SELECTED MATHEMATICA TOOLS FOR MATHEMATICAL PROGRAMMING

Derek Holmes
Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, MI 48109-2117

Technical Report 93-26

September 1993

Selected *Mathematica* Tools for Mathematical Programming

Derek Holmes
Department of Industrial and Operations Engineering
University of Michigan
Ann Arbor, MI 48109-2117

September 1, 1993.

Abstract: *Mathematica* is a widely used computer software system for computational mathematics which is easily programmable in a higher level language than either Fortran or C. The *Mathematica* language is well suited for algorithmic prototyping and model manipulation. This report describes several *Mathematica* functions which may be of use in linear and stochastic programming. These functions, organized into "packages," are divided into three categories: packages used for determinisitc linear programming, packages used for stochastic programming, and a package which may be used to manipulate LPs and SPs. The last package may be used to transform a general two-stage stochastic program into a multi-stage stochastic program of any size.

Contents

1 Introduction

Mathematica [1], a computer package developed by Wolfram Research, is described as "a system for doing mathematics by computer." The package includes several very high level functions which are of interest to operations researchers, including linear and nonlinear programming, linear systems solutions, and matrix manipulation routines. Of more interest is *Mathematica*'s ability to be programmed in a language flexible enough to allow succint and quick prototyping of any algorithm or concept. For example, complete set operations, string manipulation operations, and procedural programming constructs can be combined with numerical integration routines to read in an arbitrary linear program and solve its continuously distributed stochastic counterpart.

This technical report briefly describes the design and use of several procedures which have been developed by the author for research in stochastic programming. These procedures are organized into sets of functions called "packages", which can be loaded into *Mathematica* and used as any other (built-in) function. (We assume that the reader has a working knowledge of *Mathematica* and is familiar with *Mathematica* terminology.)

The packages are organized into three categories, corresponding to their scope. Packages in the first category are used for general deterministic programming, and include functions to read and write LP and QP MPS files (see, e.g. [2] or [3]) , solve LPs using the simplex method and interior point methods, and solve fractional LPs and parametric LPs. Matrix visualization tools that utilize *Mathematica*'s graphics capability are also included. Packages used for stochastic programming include routines for the Van Slyke and Wets L-shaped algorithm (Van-Slyke and Wets, [4]) and the Edmondson-Madansky and Jensen bounds (see, e.g [?]. Functions in the last category are used to generate arbitrary multistage stochastic linear programs (for an introduction, see, e.g. Ermoliev and Wets, Chapter 1 [5]) from determinisitic LP formulations.

Note: The author makes no guarantees regarding the efficacy or stability of these functions. They are experimental only and are intended for individual modification to a given task.

2 General Determinisitic Programming

This section includes *Mathematica* routines which are of interest to the general operations research practitioner. The routines contained in this section are designed for small problems, but can tackle mesium sized to large tasks given adequate hardware. (In the future, *Mathematica* can be linked with large scale optimizers to allow larger problems to be tackled.)

2.1 Problem manipulation in MPS format (mps.m)

`mps.m` is a package which reads and writes linear programs in MPS format. Data in *Mathematica* are usually described as nested *lists*. Since the elements of the list are not restricted to be of a single data type, they can be used as complex data structures. The routines described in this package use a standard nested list (denoted here by (LP)) to describe the linear program

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & A_i.x \diamond b_i, \quad i = 1, \ldots, m \\ & x \geq 0 \end{array}$$

where $\diamond \in \{=, \leq, \geq\}$ is { `c`, `A`, `b`, `rhstype`, `colnames`, `rownames`, `problemname` }. Here, `c` is a (dense) list of objective elements, `b` is a list of the RHS elements and `A` can either be a dense or sparse representation of the constraint matrix. The lists corresponding to these representations are described below.

Dense	$\{\{a_{11}, \ldots, a_{1n}\}, \ldots, \{a_{m1}, \ldots, a_{mn}\}\}$
Sparse	$\{\{i_1, j_1, a_{i_1,j_1}\}, \ldots, \{i_p, j_p, a_{i_p,j_p}\}\}$

where $a_{i_1,j_1} \ldots, a_{i_p,j_p}$ are the p nonzeros of the constraint matrix.

We define `rhstype` to be a list of m elements, where `rhstype[[i]]` is in $\{0, 1, -1\}$ when constraint i is an (equality, $\leq$, or $\geq$) constraint. (The value corresponds to the multiplier of a slack variable added to ensure

that each constraint is an equality.) **rownames** and **colnames** are lists of $m+1$ and n strings describing row and column names. The first element of **rownames** is always the name of the objective row. Finally, **problemname** is a string identifying the problem.

Given this background, the functions performed by the routines in **mps.m** can be described.

2.1.1 Reading an MPS file

An MPS file is a test file containing sections for describing rows, columns, RHSs, ranges, and bounds. A problem in theis format may be read in using

(in mps.m)	ReadMPS[stream, Options]
Inputs	
stream	An input stream for the MPS file, typically OpenRead[*file*]
Options	
Sparse->True	Output matrix in Sparse format. Use False for dense format.
RemoveNulls->False	Remove rows and columns that contain no nonzeros.

where the output of the function is a list of the form (LP), defined above. (**Note:**) Ranges and Bounds are incorporated into the constraint matrix within **ReadMPS**. New row and column names are formed from the source ranges and bounds.

2.1.2 Writing an MPS file

An MPS file can be written using

(in mps.m)	WriteMPS[stream, c, A, b, Options]
Inputs	
stream	An output stream for the MPS file, typically OpenWrite[*file*]
c	A dense objective vector.
A	A sparse or dense constraint matrix.
b	A dense requirements vector.
Options	
RHSformlist->{list}	constraint form (see above). If missing, all constraints are assumed =.
RowNames->{list}	$(m+1)$-list of strings for the problem rows.
ColumnNames->{list}	(n)-list of strings for the problem columns.
ProblemName->""	The problem name.
Sparse->True	Input matrix is in Sparse format.
Sort->True	Sort columns if constraint matrix is in Sparse form.

The the output of the function is **True** if no problems occurred, **False** otherwise. If **RowNames** or **ColumnNmaes** are not specified, then **WriteMPS** creates its own.

2.1.3 Writing a Quadratic MPS file

The following function outputs *only* the quadratic portion of an MPS file (for format guides, see, e.g. [3]):

(in mps.m)	WriteQuadraticMPS[stream, Q, Options]
Inputs	
stream	An output stream for the MPS file, typically OpenWrite[*file*]
Q	The (dense) quadratic matrix, i.e. Q for $\min x^T Q x + c^T x$.
Options	
ColumnNames->{list}	(n)-list of strings for the problem columns.
ProblemName->""	The problem name.
MulitplybyTwo->False	Mulitplies all entries in Q by 2.

where the output of the function is **True** if no problems occurred, **False** otherwise.

2.1.4 Obtaining submatrices and subproblems

Parts of problems may be separated out by using

(in mps.m)	SelectSubLP[LP, BeginningCoords, EndCoords]
Inputs	
LP	A (Sparse) linear program in the format described for (LP) above.
BeginningCoords	(row,col) coordinates for the upper left hand corner of the subproblem.
EndCoords	(row,col) coordinates for the lower right hand corner of the subproblem.

where the output of the function is the subproblem in the (LP) format and

(in mps.m)	Submatrix[A, BeginningCoords, EndCoords, Options]
Inputs	
A	A matrix in sparse (default) or dense format.
BeginningCoords	(row,col) coordinates for the upper left hand corner of the submatrix.
EndCoords	(row,col) coordinates for the lower right hand corner of the submatrix.
Options	
Sparse->True	Output matrix in Sparse format. Use False for dense format.

where the output is the submatrix in the same form (Dense or Sparse) as the input matrix.

2.1.5 Reading nonzero fragments

A common situation in stochastic programming occurs when a fragment of nonzeros in MPS format must be read and matched up with the column and row names found from a previously read LP. For example, stoch files frequently contain BLOCKS or SCENARIOS sections with submatrix realizations. The following function reads in these fragments.

(in mps.m)	NonzeroFragment[stream, LP]
Inputs	
stream	An input stream with the nonzeros in a form similar to MPS's COLUMNS section.
LP	A linear program in (LP) format, described above.

Output of the function is the submatrix corresponding to the nonzero fragment in sparse form.

2.2 Problem Plots (lp.m, lpeduc.m)

One of *Mathematica*'s greatest strengths is its graphics capabilities. These capabilities can be used to visualize the nonzero structures of large problems, or to visualize a particular algorithm for educational purposes. We split these functions into problem visualization tools and algorithm visualization tools.

2.2.1 Problem Visualization

The following functions use these capabilities to display the nonzero structure of a given problem.

(in mps.m)	LPPlot[LP, Options]
Inputs	
LP	A linear program in (LP) format, described in Section 1.1.
Options	
Sparse->True	Input matrix is in Sparse format. Use False for dense format.
HighlistList->{}	Output 3-D graph in which the (sparse) nonzeros in HighlistList are taller than the underlying LP.
Output->False	Function returns Graphics object if True, or shows graph and returns True.

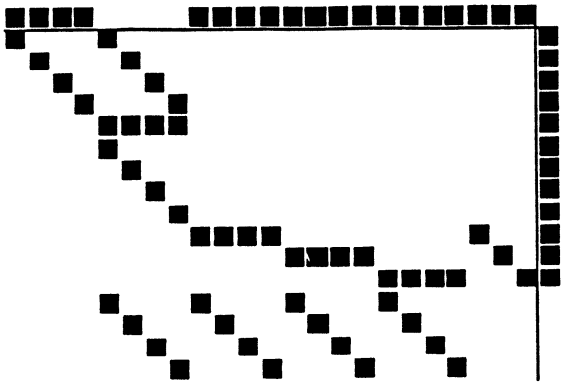

Figure 1: CEP1 matrix

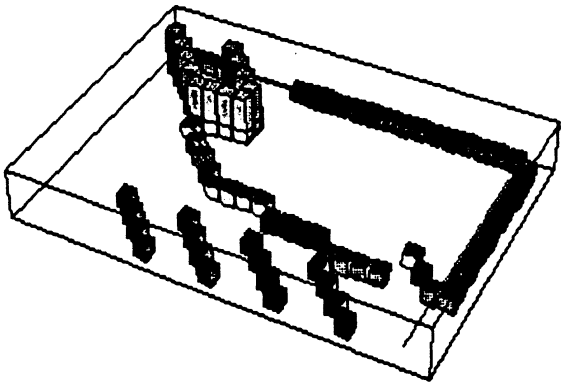

Figure 2: CEP1 with first stage highlighted

Here, the output depends on the `Output` option shown above. Two examples of the output of this function applied to a problem from the literature are shown below. The first is the problem without highlights. The second example shows the problem with the first period constraints highlighted. (Higlights don't work very well for large problems)

Prototype: (in `mps.m`) `MatPlot[c,A,b,Options]` where the inputs are the elements of an LP and the options and output are identical to `LPPlot`.

For educational purposes, it is frequently desirable to plot arbitrary two-dimensional polytopes, and show the progress of algorithms within these regions. The following function determines the Graphics objects to visulaize the polytope described by $\{x|Ax \leq b\}$.

(in `lpeduc.m`)	`PolytopeGraph[A,b]`
Inputs	
`A,b`	The constraint matrix and RHS vector in $Ax \leq b$.

and the output is a list of Graphics objects. To see the graph, type `Show[PolytopeGraph[A,b], Axes->True]`.

2.3 Solving linear programs and their variants (lp.m, lpeduc.m)

Although *Mathematica* has several optimization functions, including one for Linear Programming, these are frequently inadequate for research purposes. The biggest problem with the `LinearProgramming` function is that it returns only a primal solution; no dual solution is available when (e.g.) a problem is unbounded.

The first package, `lp.m` is the author's own implementation of common simplex-based algorithms for linear programming and its variants. The algorithms are taken from (Bazaraa, Jarvis, and Sherali, [7]) and are given after some problem manipulation routines.

2.3.1 Problem Manipulation Routines

A table of some useful problem manipulation routes is given below.

Forming Problems	
`AppendColumns[`$M_1, M_2, \ldots$`]`	Returns a submatrix with matrices M_i stacked on top of each other.
`AppendRows[`$M_1, M_2, \ldots$`]`	Returns the matrix $(M_1 \vdots, M_2, \vdots, \ldots)$.
`BlockMatrix[`M`]`	Returns a valid matrix of elements when M is a matrix formed from submatrices.
`ZeroMatrix[m]`	Returns a square zero matrix of m elements.
Changing Problems	
`MakeEqualityConstraints[ c,A,b,rhstype, Sparse->True]`	Returns an equality constrained problem formed from an LP in (LP) form.
`LTtoEQ[c,A,b]`	Returns an equality constrained problem formed from $\min\{c^T x \mid Ax \leq b, x \geq 0\}$.
Vector Tests	
`VecGT[vec,no]`	returns True if `vec[[i]]` $>$ no for all i.
`VecGE[vec,no]`	returns True if `vec[[i]]` $\geq$ no for all i.
`VecLT[vec,no]`	returns True if `vec[[i]]` $<$ no for all i.
`VecLE[vec,no]`	returns True if `vec[[i]]` $\leq$ no for all i.
`VecEQ[vec,no]`	returns True if `vec[[i]]` $=$ no for all i.

A useful tool for "debugging" linear programs and computer codes that manipulate them is to find the set of nonzeros that are not common to two linear programs. A function for finding these nonzeros is

(in `mps.m`)	`LPDiff[LP1, LP2]`
Inputs	
`LP1,LP2`	Two (sparse) linear programs in (LP) format to be compared.

whose output is an LP in (LP) format with nonzero elements that are in either `LP1` or `LP2`, but not both.

2.3.2 Linear Programming using the Primal Simplex Method

Several functions in this package fit together to solve an LP problem. Most users will call LP, which uses the primal simplex method. Feasibility is obtained using the Big-M method (M is user settable). The Big-M method implemented here adds an artificial variable to each constraint and attempts to force it out of the basis first. Also, LP prevents cycling with an implementation of Bland's Rule (discussed in, e.g. [7]). (A warning message will be printed when 40 zero steplength pivots are performed, and Bland's Rule will be invoked after 80 zero steplength pivots.)

Inputs	
(in lp.m)	LP[c,A,b, Options]
c, A, b	A (dense) linear program of the form $\min\{c^T x \mid Ax = b, x \geq 0\}$.
Options	
IdentityColumns->{}	Use an identity in A given by a list of column numbers as artificials.
Plevel->0	Print more detail about algorithmic progress. Maximum: 3.
Math->False	If True, use *Mathematica*'s LinearProgramming function.
BigM->100000	Penalty for infeasibilities.

and the output is a nested list of the form { z, xstar, pistar }, where z is the optimal objective function value, and xstar and pistar are the optimal primal and dual solutions, respectively.

LP sets up the problem by adding artificial columns and declaring these to be an intial basis. The routine then calls LPwithBasis, with prototype

(in lp.m)	LPwithBasis[Basislist, c,A,b, Options]
Inputs	
Basislist	is a list of columns that define an invertible submatrix.
c, A, b	A (dense) linear program of the form $\min\{c^T x \mid Ax = b, x \geq 0\}$.
Options	
Artificials->{}	List of artificial column numbers.
Plevel->0	Printing detail.
Bland->False	If True, start with Bland's Rule.

whose output is a nested list of the form { z, xbasic, pistar, Basislist } where z and pistar are defined as above, xbasic is the optimal values of the basic variables contained in Basislist.

2.3.3 Parametric and Fractional Linear Programming

A parametric linear program has the general form $\min\{(c + \lambda c')^T x \mid Ax = b, x \geq 0\}$, where $\lambda \geq 0$. LPPpara solves problems of this sort by performing basis change analyses within the context of the simplex method.

(in lp.m)	LPpara[plevel,iform,c,c',A,b,Options]
Inputs	
plevel	is the level of output printed by the algorithm (0=none).
iform	is 1 if the constraints are of the form $Ax \geq b$, 0 if $Ax = b$.
c, c', A, b	describe the parametric program.
Options	
Limits->{lo,hi}	Perform parametric anlaysis for $\lambda \in$ { lo,hi }.

The optimal value of a parametric linear program is a convex piecewise linear function. So, the solution from the algorithm is a list of p piecewise linear terms of the form $z + \lambda z'$ for some range of $\lambda \in \{\underline{\lambda}_i, \bar{\lambda}_i\}, i = 1, \ldots, p$. Specifically, the output is a nested list of the form { z, xsollist, pi, paralist } where xsollist = $\{x_1^*, \ldots, x_p^*\}$ is a list of optimal primal solutions for the p piecewise linear terms and paralist is a list of p lists of the form $\{\underline{\lambda}, \bar{\lambda}, z, z'\}$. Here, the first two terms are the ranges of λ for which a given basis is optimal and second two terms define the optimal objective value $z + \lambda z'$ over that range.

A fractional linear program has the general form

$$\min \frac{p^T x + \alpha}{q^T x + \beta}$$

subject to $x \in \{x|Ax \leq b, x \geq 0\}$. The method of Charnes and Cooper [8]) rewrites this nonlinear program as a linear program of the form $\min p^T y + \alpha z$ subject to $Ay \leq bz$, nonnegativity constraints, and $q^T y + \beta z = 1$. The optimal solution to the original program is $x^* = y^*/z^*$. The following function solves this problem.

(in `lp.m`)	`LPfrac[plevel,iopt, numvec, numcons, denvec, dencons, A,b]`
Inputs	
`plevel`	is the level of output printed by the algorithm (0=none).
`iopt`	is 1 if the denominator is always > 0, -1 otherwise.
`numvec,numcons`	describe the objective numerator.
`denvec,dencons`	describe the objective denominator.
`A,b`	describe the constraint set.

The output is a list of the form `{zstar, xstar}` where `zstar` is the (scalar) optimal objective function value and `xstar` is an optimal solution.

2.3.4 Small scale Interior Point Algorithms

`lpeduc.m` contains several functions for experimenting and visualizing interior point algorithms. Two types of algorithms are implemented: the affine scaling algorithm and the projective scaling algorithm. Details of the development and notation used for the algorithms are due to (Birge [12]) and are included as an appendix to this technical report. The relevent routines are `AffScaling` and `ProjScaling`.

`AffScaling` runs the affine scaling algorithm until the stopping criterion $||s_k||^2 < \epsilon$ is met, where s_k is the kth search direction.

(in `lpeduc.m`)	`AffScaling[AffScaling[A,b,c,x,gamma,eps,Options]`
Inputs	
`c, A, b`	A (dense) linear program of the form $\min\{c^T x|Ax = b, x \geq 0\}$.
`x`	is an intial interior starting point $x \in \{x|Ax < b\}$.
`gamma`	is the fraction of the largest stepsize to take.
`eps`	is the stopping criteria mentioned above.
Options	
`Outputs->{}`	List of output options (see below).
`Print->{}`	List of printing options (see below).
`EllipseRanges->{lo,hi}`	Range of iterations for which ellipses are printed (if enabled).
`MatrixUpdate->0.8`	Update projection matrix only if relative change exeeeds this number.
`EllipseStepSize->True`	Use ellipse step sizes if `True`, γ of max if `False`

A list of the output options which can be included in the `Outputs` and `Print` options is given below:

Option	Meaning
`Solution`	Final Solution and objective
`Ellipses`	Ellipses at each iteration
`EllipseGraphics`	Ellipse graphics at each iteration
`Iterates`	Sequence of Iterates (points)
`Bounds`	Lower/Upper Bounds
`sNorms`	Norms of Search Directions
`StepSizes`	Actual StepSizes
`SearchDirections`	Search Directions
`Duals`	Dual iterates
`All`	All of the above

The projective scaling implementation is similar. For details, refer to (Birge, [12]).

(in lpeduc.m)	ProjScaling[ProjScaling[A,b,c,x,gamma,eps,UB, Options]
Inputs	
c, A, b	A (dense) linear program of the form $\min\{c^T x \mid Ax = b, x \geq 0\}$.
x	is an intial interior starting point $x \in \{x \mid Ax < b\}$.
gamma	is the fraction of the largest stepsize to take.
eps	is the stopping criteria mentioned above.
UB	is an initial upper bound on $x^T x^*$.
Options	
Outputs->{}	List of output options (see below).
Print->{}	List of printing options (see below).
EllipseRanges->{lo,hi}	Range of iterations for which ellipses are printed (if enabled).
MatrixUpdate->0.8	Update projection matrix only if relative change exeeeds this number.
PolynomialStepSize->alpha	Use polnomial stepsize with a given α.

A list of the output options which can be included in the Outputs and Print options is given below:

Option	Meaning
Solution	Final Solution and objective
Ellipses	Ellipses at each iteration
xInnerEllipseGraphics	Inner Ellipse graphics (in x-space)
xOuterEllipseGraphics	Outer Ellipse graphics (in x-space)
zInnerEllipseGraphics	Inner Ellipse graphics (in z-space)
zOuterEllipseGraphics	Outer Ellipse graphics (in z-space)
Iterates	Sequence of Iterates (points)
Bounds	Lower/Upper Bounds
sNorms	Norms of Search Directions
StepSizes	Actual StepSizes
SearchDirections	Search Directions
UBs	Sequence of Upper Bounds
Duals	Dual iterates
All	All of the above

A particularly common linear program solved in stochastic programming is an LP of the form $Q(x) = \min\{q^T y \mid Wy = h - Tx, y \geq 0\}$. This program may be solved by calling (in lp.m) Q[x,h,W,T,q] where the inputs are the (dense) vectors and matrices described above.

3 Stochastic Programming

This section describes a few routines which may be of interest to stochastic programmers. *Mathematica* is especially useful for approximation algorithms, since it has powerful list manipulation capabilities. These capabilities are most useful for extremal approximations such as the Edmundson-Madansky upper bound and the Jensen lower bound. (See, e.g. [11]) Currently, two main packages have been developed. The first is for approximations, and the second implements the Van Slyke and Wets' L-Shaped algorithm for solving two stage linear programs.

3.1 Approximations

One of the more difficult tasks in stochastic porgramming is calculating the expected value of the second stage solution. This calculation requires a multidimensional integration of an implicitly defined function.

The most common method for approaching this problem is to use approximations to the second stage functional. An extensive set of approximations have been developed (see, e.g. Birge and Wets [13]). The most common procedure is to find a set of discrete realizations which closely approximates the underlying distribution. Doing so allows the integral to be written as a finite sum of solutions to second stage optimization problems.

The two most common discretization procedures are based on Jensen's inequality and the Edmundson-Madansky inequality. (again for detail, see [13]). The problem we wish to approximate is a two-stage stochastic linear program with recourse, which can be written as

$$\min c^T x + Q(x)$$

$$\text{s. t. } Ax = b, x \geq 0$$

where $Q(x) = E[Q(x,\xi)]$ and $Q(x,\xi) = \min\{q^T y | Wy = \xi - Tx, y \geq 0\}$. The Jensen inquality finds a lower bound on $Q(x)$ using the inquality

$$Q(x) = E[Q(x,\xi)] \geq \sum_{l=1}^{\nu} p^l Q(x, E[\xi|S^l]),$$

where $(S^1,\ldots,S^\nu)$ is a partition of $\xi's$ support (Ξ) and $p^l = P[\xi \in S^l]$.

The Edmundson-Madansky upper bound implemented here assumes that the support Ξ is of the form $[a_1,b_1]\times,\ldots,\times,[a_N,b_N]$, and that the components of ξ are independent. The bound is

$$Q(x) \leq \sum_{e \in \text{ext } \Xi} (\prod_{i=1}^{N} \frac{|\bar{\xi}_i - e_i|}{b_i - a_i}) Q(x,e).$$

Refinements are taken by partitioning the support Ξ into several subrectangles $(S^1,\ldots,S^\nu)$ and calculating the conditional bounds over each partition. The partitioning method implemented follows that of Frauendorfer and Kall [11], which (1) splits the cell with the greatest error between upper and lower bounds, (2) considers all 2^{m+1} points to find the point and direction which maximizes the minimum linearization error. (The functions implemented here also assume the underlying continuous distribution is multiariate uniform.)

Although both bounds require a description of the support, keeping more data can greatly speed up the calculation of the bound. Specifically, keeping the LP solution at each extremal point of the support and the mean solution eliminates the need to recalculate them each time a new bound is calculated. Each of these sets of data must be kept for each partition. The following lists are the "data structures" which hold this data:

Name	Contents	Format
PARTN	Extreme points defining partition	{ pt 1,.., pt 2^n }
SUPPTS	Set of partitions	{ PARTN(1),..,PARTN(K) }
MEANS	Set of means of a parition	{ {mean 1},..,{mean k} }
PROBS	Marginal probabilities of a parition	{ p1,..,pk }
SUPPORT	Complete description of a support	{ SUPPTS, MEANS, PROBS }
SOLN	LP Solution to a point (z, x^*, π^*)	{ z, xstar, pistar }
PTSOLS	List of solutions of extrema of a partition.	{ SOLN(1),..,SOLN(2^n) }
SOLVEDSUPPORT	SUPPORT with extrema solutions.	{ SUPPORT , { PTSOLS(1),,..,PTSOLS(K) } }
TIMES	Cumulative timing data (LB times, UB times, Partitioning times)	{ Time, Time, Time }
TIMEDSUP	SOLVEDSUPPORT with timing data	{ SOLVEDSUPPORT, TIMES }

These lists serve as input to the main partitioning functions `CP` and `CompletePartition`:

(in approx.m) (in approx.m)	CP[iteration, timedsup, x, hconst, W,T,q, totalarea, streams] CompletePartition[iteration, solvedsupport, x, hconst, W,T,q, totalarea, cumt, streams]
Inputs	
iteration	The iteration or refinement number.
timedsup	A list in TIMEDSUP form (shown above).
x	The current primal iterate.
hconst	$(h_1, \ldots, h_k)$ in the second stage RHS $(h_1, \ldots, h_k, h_{k+1}, \ldots, h_m)$.
W,T,q	Describe the second stage linear program.
totalarea	A divisor for normalizing the total initial probability to 1.
streams	A set of 3 streams for outputting (all dual solutions, not used, not used). Note: Production versions will eliminate streams.
solvedsupport	A list in SOLVEDSUPPORT form (shown above).
cumt	A list in TIMES form (shown above).

Both calculate an upper and lower bound based on the refined support given and refine the support for another iteration. The output is a list of the form TIMEDSUP.

These functions call several other functions to find the bounds and the partition. These functions are described in the following sections.

3.1.1 Upper Bounds

The main function that calculates the upper bound is **UBEM**, which has the prototype

(in approx.m)	UBEM[itertation, x, support, hconst, W,T,q,probsup, opts]
Inputs	
x, hconst, W,T,q	are defined as above.
iteration	The iteration or refinement number.
support	A list in SUPPTS form (shown above).
probsup	A list in PROBS form (shown above).
Options	
Streams->{}	A list of three streams for outputting dual information (see below).
LBs->{}	A list of Lower Bounds for each cell to determine the cell to split.
Optpi->{}	A list of the optimal dual mulitpliers at x for comparison purposes.
PrintLevel->1	is the level of debugging detail.

The **Optpi** and **Streams** options are used to determine the distance of the approximated dual solutions from the actual dual solutions. Solutions to each new point are output to **Streams[[1]]**. Expected duals within each parition and their 2-norm distances to the optimal (marginal) π^* are printed to **Streams[[2]]**, while the overall expected dual estimate and its distance to π^* are printed on **Streams[[3]]**.

UBEM returns a compund list { **UB, newpis, UBlist, errorlist, celltosplit**}, where the overall upper bound and new duals are the first two items. The third item is a list of upper bounds for each partition, and the last two items are the differences $UB_i - LB_i$ in each cell (which is nonempty only if **LBs** is specified), and the cell which maximizes this error.

To save computation, UBEM develops the upper bound from a list of solutions which are generated using the function:

(in approx.m)	SolvePointList[solnlist, support, x, hconst, W,T,q]
Inputs	
x, hconst, W,T,q	are defined as above.
solnlist	A list of solutions of the form PTSOLS.
support	A list in SUPPTS form (shown above).

The output is a new **solnlist** in PTSOLS form.

3.1.2 Lower Bounds

The main function that calculates the Jensen lower bound is LB, which has the prototype

(in approx.m)	LB[itertn, hbar, hconst, probsup, x, W,T,q, opts]
Inputs	
iteration, x, hconst, W,T,q	are defined as above.
hbar	A list of expected RHSs in each partition.
probsup	A list in PROBS form (shown above).
Options	
Streams->{}	A list of three streams for outputting dual information (see below).
Optpi->{}	A list of the optimal dual mulitpliers at x for comparison purposes.
PrintLevel->1	is the level of debugging detail.

Again, the Optpi and Streams options print the lower bounds and differences between the approximated dual solution and the optimal dual solution. LB returns a compound list of the form {LB, LBlist}, which contains the overall lower bound and the partition-specific lower bounds.

3.1.3 Refinement Procedures

As mentioned earlier, the single partitioning strategy implemented is due to Fraundorfer and Kall, who assume a rectangular support. Several functions determine the refinement to be taken, all of which are called by

(in approx.m)	FindFKRefinement[Sup, sollist, partition,pivec,hbar, probsup,totalarea, opts]
Inputs	
Sup	A list in SUPPTS form (shown above).
sollist	A list of solutions of the form PTSOLS.
partition	The partition number to split.
pivec	A list of duals for the cell to be partitioned. (Use TrimPi[ubout[[2]],hconst] where ubout is from UBEM).
hbar	A list of expected RHSs in each partition.
probsup	A list in PROBS form (shown above).
totalarea	A divisor for normalizing the total initial probability to 1.
Options	
PrintLevel->1	is the level of debugging detail.

The returned list is of the form SOLVEDSUPPORT, with the new partitions replacing the old partition in the list. FindFKRefinement calls the function MaptoExtrema[pointlist, extremalist], which maps a list of references to extreme points to their actual values.

3.1.4 Miscellaneous functions

A few other functions have been developed to assist researchers to experiment with approximations. We review their prototypes here.

- (in approx.m) Findsupportdata[support, option, totalarea]

  returns the mean value and probability of partitions of a rectangular support, assuming multivariate uniform distributions. support and totalarea are defined above, and option is 1 for the first call to the function,where and 2 if not.

- (in approx.m) FindPartition[Sup, chi, hconst, options]

  returns the partition number in Sup (of the form SUPP defined above) which contains the right-hand-side (Chi, hconst). If the RHS is not in Sup, the function returns zero. The only option for this function is PrintLevel.

- (in approx.m) `FindEncSimplex[basept,Basis,A,b,options]`

  returns a list of $n+1$ points which describe a simplex which encloses the region $K = \{x|Ax \geq b, x \geq 0\}$. The function requires a starting extreme point of K called **basept** and a list of columns defining basespt's basis. The only option for this function is `PrintLevel`.

- (in approx.m) `FormSimplexCsontraints[Simplex, options]`

  returns a feasible region of the form $K = \{x|Ax \geq b\}$ which describes the simplex **Simplex** (a list of $n+1$ points). The returned compound list is of the form `{A,b}`. Again, the only option for this function is `PrintLevel`.

3.2 L-Shaped Method

The most common specialized algorithm for solving two stage stochastic linear programs is the *L-Shaped* method (Van Slyke and Wets [4]), which is based on Bender's decomposition. The algorithm, which is also approximates the value of the recourse function (second-stage solution) by finding *cuts*, which are constraints on both the approximation and the decision variables. A full discussion of this algorithm is beyond the scope of this report, so we just describe the main functions and their subfunctions.

There are two main functions which implement the two-stage cutting plane algorithm. The first solves discrete problems and the second uses the approximations described above to solve problems with uniformly distributed RHSs. The prototype for the first algorithm is:

(in cpa.m)	`CutPlane[Supin,hconst,A,b,c,W,T,q, options]`
Inputs	
`Supin`	is a list of realizations and their probabilities of the form `{{h1,...,hk},{p1,...,pk}}`
`A,b,c`	First Stage data.
`W,T,q`	Deterministic Second stage data.
`hconst`	Deterministic RHS for second stage . (Deterministic rows come first in $Wy = h - Tx$).
Options	
`PrintLevel->1`	is the level of debugging detail.

The function returns the optimal first stage decision x. Of course, the primary use of this function is to modify it to output other items of interest or try variants on the algorithm.

The second implementation uses the Jensen and Edmunson-Madansky upper bounds in `approx.m` to solve a two stage stochastic linear program with uniformly distributed RHSs. The prototype for the function is

(in cpa.m)	`CutPlanewApprox[maxits,Supin,hconst,A,b,c,W,T,q, options]`
Inputs	
`maxits`	is the maximum number of complete iterations allowed.
`Supin`	is a support of the form SUPPTS.
`A,b,c`	First Stage data.
`W,T,q`	Deterministic Second stage data.
`hconst`	Deterministic RHS for second stage . (Deterministic rows come first in $Wy = h - Tx$).
Options	
`PrintLevelAlg->1`	is the level of algorithmic debugging detail.
`PrintLevelBounds->1`	is the level of bounding debugging detail.

`CutPlanewApprox` also returns the optimal first stage decision x.

One of the most lucrative improvements suggested for the L-Shaped algorithm dur to (Birge and Louveax, [16]) is the inclusion of multiple cuts at each iteration. This modification has been implemented in `cpa.m`, and may be run using the functions:

- (in cpa.m) `MultiCutPlanewApprox[maxits,Supin,hconst,A,b,c,W,T,q, options]`

- (in cpa.m) `MultiCutPlane[Supin,hconst,A,b,c,W,T,q, options]`

The inputs, options, and outputs are the same as in `CutPlane` and `CutPlanewApprox`.

3.3 Miscellaneous Functions

To date, only one function has been written which cannot be classified into the other categories. This function (and future ones like it) is in `stopro.m` and finds the size (in term of rows and columns) of a deterministic representation to a multistage stochastic program. The function returns a list `{matrices, nodes, rows, cols}` with the total number of matrices, nodes, rows, and columns in the determinsitic equivalent. The prototype and arguments are

(in stopro.m)	DetEquiv[scenlist, pds, Adims, WMdims, WEdims]
Inputs	
scenlist	is a list with the number of distinct realizations in each period.
pds	is the number of periods.
Adims	is a list with the dimensions of the first period matrix A.
WMdims	is a list with the dimensions of the middle period matrices.
WEdims	is a list with the dimensions of the last period matrix.

4 Stochastic Program generation and manipulation

The last category of functions concerns the manipulation and generation of large multistage stochastic programs (MSLPs). Currently, there are two known programs for solving large MSLPs, MSLiP (Gassman, [14], [15]) and ND, (Donohue, [17]). MSLiP will accept stochastic programs in a standard format (Edwards, et. al. [18]), while ND uses a more customized format.

The functions in `makesto.m` will take an arbitrary linear program in MPS format (see section 2), and, given a description of stochasticity and the coordinates necessary to split the problem into periods, generate the input files necessary for both programs. The presentation of these functions will be split into three parts: reading in data, forming stochastic descriptions, and writing output files. A full example will be presented in a fourth section.

4.1 Reading in Data

The basic premise of the `makesto` package is that a multistage stochastic linear program can be split into periods, and that constraints on first period decisions, intermediate period decisions, and last period decisions may all be different.

The constraints for all periods are generally contained in one MPS file (called a CORE file). Since decisions in each period cannot depend on future decisions, the CORE file must be lower block triangular. The functions in `makesto.m` make the additional assumption that the problems are Markovian in nature. This implies that the CORE file will actually be block diagonal.

Most practical problems will have CORE files that describe at most the first, intermediate, and final periods. (CORE files that describe more than that may be cut down for `makesto.m`'s purposes. As a result, the only information beyond the CORE file needed to extend the stochastic program is the upper left-hand coordinates of the *Repeating Period* and the *Ending Period*. (These coordinates will henceforth be abbreviated as `repc` and `endc`.) In standard format, these coordinates are expresseed as row and column names, and are in a separate file called a TIME file.

In some cases, constraints on all decisions in periods $2,\ldots,T$ may all be the same. If this is the case, then the repeating period and the ending period have the same starting coordinates.

Assuming that `ReadMPS` (described above) is used to read in the CORE file, `repc` and `endc` may be found by looking at the matrix with `LPPlot`, or by using `FindCoords`, which finds the coordinates of a named column and row, or by reading an existing TIME file. The following two functions may be useful in this regard.

(in makesto.m)	FindCoords[LP, row, col]
Inputs	
LP	is a complete linear program in (LP) format.
row	is a string (8 characters or less) containing a row name.
col	is a string (8 characters or less) containing a column name.

(in makesto.m)	ReadTimeFile[filename, colnames, rownames]
Inputs	
filename	A TIME filename.
colnames,rownames	are lists of strings obtained from ReadMPS.

The output of the first function is the coordinates of the desired element, and the output of the second function is a list of starting coordinates for each time period. Once these coordintes have been obtained, SplitCore can be used to separate out the CORE file into its constituent components.

(in makesto.m)	SplitCore[LP, RepeatPeriodStart, EndPeriodStart, options]
Inputs	
LP	is a complete linear program in (LP) format.
RepeatPeriodStart	is the upper left hand corner of the repeating period constraints.
EndPeriodStart	is the upper left hand corner of the final period constraints.
Options	
WriteLPform->True	Writes each block as a full LP in (LP) format.
AddNames->True	includes row and column names in the output LPs.

Extending stochastic programs may also require input of an existing description of stochasticity. In standard format, stochsticity is described using a number of formats in a STOCH file. Currently, only one format can be read using the functions in makesto.m. This format is the BLOCKS format, which is used to described problems with stochastic T (off-diagonal) matrices. The function ReadBlockFile, described below, will return a list of the form {{period, {{p1,T1}, ..., {pn,Tn}}, ... }. Here, the technology matrix in period period takes on one of n different realizations T_i, each with probability p_i.

(in makesto.m)	ReadBlockFile[filename, nperiods, pdnamelist, LPlist]
Inputs	
filename	is the STOCH filename to be read.
nperiods	is the number of periods in the STOCH file.
pdnamelist	is a list of period names obtained from a TIME file.
LPlist	is a list of LPs, obtained directly by running SplitCore with WriteLPform->True.

4.2 Describing stochasticity

Stochasticity in either the RHS or the technology matrix T_i can be described in several ways. To make these descriptions as succinct and as easy to use as possible, the following options have been developed:

- **IndependentRHS**, which requires as input a list of the form

$$\{\ldots,\{t,i,\{\{b_i(1),p_i(1)\},\ldots,\{b_i(k),p_i(k)\}\}\},\ldots\}.$$

  Each RHS b_i in period t is independently distributed with the takes on the value $b_i(k)$ with probability $p_i(k)$.

- **SampledRHS** which is a list of the form $\{seed,\{t,p,n,\bar{b},\bar{\ell},\bar{u}\},..\}$ where *seed* is a random number seed, t is the period for the sample, p is the maringal probability for each item in the list, and n is the number of samples generated (each with probability p/n). Samples are generated using the formula $b_i = (1+\alpha)\bar{b}_i$ where α is unformly distributed between $\bar{\ell}_i$ and $\bar{u}_i$. This option is best for generating huge problems.

- **Blocks** which is a list of the form

$$\{\ldots,\{t,\{p_1,T_1\},\ldots,\{p_k,T_k\}\},\ldots\}$$

where T_i is a matrix realization in *sparse* form. This option is the same as the BLOCKS format in STOCH files.

The lists described above are *Mathematica* lists. In most cases, **BLOCKS** lists will be generated using **ReadBlockfile**, **SampledRHS** lists will be generated by using **ReadList** or **ReadMPS** to read an existing RHS, and **IndependentRHS** lists will be generated manually, or by using the following function

(in **makesto.m**)	**Makestochlist[period, element, mean, diff, n, options]**
Inputs	
period,element	The stochastic element.
mean	is a base value from which to generate realizations.
diff	is an increment value (see below)
n	is the number of realizations.
Options	
Range->{lo,hi}	is a range specification for values.

Realizations are usually incrementally generated using the formula **mean** $\pm$ **i*diff**, where **i** is incremented until n realizations have been formed. In other words, **diff** is successively added and subtracted from the highest and lowest realizations until **n** realizations have been formed. The result returned is this list of numbers.

If **Range** has been specified, then **mean** and **diff** are ignored, and **n** relizations uniformly spaced between the ranges specified by **Ranges** are returned.

4.3 Outputting stochastic programs

Once the split stochastic program has been loaded into memory and a suitable definition of the stochasticity has been created, the following functions will form an arbitarily sized stochastic program and output it in standard format.

(in **makesto.m**)	**MSCoretoCore[infile, outcorefile, outtimefile, repc, endc, stages, options]**
Inputs	
infile	is an input CORE file.
outcorefile	is an output CORE file with **stages** stages.
outtimefile	is an output TIME file with **stages** stages.
repc,endc	are the repeating and ending block coordinates.
stages	are the number of stages to output.
Options	
SplitCoreLP->{}	An existing split core from **SplitCore**
CoreLP->{}	An existing core LP from **ReadMPS**.

The result returned will be **True** unless an error occurs. The companion function to **MDCoretoCore** is **WriteStochfile**.

(in **makesto.m**)	**WriteStochfile[outstochfile, stages, stochlist, options]**
Inputs	
outstochfile	is an output STOCH file with **stages** stages.
stages	are the number of stages to output.
stochlist	is a list describing the stochasticity.
Options	
StochlistForm->"IndependentRHS"	Form of **stochlist**.

The multistage solver ND accepts problems using two files, a *fort* file, which contains diagonal constraint matrices W_t for each period t, and a *scendata* file, which contains the T matrices and RHSs. The scendata file does not require all nodes in a decision tree, but only those which can be repeated for a given stage.

(in makesto.m)	MSCoretoND[infile, outfortfile, outscenfile, repc, endc, stages, stochlist, options]
Inputs	
infile	is an input CORE file.
outfortfile	is an output fort file with stages stages.
outscenfile	is an output scendata file with stages stages.
repc,endc	are the repeating and ending block coordinates.
stages	are the number of stages to output.
stochlist	is a list describing the stochasticity.
Options	
StochlistForm->"IndependentRHS"	Form of stochlist.
SplitCoreLP->{}	An existing split core from SplitCore
CoreLP->{}	An existing core LP from ReadMPS.

Again, the result returned is True if all files were sucessfully written, and False otherwise.

4.4 An example

As a brief demonstration for how these routines fit together, the following notebook fragment will read in a stochastic program in standard format and output several versions of increasing size.

```
Input files and partition
<<d:\r\math\makesto.m
<<d:\r\probs\ndprobs\agr7\stochs.m
CORE=ReadMPS[str=OpenRead["e:core"], Sparse->True]; Close[str];
repc={16,21}; endc={35,41}; SPLITCORE=SplitCore[CORE,repc, endc];
Off[NumberForm::sigz];
Output 4 Stages
MSCoretoCore["e:core","e:p4.cor","e:p4.tim",repc,endc,4,
StochlistForm->"IndependentRHS", SplitCoreLP->SPLITCORE];
WriteStochfile["e:p4s2.sto",4,s4s2,StochlistForm->"IndependentRHS"]
WriteStochfile["e:p4s4.sto",4,s4s4,StochlistForm->"IndependentRHS"]
WriteStochfile["e:p4s8.sto",4,s4s8,StochlistForm->"IndependentRHS"]
WriteStochfile["e:p4s16.sto",4,s4s16,StochlistForm->"IndependentRHS"]
WriteStochfile["e:p4s32.sto",4,s4s32,StochlistForm->"IndependentRHS"]
```

The file **stochs.m** contains several lists describing the stochasticity to be modelled:

```
(* --- 3 stage------------------------------------------------------------ *)
s3s2 := List @@
   { Makestochlist[1,5,1700,200,2],
     Makestochlist[2,11,1700,200,2]};
(* --- 4 stage------------------------------------------------------------ *)
s4s2 := List @@
   { Makestochlist[1,5,1700,200,2], Makestochlist[2,5,1700,200,2],
     Makestochlist[3,11,1700,200,2]};
s4s4 := List @@
   { Makestochlist[1,5,1700,200,4], Makestochlist[2,5,1700,200,4],
     Makestochlist[3,11,1700,200,4]};
s4s8 := List @@
   { Makestochlist[1,5,1700,125,8], Makestochlist[2,5,1700,125,8],
```

```
    Makestochlist[3,11,1700,125,8]};
s4s16 := List @@
   { Makestochlist[1,5,1700,75,16], Makestochlist[2,5,1700,75,16],
     Makestochlist[3,11,1700,75,16]};
s4s32 := List @@
   { Makestochlist[1,5,1700,50,32], Makestochlist[2,5,1700,50,32],
     Makestochlist[3,11,1700,50,32]};
```

References

[1] Wolfram, S., 1992. *Mathematica*, **A System for doging mathematics by computer.**, Addison Wesley, New York, NY.

[2] B.A. Murtaugh and M.A. Saunders, 1983. "MINOS 5.1 User's Guide," Technical Report SOL-83-20R, Systems Optimization Laboratory, Stanford University, Stanford, California.

[3] International Business Machines Corp., 1991. "Optimization Subroutine Libary Guide and Reference, Release 2," document SC23-0519-02, International Business Machines Corp.

[4] Van Slyke R., and R. Wets, 1969. " L-shaped linear programs with applications to optimal control and stochastic programming," *SIAM Journal of Applied Mathematics*, 17, pp. 638-663.

[5] Ermoliev, Yu. and Wets, R. 1980. "Stochastic Programming, an Introduction" in *Numerical Techniques for Stochastic Optimization*, Y. Ermoliev and R. J.-B. Wets (eds.), Springer-Verlag, Berlin.

[6] Birge, J., 1985. "Decomposition and Partitioning Methods for Stochastic Linear Programs, " *Operations Research, 33*, 989-1007.

[7] Bazaraa, M. S., J. J. Jarvis, and H. D. Sherali, 1990. **Linear Programming and Network Flows**, Second Edition, Wiley and Sons, New York, NY.

[8] Charnes, A. and W. W. Cooper, 1962. "Programming with Linear Fractionals," *Naval Research Logistics Quarterly* bf 9, pp. 181-186.

[9] Bazaraa, M. S., and C. M. Shetty, 1979. **Nonlinear Programming**, Wiley and Sons, New York, NY.

[10] Kall, P., A. Ruszczyński, and K. Frauendorfer. 1980. "Approximation Techniques in Stochastic Programming" in *Numerical Techniques for Stochastic Optimization*, Y. Ermoliev and R. J.-B. Wets (eds.), Springer-Verlag, Berlin.

[11] Kall, P., A. Ruszczyński, and K. Frauendorfer. 1980. "Approximation Techniques in Stochastic Programming" in *Numerical Techniques for Stochastic Optimization*, Y. Ermoliev and R. J.-B. Wets (eds.), Springer-Verlag, Berlin.

[12] Birge, J. R. "Interior Point Notes," Class Materials, Industrial and Operations Engineering 510, Ann Arbor, MI 48109.

[13] Birge, J. and R. Wets. 1986. "Designing approximation schemes for stochastic optimization problems, in particular for stochastic programming problems with recourse." *Mathematical Programming Study*, 27 54-102.

[14] Gassman, H.I., 1990. " MSLip: A Computer Code for the Multistage Stochastic Linear Programming Problem, " *Mathematical Programming* 47, 407-423.

[15] Gassman, H.I., 1990. " MSLip User's Guide, " Working Paper WP-90-4, School of Business Administration, Dalhousie University, Halifax, Canada.

[16] Birge, J. R. and F. Louveuax, 1988. "A multicut algorithm for two-stage stochastic linear programs," *European Journal of Operations Research* 34, pp. 384-392.

[17] Donohue, C., 1993. Private communication, techincal report under preparation, University of Michigan, Ann Arbor, MI 48109.

[18] Edwards, J.. 1980. "A proposed Standard Input format for computer codes which solve stochastic programs wit recourse. " in *Numerical Techniques for Stochastic Optimization*, Y. Ermoliev and R. J.-B. Wets (eds.), Springer-Verlag, Berlin.