-side elements with a given discrete probability distribution, and penalty vectors q^+ and q^- associated with shortage and surplus, respectively, in the recourse stage (1.1).

Thus, (1.1) may be written

$$Q(x,w) = \min_{y^+,y^- \geq 0} [q^+y^+ + q^-y^- \mid Iy^+ - Iy^- = h(w) - Tx] \qquad (1.1')$$

where

$$q^+, q^- \geq 0$$

and the ith row of $h(w)$ may assume one of the values $h_{i1}, \ldots, h_{ik_i}$, where $h_{i,j} < h_{i,j+1}$, with probabilities $p_{i1}, \ldots, p_{ik_i}$. SPORT also allows the user to specify a weight for the recourse value [effectively adding a factor ρ to $Q(x,w)$ in (1)].

The Methods

Both ILSRDD and BVSRDD require that the problem be cast into a more tractable form before it is solved. Since the technology matrix is fixed, the substitution $\chi = Tx$ may be made, thereby introducing the variables χ, called 'tenders,' into (1). This transformation is useful because it generates a nonlinear program in which the number of variables occurring nonlinearly is m_2 rather than n_1, and usually $m_2 \ll n_1$. The problem then becomes

$$\text{minimize } cx + \Psi(\chi) \qquad (2)$$

subject to

$$\begin{aligned} Ax &= b \\ Tx - \chi &= 0 \\ x &\geq 0 \quad . \end{aligned}$$

where

$$\Psi(\chi) = E_w[\psi(\chi,w)] \quad ,$$

$$\psi(\chi,w) = \min_{y \geq 0}[qy \mid Wy = h(w) - \chi] \quad ,$$

and the vector q and the matrix W are reintroduced for notational convenience.

A further transformation involves introducing second stage activities into the first stage. It is shown in [2] that (2) is equivalent to

$$\text{minimize } cx + qy + \Psi(\chi) \qquad (3)$$

subject to

$$\begin{aligned} Ax &= b \\ Tx + Wy - \chi &= 0 \\ x \geq 0 \ &, \ y \geq 0 \end{aligned}$$

where Ψ and ψ are defined as in (2). This form has significant advantages from a computational standpoint.

Both ILSRDD and BVSRDD exploit the separability of Ψ, which is due to the presence of simple recourse and to the separability of the cost vector. Thus, $\Psi(\chi)$ may be written

$$\Psi(\chi) = \sum_{i=1,m_2} \Psi_i(\chi_i) \qquad (4)$$

Furthermore, since each component of $h(w)$ is discretely distributed and since the cost vector is two-piece linear, each $\Psi_i(\chi_i)$ is piecewise linear, viz.

$$\Psi_i(\chi_i) = \max_{l=0,\ldots,k_i} (s_{il}\chi_i + e_{il}) \tag{4.1}$$

where

$$s_{il} = (q_i^+ + q_i^-) \sum_{t=1,l} p_{it} - q_i^+ \quad , \quad 0 \le l \le k_i \quad ,$$

$$e_{il} = q_i^+\bar{h}_i - (q_i^+ + q_i^-) \sum_{t=1,l} h_{it} p_{it} \quad , \quad 0 \le l \le k_i \quad .$$

$\bar{h}_i$ is the expected value of the ith row of $h(w)$, and by convention $\sum_{t=1,0} = 0$.†

ILSRDD is based on Wolfe's Generalized Linear Programming method (GLP), which solves a sequence of problems obtained by inner (or grid) linearization of (3) over the convex hull of the set of tenders $\{\chi^1, \ldots, \chi^K\}$. (Actually, because the amount of memory in the computer is finite, SPORT uses a smaller set with a fixed number of tenders, $\{\chi^{K-n+1}, \ldots, \chi^K\}$.) The problems are of the form

$$\text{minimize } cx + q^+y^+ + q^-y^- + \sum_{k=1,K} \lambda_k \Psi(\chi^k) \tag{5}$$

subject to

$$Ax = b$$

$$Tx + Iy^+ - Iy^- - \sum_{k=1,K} \lambda_k \chi^k = 0 \tag{5.1}$$

$$\sum_{k=1,K} \lambda_k = 1 \tag{5.2}$$

$$x, y, \lambda_k \ge 0 \quad .$$

The tenders $\chi^1, \ldots, \chi^K$ are assumed to have been generated previously. χ^1 is set to the expected value of $h(w)$ prior to the first iteration. A new tender is obtained in each cycle by solving the Lagrangian subproblem

$$\underset{\chi}{\text{minimize}}\ \Psi(\chi) + \pi^K\chi \tag{6}$$

where π^K is formed by the dual multipliers associated with the constraints (5.1) in the optimal solution of (5).* The optimal solution of (6), χ^{K+1}, represents an improvement provided

$$\Psi(\chi^{K+1}) + \pi^K\chi^{K+1} < \vartheta^K \quad ,$$

where ϑ^K is formed by the dual multipliers associated with the constraints (5.2) in the optimal solution of (5). If no such χ can be found, the current solution is optimal. Generally speaking, problem (5) lends itself to solution because only a few tenders will have nonzero coefficients in the optimal solution and because a good set of initial tenders can be provided given the underlying recourse program.

The properties of $\Psi(\chi)$, particularly the convexity and piecewise linearity of $\Psi_i(\chi_i)$, permit the use of a simpler iterative technique. BVSRDD introduces new variables z_{il} for each interval over which $\Psi_i(\chi_i)$ is linear. It follows that

$$\Psi_i(\chi_i) = \Psi_i(\chi_i^0) + \min[\sum_{l=0,k_i} s_{il} z_{il} \mid \chi_i = \chi_i^0 + \sum_{l=0,k_i} z_{il}]$$

where χ_i^0 is the ith component of χ^0, the *base* tender, and z_{il} is bounded below

† Because Ψ_i is convex, the s_{il}s form an increasing sequence (in fact, $-q_i^+ \le s_{il} \le q_i^-$, $0 \le l \le k_i$). It also happens that the e_{il}s form a nonincreasing sequence.

* Since $\Psi(\chi)$ is separable, (6) is easily solved. See [2] for more details.

by zero and above by the length of the lth interval. With this substitution for $\Psi_i(\chi_i)$, (2) becomes

$$\text{minimize } cx + \sum_{i=1,m_2} \sum_{l=0,k_i} s_{il} z_{il} \tag{7}$$

subject to

$$Ax = b$$

$$T_i x - \sum_{l=0,k_i} z_{il} = \chi_i^0 \quad , \quad i=1,\ldots,m_2$$

$$x \geq 0$$

where T_i is the ith row of the matrix T. The z_{il}s are constrained to the length of the lth interval as before. This is a straightforward linear program.

Because of its dependence on the properties of Ψ, BVSRDD is fairly limited in its range of possible application. The algorithm has been implemented primarily to provide some assurance that ILSRDD is working properly.

The program itself is essentially a front end to a customized version of MINOS [4]. The program reads the data and establishes the appropriate form of the problem [(5) or (7)], then calls MINOS routines to solve the resulting linear program(s). Because there are two methods implemented, one of which requires the solution of several linear programs, some of the MINOS procedures have been modified.

Input Overview

SPORT requires three logically distinct sets of data. The first set is control information. The second set contains most of the nonstochastic data for the problem. The final set provides information about the tenders and the distribution of the random vector $h(w)$. It is anticipated that in normal practice the three sets of data will reside in three separate files.

Standard Input, Output, and Error Files

The program reads its control information from the standard input file (usually connected to unit 5) and writes its results and error messages to the standard output file (usually connected to unit 6). These files are assumed to be standard from system to system and consequently they are not opened by the program. The user may alter the standard input, standard output, and standard error unit numbers as described in the section entitled 'User-Accessible Parameters' below. The user is responsible for opening these files if necessary.

Control Information

The user must supply the program with various limits, file names and unit numbers, and other options. This control information resides in a 'specs' file. The user must connect this file to the standard input unit before the program is invoked.

The specs file contains a number of sections, some of which may be empty. Each section is identified by a keyword which begins in column 1. In general, entries within a section are identified by a keyword which begins in column 5 and the actual values begin in column 23. The keywords that identify each section must appear even if default values are selected, although in this case the section need not contain any entries. Unless otherwise specified, all character values are read using a 2A4 format and all numeric values are read using an I8 format. Keywords must be capitalized and only the first four characters of a keyword (including trailing blanks) are significant. The sections are listed below with their keywords in parentheses. The sections must appear in the order specified.

- **name of the method ('ISLRDD' or 'BVSRDD').** This section selects which algorithm shall be used.
- **start of control information ('BEGIN').** This identifies the beginning of the control file.
- **file names and unit numbers ('UNIT').** SPORT uses nine files, three of which are assumed to be standard on all systems. This section specifies the unit numbers and file names for the remaining files used by the program. There may be as many as six entries in this section and they may appear in any order. Each entry consists of two cards. The first card contains a keyword beginning in column 5 and an integer beginning in column 23. The second card contains a character string in columns 1 through 20. The keyword identifies one of the files used by the program. The integer specifies a unit number for the file and the string is the file name for the file. Unless otherwise specified, unit numbers and file names must be unique. Although zero is a legal unit number on some systems, the program will generate an error if the user attempts to assign this unit to the core file, to the stochastics file, or to either of the MINOS files. Valid entries and their keywords are
  - **core file ('CORE')** This entry specifies the unit number and file name for the 'core' file (see the section entitled 'Core File' below). If the unit number is the same as the standard input unit number the file name is read but is not used (i.e., the core data is assumed to follow the control information in the specs file). The unit number may not be the same as the standard output unit number or the standard error unit number.
  - **stochastics file ('STOCHASTICS')** This entry specifies the unit number and file name for the 'stochastics' file (see the section entitled 'Stochastics File' below). If the unit number is the same as the standard input unit number or the core file unit number, the file name is read but is not used (i.e., the stochastics data is assumed to follow the control information or the core data in the appropriate file). The unit number may not be the same as the standard output unit number or the standard error unit number.
  - **MINOS specifications file ('SPECS')** This entry specifies the unit number and file name for the MINOS control file (see the section entitled 'MINOS Files' below). The unit number may not be the same as that of any of the standard files.
  - **MINOS data file ('MPS')** This entry specifies the unit number and file name for the MINOS data file (see the section entitled 'MINOS Files' below). The unit number may not be the same as that of any of the standard files.
  - **debug file ('DEBUG') (optional)** This entry specifies the unit number and file name for the debug file, which contains the values of s_{il}, e_{il}, q, χ, and some variables internal to the program at various points of execution. If ths entry is omitted, no debug file is produced. A unit number of 0 also inhibits production of

the debug file. If the unit number is the same the standard output unit number or the standard error unit number, the file name is read but is not used (i.e., the debug information is written to the appropriate standard file). The unit number may not be the same as the standard input unit number.

  - **log file ('LOG') (optional)** This entry specifies the unit number and file name for the log file, which contains a trace of the program's execution and the contents of the stochastics file and part of the core file as read by the program. If this entry is omitted, no log file is produced. A unit number of 0 also inhibits production of the log file. If the unit number is the same the standard output unit number or the standard error unit number, the file name is read but is not used (i.e., the log information is written to the appropriate standard file). The unit number may not be the same as the standard input unit number.

- **maximum memory requirements ('DIMENSIONS')** This section provides the program with the information necessary to set up a number of arrays. The entries in this section may appear in any order and all are optional. Each entry contains a keyword beginning in column 5 and a value beginning in column 23 (an exception is the 'TENDERS' entry; see below). If an entry is omitted, the corresponding variable assumes the default value indicated. These defaults may be changed as described in the section entitled 'User-Accessible Parameters.' Valid entries and their keywords are
  - **maximum number of right-hand-side elements ('ELEMENTS')** This is the maximum number of nonzero elements in the augmented matrix, i.e., in $[A\,|\,\chi]$ after the transformations in (2) and (3) have been applied. This value must provide enough space for generated tenders as well as those that are entered in the input data. The default value is 5 times the maximum number of matrix columns (see below).
  - **maximum number of matrix rows ('ROWS')** This is the maximum number of rows that may appear in the combined matrix $\begin{bmatrix} A \\ T \end{bmatrix}$. The default value is 100.
  - **maximum number of technology rows ('TECHNOLOGY')** This is the maximum number of rows that may appear in the technology matrix, T. This value may not exceed the maximum number of matrix rows as specified in the 'ROWS' entry. The default value is 20.
  - **maximum number of matrix columns ('COLUMNS')** This is the maximum number of columns that may appear in the combined matrix $[A\,|\,\chi]$, where χ is the matrix formed by the set of tenders (each of which is a column vector with m_1 rows) used in problems (5). The default value is 3 times the maximum number of matrix rows (see above).
  - **maximum number of values for $\mathbf{h_i(w)}$ ('PROBABILTIES')** This is the maximum number of points in the support of any row of $h(w)$ [i.e., the maximum permissible k_i in (4.1)]. The default value is 30.

  - **tender information (χs) ('TENDERS')** This entry contains three subentries. Each subentry contains a keyword beginning in column 9 and a value beginning in column 23. The subentries may appear in any order and all are optional. Valid subentries and their keywords are
    - **maximum number of user supplied tenders ('INPUT')** This is the maximum number of tenders that may be entered in the stochastics file (see below). For BVSRDD, this value must be 1. For ILSRDD, this value may not exceed the maximum number of tenders saved as specified in the 'GENERATED' entry. The default value is 1.
    - **maximum number of tenders saved ('GENERATED')** This is the maximum number of tenders that will appear in the set used to generate each problem (5) in ILSRDD (i.e., n). The default value is 20.
    - **maximum number of tender elements ('ELEMENTS')** This is the maximum number of nonzero elements that may appear in the tenders in the set used to generate each problem (5) in ILSRDD or that may appear in the base tender in BVSRDD. The default value is 2000.
- **row and vector names ('SELECTORS')** This section identifies which row is the objective row and which vectors in the 'RHS,' 'BOUNDS,' and/or 'RANGES' sections of the core file are to be used. The entries in this section may appear in any order. Each entry contains a keyword beginning in column 5 and a value beginning in column 23. If an optional entry is omitted, the first applicable row or vector found in the core file is taken to be the desired entry. Valid entries and their keywords are
  - **name of the objective row ('OBJECTIVE')** The type 'N' row with this name in the 'ROWS' section of the core file is taken to be the objective row.
  - **name of the right-hand-side vector ('RHS')** The entries in the 'RHS' section of the core file that contain this name in the first name field (columns 5 through 12) form the vector b.
  - **name of the bounds vector ('BOUNDS') (optional)** The entries in the 'BOUNDS' section of the core file that contain this name in the first name field (columns 5 through 12) provide bounds on the decision variables.
  - **name of the ranges vector ('RANGES') (optional)** The entries in the 'RANGES' section of the core file that contain this name in the first name field (columns 5 through 12) provide ranges for the decision variables.
- **miscellaneous control information ('CONTROL')** This section contains a number of miscellaneous control parameters. The entries in this section may appear in any order and all are optional. Each entry contains a keyword beginning in column 5 and a value beginning in column 23. If an entry is omitted, the corresponding variable assumes the default value indicated. These defaults may be changed as described in the section entitled 'User-Accessible Parameters.' Valid entries and their keywords are

  - **print control ('OUTPUT')** This entry specifies how much information is written to the log file and to the standard output file. The default value is 2 and causes the program to place a certain amount of information in the output files. A value less than 2 causes less information to be written to the log file and a value greater than 2 causes additional information to be written to the output file. See the sections describing the output and log files below for more details.
  - **maximum number of cycles ('CYCLE')** This is the maximum number of linear programs (5) to solve when applying ILSRDD (i.e., the maximum number of tenders generated or the maximum permissible value for K). It is not used by BVSRDD. The default value is 1.
  - **scale factor ($\rho \times 100$) ('SCALE')** This is the factor by which the recourse value is multiplied. It is expressed as a percentage (i.e., a value of 100 results in a factor of 1.00). The default value is 100.
- **MINOS specifications ('MINOS')** This section contains any additional MINOS options desired. Any cards whose first four columns are blank will be echoed in the MINOS specifications file after the cards that specify the objective row, rhs vector, bounds vector, and/or ranges vector and before the card that gives the cycle limit.

  This section need not contain any entries.
- **END card ('END')** This card marks the end of the control information.

Core File

The core file contains the data for the decision variables. It specifies

- the name and type of each row in the problem,
- the objective, c,
- the nonzero elements of A and T,
- the deterministic right-hand-side, b, and
- the bounds on the decision variables.

The core file is specified in standard MPS format [5]. The 'ROWS' section contains an entry for the objective and for each row of A and of T. The rows of A and of T may be interleaved. The rows of T are normally equality rows, but the program performs the necessary conversions if this is not the case (e.g., if there is no penalty on surplus). The 'COLUMNS' section contains the elements of c and the nonzero elements of A and of T. The 'RHS' section contains the nonzero elements of b. Nonzero elements of b that correspond to rows of the technology matrix are ignored. The 'BOUNDS' and 'RANGES' sections may be used to impose limits on the solution as in normal practice.

Stochastics File

The stochastics file specifies the information pertaining to the recourse problem. It specifies

- the names of the rows that constitute the technology matrix,

- the distribution of each row of the stochastic rhs vector, $h(w)$,
- the penalties on shortage and surplus, q^+ and q^-, and
- the set of initial tenders for ILSRDD or the base tender for BVSRDD.

The stochastics file is specified in a subset of an extended MPS format developed for stochastic linear programs with recourse [6]. Like the core file, the stochastics file consists of a number of sections. Field conventions similar to those in standard MPS are employed. That is, section names begin in column 1; options on the same line as a section name begin in column 15. Data lines have six fields (some of which may be blank): a code field (columns 2 and 3), three name fields (columns 5 through 12, 15 through 22, and 40 through 47), and two numeric fields (columns 25 through 36 and 50 through 61). The contents of the code and name fields are interpreted as character strings. The sections are

NAME - This is an informative header card. The user may enter any characters desired in columns 15 through 72.

TECHNOLOGY - This section of the stochastics file specifies which of the rows listed in the 'ROWS' section of the core file constitute the technology matrix, T. The data consists of a list of names corresponding to a subset of the list of row names specified in the 'ROWS' section of the core file. The contents of these rows (as specified in the COLUMNS section of the core file) constitute the technology matrix. One name appears per line, in the first name field (columns 5 through 12).

DISTRIBUTIONS - This section of the stochastics file specifies the probability distribution of the r.h.s., $h(w)$. The data consists of sets of entries of the form "rowname value probability." There is one such set for each of the rows named in the TECHNOLOGY section. "Rowname" specifies the row associated with the entry; it occupies the first name field on a line (columns 5 through 12). "Value" and "probability" give a value for the row and its likelihood, respectively. They occupy the first and second numeric fields (columns 25 through 36 and 50 through 61), respectively.

The sum of the probabilities for a given row must be unity. Entries for different rows must not be mixed together.

RECOURSE - This section of the stochastics file is included to provide compatability with files specified strictly according to the format described in [6]. It contains no data.

OBJECTIVES - This section of the "stochastics" file specifies the recourse objectives, q. The data consists of entries of the form "name value value", where "name" gives the name of a row of T. the first value gives the corresponding value of q^+, and the second value gives the corresponding value of q^-. The name occupies the first name field on a line (columns 5 through 12) and the values occupy the first and second numeric fields (columns 25 through 36 and 50 through 61), respectively.

TENDERS - This section specifies the value(s) for the initial tender(s) used by the algorithms. The data consists of entries of the form "name rowname value," where "name" is a name provided for the tender, "rowname" specifies the row of T associated with the entry, and "value" is the value of the corresponding row in the tender vector. "name" is repeated for each row in the tender and there is one such name for each tender specified. "name" and "rowname" occupy the first two name fields (columns 5 through 12 and 15 through 22, respectively) and "value" occupies the first numeric field (columns 25 throgh 36). If a set of tenders is provided for ILSRDD, the first is used by BVSRDD (although all are read).

ENDATA - Indicates the end of the "stochastics" file.

MINOS Files

The program generates the MINOS specifications file and the MINOS data file from the specs file and from the core and stochastics files, respectively. The user may pass options to MINOS as described in the paragraph on MINOS specifications in the section entitled

Output File

This file contains the results generated by the program. If the print control variable is 2 or less, the program writes certain error messages and the following data to the output file:

- the contents of the specs file,
- the proposed tender, current objective value, and lower bound for the optimal objective value after each cycle,
- the standard MINOS output for the *last* linear program solved, and
- the first and second stage costs, the optimal tender, the dual multipliers (prices) associated with the technology rows in the optimal solution, and the probabilities of the equivalent change constrained programs.

If the print control variable is greater than 2, the output includes the standard MINOS output for *each* linear program solved.

Log File

This file contains a trace of the program's execution. If the print control variable is 2 or more, the program writes the following data to the output file:

- various messages concerning the program's activity, e.g., reading stochastics file, finished writing MINOS specs file,
- the contents of the stochastics file, and
- the 'NAME,' 'ROWS,' and 'RHS' sections of the core file.

If the print control variable is less than 2, the 'ROWS' and 'RHS' sections of the core file are not printed.

Data Structures

The matrices in problem (1) tend to be rather sparse, and the program represents them in a compact fashion to save space. To represent a large, sparse, two-dimensional array, the program uses three smaller one-dimensional arrays. The first array contains the nonzero elements of the matrix. These elements are ordered by column. Each element of the second array contains the row index within the sparse matrix of the corresponding element in the first array. The third array contains the indices within the first two arrays where the entries for each column of the sparse matrix begin. The ith entry in the third array is a pointer to the beginning of the ith column.

The program uses one (very large) array to hold the contents of most of the matrices and vectors used by the algorithm. A similar scheme is used by MINOS, and this array is passed to the MINOS routines.

User-Accessible Parameters

The default values for the standard unit numbers and for the variables in the specs file may be changed by changing the values of the appropriate variables when they are initialized at the beginning of the appropriate routines. Items with default values are listed in the table below together with the variable that contains the value of the item and the subroutine in which the default value is established.

Value	Variable	Subroutine
standard input unit number	in	sport
standard output unit number	iprint	sport
standard error output unit number	ierprt	sport
debug file unit number	idebug	readsp
log file unit number	iout	readsp
maximum number of rhs elements	maxele	readsp
maximum number of matrix rows	maxrow	readsp
maximum number of technology rows	maxtrw	readsp
maximum number of matrix columns	maxcol	readsp
maximum number of values for $h_i(w)$	maxpro	readsp
maximum number of user supplied tenders	maxten	readsp
maximum number of tenders in set	maxgtn	readsp
maximum number of nonzero tender elements	maxtel	readsp
objective row name	mobj	readsp
rhs vector name	mrhs	readsp
bounds vector name	mbou	readsp
ranges vector name	mran	readsp
print control	lvout	readsp
maximum number of cycles	ncycle	readsp
scale factor (ρ)	scale	readsp*

* The value specified in the program for the scale factor is *not* a percentage. That is, if ρ is one half, the variable 'scale' should be set to 0.5, not to 50.

The variable 'ctol,' set in subroutine sport, is used as a tolerance to determine convergence. If

$$\Psi(\chi^{K+1}) + \pi^K \chi^{K+1} - \vartheta^K \geq \text{ctol}$$

the optimal solution is taken to have been found. 'ctol,' which must be negative, is currently $-1 \cdot 10^{-7}$.

There are a number of machine dependent parameters. They are

Parameter	Variable	Subroutine	Current Value
positive infinity	plinfy	sport	10^{20}
double precision tolerance	eps	m1init	2^{-55}
number of integers per real*8 variable	nwordi	sport & m1init	2
number of reals "	nwordr	m1init	2
number of integer*2 variables "	nwordh	m1init	4

Subprocedure Hierarchy

See the attached figure. The MINOS routines shown have been modified.

Library Routines

The program uses version 4.9 of MINOS. Several routines have been modified to reflect the special requirements of the program.

References

[1] J.L. Nazareth and R. J.-B. Wets, 'Stochastic Programs with Recourse: Algorithms and Implementation,' IIASA Working Paper, forthcoming

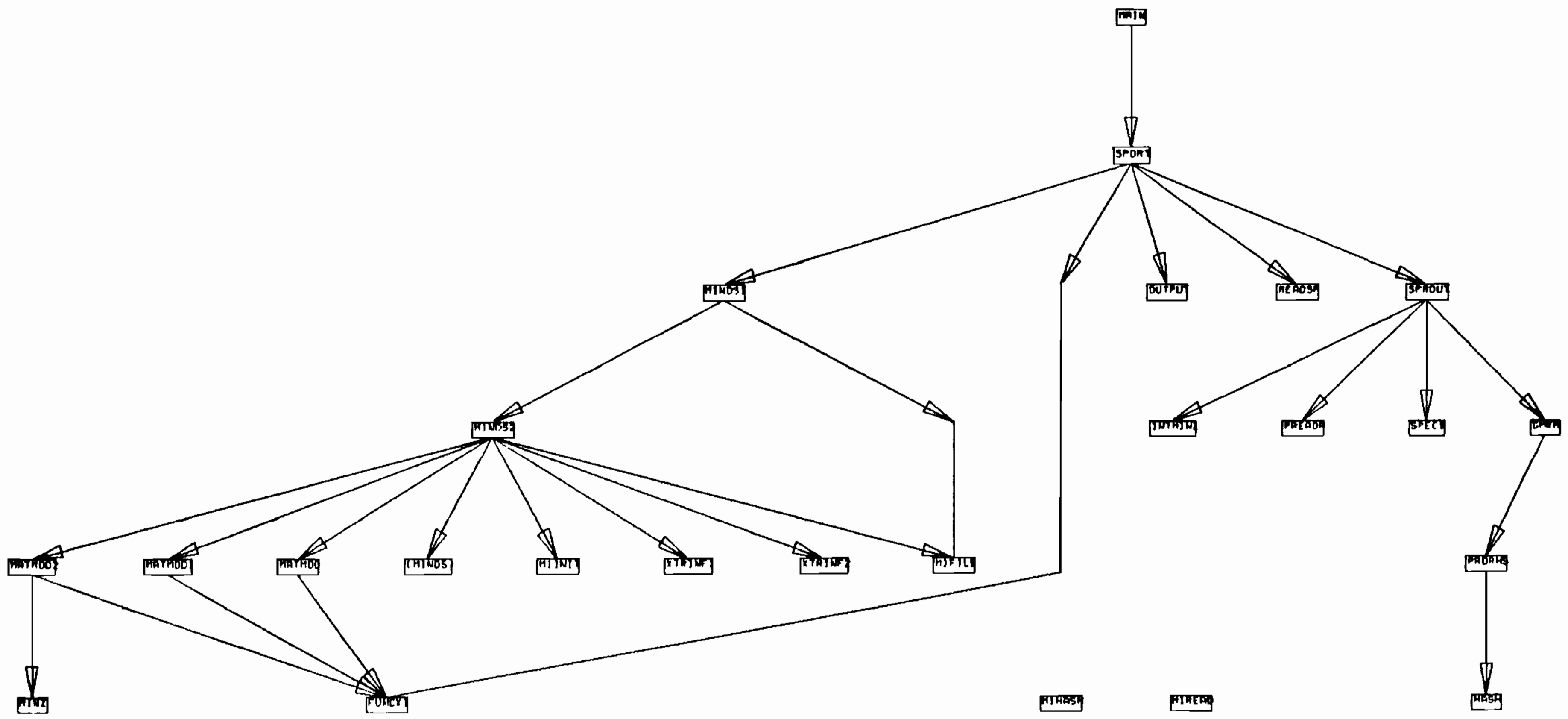
[2] J.L. Nazareth, 'Algorithms Based Upon Generalized Linear Programming for Stochastic Programs with Recourse,' IIASA Working Paper WP-84-81, 1984

[3] J.L. Nazereth and R. J.-B. Wets, 'Algorithms for Stochastic Programs: The Case of Nonstochastic Tenders,' IIASA Working Paper WP-83-5, 1983

[4] B. Murtagh and M. Saunders, 'Large Scale Linearly Constrained Optimization,' *Mathematical Programming* v. 14, pp. 41-72, 1978

[5] IBM Corp., **Mathematical Programming Subsystem - Extended (MPSX) and Generalized Upper Bounding (GUB) Program Description**, document number SH20-0968-1

[6] J. Edwards, J. Birge, A. King, and L. Nazareth, "A Standard Input Format for Computer Codes which Solve Stochastic Programs with Recourse and a Library of Utilities to Simplify Its Use," IIASA Working Paper, forthcoming


MAIN
SPORT
OUTPUT
READSA
SPROUT
PREADA
SPECT
MATMOD2
MATMOD1
MATMOD
CMINOS
MTINIT
XTRINF1
XTRINF2
MJFILL
FUNCT

FIM User's Manual

J. Edwards

Introduction

This program implements Qi's Forest Iteration Method (FIM) for stochastic transportation problems. The method is described in [1]. The program was written by Liqun Qi at the University of Wisconsin and later was modified and translated at IIASA. It is written in FORTRAN 77. The following description of the problem and the discussion of the algorithm are taken from [1].

The Problem

The standard formulation of the stochastic transportation problem with a dummy node is to find

$$\min_{x,\chi}\left\{\sum_{i=1,m}\left[\sum_{j=1,n} c_{ij}x_{ij} + \sum_{j=1,n}\varphi_j(\chi_j)\right]\right\}$$

subject to

$$\sum_{j=1,n+1} x_{ij} = a_i \quad , \quad i=1,\dots,m$$

$$\sum_{i=1,m} x_{ij} = \chi_j \quad , \quad j=1,\dots,n+1$$

$$x_{ij} \geq 0 \text{ for all } i,j$$

where a_i is the total inventory at node i ($a_i > 0$), c_{ij} is the cost of shipping one unit from node i to node j ($c_{ij} \geq 0$), x_{ij} is the amount of inventory shipped from node i to node j, χ_j is the amount of inventory received at node j, φ_j is the penalty due to surplus or deficit at node j, and node n+1 is the dummy node. φ_j has the following form

$$\varphi_j(\chi_j) = q_j^+ \int_{z_j<\chi_j} (\chi_j - z_j)dF_j(z_j) + q_j^- \int_{z_j>\chi_j} (z_j - \chi_j)dF_j(z_j)$$

where z_j is the observed value of $\tilde{z}_j$, the random variable for demand at j, F_j is the marginal distribution function of $\tilde{z}_j$ and is known, q_j^+ is the salvage cost per unit of excess inventory at node j ($q_j^+ \geq 0$), q_j^- is the penalty per unit of inventory shortage at node j ($q_j^- \geq 0$).

The Method

According to convex program theory, (x,χ) is an optimal solution if and only if there exist u in R^m and v in R^{n+1} such that

$$\sum_{j=1,n+1} x_{ij} = a_i \quad , \quad i=1,\dots,m$$

$$\sum_{i=1,m} x_{ij} = \chi_j \quad , \quad j=1,\ldots,n+1$$

$$x_{ij} \geq 0 \text{ for all i,j}$$

$$u_i + v_j \leq c_{ij} \text{ for all i,j}$$

$$x_{ij}(c_{ij} - u_i - v_j) = 0 \text{ for all i,j}$$

$$-v_j \in \partial\varphi_j(\chi_j) \quad , \quad j=1,\ldots,n \text{ and}$$

$$v_{n+1} = 0$$

where $c_{i,n+1} = 0$, i=1,...,m.

The algorithm starts with an estimate χ' of the optimal solution χ^* (e.g., χ_j could be set to the mean value of $\tilde{z}_j$). By fixing $\chi = \chi'$ in the original set of equations, the problem becomes an ordinary transportation problem of the same size. This is solved and x', the optimum value for x given $\chi = \chi'$, obtained. The graph of all positive elements of x' forms a forest, f'. The triple (x',χ',f') is called a forest triple.

The algorithm solves a reduced version of (1), wherein the components of x not on f' are restricted to zero and the nonnegative restriction on the rows of x is removed. Let the optimal solution to the reduced problem be $(\hat{x},\hat{\chi})$. If $\hat{x}$ is nonnegative (i.e., is a feasible solution of the original problem), the forest triple $(\hat{x},\hat{\chi},f')$ is called a base forest triple.

If $\hat{x}$ is not a feasible solution of the original problem, the algorithm uses a technique called "cutting" (fully discussed in [1]) to obtain a new forest triple $(x-,\chi-,f-)$ whose x component (i.e., $x-$) is a feasible solution. The cutting technique guarantees that the value of the objective function for $(x-,\chi-,f-)$ is strictly less than the value of the objective function for $(\hat{x},\hat{\chi},f')$. The algorithm "cuts" until a base forest triple is obtained.

Once a base forest triple has been found, the conditions involving u_i and v_j are used to determine whether the base forest triple is the optimal solution to the original problem. If $u_i + v_j > c_{ij}$ for some i and j, then the algorithm uses one of two techniques (called "pivoting" and "connecting" in [1]) to obtain a new forest triple. Again, these techniques guarantee that the value of the objective function for this new triple is strictly less than the value of the objective function for the base forest triple from which the new triple was derived. However, this new triple may not be a base forest triple, and the algorithm again "cuts" as described in the preceding paragraph.

Since there are only finitely many base forest triples and since the techniques employed by the algorithm produce strictly decreasing objective function values, it is clear that the optimal solution is reached in finitely many steps.

User Supplied Routines

The program reads the values for m, n, the a_is, the c_{ij}s, and an initial estimate for the χ_js and proceeds to find the optimum solution based on values for φ_j supplied by the user. The user must write two routines, **penalt** and **repenl**, toward this end. Penalt should be declared as

```
double precision function penalt(j,chij)
integer*2 j
double precision chij
```

Penalt calculates the value of $\varphi_j(\chi_j)$ given the values of j and χ_j. Repenl should be declared as

```
    double precision function repenl(j,A,toler)
    integer*2 j
    double precision A,toler
```

Repenl returns a value from the subdifferential of φ_j $(\partial\varphi_j)$ that is closest to the constant A given the values of j and A and a tolerance.

If φ_j is continuous and strictly convex, repenl's task reduces to finding the value of χ_j so that $\varphi'_j(\chi_j) = A$. If φ_j is discrete (as it often is) or continuous but not convex, repenl must generate a value based on $\partial\varphi_j$ and the tolerance.

If the tolerance is close to zero, repenl should return $\max\{\chi_j \mid A \in \partial\varphi_j(\chi_j)\}$. If the tolerance is not close to zero, but is less than the difference between the minimum and maximum values in the set, repenl should return the minimum value in the set instead of the maximum value and should set the tolerance to its old value plus the difference between the maximum and minimum values. If the tolerance is greater than the difference between the minimum and maximum values, repenl should return the maximum value plus the tolerance and should set the the tolerance to zero.

To aid in obtaining any data that might be required to perform these calculations, this program calls the subroutine **usedat** after it reads its own data. The user may place any code necessary to read or to initialize the appropriate variables in this subroutine. The user may also access vital parameters (e.g., dimensions, I/O unit numbers) via the common block "forest." The contents of this block and their meanings are described later in this manual.

Input

FIM takes its input from unit 5. It is the user's responsibility to connect this unit to the proper file before execution. The input to this program consists of the following data in the order specified:

number of rows (m)
number of columns (n)
total inventory at each node (a_is)
cost matrix (c_{ij}s)
initial estimate for the inventory received at each node (χ_js)
data for user routines

The cost matrix is entered by row and each row must start a new line. To simplify the process, all negative values are converted to a large number. Thus, to indicate an unavailable element, -1 may be used. The program reads all its data in free format (i.e., '*').

Output

FIM writes its output to unit 6. The output consists of a listing of the input data, a trace of the program's execution, and the optimal values for the u_is, the v_js, and the χ_js, the optimal direct, penalty, and total costs, and the values of the nonzero flows.

After each normalization, the program writes the number of normalizations that have been performed, the number of cuttings required by the latest normalization, and the current values of the direct, penalty, and total costs. After a forest has been normalized and altered to satisfy the necessary conditions, the program writes one of two messages depending on which technique ('pivoting' or Once the optimal forest is found, an appropriate message is written.

Data Structures

The program uses four data structures (each of which requires several separate arrays) to represent a transportation tableau and a forest of trees within it. The tableau itself consists of four (m+1) by (n+2) arrays of cells (the actual tableau only requires m by (n+1) arrays; the additional elements are used for various bookkeeping chores). "celflo(i,j)" contains the value of x_{ij}. "celnxr(i,j)" and "celnxc(i,j)" contain information about the structure of the tree of which the cell is a part and "celopf(i,j)" contains a value used to determine whether the tree of which the cell is a part is a base tree.

The algorithm requires occupied cells (i.e., those whose value of x_{ij} is positive) that are in the same row or in the same column to be members of the same tree, although the order in which nodes are visited as the tree is traversed does not necessarily follow column or row index. For each occupied cell i,j, "celnxr(i,j)" contains the row index of a cell in column j and "celnxc(i,j)" contains the column index of a cell in row i; these cells are cell i,j's children. The leaves of a tree (which have no children) have -1 for these indices.

Since all occupied cells in a given column or row must belong to the same tree, it is convenient to group a number of columns (and rows) together to form a tree. There are two data structures (one for rows and one for columns) to accomplish this. Each structure requires four arrays. For each column (row) i, "clsdex(i)" ("rwsdex(i)") specifies the row (column) index of the cell in column (row) i to be visited first as the tree is traversed, "clsnxt(i)" ("rwsnxt(i)") contains the index of the next column (row) in the tree, and "clsdad(i)" and "clslmp(i)" ("rwsdad(i)" and "rwslmp(i)") contain two values used by the algorithm for general bookkeeping.

Finally, there is a data structure (consisting of three arrays) which organizes the trees into a forest. For each tree i, "trsdex(i)" specifies the column index of the root node (the row index may be obtained from the column data structure described in the previous paragraph), "trsslv(i)" contains a flag indicating whether the tree is a base tree, and "trsnxt(i)" contains the index of the next tree in the forest.

Common Blocks and User-Accessible Parameters

The data structures described in the previous section and several other variables appear in the named common block "incl.forest." Most of these global variables are adequately commented in the source code and are of little concern to the user. Of potential interest, however, are the three constants MAXROW, MAXCLM, and MAXVAL. The first two give the maximum number of rows and the maximum number of columns, respectively, that may appear in the transportation tableau and the cost matrix. If the preset limit of 51 rows and 51 columns is insufficient, these constants must be changed and the program recompiled. MAXVAL is the value placed in the cost matrix when negative numbers are read in the input file. The preset value of $5 \cdot 10^{30}$ may be too large for some machines.

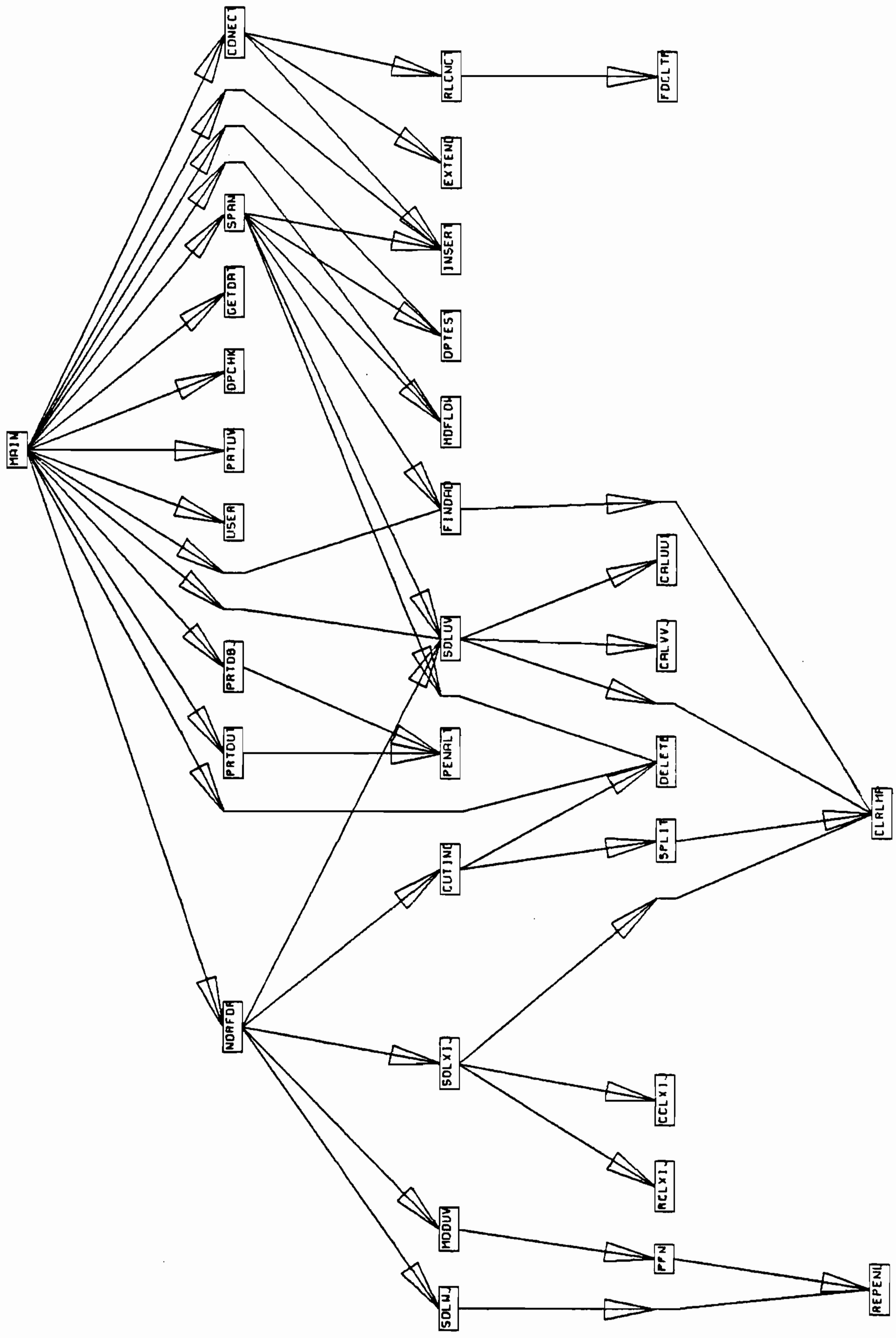
Among the variables that the user is likely to need are numrow and numclm, which give the actual number of rows and columns, respectively, in the transportation tableau.

Subprocedure Hierarchy

See the attached figure.

References

[1] L. Qi, "Forest Iteration Method for Stochastic Transportation Problems," to appear in *Math. Prog. Study*


MAIN
CONECT
SPAN
GETDAT
OPCHK
PRTUV
USER
PRTDB
PRTOUT
NORFOR
RLCNCT
EXTEND
INSERT
OPTEST
MDFLOW
FINDAD
SOLUV
PENALT
CUTINC
SOLXI
MODUV
SOLWJ
FDCLTR
CALUUT
CALVV
DELETE
SPLIT
CCLXI
RCLXI
PFN
CLRLMR
REPENL

STOSUB User's Manual

J. Edwards

Introduction

This package contains routines to solve stochastic programs using an algorithm that employs "stochastic subgradients" at each step. The method is described in [1]. The program was developed by Andrzej Ruszczynski at the Institut fur Operations Research of the Universitat Zurich in Switzerland and is completely described in [2]. The program is written in FORTRAN 77. The contents of this manual are largely adapted from [2].

The Problem

The stochastic linear program under consideration is

minimize

$$[F(x) = \mathrm{E}\{f_1(x, \vartheta_1(\omega))\} + \mathrm{E}\{f_2(x, \vartheta_2(\omega))\}] \tag{1}$$

subject to

$$Ax \begin{bmatrix} \le \\ = \end{bmatrix} b$$

$$x_j \ge 0,\ j \in \mathrm{J},$$

where x is an n_x vector of decision variables, ϑ_1 and ϑ_2 are vectors of random problem parameters, f_1 and f_2 are real-valued functions, "E" denotes mathematical expectation, A is an $m_x \times n_x$ matrix, b is an m_x vector, and J is a subset of $\{1,2,\ldots,n_x\}$.

The function f_1 is assumed to be explicitly defined, e.g., $f_1(x, \vartheta_1) = cx$, and it must be possible at each (x, ϑ_1) to calculate the gradient or subgradient of f_1 with respect to x.

The function f_2 is assumed to be the optimal value of the linear programming problem (the "recourse" or "second-stage" problem)

minimize

$$q^{\mathrm{T}}(\omega)y \tag{2}$$

subject to

$$\mathrm{T}(\omega)x + \mathrm{W}(\omega)y \begin{bmatrix} \le \\ = \end{bmatrix} h(\omega)$$

$$y \ge 0$$

dependent on x and the random parameters $\vartheta_2(\omega) = [q(\omega), \mathrm{T}(\omega), \mathrm{W}(\omega), h(\omega)]$. The user must provide realizations of these parameters to the package.

The Method

The method used to solve the above problem is a recursive stochastic algorithm that employs so-called stochastic subgradients, ξ^k, of the objective function F in (1) at current points x^k, $k = 0,1,\ldots$. Stochastic subgradients are random vectors and have the property that

$$E\{\xi^k \mid x^k\} = \nabla F(x^k)$$

if F is differentiable or

$$E\{\xi^k \mid x^k\} \in \partial F(x^k)$$

if F is a nondifferentiable convex function (here, ∂F denotes the subdifferential of F). These random vectors are used to calculate directions, d^k, $k = 0,1,\ldots$, by the formula

$$d^0 = \xi^0,$$
$$d^k = \frac{\xi^k + \gamma_k d^{k-1}}{1+\gamma_k}, \quad k = 1,2,\ldots \ .$$

Successive steps are made according to

$$x^{k+1} = \mathrm{Proj}_X(x^k - \tau_k d^k) \quad ,$$

where Proj_X denotes orthogonal projection onto the set defined by the constraints in problem (1). The coefficients γ_k and step sizes τ_k are controlled automatically by the algorithm.

The package uses the IMSL double-precision subroutine zx3lp to solve linear programs.

Stochastic Subgradients and User Supplied Routines

The stochastic subgradients ξ^k of the function F are composed of two parts, ξ_1^k and ξ_2^k, corresponding to the two parts of F. The user must prepare a subroutine to calculate the stochastic subgradients of $E\{f_1(x,\vartheta_1)\}$; the stochastic subgradients of $E\{f_2(x,\vartheta_2)\}$ are calculated automatically by the package. Some techniques for calculating ξ_1^k are

- if f_1 is smooth and does not depend on ϑ_1, set $\xi_1^k = \nabla f_1(x^k)$;
- if f_1 is smooth in x for all ϑ_1, draw at random ϑ_1^k and set $\xi_1^k = \nabla f_1(x^k,\vartheta_1^k)$ [finite-difference estiation of the gradient of $f_1(\cdot,\vartheta_1^k)$ is also acceptable];
- if f_1 is nonsmooth and convex in x for all ϑ_1, sample ϑ_1^k and choose $\xi_1^k \in \partial_x f_1(x^k,\vartheta_1^k)$.

The user's routine may have any desired name; the calling sequence is

call name(nx,my,myi,ny,t,w,h,c,q)

The values of nx, my, and ny are passed to the subroutine, which must return valid data in myi, t, w, h, c, and q. The parameters, their types, and their dimensions (where applicable) are listed below.

nx (integer) is the dimension of the first stage vector, x (i.e., nx = n_x).

my (integer) is the number of constraints in the second stage problem, (2).

myi (integer) upon return contains the number of inequality constraints in the second stage problem. The first myi constraints in (2) are inequalities and the remaining my-myi constraints are equations.

ny (integer) is the number of second stage variables.

t(my,nx) (real*8) upon return contains the second stage constraint coefficients that correspond to x in (2) (i.e., the contents of the matrix T).

w(my,ny) (real*8) upon return contains the second stage constraint coefficients that correspond to y in (2) (i.e., the contents of the matrix W).

h(my) (real*8) upon return contains the right sides of the second stage constraints.

c(nx) (real*8) upon return contains the (stochastic) gradient of the first stage cost.

q(ny) (real*8) upon return contains the cost coefficients for the second stage variables in (2).

All output parameters may be random, in which case their values should be generated in the subroutine by pseudo-random generators with the appropriate distributions.

If there is no second stage in the problem, set ny and nx to zero and use real variables for t, w, h, and q. The subroutine should return *without* assigning any values to them.

Invoking the Package

The user must write a driver for the package. The driver may do no more than define a work area and call the package with some preset data, or it may read a data file and print intermediate solutions as well. The calling sequence for the package is

```
   call stoslp(mx,mxi,nx,nxi,my,ny,x,a,b,
  *        name,nex,rk,eps,lsm,is,ip)
```

All values are passed to the subroutine, which returns data in x and ip. The parameters, their types, and their dimensions (where applicable) are listed below.

mx (integer) is the number of constraints in the first stage problem (i.e., the number of rows in A in (1)).

mxi (integer) is the number of inequalities in the constraints in (1). The first mxi constraints are inequalities and the remaining mx-mxi constraints are equations.

nx (integer) is the dimension of the first stage decision vector, x.

nxi (integer) is the number of components of x that are restricted in sign. The first nxi components are bounded below by zero and the remaining nx-nxi components may have any sign.

my (integer) is the number of constraints in the second stage problem, (2).

ny (integer) is the number of second stage variables.

x(nx) (real*8) contains the starting point, which need not be feasible. Upon return, it contains the coordinates of the (intermediate) solution.

a(mx,nx) (real*8) contains the coefficients of the first stage constraints.

b(mx) (real*8) contains the right hand sides of the first stage constraints.

name (subroutine) is the name of the user supplied routine to generate stochastic subgradients of the first stage cost and data for the second stage problem.

nex (integer) is the number of observations of the stochastic subgradients to generate at each iteration.

rk (real*8) is the initial change in x per iteration (Euclidean norm).

eps (real*8) is a stopping criterion. The package halts if two successive values of x differ by this amount or less.

lsm (integer) is the maximum number of iterations. When combined with the "ip" parameter (see below), lsm can be used to periodically halt the execution of the package so that intermediate results may be displayed.

is (integer) is the dimension of the work vector w in the common block optc (see below). This variable must be at least 6×nx + mx×nx + 5×mx + mxi + 3×nxi + (my + 1)×(nx + 6) + (my + 3)×(ny + 2) + max(my,ny) + $(my + 2)^2$ + (mx − mxi)×nx.

ip (integer) is a control parameter and a return code. If ip is one when passed to stoslp, the package performs a number of initialization steps and begins to solve the problem. If ip is greater than one when passed to stoslp, the package bypasses the initialization, i.e., continues where it left off. Thus, the caller could print all intermediate solutions by setting the maximum number of iterations (see lsm above) to one and calling stoslp without changing lsm.

ip is also used as a return code. Table 1 shows possible return values and their meanings.

Value	Meaning
2	"Convergence" achieved (successive x values are within eps of each other)
3	Maximum number of iterations (lsm) reached
4	Insufficient space in the work vector, w
5	Inconsistent constraints in problem (1)
6	The second stage problem has an unbounded solution
7	Maximum number of iterations exceeded in the IMSL routine zx3lp
8	The feasible set for the second stage problem is empty

Common Blocks

The program requires the user to declare a workspace in the named common block optc. This workspace should be a double precision array whose dimension is greater than or equal to the parameter "is" passed to the package (see above).

Library Routines

The program uses a routine from the IMSL double precision library to solve linear programs.

Subprocedure Hierarchy

See the attached figure.

Notes

STOSUB only takes advantage of the general statistical properties of (1) and is therefore applicable to a broad class of problems with a nonlinear first stage cost and arbitrary distributions of ϑ_1 and ϑ_2. This generality makes it inefficient for linear problems with easy-to-handle distributions and implies that even for simple problems the package must perform a large number of iterations (usually more than 100) to obtain a sufficiently good approximation of the solution. On the other hand, the accuracy after 300 to 500 iterations is within about 5 percent of optimal and can hardly be improved.

Although STOSUB determines stepsizes automatically, its efficiency does depend on the value of the input parameter rk, which is used to calculate the initial stepsize. It is advisable to choose rk so that $0.1 \times r \leq rk \leq r$, were r is the estimate of the distance from the starting point x^0 to the solution.

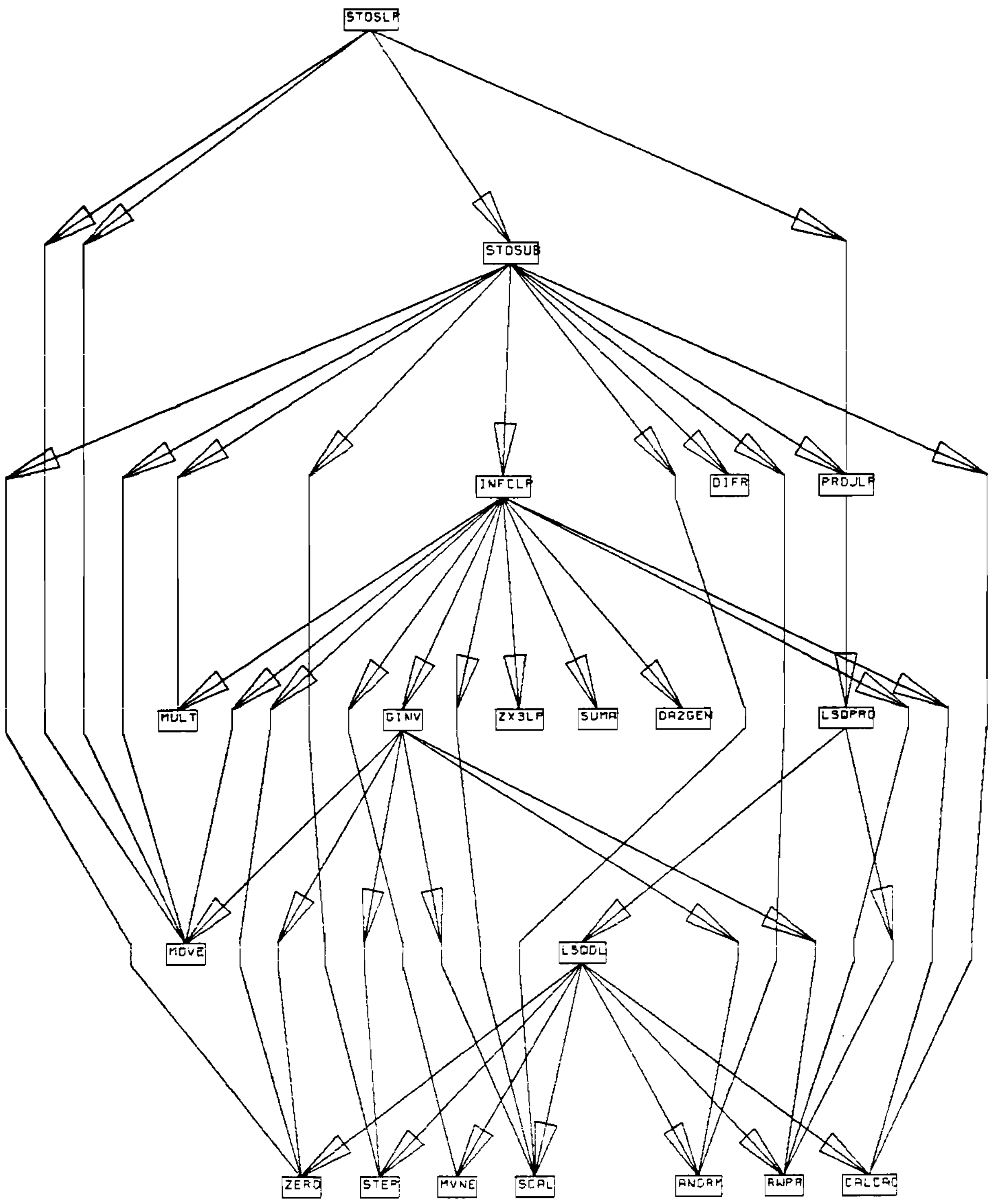
Because of the stochastic nature of the method, extremely high accuracy is not possible. A choice of 0.01xrk for the parameter eps is usually sufficient. The current average of the increments made to the decision variables (which is compared to eps to determine when to stop) can be found in the variable shift in the named common block opts.

Finally, it is advisable to run the method with many different starting points and to compare the solutions obtained in order to gain insight into the real accuracy provided.

References

[1] A. Ruszczynski and W. Syski, "A Method of Aggregate Stochastic Subgradients with On-Line Stepsize Rules for Convex Stochastic Programming Problems," report R.I. 21/83, Instytut Automatyki, Politechnika Warszawska, 1983 (to appear in *Mathematical Programming Study*).

[2] A. Ruszczynski, "STOSUB: A Package for Solving Stochastic Programming Problems - User's Manual," manuscript from the Institut fur Operations Research der Universitat Zurich, October 1984


STOSLP
STOSUB
INFCLP
DIFR
PROJLP
MULT
GINV
ZX3LP
SUMA
DAZGEN
LSQPRO
MOVE
LSQDL
ZERO
STEP
MVNE
SCAL
ANORM
RWPA
CALCAC

PCSP User's Manual

T. Szantai and J. Edwards

Introduction

This program solves problems in probabilistic constrained stochastic programming. It is based on Veinott's supporting hyperplane algorithm. The method is described in [1]. The program was developed by Tamas Szantai at the Technical University in Budapest and later was modified at IIASA. It is written in FORTRAN 77.

The Problem

The program solves stochastic programs with probabilistic constraints of the form

minimize

$$c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{1}$$

subject to

$$Ax = b$$
$$x \geq 0,$$

and

$$P(Dx \geq \beta) \geq p$$

where A is a known m×n matrix, D is a known s×n matrix, b and p are known and of the appropriate dimensions, and $\beta_1, \ldots, \beta_s$ have joint normal probability distribution with expected values

$$E(\beta_1) = \mu_1, \ldots, E(\beta_s) = \mu_s \quad ,$$

with variances

$$D^2(\beta_1) = \delta_1^2, \ldots, D^2(\beta_s) = \delta_s^2 \quad ,$$

and with correlation matrix

$$R = \begin{bmatrix} 1 & r_{12} & \cdots & r_{1s} \\ r_{21} & 1 & \cdots & r_{2s} \\ \cdots & \cdots & \cdots & \cdots \\ r_{s1} & r_{s2} & \cdots & 1 \end{bmatrix}$$

The linear constraints may include inequalities as well and it is also possible to specify explicit upper bounds on the variables.

The Method

A complete description of the supporting hyperplane algorithm is given in [1]. To obtain a starting point in the interior of the feasible domain, the program solves the linear program

maximize

$$\sum_{i=1,n} (d_{i1}x_1 + \cdots + d_{in}x_n - \mu_i) \,/\, \delta_i$$

subject to

$$Ax = b$$

$$Dx \geq \mu + t\delta$$

$$x \geq 0$$

where d_{ij} is the element of D in the ith row and jth column and t is a constant specified by the user. t's value should be chosen based on the desired probability level, p; 3.0 is recommended for high probabilities.

To obtain a starting point outside the feasible domain, the program solves the linear program

minimize

$$c_1x_1 + \cdots + c_nx_n$$

subject to

$$Ax = b$$

and

$$x \geq 0$$

In the case of an unbounded objective, one must provide additional constraints on the variables which do not disturb the probabilistic constraint.

To find the boundary point of the probabilistic constraint at each iteration the program uses an interval bisection algorithm with a sophisticated stopping rule. The values of the joint normal probability distribution function are calculated by a Monte Carlo simulation technique. This technique is also used to determine the gradient vector of the distribution function.

To solve the linear programs the program uses code from Land and Powell ([2]).

Input Overview

PCSP requires two data files and a control file. It takes its control data from unit 5; it is the user's responsibility to connect this unit to the appropriate file before the program is invoked. PCSP writes prompts for the control information to unit 6 and so the user may provide that data interactively.

Control Information

PCSP's control information consists of the names of four files, all of which are described in subsequent sections. File names may contain up to 80 characters. They are read in the following order:

- linear data file name
- stochastic data file name
- output file name
- log file name.

Linear Data File

This file is connected to unit 7. It contains some control parameters as well as data specifying the linear portion of the problem. It contains the following information:

- control parameters. Six integers appear on the first line of the linear data file. They are read using an I10 format. The numbers are

  the number of constraints, m.

  the number of variables, n.

  the number of bounded variables. If this number is -1, all the variables are assigned an upper bound of 1.

  print control parameter. This parameter controls whether the program prints the input cards and the contents of the matrix after each inversion. Possible values and the data printed in each case are shown in the table below.

Value	Input Cards Printed?	Inverse Printed?
0	yes	no
1	yes	yes
2	no	yes
3	no	no

  the maximum number of iterations. If this number is 0, the maximum number of iterations is set to $3\times(m+n+$ the number of bounded variables).

  the maximum number of re-inversions.

- the nonzero columns of the objective function, c. As many lines as are necessary to specify the contents of the nonzero columns of the objective function appear. Each line contains up to eight entries of the form "index value," where "index" is the column index of a nonzero column of c and "value" is the corresponding value. Each entry is read using a (I3,X,F6.0) format. The first entry on a line whose "index" is 0 (i.e., a blank entry) signals the end of the line.
- a line marking the end of the section specifying the nonzero elements of the objective function. This line contains the number 9999999999 (ten 9s) in the first ten columns.
- an optional section specifying the upper bounds on any bounded variable. If there are no bounded variables, this section does not appear. If there are bounded variables, this sections contains as many lines as are necessary to specify the upper bounds on the variables. Each line contains up to eight entries of the form "index value," where "index" is the row index of a bounded row of x and "value" is the corresponding upper bound. Each entry is read using a (I3,X,F6.0) format. The first entry on a line whose

"index" is 0 (i.e., a blank entry) signals the end of the line.

- a line marking the end of the section specifying the upper bounds on the variables. This line contains the number 9999999999 (ten 9s) in the first ten columns. This line does not appear if there are no bounded variables.
- the elements of the constraint vector, b, and the type of inequality for each row. As many lines as are necessary to specify the contents of the constraint vector appear. (Any rows that are not specified are assumed to be equality rows and the value of such rows of b is taken to be a large number.) Each line contains up to eight entries of the form "index type value," where "index" is the column index of a nonzero column of c and "value" is the corresponding value. "type" indicates the type of inequality; it is 0 for "=" rows, 1 for "≤" rows, and 2 for "≥" rows. Each entry is read using a (I3,I1,F6.0) format. The first entry on a line whose "index" is 0 (i.e., a blank entry) signals the end of the line.
- a line marking the end of the section specifying the contents of the constraint vector. This line contains the number 9999999999 (ten 9s) in the first ten columns.
- the nonzero elements of the constraint matrix, A. As many lines as are necessary to specify the contents of the constraint matrix appear. Each line contains the index of a row of A in columns 5 through 10, followed by up to seven entries of the form "index value," where "index" is the column index of a nonzero element of A in the current row and "value" is the corresponding value. The first entry begins in column 11 and each entry is read using a (I3,X,F6.0) format. The first entry on a line whose "index" is 0 (i.e., a blank entry) signals the end of the line.

  All the elements on a line must belong to the same row but need not be in correct column order. However, the rows must appear in strictly ascending order.
- a line marking the end of the section specifying the contents of the constraint matrix. This line contains the number 9999999999 (ten 9s) in the first ten columns.
- a line indicating whether the data for additional problems appears in the linear data file. If the value in the first ten columns of this line is nonzero, all the sections previously described are repeated and define another problem. The last card of the second problem may specify that there is a third problem, and so on. PCSP repeats its calculations for each problem that is defined in this file and in the stochastic data file.

Stochastic Data File

This file is connected to unit 8. It contains information describing the distribution of the stochastic elements in the problem. It contains the following information:

- control parameters. Five integers appear on the first line of the linear data file. They are read using an I10 format. The numbers are

  the number of random variables, s.

  the maximum number of generated supporting hyperplanes.

  the probability level, p.

  a tolerance on the probability level, p_ε. The actual effective probability level is $p \pm p_\varepsilon$.

the constant, t, used to find the initial inner point.

- the expected values, μ, of the random variables. As many lines as are necessary to specify the expected value of each random variable appear. Each line contains up to eight entries of the form "index value," where "index" is the row index of a row of β and "value" is the corresponding expected value. Each entry is read using a (I3,X,F6.0) format. The first entry on a line whose "index" is 0 (i.e., a blank entry) signals the end of the line.
- a line marking the end of the section specifying the expected values of the random variables. This line contains the number 9999999999 (ten 9s) in the first ten columns.
- the dispersions, δ, of the random variables. As many lines as are necessary to specify the dispersion of each random variable appear. Each line contains up to eight entries of the form "index value," where "index" is the row index of a row of β and "value" is the corresponding dispersion. Each entry is read using a (I3,X,F6.0) format. The first entry on a line whose "index" is 0 (i.e., a blank entry) signals the end of the line.
- a line marking the end of the section specifying the dispersions of the random variables. This line contains the number 9999999999 (ten 9s) in the first ten columns.
- the elements of the correlation matrix, R. As many lines as are necessary to specify the contents of the correlation matrix appear. Each line contains the index of a row of R in columns 5 through 10, followed by up to seven entries of the form "index value," where "index" is the column index of an element of R in the current row and "value" is the corresponding value. The first entry begins in column 11 and each entry is read using a (I3,X,F6.0) format. The first entry on a line whose "index" is 0 (i.e., a blank entry) signals the end of the line.

  All the elements on a line must belong to the same row but need not be in correct column order. However, the rows must appear in strictly ascending order.
- a line marking the end of the section specifying the contents of the correlation matrix. This line contains the number 9999999999 (ten 9s) in the first ten columns.
- the elements of the probabilistic constraint matrix, D. As many lines as are necessary to specify the contents of the probabilistic constraint matrix appear. Each line contains the index of a row of D in columns 5 through 10, followed by up to seven entries of the form "index value," where "index" is the column index of an element of D in the current row and "value" is the corresponding value. The first entry begins in column 11 and each entry is read using a (I3,X,F6.0) format. The first entry on a line whose "index" is 0 (i.e., a blank entry) signals the end of the line.

  All the elements on a line must belong to the same row but need not be in correct column order. However, the rows must appear in strictly ascending order.
- a line marking the end of the section specifying the contents of the probabilistic constraint matrix. This line contains the number 9999999999 (ten 9s) in the first ten columns.

The sections previously described are repeated as many times as is necessary to define the number of problems that appear in the linear data file. PCSP repeats its calculations for each problem that is defined in this file and in the linear data file.

Output File

PCSP echoes its input and writes its solution to this file, which is connected to unit 10.

Log File

PCSP writes the actual inner point, limit point, and cutting plane coefficients at each iteration as well as the corresponding objective function values and probability levels to this file, which is connected to unit 9.

Limits and Extensions

Several limits are currently imposed on the problem (e.g., no more than 25 rows in the x vector, 300 nonzero elements in A). To extend these limits, the values of the appropriate constants in the include file "constants.h" must be increased. The constants are

- **MAXVAR**, the maximum number of variables (n)
- **MAXCTR**, the maximum number of deterministic constraints (m)
- **MAXELM**, a bound on the number of elements in the deterministic constraint matrix, A. MAXELM should be twice the maximum number of nonzero elements in A.
- **MAXPLM**, the maximum number of nonzero elements in the probabilistic constraint matrix, D
- **MAXSTO**, the maximum number of probabilistic constraints (s)

Library Routines

PCSP uses the GGNSM module from the International Mathematical and Statistical Libraries, Inc., IMSL library.

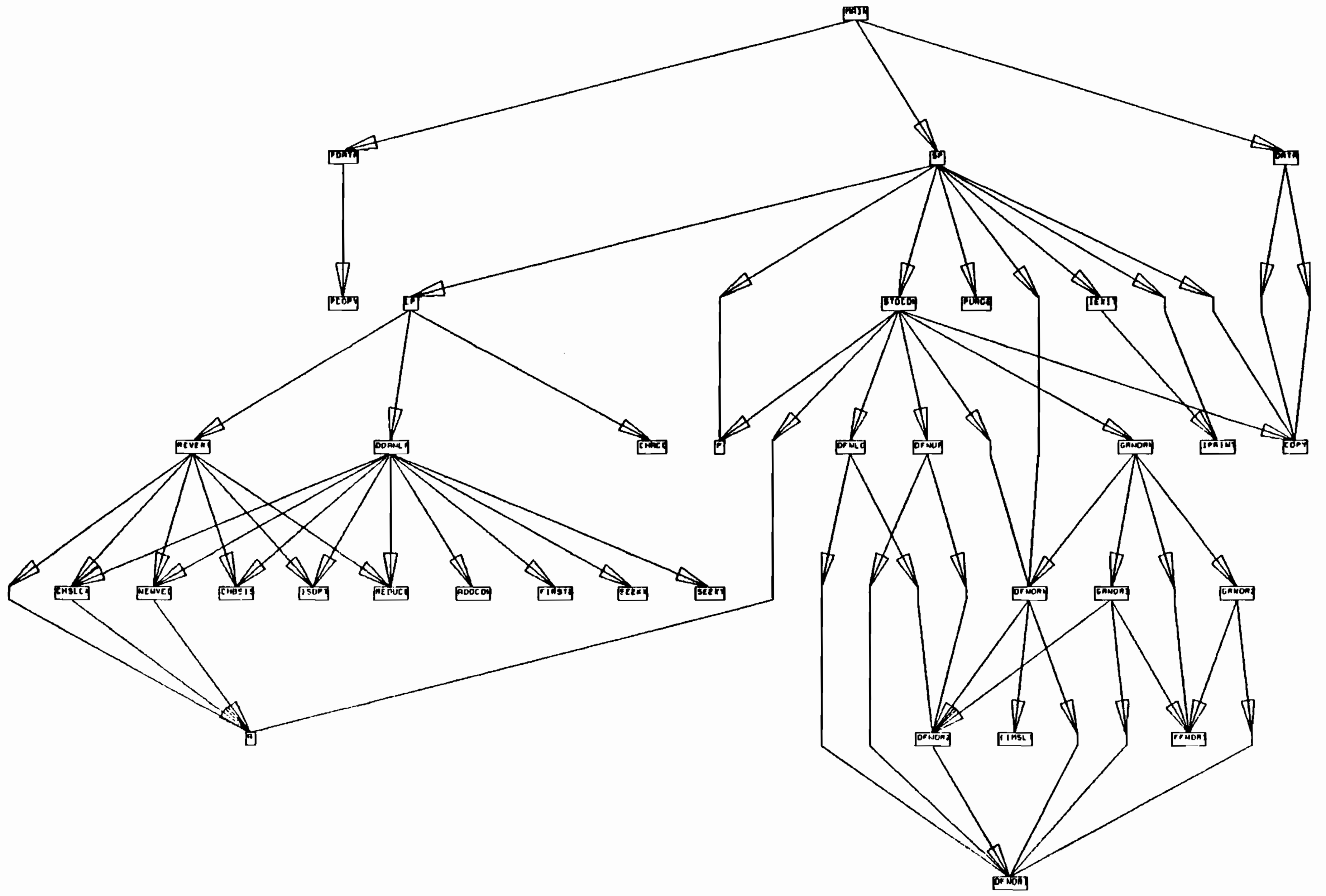
Subprocedure Hierarchy

See the attached figure.

References

[1] A. F. Veinott, "Supporting Hyperplane Method," *Operations Research* v. 15, pp. 147-152, 1967

[2] A. Land and S. Powell, **FORTRAN Codes for Mathematical Programming**, John Wiley, 1973



MAIN
PDATA
SP
DATA
PCOPY
LP
STOLDN
PURGE
IEXIT
REVERS
ODRNLP
CHNEG
P
DFNLC
DFNUP
GRNOR1
IPRINT
COPY
CHSLCT
NEWVEC
CHBSIS
ISUPT
REDUCE
ADDCON
FIRSTE
SEEKX
SEEKY
DFNOR1
GRNOR3
GRNOR2
R
DFNOR2
FIMSL
FFNOR1
DFNOR3

STOCHNET User's Manual

J. Edwards

Introduction

This program solves stochastic programs whose recourse is a pure network. It is based on the L-shaped method of Van Slyke and Wets for two-stage stochastic linear programs. The method is described in [1]. The algorithm particular to this implementation (which uses Schur complements for pure networks to update the basis without a matrix inversion) was invented by Stein W. Wallace and is described in [2]. The program was developed by Stein W. Wallace at Chr. Michelsen Institute in Bergen, Norway. It is written in FORTRAN 77. The following description of the problem and the discussion of the algorithm are adapted from [2].

The Problem

The multi-stage stochastic linear program under consideration is

minimize

$$cx + \mathcal{Q}(x) \tag{1}$$

subject to

$$Ax = b$$
$$x \geq 0,$$

where

$$\mathcal{Q}(x) = \int Q(x,\eta)dF(\eta),$$
$$Q(x,\eta) = \min\{qy \mid Wy = \eta - Tx,\ y \geq 0\},$$

A is a known $m_1 \times n_1$ matrix, W is a known $m_2 \times n_2$ matrix, b, c, and T are known and of the appropriate dimensions, η is a stochastic variable and F is its distribution function.

Since the recourse is a pure network, W may be written as

$$\begin{bmatrix} E & 0 \\ I & I \end{bmatrix}$$

where E is the node-arc incidence matrix for the network with the first row deleted (the program arbitrarily takes the first node to be the dummy node).

The program assumes that the stochastic variable, η, is finitely distributed.

The Method

The L-shaped method of Van Slyke and Wets is an outer linearization procedure that approximates the convex objective term in the stochastic program by successively appending supporting hyperplanes. It introduces the new variable ϑ into the problem, which becomes

minimize

$$cx + \vartheta \tag{2}$$

subject to

$$Ax = b$$
$$\vartheta \geq Q(x)$$
$$x \geq 0.$$

The method begins with an approximation x_1 to the actual solution (x_1 need be little more than a wild guess) and the problem

minimize

$$cx + \vartheta \tag{3.0}$$

subject to

$$Ax = b \tag{3.1}$$
$$U_i x \geq u_i \quad , \quad i=1,\dots,s \tag{3.2}$$
$$Z_i x + \vartheta \geq z_i \quad , \quad i=1,\dots,t \tag{3.3}$$
$$x \geq 0 \quad . \tag{3.4}$$

where Z_i and z_i are chosen so that $Q(x) = \max_i \{z_i - Z_i x\}$ [i.e., (3.3) is equivalent to $\vartheta \geq Q(x)$]. s and t count the number of constraints (3.2) and (3.3), respectively, and are initially set to zero.

The first step in the algorithm is to determine $Q(x_1)$ If $Q(x_1)$ is infeasible, the algorithm creates a "feasibility cut" (3.2) and adds one to the value of s. If $Q(x_1)$ is feasible, the algorithm creates an "optimality cut" (3.3) and adds one to the value of t.

The algorithm generates successive approximations to the optimal solution by solving (3) given some number of constraints (3.2) and (3.3). Let the optimal solution at the νth step be $(x_{\nu+1}, \vartheta_{\nu+1})$. If $Q(x_{\nu+1})$ is infeasible, the algorithm creates a new feasibility cut, adds 1 to s, and solves (3) with the new cut added. If $Q(x_{\nu+1})$ is feasible, the algorithm checks for convergence. If $\vartheta_{\nu+1} \geq Q(x_{\nu+1})$, the optimal solution has been found. Otherwise, the algorithm creates a new optimality cut, adds 1 to t, and solves (3) with the new cut added. Of course, Q may no longer be feasible.

Calculating $Q(\mathbf{x})$

It is important to use an efficient method to find $Q(x)$ in (1). The following scheme is used: let $\xi = \eta - Tx$. Since η is finitely distributed, so is ξ. Find $\underline{\xi}$, the expected value of ξ, and calculate $Q(\underline{\xi})$. Associated with the optimal solution of $Q(\underline{\xi})$ is a basis, W_1, with the property that $W_1^{-1}\underline{\xi} \geq 0$. Furthermore, because the solution is optimal, all reduced costs are nonpositive. Thus, W_1 is the optimal basis for all values of ξ such that $W_1^{-1}\xi \geq 0$. Since the L-shaped decomposition requires the dual solution and since all values of ξ with the same optimal basis also share a single dual solution, it is sufficient to retain a fairly small number of bases, all of which satisfy dual feasibility. It is also necessary to record the total probability mass of the values of ξ for which each basis represents a primal feasible, and therefore optimal, solution. The coefficients needed to create an optimality cut may then be found from

$$\pi = \sum_i p_i \pi_i$$

where p_i is the total probability mass of the values of ξ for which $W_i^{-1}\xi \geq 0$ and π_i is the dual solution associated with W_i. Then

$$Z = \pi T$$

and

$$z = \sum_i [p_i \pi_i \sum_{j \in I(i)} \eta_j] \quad .$$

where $I(i)$ is the set containing the indices of the ξs for which W_i was the optimal basis. (Recall that $Q(x)$ may be defined in terms of z and Z.)

The program generates a search tree, whose nodes contain the Schur complements of the bases that satisfy dual feasibility. For each value of ξ, the program systematically searches the tree to find a basis for which $W_i^{-1}\xi \geq 0$. If none exists, the program pivots a column out of the last basis checked to obtain a new basis. The pivoting continues until an optimal basis is generated and the new basis is added to the search tree.

Approximations

In practice, an approximation to $Q(x)$ must be used since the number of right hand sides can become prohibitively large even for simple problems. The approximation used by STOCHNET (see [3]) generates upper and lower bounds $U(x)$ and $L(x)$ on $Q(x)$. To test for convergence, the program does not compare ϑ to $Q(x)$ as described above; rather, STOCHNET uses the bounds as follows: if ϑ is close to the lower bound and the bounds are close to each other, the program halts. Specifically, if $\vartheta_{\nu+1} \geq L(x_{\nu+1})(1 - \alpha)$ and $U(x_{\nu+1}) \leq L(x_{\nu+1})(1 + \beta)$, the program prints the solution. Otherwise processing continues. α and β are constants and are initialized via a "DATA" statement in the main program. The current values are $\alpha = 0.035$ and $\beta = 0.05$.

Input Overview

STOCHNET requires four data files and a control file. It takes its control data from unit 5; it is the user's responsibility to connect this unit to the appropriate file before the program is invoked. STOCHNET writes prompts for the control information to unit 6 and so the user may provide that data interactively.

Control Information

STOCHNET's control information consists of the names of five files, all of which are described in subsequent sections. File names may contain up to 80 characters. They are read in the following order:

- dump file name
- network data file name

- first stage data file name
- right hand side data file name
- initial x data file name.

Network Data File

This file is connected to unit 7. It defines the network that is used to generate the recourse matrix, W and contains the following information:

- "BEGIN" line. This line indicates the beginning of the network data file. It contains the characters "BEGIN" in columns 1 through 5.
- name line. This line contains the name of the problem. The user may enter any characters desired in columns 1 through 80.
- "ARCS" line. This line indicates the beginning of the arcs description section. It contains the characters "ARCS" in columns 1 through 4.
- arcs description section. This section contains a line describing each arc in the network. The line for each arc contains

  the index of the node at which the arc originates (columns 7-12),

  the index of the node at which the arc terminates (columns 13-18),

  the cost of transporting one unit along the arc (columns 21-30), and

  the capacity of the arc (columns 31-40).

  All these values are integers. The entries in this section must be ordered by the index of the originating node (e.g., the lines for all arcs which originate at node 1 must appear before the lines for arcs originating at node 2).
- "END" line. This line indicates the end of the network data file. It contains the characters "END" in columns 1 through 3.

First Stage Data File

This file is connected to unit 8. It contains the dimensions and contents of the matrices A and T and the elements of the vectors c and b. These values are read using a free format (i.e., '*') and must appear in the order shown below. Each major section must begin a new line.

- the number of rows and the number of columns in A, the constraint matrix.
- the contents of the matrix A and the vector b along with the type of inequality for each row. This information is expressed in the format used by the routines of the Numerical Analysis Group ("NAG"): the contents of a row of A appear, followed by an integer that gives the type of inequality for the row (-1 implies $\leq$, 1 implies $\geq$, and 0 implies $=$), followed by the element of b that corresponds to the row. The entry for each row must begin a new line.
- the contents of c, the cost vector.
- the number of nonzero rows and the number of columns in T, the technology matrix.
- the indices of the nodes to which the nonzero rows of T correspond.
- the contents of T, specified by row. The entry for each row must begin a new line.

Right Hand Side Data File

This file is connected to unit 10. It defines the distribution of the stochastic right hand side, η. It provides the number of random elements of η, their indices, and their distributions. These values are read using a free format (i.e., '*') and must appear in the order shown below. Each major section must begin a new line.

- the number of stochastic elements in η.
- the indices of the nodes to which the stochastic elements of η correspond.
- the distribution of each stochastic element of η. An entry defining the distribution contains two parts and there is one such entry for each stochastic element of η. The first line in an entry contains n, the number of values the element may assume, and s, the smallest such value. The program assumes that permissible values for the element are s, $s+1$, ..., $s+n-1$. Subsequent lines give the probabilities that the element assumes each value; the ith number is the likelihood of $s+i-1$.

Initial x Data File

This file is connected to unit 7. It contains the initial approximation to the actual solution (x_1) used by the algorithm to generate the first cut (3.2) or (3.3).

Output File

The program writes prompts for control information and the optimal values for x and ϑ to unit 6.

Dump File

STOCHNET writes a trace of its execution to the dump file, which is connected to unit 9. The trace includes the current values of x, ϑ, Tx, and the expected value of b. The coefficients of the vectors U_i and Z_i and the values of u_i and z_i that appear in the current set of constraints (3.2) and (3.3) are also printed. These constraints appear in a modified NAG format: the first number on a line indicates the type of inequality, the next several numbers give the coefficients by which the rows of the x vector and ϑ are multiplied, the number just before the last number identifies the type of constraint (0 implies a feasibility cut, 1 implies an optimality cut), and the last number on the line is the value of the appropriate row of the right hand side. For example,

$$-1 \quad 10 \quad -4 \quad 20 \quad 0 \quad 0 \quad 5$$

represents the feasibility cut

$$10x_1 - 4x_2 + 20x_3 \leq 5$$

and

$$-1 \quad 10 \quad -4 \quad 20 \quad -1 \quad 1 \quad 10$$

represents the optimality cut

$$10x_1 - 4x_2 + 20x_3 - \vartheta \leq 10 \quad .$$

The dump also includes the cuts which could be generated for a given value of x and the corresponding values of Q.

Limits and Extensions

Several limits are currently imposed on the problem (e.g., no more than 21 arcs, 100 nodes, 6 rows in the x vector). To extend these limits, the values of the appropriate constants in the include file "Head.h" must be increased. Among the constants are the maximum number of arcs, the maximum number of nodes, the maximum number of rows in the x vector, the maximum number of points in the support of any row of η, the maxmum number of stochatic elements in η, and the maximum number of rows in the technology matrix. These constants are adequately documented in the source code.

Library Routines

STOCHNET uses the standard NAG library.

Subprocedure Hierarchy

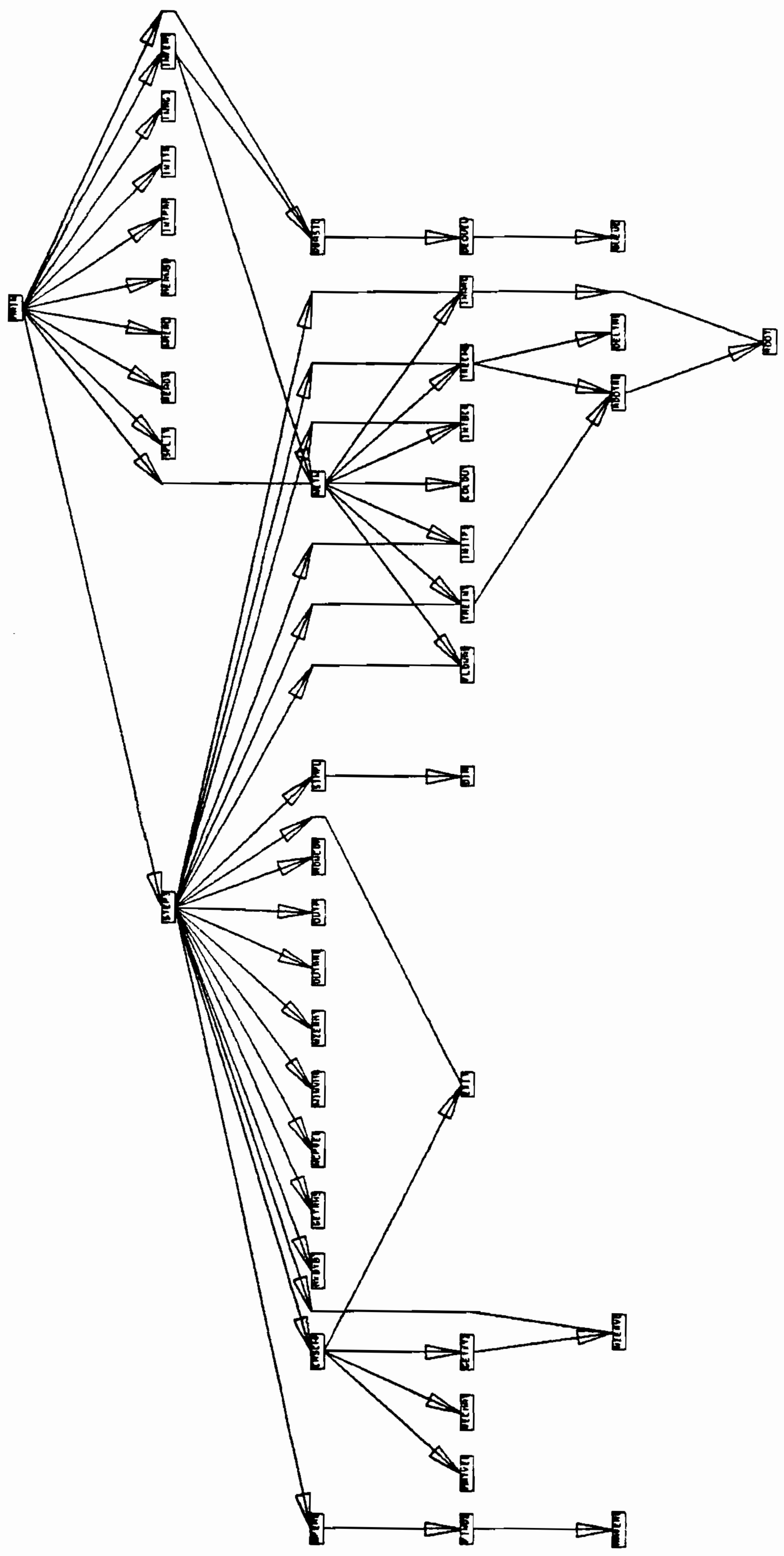
See the attached figure.

References

[1] R. Van Slyke and R. Wets, "L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Linear Programs," *SIAM Journal on Applied Mathematics* v. 17, pp. 638-663, 1969

[2] S. W. Wallace, "On Network Structured Stochastic Optimization Problems," Report no. 842555-8, Chr. Michelsen Institute, 1984

[3] J. Birge and R. J.-B. Wets, "Designing Approximation Schemes for Stochastic Optimization Problems, in particular for Stochastic Programs with Recourse," IIASA Working Paper WP-83-111, 1983


MAIN
STEPS
ROOT