Modeling Languages in Computational Economics: GAMS

Stavros A. Zenios
Operations & Information Management
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104
and University of Cyprus
Nicosia, CYPRUS.

January 2, 1994

Abstract

This working paper is intended as a Chapter in the *Handbook of Computational Economics*, H.M. Amman, D. Kendrick and J. Rust (eds.), North-Holland. It gives an overview of the state-of-the-art in algebraic modeling languages. Such languages are in widespread use for the implementation of economic models, especially those that can be formulated as mathematical programs (e.g., linear programs, nonlinear equilibrium models, transportation models, models for estimating Social Accounting Matrices). Emphasis is placed on the description of GAMS of Brooke, Kendrick and Meeraus (1992) — a General Algebraic Modeling System — that originated at the World Bank. Example models are presented.

1 Introduction

Mathematical modeling and computer analysis is a cornerstone of computational economics. A wide range of economic problems can be represented by systems of equations or by optimization programs over systems of inequalities. Such models, and the underlying economic applications, are discussed in other Chapters of this Handbook on General Equilibrium Models, Game Theory, and Sectoral Models.

These mathematical models are used to represent real and observable systems. The models are, therefore, developed using economic observations (i.e., data). They are not merely abstract mathematical descriptions. The instanciation of an abstract mathematical model using economic data, and its solution on the computer, is facilitated using *high-level modeling languages*. This Chapter provides an introduction to one particular algebraic modeling language, GAMS of Brooke, Kendrick and Meearus (1992). It gives an overview of the language and illustrates its use in modeling a problem in transportation and a problem in the estimation of Social Accounting Matrices.

A significant part of the time required to develop a model involves data preparation and transformation, and report generation. The model is transformed from a form that is understandable to the modeler, to a form that is readable by the computer. Until the 1970's such transformations were handled by programs tailored to each specific application. Such programs, known as *matrix generators*, required several hours of programming time, were difficult to alter to accommodate changes in the model, were accessible only to the specialist who wrote them and not to the analyst who developed the model and, therefore, were difficult to debug and correct.

In the late 1970's the case was made that matrix generators should give way to *algebraic modeling languages*, Bisschop and Meeraus (1982) and Fourer (1983). A modeling language could integrate ideas from relational database theory with the rapidly expanding field of mathematical programming. Relational databases provide the framework for data organization, while mathematical programming provides a way for describing a variety of problems and offers algorithms for their solution. The first steps towards the development of an algebraic modeling language concentrated on linear programming problems, Bisschop and Meeraus (1982). Extensions where then made to handle nonlinear programs, Brooke, Drud and Meeraus (1984), integer programs, Brooke, Kendrick and Meeraus (1992), network problems, Zenios (1990), variational and complementarity problems, Rutherford (1992).

The first modeling language — GAMS, a General Algebraic Modeling System — was developed at the World Bank in the late 1970's. This system provides a high-level (algebraic) language for the representation of large and complex models. It allows for unambiguous statements of algebraic relations that define an abstract system of variables and equations. It also provides several mechanisms for data management. The system performs appropriate data transformations to create a specific instance of the model, starting from the abstract representation. Since the model description is algebraic the GAMS statement of the model provides a readable documentation. The data management mechanisms also facilitate the preparation of reports.

The use of an appropriate algorithm to solve the model is also handled by GAMS. From the user's perspective, the model is independent of the solution algorithm. This approach ensures portability. Only the GAMS statement of the model needs to be ported among different computer platforms, and the GAMS statements are machine independent. (Although a model statement in a later release of GAMS, like 2.25, may be using features that are not supported on a computer running an earlier release, like 2.05.) A GAMS model can be ported, today, to a wide range of computer platforms ranging from personal computers, mainframe and workstation systems, attached array processors and CRAY vector supercomputers.

GAMS is not the only language currently available, although it is the most widely used due to the experiences accumulated by the World Bank analysts. For other developments in modeling languages refer to Geoffrion (1987) and Fourer, Gay and Kernighan (1993).

2 Overview of the GAMS modeling language

A GAMS model is a collection of statements in the GAMS language. These statements define the variables of the model, specify the symbolic relationships between them in the form of equations, specify data structures and assign values to them, and instructs the computer to generate and solve the model. Other GAMS statements are used to handle output. In this section we describe the basic components of GAMS, without going into details on the precise syntax. Readers should get a general overview of the language and its capabilities. Detailed documentation is provided in the GAMS User's Guide, Brooke, Kendrick and Meeraus (1992). We use upper case, typewriter font for all expressions that are part of the GAMS language, such as `EQUATIONS` and `SOLVE`.

Data structures, data initialization and symbolic relationships are specified by writing GAMS *statements* on GAMS *symbols*. Symbols must first be declared as to type, before they can be used. Each symbol must be declared to belong to one of the following six classes:

```
SETS                           VARIABLES
PARAMETERS (SCALAR, TABLE)     EQUATIONS
ACRONYMS                       MODELS
```

Statements in GAMS are classified into one of two groups:

1. Declaration and definition statements.
2. Execution statements.

Declaration statements specify the class of a symbol, and a definition statement provides values for a declared symbol. For example, the GAMS statement:

```
    SETS J markets ;
```

specifies a symbol J as being a set, which is explained in the declaration statement to be the set of "markets". Elements of this set, i.e., actual markets, are defined in the following GAMS statement that combines the declaration of the symbol with its definition:

```
    SETS J markets /NEW-YORK, CHICAGO, TOPEKA /;
```

Declarations have some common characteristics, as illustrated in the following example:

`PARAMETER`	`A`	`(I,J)`	`Input-Output Matrix`
Symbol-class-keyword	Identifier	Domain (Sec. 2.1)	Text

Execution statements are instructions to carry out actions such as data transformation, model generation, model solution and preparation of reports. The execution statements are:

```
        OPTION      DISPLAY     ABORT
        ASSIGMENT   LOOP        SOLVE
```

2.1 Domains of parameters and variables

The sets in a GAMS model are used to specify the domains of parameters and variables. Equations are also defined over domains as discussed in sec. 2.6. For example,

```
    PARAMETER D(J) demand of a product at market J ;
```

declares a parameter over the set of markets J. For the set J defined above, this parameter definition is equivalent to the three parameters D("NEW-YORK"), D("CHICAGO"), D("TOPEKA").

Variables are also defined over sets, as in the example

```
VARIABLE X(I,J) ;
```

which defines a variable over the product space of sets I and J.

The domain sets can be used to manipulate and transform data by using indexed operators, as explained in section 2.3. In the context of declaration and definition of equations the domains are used to specify the symbolic relationships among variables.

2.2 The GAMS arithmetic operations

GAMS supports arithmetic expressions on parameters and variables. These expressions can be used either for data manipulation, or for defining symbolic relationships. The standard arithmetic expressions are

**	exponentiation
* /	multiplication and division
+ -	addition and subtraction

In addition, GAMS supports many commonly used standard functions such as exponentiation, logarithms, trigonometric functions, absolute value functions etc.

2.3 The GAMS summation and product operators

Algebraic manipulations of GAMS symbols are facilitated with the use of indexed operators such as SUM (for summation) and PROD (for product). The format of these operators is based on the idea that both operators have two arguments: The domain, that is the index set, over which the operator is executed and an operand. A simple example is:

```
SUM(J, X(I,J));
```

which is equivalent to the standard algebraic expression $\sum_{j \in J} x_{ij}$. Similarly

```
PROD ((I,J), X(I,J));
```

is equivalent to $\prod_{i \in I} \prod_{j \in J} x_{ij}$.

GAMS supports two additional indexed operators, SMAX and SMIN, that find the largest and smallest values over the domain of an indexed set. For example, SMAX(J,D(J)) finds the largest value of the parameter D(J).

2.4 Data entry and manipulations

GAMS allows three different formats for entering data:

1. Lists,
2. Tables,
3. Direct assignments.

Consider for example the PARAMETER D(J) for the demand of a product at each member of the market set J. Assume that demands are identical, let us say 300, at all markets. The simplest way to initialize the demand parameter is by the assignment statement

```
    D(J) = 300;
```

The following list format is equivalent to the assignment statement:

```
PARAMETERS D(J)     demand at market j in cases
                    /NEW-YORK 300
                    CHICAGO 300
                    TOPEKA 300 /;
```

The right-hand-side of the assignment statement does not have to be a number. It could be a GAMS expression that transforms other data structures. For example, the demand in every market region can be split over multiple customers. If the SET I defines the set of customers, and TABLE C-DEMAND(I,J) is the demand for each customer in each region, then the total demand in each region can be initialized by using the following assignment statement:

```
    D(J) = SUM(I, C-DEMAND(I,J)) ;
```

C-DEMAND(I,J) is an example of a two-dimensional table (or matrix). GAMS provides the TABLE format for initializing tables. The following statement illustrates the data initialization of the demand for two customers (BIG and SMALL) that are present in each one of the three markets in J.

```
    TABLE C-DEMAND (I,J) demand for each customer in each market

              NEW-YORK  CHICAGO  TOPEKA
     BIG           100      100     100
     SMALL         200      200    200;
```

GAMS allows the declaration and initialization of tables with more than two dimensions.

Data structures can be manipulated using standard arithmetic operations, indexed operations and functions. For example, the statement

```
D(J) = SUM(I, C-DEMAND(I,J)) ;
```

manipulates the two-dimensional table `C-DEMAND(I,J)` using the `SUM` indexed operator in order to initialize the parameter `D(J)`. Note that both the right and left expressions of the assignment statement are defined over the domain set `J`. GAMS produces an error statement if the domains of the two sides of an assignment statement are not consistent with each other.

A simple arithmetic operation `D(J) = 2 * D(J)` doubles the level of demand. Functions can also operate on GAMS data structures as in

```
LOGD(J) = LOG (D(J));
```

that transforms the demand values to logarithmic scale.

2.5 The GAMS relational operators

A relational operator allow the specification of relations between its left and right arguments. GAMS supports relational operators in two ways: in the definition of equations and in logical expressions.

In the definition of equations a relational operator is used to specify the type of the relationship. For example, `=E=` is used to define equality relationships, `=G=` is used to define greater-or-equal ($\geq$) inequalities, and `=L=` is used to define less-or-equal ($\leq$) inequalities.

In logical expressions the symbols `EQ`, `NE`, `LT` and so on are used to specify a required relationship between two values. These three symbols correspond to the relationships $=, \neq$ and $<$ respectively. GAMS also supports boolean relational operators (`NOT`, `AND`, `OR`, `XOR`) although it does not support a boolean data type. It follows the convention that the result of a relational operator is 0 if the assertion is false, and 1 if it is true. (Programmers familiar with the C programming language will notice the similarity).

2.6 Declaration and definition of equations

`EQUATIONS`, like all GAMS symbols, must first be declared before they can be defined and used. The declaration is a list of names (these are the names of the equations), each followed by a domain and by some explanatory text. We give two examples:

```
EQUATIONS    COST       Cost definition
             DEMAND(J)  Constraint on required demand in market J ;
```

The COST equation is a single equation, while for SUPPLY(J) we have one equation for each element of the set J. The *domain* of the demand equation is the set J. The above statements define two blocks of equations; the actual number of generated equations is equal to the cardinality of the set J, plus one more for the COST equation.

The next statement specifies the symbolic relationships that define the equations. First, we define the variables and some additional parameters that are needed in the specification of the equations. The definition of the equations follows. It starts with the equation identifier followed by .. , and then it gives the symbolic expression.

```
VARIABLES    TRCOST   Total transportation cost
             X(I,J)   Shipment from origin I to destination J ;
PARAMETERS   D(J)     Total demand at each market
             C(I,J)   Per unit transportation cost from I to J;

COST..        TRCOST =E= SUM((I,J), C(I,J)*X(I,J));
DEMAND(J)..   SUM(I,X(I,J)) =G= D(J);
```

2.7 Exception handling capabilities

The specification of complex relationships requires a mechanism for handling exceptions. One of the most powerful features of GAMS, in this respect, is the *Dollar ($) Operator*. This operator can be used in both arithemic expressions and in the definition of equations. Conceptually, the dollar operator is equivalent to an "IF" statement of programming languages. Its general structure is the following:

```
    A $ B.
```

It specifies that the expression A is evaluated (if it is a definition statement) or is executed (if it is an execution statement) IF expression B is true. We illustrate the use of this operator with two simple examples. Detailed explanations can be found in the GAMS User's Guide.

Consider the following:

```
    SCALAR X,Y;
    Y=2; X=1;
    X = 2$(Y GT 1.5);
```

This statement assigns the value 2 to X if Y is greater than 1.5.

The next example uses the dollar operator to control an indexed operation. Assume that we are given the demand parameter D(J), but for some markets in the set J the demand is unavailable and is assigned a value of -INF (i.e., -infinity). The total demand can be calculated by the following expression:

```
    PARAMETER TOTAL;
    TOTAL = SUM(J$( D(J) NE -INF), D(J));
```

2.8 GAMS solvers

While the GAMS language provides the flexibility for the specification of a wide variety of models, the GAMS system is interfaced with several optimization solvers that allow the solution of the models. The basic system is usually configured with two linear programming solvers (BDMLP and MINOS). GAMS/MINOS (Murtagh and Saunders (1977)) can also handle nonlinear programs. Other linear and nonlinear programming codes include GRG2 of Lasdon *et al.* (1978)) and CONOPT (Drud (1985)). Integer programs can be solved using GAMS/ZOOM (Singhal, Marsten and Morin (1989)). Network problems, linear and nonlinear, can be solved using GAMS/GENOS (Zenios (1990)). More specialized solvers are also available, like HERCULES of Drud and Kendrick (1986) for large, economywide models, GAMS/MATBAL of Zenios, Drud and Mulvey (1989) for solving matrix estimation problems, and GAMS/CPLIB of Dirkse et al. (1992) that allows the interfacing of GAMS with solvers for the mixed complementarity problem.

Most of these solvers are available on several machines. Those range from personal computers, to workstations, mainframes and vector supercomputers. More information on the availability of solvers is given in the GAMS manual.

2.9 The GAMS libraries of economic and financial models

It is often useful to build a model for an economic system by modifying the model of a closely related economy. This practice facilitates the development of the conceptual model, building on prior experiences by others. It also makes it easier to implement the actual model on the computer. The GAMS system includes a large library of 100 models, called GAMSLIB. Some of the models is the library are included to illustrate the capabilities of GAMS. Others are included because they represent classical and widely used models. Of particular interests to economists are the models on *agricultural economics* (those include several country-wide models for Pakistan, Egypt, Turkey, Brazil), *general equilibrium, economic development, energy economics* (including again country-wide models for Korea, Turkey, USA), as well as models in *micro-* and *macro- economics* and *econometrics*. Detailed information on GAMSLIB is given in Brooke, Kendrick and Meeraus (1992).

A library on financial models expressed in GAMS is also available. This library contains most of the standard optimization models from corporate finance (Markowitz's mean-variance models, portfolio dedication and portfolio immunization), as well as more specialized and complex models for structuring collateralized mortgage obligations (CMOs), term- structure estimation and so on. Detailed information on these models is given in Dahl, Meeraus and Zenios (1993).

3 Example Applications

A GAMS model typically consists of the following statements:

Specification of the data: This part of the statement does the following:
Declare and define sets
Declare and define parameters
Assign data to parameters
Display the values for inspection purposes.

Specification of the model: This part of the statement does the following:
Declare variables
Declare equations
Define equations
Define a model

Solution of the model: This part of the statement solves the model and displays results.

The next section illustrates the use of these basic GAMS features.

3.1 A simple transportation model

We consider as an example a simple transportation problem. In this problem we are given a set of production plants and a set of potential markets. Each plant has a given level of production, and each market has a known level of demand. The cost of shipping one unit of the product from the plants to the markets is also given.

A simple, algebraic, statement of the problem is the following: Let i and j be indices for the plants and markets respectively. Denote by s_i the supply of each plant, and by d_j the demand at each market. Let also x_{ij} denote the decision variables, indicating the amount supplied to market j by plant i, and let c_{ij} denote the cost of shipping one unit from i to j. The following linear program determines the least-cost solution:

Problem 3.1

$$\underset{x}{Minimize} \quad \sum_i \sum_j c_{ij} x_{ij} \tag{1}$$

$$s.t. \quad \sum_j x_{ij} \leq s_i \qquad \textit{for all } i \tag{2}$$

$$\sum_i x_{ij} \geq d_j \qquad \textit{for all } j \tag{3}$$

We consider now a specific instance of this problem, described in Dantzig (1963, ch. 3). In this example there are two plants and three markets. A complete GAMS statement of this model is given next. Lines starting with a * are comments. Note that the model is self explanatory.

```
* Declare two sets, and define their member elements.

SETS
        I      production plants        /SEATTLE, SAN-DIEGO/
        J      markets                  /NEW-YORK, CHICAGO, TOPEKA/;

* Declare the supply and demand parameters, and define their numerical values.
PARAMETERS
        S(I)   capacity of plant i in cases
               /SEATTLE 350
               SAN-DIEGO 600 /
        D(J)   demand at market j in cases
```

```
                    /NEW-YORK 325
                    CHICAGO 300
                    TOPEKA 275 /;

* Declare a table of distances from each plant to each market,
* and define the numerical entries of the table.
TABLE D(I,J) distance in thousands of miles


                    NEW-YORK   CHICAGO   TOPEKA
 SEATTLE                 2.5       1.7      1.8
 SAN-DIEGO               2.5       1.8      1.4

* Define scalar parameters, and perform some data transformations to
* convert the distance between each pair of plant/market to a monetary cost.


SCALAR          F           freight in dollars per case per thousands of miles /90/;
PARAMETER       C(I,J)      transport cost in thousands of dollars per case ;
                C(I,J) = F * D(I,J) / 1000 ;

* Declare the variables of the model
VARIABLES
                X(I,J)      shipment quantity (in cases) from plant i to market j
                Z           total transportation costs in thousands of dollars
POSITIVE VARIABLE X;

* Declare the equations, and specify the symbolic relationships that define
* each equation. There are three groups of equations. One equation is the
n
* objective function. A second group specifies constraints on the available
* supply at each plant. A third group specifies constraints on the required
* demand at each market.

EQUATIONS
                    COST define the objective function
                    SUPPLY(I) observe supply limit at plant i
                    DEMAND(J) satisfy demand at market j;
COST ..             Z =E= SUM((I,J), C(I,J) * X(I,J));
```

```
SUPPLY(I)..     SUM(J,X(I,J)) =L= S(I);
DEMAND(J)..     SUM(I,X(I<J)) =G= D(J);
```

* Define a model, called TRANSPORT, that contains all the
* equations declared and defined above.
```
MODEL TRANSPORT /ALL/;
```

* Solve the model, using a linear programming package.
```
SOLVE TRANSPORT USING LP MINIMIZING Z;
```

* Solve the model again, using a network optimization solver.
```
OPTION LP=GENOS;
SOLVE TRANSPORT USING LP MINIMIZING Z;
```

* Display the level (.L) and marginal values (.M) of the variables
```
DISPLAY X.L, X.M;
```

3.2 The SAMBAL system: estimating Social Accounting Matrices

We now describe a system, developed based on GAMS, to facilitate the representation and solution of matrix estimation problems. The GAMS/SAMBAL system is a custom-made template of the GAMS language that allows easy specification of models for estimating a Social Accounting Matrix. This system is specialized for the particular application, but the full set of GAMS features explained above remain available.

The matrix estimation problem is typically posed as follows:

> Given a rectangular matrix A, determine a matrix X that is *close to* A *and satisfies a given set of linear restrictions on its entries.*

A matrix that satisfies the linear restrictions is said to be *balanced.* For a survey of models and algorithms for this problem see Schneider and Zenios (1990). *In this section we are interested in the estimation of Social Accounting Matrices.*

A Social Accounting Matrix, or SAM, is a square matrix A whose entries represent the flow-of-funds between the national income accounts of a country's economy at a fixed point in time. Each index of a row or a column of A represents an account, or agent, in the economy. Entry a_{ij}

is positive if agent j receives funds from agent i. A SAM is a *snapshot* of the critical variables in a general equilibrium model describing the circular flow of financial transactions in an economy. For balancing problems arising from estimating SAMs, the linear restrictions are the *a priori* accounting identities that each agent's total expenditures and total receipts must be equal. That is, for each index i of the matrix A, the sum of the entries in row i must equal the sum of the entries in column i. The volume by Pyatt and Round (1985) provides an introduction to Social Accounting Matrices.

The agents of an economy in a simplified SAM include institutions, factors of production, households, and the rest-of-the-world (to account for transactions with the economies of other countries). Briefly, the production activities generate value-added which flows to the factors of production — land, labor, and capital. Factor income is the primary source of income for institutions — households, government and firms — who purchase goods and services supplied by productive activities, thereby completing the cycle. Of course, to be useful for equilibrium modeling, this highly aggregated model must be desegregated into subaccounts for each sector of the economy.

The compilation of a SAM is a difficult task due to data inconsistencies. Inconsistent data is an inherent problem when statistical methods are used to estimate underlying economic models. Morgenstern (1963) devoted his book to the problem of inconsistency in economic measurements. In particular, the direct estimate of a SAM is never balanced. The following quote of Sir Richard Stone (Van der Ploeg (1982, p.186) summarizes the sources of inconsistency in SAM modeling.

> " ... it is impossible to establish by direct estimation a system of national accounts free of statistical discrepancies, residual errors, unidentified items, balancing entries and the like since the information available is in some degree incomplete, inconsistent and unreliable. Accordingly, the task of measurement is not finished when the initial estimates have been made and remains incomplete until final estimates have been obtained which satisfy the constraints that hold between their true values. "

Therefore, the raw estimates of a SAM must be adjusted so that the consistency requirements are satisfied. This problem motivates much of the work on matrix balancing for economic modeling.

A matrix balancing problem also arises when *partial survey* methods are used to estimate a SAM. Frequently, estimates are available of the total expenditures and receipts for each agent in an economy but current data are not available for the individual transactions between the agents. If a complete (balanced) SAM is available from an earlier period, then the SAM must be updated to reflect the recent index totals. The problem is then to adjust the entries of the old matrix A so that the row and column totals equal the given fixed amounts. A similar balancing problem occurs when the entries of an input-output matrix must be updated to be consistent with exogenous estimates of the total levels of primary inputs and final demands.

The matrix balancing application we describe here can be formulated as follows:

Problem 3.2 *Given an $n \times n$ nonnegative matrix $A = (a_{ij})$, determine a "nearby" non-negative matrix $X = (x_{ij})$ (of the same dimensions) such that*

$$\sum_{j=1}^{n} x_{ij} = \sum_{j=1}^{n} x_{ji}, \quad i = 1, 2, \ldots, n, \tag{4}$$

and $x_{ij} > 0$ only if $a_{ij} > 0$.

In general, there are infinitely many matrices satisfying the consistency restrictions (4). For the problem to be well-posed the notion of a *nearby matrix* has to be defined. This notion is defined by some distance function $f(X; A)$ which measures the "distance" between X and A. The choice $f(X; A) \doteq \| X - A \|_F^2$, where $\| \cdot \|_F$ denotes the Frobenius norm, leads to a linearly constrained quadratic optimization problem. Another commonly used objective is the negative entropy functional:

$$\sum_{(i,j)} x_{ij} \left[\ln \left(\frac{x_{ij}}{a_{ij}} \right) - 1 \right]. \tag{5}$$

The GAMS/SAMBAL system specifies exactly how data structures for a matrix balancing model should be set up. A GAMS statement `ACRONYMS` is used to specify the distance functions. Additional data structures are provided to hold the data of the problem by using two sets of GAMS symbols. One set, prefixed with `T`, is used to provide information about the entries of the Social Accounting Matrix (e.g., initial values, upper/lower bounds on the estimated values, specification of the distance measure). The other set, prefixed by `Y`, is used to specify information about the row and column totals. Though problem 3.2 only specifies that row sums should be equal to column sums, it is also possible that some a priori target values are given for these totals. It is possible to specify a problem whereby row sums are required to be equal to column sums, while minimizing some distance from the prespecified values.

The following data structures are available for the specification of the Social Accounting Matrix.

`TINIT`	the initial values of the matrix
`TMAX`	upper bounds on the entries of the balanced matrix
`TMIN`	lower bounds on the entries of the balanced matrix
`TFUNC`	functional form of the distance function, chosen from the list of `ACRONYMS`
`TWEIGHT`	weighing coefficients of the penalty term for each entry of the matrix
`TBASE`	the balanced values of the matrix

The same data structures, prefixed with `Y` — `YINIT, YMAX, YMIN, YFUNC, YWEIGHT, YBASE` — are available to store information about the row and column totals. For example, `YINIT` specifies the initial, target, values for the row and column totals. Not all of the above data structures need to be provided for a well-specified model. Minimal data requirements include the specification of the initial matrix values `TINIT` and the functional from of the distance norms. Other information is incorporated in the model if it is provided.

Figure 1 illustrates the output of a simple model in GAMS/SAMBAL. This model estimates the entries of a 5×5 social accounting matrix. For three of these rows/columns we are given an estimate of the totals (see line 31), and a quadratic distance function is specified (line 49). For the remaining totals no prior information is given, and a residual distance function (i.e., a penalty identically equal to zero) is specified in line 50.

```
SET ACC ACCOUNTS  / LABOR, HOUSE1, HOUSE2, PROD1, PROD2 /
ALIAS(ACC,ACCP)

ACRONYMS  QUAD0  QUADRATIC PENALTY FUNCTION
          RESID  SAM VALUE IS RESIDUAL;

TABLE SAM0(ACC,ACC) INITIAL UNBALANCED SAM ESTIMATES
         LABOR  HOUSE1  HOUSE2  PROD1  PROD2
LABOR             15       3     130    80
HOUSE1
HOUSE2
PROD1             15     130            20
PROD2             25      40      55

TABLE SPEC(ACC,ACC)  FUNCTIONAL FORMS OF PENALTY FUNCTION
         LABOR  HOUSE1  HOUSE2  PROD1  PROD2
LABOR           QUAD0   QUAD0   QUAD0  QUAD0
HOUSE1   RESID
HOUSE2   RESID
PROD1           QUAD0  QUAD0           QUAD0
PROD2           QUAD0  QUAD0    QUAD0

TABLE VAR(ACC,ACCP) VARIANCE OF UNBALANCED ESTIMATES
         LABOR  HOUSE1  HOUSE2  PROD1  PROD2
LABOR           0.167   0.833  0.038  0.063
HOUSE1
HOUSE2
PROD1           0.167   0.019         0.071
PROD2           0.400   0.063  0.091

PARAMETER TOTAL(ACC)  ESTIMATES OF ACCOUNT TOTALS  /
                      LABOR = 220, PROD1 = 190, PROD2 = 105 /
  WEIGHT(ACC)  WEIGHTS OF ESTIMATED ACCOUNT TOTALS  /
               LABOR = 22, PROD1 = 38, PROD2 = 21 /;

SET ACCA(ACC) ACCOUNTS WITH ESTIMATES OF ACCOUNT TOTALS
    ACCN(ACC) ACCOUNTS WITHOUT ESTIMATES OF ACCOUNT TOTALS;
ACCA(ACC) = YES $ TOTAL(ACC);
ACCN(ACC) = YES ; ACCN(ACCA) = NO;

PARAMETER CT(ACC,ACC,*)  CELL DESCRIPTION TABLE
          AT(ACC,*)      ACCOUNT DESCRIPTION TABLE;

CT(ACC,ACCP,'TINIT')   = SAM0(ACC,ACCP);
CT(ACC,ACCP,'TFUNC')   = SPEC(ACC,ACCP);
CT(ACC,ACCP,'TWEIGHT') = VAR (ACC,ACCP);

AT(ACCA,'YINIT')   = TOTAL(ACCA);
AT(ACCA,'YFUNC')   = QUAD0;
AT(ACCN,'YFUNC')   = RESID;
AT(ACCA,'YWEIGHT') = WEIGHT(ACCA);

MODEL BALANCE / ACC, AT, CT /

SOLVE BALANCE USING SAMBAL;
```

Figure 1: The GAMS/SAMBAL statement of a simple model for estimating a Social Accounting Matrix

References

[1] J. Bisschop and A. Meeraus. On the development of a general algebraic modeling system in a strategic planning environment. *Mathematical Programming Study*, 20:1–29, 1982.

[2] A. Brooke, A. Drud, and A. Meeraus. High level modeling systems and nonlinear programming. In P.T. Boggs, R.H. Byrd, and R.B. Schnabel, editors, *Numerical Optimization 1984*. SIAM, 1984.

[3] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide, Release 2.25.* The Scientific Press, 1992.

[4] H. Dahl, A. Meeraus, and S.A. Zenios. Some financial optimization models: I. risk management. In S.A. Zenios, editor, *Financial Optimization*, pages 3–36. Cambridge University Press, 1993.

[5] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, 1963.

[6] S. Dirkse, M. Ferris, P.V. Preckel, and T. Rutherford. The GAMS callable program library for variation and complementarity solvers. Working paper, Computer Science Department, University of Wisconsin, Madison, WI, 1992.

[7] A. Drud. CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems. *Mathematical Programming,* 31:153–191, 1985.

[8] A. Drud and D. Kendrick. HERCULES: a system for large economy-wide models. Technical report, Development Research Department, The World Bank, 1986. Technical report.

[9] R. Fourer. Modeling languages versus matrix generators for linear programming. *ACM Transactions on Mathematical Software*, 9(2):143–183, June 1983.

[10] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* The Scientific Press, 1993.

[11] A. M. Geoffrion. Introduction to structured modeling. *Management Science*, 33(5):547–588, 1987.

[12] L.S. Lasdon, A.D. Waren, A. Jain, and M. Ratner. Design and testing of a generalized reduced gradient code for nonlinear programming. *ACM Transactions on Mathematical Software*, 4:34, 1978.

[13] O. Morgenstern. *On the accuracy of economic observations.* Princeton University Press, Princeton, New Jersey, 1963.

[14] F. Van Der Ploeg. Reliability and the adjustment of sequences of large economic accounting matrices. *Journal of the Royal Statistical Society*, 145:169–194, 1982.

[15] G. Pyatt and J. I. Round, editors. *Social Accounting Matrices: A Basis for Planning.* The World Bank, Washington, D.C., 1985.

[16] T. Rutherford. Extensions of GAMS for complementarity problems and variational inequalities with examples arising in economic equilibrium analysis. Working paper, Department of Economics, University of Western Ontario, 1992.

[17] M. H. Schneider and S. A. Zenios. A comparative study of algorithms for matrix balancing. *Operations Research*, 38:439–455, 1990.

[18] J. Singhal, R.E. Marsten, and T.L. Morin. Fixed order branch-and-bound methods for mixed-integer programming: the ZOOM system. *ORSA Journal on Computing*, 1(1):44–51, 1989.

[19] S. A. Zenios. Incorporating network optimization capabilities into a high-level programming language. *ACM Transactions on Mathematical Software*, 16:113–142, 1990.

[20] S.A. Zenios, A. Drud, and J.M. Mulvey. Balancing large social accounting matrices with nonlinear network programming. *Networks*, 17:569–585, 1989.

Mathematica for Economists

Hal R. Varian University of Michigan January 1994

Introduction

Mathematica is a computer program that can help you do mathematics. You can use it to do symbolic, numeric and graphical analysis. *Mathematica* is sold by Wolfram Research, Inc and runs on a variety of computers including MS-Windows, Macintosh, and Unix platforms. The cost of *Mathematica* depends on the version and the platform; it ranges from about $200 for a student version to several thousand for a multiple-user workstation version. You can contact Wolfram Research, Inc. by sending e-mail to **info@wri.com**, or calling them at 217-398-0700.

Design of *Mathematica*

There are two parts to the *Mathematica* program: the kernel and the front end. The kernel is the basic computational engine and is more-or-less platform independent; the front-end is slightly different for each platform. These two programs can be run separately: the front end can run on a lowly Macintosh while the kernel executes on a remote workstation or a supercomputer. This allows you to do your computations on whatever size computer you choose and still work in exactly the same user environment.

Wolfram Research has developed a set of protocols known as *MathLink* that allow the *Mathematica* kernel to communicate with other programs running on a given machine. This feature allows the user to combine functionality of various programs in a convenient way. For example, you can manipulate numbers in a spreadsheet and then send them to *Mathematica* for further processing. Or you can process *Mathematica* output in T_EX or some other formatting system. You can also send parts of *Mathematica* computations off to a special-purpose computer package such as IMSL or S. Various packages are available from third party suppliers that make this kind of inter-process communication very easy to implement.

You load packages into *Mathematica* using<< as in the following examples.

```
In[1]:=  <<Statistics`DescriptiveStatistics`
         <</Users/hal/Papers/Mathematica/nash.m
```

The front end

Mathematica keeps a record of a session in a format known as a Notebook. This is an ASCII file and is essentially machine independent. It allows the input and output of *Mathematica* (including graphical output) to be organized in a convenient way. A Notebook has an outline structure that allows parts of the session to be hidden or open as the user desires. A Notebook can serve as an "audit trail" to ensure that calculations or manipulations of data can be easily reproduced.

Notebooks can themselves be used as inputs to other programs. For example, WRI distributes a package that will convert Notebooks to T_EX format so that the material in the Notebook can be typeset. Several books have been produced using this technique. In fact, this article has been produced using this system.

■ Programming

Mathematica contains a complete programming language that can be used to automate various kinds of computations. The language is based on the philosophy of "functional programming." This means that the fundamental operation in the language is the application of a function.

Adherents of functional programming argue that it is a very efficient way to program. Functional programming allows you to build up small pieces of a program, interactively debug them, and string them together to achieve a desired end. Other functional languages are APL and Lisp. People who have used these languages will find *Mathematica* programming to be quite congenial.

Mathematica also has tools for procedural programming, which is the style of programming used in Fortran, Pascal and C. However, these tools---DO loops, WHILE loops, and the like---are normally not the best way to program in *Mathematica*. One advocate has gone so far as to proclaim "If you aren't programming functionally, you're programming disfunctionally!"

■ An example

The operation of the *Mathematica* programming language is best illustrated by an example.

Suppose that you want to compute the square root of 3 using Newton's method. The difference equation that you want to iterate is:

$$x_{t+1} = \frac{1}{2}\left(x_t + \frac{3}{x_t}\right).$$

In order to write a program to calculate this expression using C or Fortran you would need to declare the variables, construct a DO or a for loop, and output the results. In *Mathematica* you simply declare the function:

```
In[1]:=  newton[x_] := N[1/2 (x + 3/x)]
```

The expression on the left is the function declaration, x_ defines the dummy variable that will be the argument to the function, and N[ ] indicates that you want the expression inside the bracket to be converted to a real number.

Once the function has been defined, you then apply the built-in function **NestList** to calculate the first 5 terms starting from $x = 1$:

```
In[2]:=  NestList[newton,1.0,5]

Out[2]=  {1., 2., 1.75, 1.73214, 1.73205, 1.73205}
```

If you want to iterate until the result no longer changes, simply use the **FixedPoint** function.

```
In[3]:=  FixedPoint[newton,1.0]

Out[3]=  1.73205
```

■ Defining functions

Mathematica contains a number of functions that operate on lists of objects. You can also write your own functions.

For example, *Mathematica* contains a derivative function that will calculate the symbolic derivative of an expression:

```
In[1]:=  D[x^n,x]
Out[1]=     -1 + n
         n x
```

You can define a gradient and Hessian function as follows:

```
In[2]:=  Grad[f_,x_] := Map[D[f,#]&,x]
         Hessian[f_,x_] := Grad[Grad[f,x],x]
```

Here the # symbol is a placeholder that will take on the values in the list **x**. The definition of **Grad** "maps" the **D[f,#]** over the list **x={x1,x2}** to produce a new list **{D[f,x1],D[f,x2]}**. The definition of Hessian applies **Grad** to **Grad**. Here are some examples.

```
In[3]:=  Grad[x1^a x2^b, {x1,x2}]
Out[3]=       -1 + a    b         a   -1 + b
         {a x1        x2  , b x1  x2        }
```

```
In[4]:=  MatrixForm[Hessian[x1^a x2^b, {x1,x2}]]
Out[4]=                  -2 + a    b              -1 + a    -1 + b
         (-1 + a) a x1          x2         a b x1         x2

               -1 + a    -1 + b                          a   -2 + b
         a b x1        x2                (-1 + b) b x1  x2
```

Similarly the following definition will produce the first-order conditions for optimizing a function:

```
In[5]:=  FOC[f_,x_] := Map[(D[f,#]==0)&,x]
```

```
In[6]:=  FOC[x1^a x2^b,{x1,x2}]
Out[6]=       -1 + a    b                 a   -1 + b
         {a x1        x2   == 0, b x1  x2          == 0}
```

Programming constructs

Mathematica contains a number of programming constructs for iterating, branching, etc. However, in general it is best to avoid iteration and indices if possible. Often there is a built in function that will do some particular sort of manipulation of a list of values. For example, recently I had a list of price vectors, (p^t) and associated consumption bundles (x^t) for $t = 1, \ldots, T$. I wanted to calculate the matrix $(p^s x^t)$ for $t, s = 1, \ldots, T$ for some revealed preference calculations. This is simple to do using iteration and indices of course, but that is quite inelegant. After a short search through the *Mathematica* book and a bit of experimentation I came up with the following solution that uses *Mathematica*'s generalized inner product function.

```
In[1]:=   vMatrix[p_,x_] := Inner[Times,p,Transpose[x],Plus]
```

To verify that this works, define some vectors and apply the function:

```
In[2]:=  p={{p11,p12},{p21,p22},{p31,p32}};
         x={{x11,x12},{x21,x22},{x31,x32}};
```

```
In[3]:=  MatrixForm[vMatrix[p,x]]
Out[3]=  p11 x11 + p12 x12   p11 x21 + p12 x22   p11 x31 + p12 x32

         p21 x11 + p22 x12   p21 x21 + p22 x22   p21 x31 + p22 x32

         p31 x11 + p32 x12   p31 x21 + p32 x22   p31 x31 + p32 x32
```

This example illustrates a nice point about *Mathematica* programming: if you know the formula for what you want, you can apply your function to symbolic values to see if it produces the right thing. Once it does, you can switch to numbers.

■ Pattern matching

At the most fundamental level, *Mathematica* operates by replacing patterns of expressions with other expressions. This means that *Mathematica* has a sophisticated, built-in pattern matching engine. This pattern matching facility is also available to the user.

For example, economists often want to solve systems of equations that have a "Cobb-Douglas" structure:

$$x_1^{a_{11}} x_2^{a_{12}} = b_1$$
$$x_1^{a_{21}} x_2^{a_{22}} = b_2.$$

An easy way to do this is to take a log transform to construct the linear system

$$a_{11}\log x_1 + a_{12}\log x_2 = \log b_1$$
$$a_{21}\log x_1 + a_{22}\log x_2 = \log b_2.$$

Mathematica won't do this kind of transformation automatically, nor should it: this particular transformation is only valid for positive real numbers. On the other hand, it would be nice to automate this sort of thing. Here's how to do this in *Mathematica*.

First we define the rules that translate the pattern $x^a y^b = c$ into $a\log x + b\log y = \log c$ and the reverse transformation. (The semicolon at the end of the expression inhibits the output which, in this case, is uninteresting.)

```
In[1]:=  Clear[x]


In[2]:=  logRules={x_^a_ y_^b_ == c_ -> a Log[x] + b Log[y]==Log[c]};


In[3]:=  eRules={(Log[x_] -> (a_*Log[b_] + c_*Log[d_])/e_) ->
           (x -> b^(a/e) c^(d/e))};
```

Now let's apply this to solving the following system of equations.

```
In[4]:=  eqns={x1^a11 x2^a21 == b1,x1^a12 x2^a22 == b2}
Out[4]=     a11   a21             a12   a22
         {x1    x2    == b1, x1     x2    == b2}


In[5]:=  logEqns=eqns/.logRules
```

```
Out[5]=  {a11 Log[x1] + a21 Log[x2] == Log[b1],

           a12 Log[x1] + a22 Log[x2] == Log[b2]}

In[6]:=  ans=Simplify[Solve[logEqns,{Log[x1],Log[x2]}]]
Out[6]=                  -(a22 Log[b1]) + a21 Log[b2]
         {{Log[x1] -> ----------------------------,
                             a12 a21 - a11 a22

                          -(a12 Log[b1]) + a11 Log[b2]
            Log[x2] -> -----------------------------}}
                            -(a12 a21) + a11 a22


In[7]:=  ans/.eRules
Out[7]=                b2/(a12 a21 - a11 a22)
                    a21
         {{x1 -> -------------------------,
                     a22/(a12 a21 - a11 a22)
                    b1

                     b2/(-(a12 a21) + a11 a22)
                     a11
            x2 -> ----------------------------}}
                     a12/(-(a12 a21) + a11 a22)
                     b1
```

This particular set of rules is pretty minimal. A really useful set of rules for doing and undoing log transforms should be more sophisticated. Nevertheless, this example illustrates some of the power of the pattern-matching capabilities.

■ Packages

Sets of *Mathematica* commands can be collected together into Packages. These are plain ASCII files that can be input into other *Mathematica* programs to do specific calculations. The *Mathematica* distribution comes with a number of Packages designed for specific sorts of calculations such as combinatorics, linear algebra, statistics, and so on. Many authors have produced Packages and Notebooks that for various uses that they have made available to other users through articles, books and on-line systems.

■ MathSource

Wolfram Research maintains a repository of contributed *Mathematica* materials that are available via e-mail and ftp. This repository is known as MathSource. The easiest way to start using MathSource is to send e-mail to **mathsource@wri.com** that contains the message `help intro`. MathSource will return some documents that explain how to retrieve files.

■ Books, magazines and newsletters

The standard reference book for *Mathematica* is written by Stephen Wolfram and titled *Mathematica*. The ISBN number is 0-201-51502-4. The basic journal is *The Mathematica Journal*, located at 600 Harrison Street, San Francisco, CA 94107. There is also a nice newsletter called *Mathematica in Education* available from the Department of Mathematics, Sonoma State University, 1801 East Cotati Avenue, Rohner Park, CA 94928. Finally, Variable Symbols, 2161 Shattuck Avenue, Berkeley, CA 94705-1313 publishes some useful *Mathematica* materials.

■ Applications in economics

In the following sections I describe some applications of *Mathematica* in economics. Obviously the list is not complete, but I hope to give the reader some idea of potential uses. Many further examples can be found in Varian (1993).

■ Comparative statics

Economists spend a lot of time analyzing optimization problems using the techniques of comparative statics. Although these computations are very simple, they can be quite tedious. *Mathematica* can help to automate this process.

For example, here is a calculation that derives the comparative statics for a profit maximizing firm with two inputs. First we define the objective function:

```
In[1]:= profit = f[x1,x2] - w1 x1 - w2 x2

Out[1]= -(w1 x1) - w2 x2 + f[x1, x2]
```

Next we calculate the first order conditions for profit maximization and the Hessian using the functions that we have defined earlier.

```
In[2]:= focs=FOC[profit,{x1,x2}]
Out[2]=
                 (1,0)                     (0,1)
         {-w1 + f      [x1, x2] == 0, -w2 + f      [x1, x2] == 0}


In[3]:= Hess = Hessian[profit,{x1,x2}]
Out[3]=     (2,0)           (1,1)
         {{f      [x1, x2], f      [x1, x2]},

            (1,1)           (0,2)
          {f      [x1, x2], f      [x1, x2]}}
```

Note *Mathematica*'s notation for derivatives: $f^{(i,j)}$ is the ith derivative of argument 1 and the j^{th} derivative of argument 2.

Next we totally differentiate the first-order conditions.

```
In[4]:= totalDerivative = Dt[focs]
Out[4]=                           (1,1)
         {-Dt[w1] + Dt[x2] f      [x1, x2] +

                     (2,0)
            Dt[x1] f      [x1, x2] == 0,

                                  (0,2)
          -Dt[w2] + Dt[x2] f      [x1, x2] +

                     (1,1)
            Dt[x1] f      [x1, x2] == 0}
```

Note that *Mathematica* uses the notation of Dt[x1] for the differential element dx_1. Now we simply solve the system of equations and substitute out for the determinate of the Hessian:

```
In[5]:=  Simplify[Solve[totalDerivative,{Dt[x1],
             Dt[x2]}]]/.{Det[Hess]->dHess}

Out[5]=                  (0,2)                          (1,1)
                 Dt[w1] f     [x1, x2] - Dt[w2] f      [x1, x2]
         {{Dt[x1] -> ----------------------------------------------,
                                   dHess

            Dt[x2] ->

                       (1,1)                       (2,0)
            -(Dt[w1] f      [x1, x2]) + Dt[w2] f      [x1, x2]
            ------------------------------------------------}}
                                 dHess
```

In standard notation, these expressions say

$$dx_1 = \frac{f_{22}dw_1 + f_{12}dw_2}{dHess}$$
$$dx_2 = \frac{f_{11}dw_2 + f_{21}dw_1}{dHess}$$

These conditions contain all of the normal comparative statics conclusions about cost minimization.

■ Dynamic programming

Dynamic programming is another calculation that is straightforward but tedious. Consider, for example, the problem of allocating consumption over time. The optimal solution can be characterized through the use of the Bellman value function:

$$V_t(w) = \max_c \; u(c) + \alpha V_{t+1}((w-c)r).$$

For certain classes of $u(c)$ it is possible to find closed-form solutions for $V_t(w)$. Solving the Bellman recursion numerically *or symbolically* is simple using *Mathematica*. For example, here is how you would write the Bellman equation for the case of log utility and a five-period time horizon:

```
In[1]:=  V[w_,5] := Log[w]
         V[w_,t_]:=
          Module[{c},
          Log[c] + alpha*V[(w-c)*R,t+1]/.Solve[D[Log[c]
                 + alpha*V[(w-c)*R,t+1],c]==0,c][[1]]]
```

The first definition gives the boundary condition. The second definition gives the recursion.

The Module construction declares c to be a local variable. Subsequently, we have the recursive definition of the value function; the notation a/.b means "substitute b into a." In this case b contains the optimal solution and a is the objective function.

Calculating $V(w, 2)$, for example, gives us:

```
In[2]:=  Simplify[V[w,2]]
```

```
Out[2]=                     w
        Log[----------------------------] +
                             2        3
            1 + alpha + alpha  + alpha

                         alpha R w
          alpha Log[----------------------------] +
                                 2        3
                    1 + alpha + alpha  + alpha

                              2  2
               2         alpha  R  w
          alpha  Log[----------------------------] +
                                  2        3
                     1 + alpha + alpha  + alpha

                              3  3
               3         alpha  R  w
          alpha  Log[----------------------------]
                                  2        3
                     1 + alpha + alpha  + alpha
```

The optimal consumption in period 2 is given by

```
In[3]:=  Solve[D[Log[c] + alpha*V[(w-c)*r,3],c]==0,c][[1]]
Out[3]=                      w
         {c -> ----------------------------}
                                 2        3
                1 + alpha + alpha  + alpha
```

Nash equilibria

In a two-person game with a finite number of strategies, calculating all Nash equilibria is a straightforward but tedious enumeration of Kuhn-Tucker conditions. Dickhaut and Kaplan (1993) have written up a *Mathematica* package that automates this calculation. For example, here are all Nash equilibria in the Battle of the Sexes.

```
In[1]:=  Nash[{{{2,1},{0,0}},{{0,0},{1,2}}}]
Out[1]=                            2  1    1  2
         {{{0, 1}, {0, 1}}, {{-, -}, {-, -}}, {{1, 0}, {1, 0}}}
                            3  3    3  3
```

Econometrics and statistics

One of the most promising and comparatively underexploited areas for applications of *Mathematica* is in econometrics and statistics. *Mathematica* can serve as a computational engine for special-purpose calculations, as a symbolic engine for deriving expressions , and as a tool for data analysis.

Symbolic expressions

Mathematica can simplify various statistical calculations. As an example, let us define a Normal distribution:

```
In[1]:=  NormalDistn[x_,m_,s_] :=
             Exp[-((x-m)/s)^2 / 2]/(s Sqrt[2*Pi])
```

Consider the problem of choosing a forecast y so as to minimize some expected loss involving x and y. The LINEX loss function (see Varian(1975) and Zellner (1986)) has the form:

```
In[2]:=  LinexLoss[a_,x_,y_] := Exp[a*(y-x)] - a*(y-x)
```

We are interested in the expected loss. The easy way to calculate this is to recognize that the first term is just the moment generating function for the Normal distribution. But if we don't recognize this, we can easily calculate the expected loss between $-\infty$ and $+\infty$ using *Mathematica*:

```
In[3]:=  ExpectedLoss1=Together[PowerExpand[
             Integrate[LinexLoss[a,x,y]*
            NormalDistn[x,m,s],{x,-Infinity,Infinity}]]]

Out[3]=                         2
            (a (-2 m + a s  + 2 y))/2
         (2 E                          + 2 a m - 2 a y -

                                                 m
                                            (a - --) s
                            2                     2
           (a (-2 m + a s  + 2 y))/2              s
          E                            Erf[----------] -
                                             Sqrt[2]

                                                 m
                                            (-a + --) s
                            2                      2
           (a (-2 m + a s  + 2 y))/2               s
          E                            Erf[-----------]) / 2
                                              Sqrt[2]
```

In *Mathematica* notation, $\mathrm{Erf}[z] = \frac{1}{\sqrt{\pi}} \int_0^z e^{-t^2}\, dt$. From the definition it is easy to see that $\mathrm{Erf}[-x] = -\mathrm{Erf}[x]$. In order to get Mathematica to make this substitution, we use the function **ExpandAll**.

```
In[4]:=  ExpectedLoss3=ExpandAll[ExpectedLoss1]

Out[4]=                  2  2
          -(a m) + (a  s )/2 + a y
         E                           + a m - a y
```

Finally we solve for the loss-minimizing estimate and use **Simplify** and **PowerExpand** to put it into a simple form.

```
In[5]:=  ans=Solve[D[ExpectedLoss3,y]==0,y][[1]]

Out[5]=                   2  2          2  2
                   a m - a  s          a  s
              Log[E            Sqrt[E      ]]
         {y -> ------------------------------}
                             a

In[6]:=  Simplify[PowerExpand[ans]]

Out[6]=               2
                   a s
         {y -> m - ----}
                    2
```

The loss minimizing estimator is the mean, biased downwards by an amount that depends on the variance of the distribution.

Special purpose calculations

The bootstrap is a well-known tool for estimating the sampling distribution of an estimator. Implementing this in *Mathematica* is trivial. Here is a function that resamples from a list.

```
In[1]:=  Resample[list_] := list[[
                   Table[Random[Integer,{1,Length[list]}],
                   {Length[list]}]]]
```

To test this we apply it to a symbolic list:

```
In[2]:=  Resample[{a,b,c,d,e,f,g,h}]

Out[2]=  {b, f, b, d, a, f, h, d}


In[3]:=  Resample[{a,b,c,d,e,f,g,h}]

Out[3]=  {e, f, c, c, e, d, h, h}
```

Now I generate a sample of 25 random numbers; the semicolon tells *Mathematica* not to print them out.

```
In[4]:=  theSample=Table[Random[],{25}];
```

Here I resample from the list 10 times and compute the mean of each resample.

```
In[5]:=  bootList=Table[Mean[Resample[theSample]],{10}]
Out[5]=  {0.529655, 0.473363, 0.62481, 0.586473, 0.543386,

           0.547182, 0.387835, 0.464449, 0.593265, 0.633631}
```

We can write a function that will take a sample and a statistic and apply this statistic to resampled draws from the original sample.

```
In[6]:=  bootIt[sample_,n_,stat_] := Table[stat[Resample[sample]],{n}]


In[7]:=  bootIt[theSample,10,Median]
Out[7]=  {0.520299, 0.482829, 0.398446, 0.526447, 0.482829,

           0.520299, 0.398446, 0.392464, 0.445672, 0.526447}
```

Data analysis

Mathematica comes with a number of standard statistical routines. Belsely (1993) has provided a number of additional routines for classical econometric calculations. Ley and Steel (1993) have provided routines for Bayesian calculations. Stine (1993) describes some methods for time series analysis.

It is also easy to write your own routines. For example, I recently used Mathematica to calculate "efficiency indices." Suppose that you have a set of factor prices, factor choices, and output levels for n firms denoted by (w_i, x_i, y_i). The efficiency index of firm i is given by

$$e_i = \min_{y_j \geq y_i} \frac{w_i x_j}{w_i x_i}.$$

Here we look at each firm that produces at least as much output as firm i to see if it's production plan would cost less than firm i's plan assuming both firms faced factor prices w_i. Furthermore, I wanted to keep track of all the elements over which the minimization was taken so I could see how *many* firms appeared to be more efficient than the firm in question.

This is not a difficult calculation, but there is no ready-made package to do it.

In *Mathematica* all I had to do was to write:

```
In[1]:=  efficOf[i_,j_,wt_,xt_] := (wt[[j]].xt[[i]])/(wt[[j]].xt[[j]])

         eff[ws_,xs_,y_] :=
                Map[Union,
                  Table[If[y[[i]] <= y[[j]],Min[efficOf[j,i,ws,xs],1],1],
                  {i,1,Length[y]},{j,1,Length[y]}]]


In[2]:=  effList=eff[w,x,y];


In[3]:=  minEff=Map[Min,effList]
```

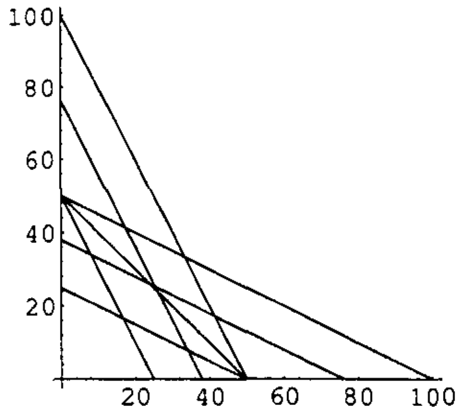
The first expression simply calculates the cost ratio. The second expression compiles a list of all the efficiencies that are less than 1 for each firm. The third expression actually does the calculations, and the fourth expression calculates the minimum efficiency. Once I had these efficiencies it was easy to look at a histogram, see how they changed when different inputs were used, and so on. Because the calculations were so flexible it was much easier to experiment than it would be if I had used a Fortran or C program to do this.

■ Graphics

One of the most useful things that *Mathematica* can do is to produce plots. Here are a few economics graphs.

Revealed preference

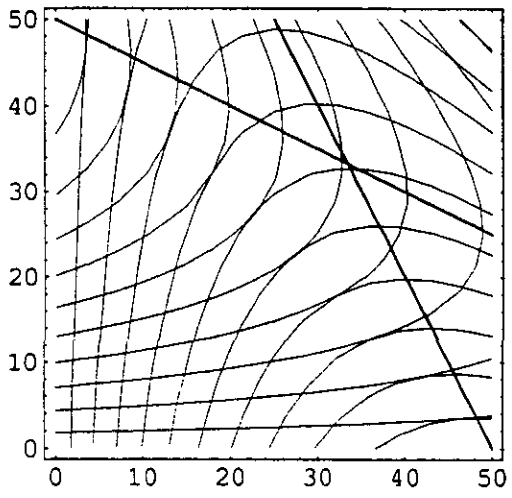
This is the output of a function that takes as input a list of price-quantity pairs, plots them, and then highlights the observations that violate the Weak Axiom of Revealed Preference.



Cournot equilibrium

Here are the commands to generate the isoprofit lines and reaction curves depicting a Cournot equilibrium. Although I've chosen a very simple example, *Mathematica* has no problem dealing with quite complicated profit functions---including ones with kinks and discontinuities.

```
In[1]:=  Profit1[x1_,x2_] := (100-x1-x2)*x1
         Profit2[x1_,x2_] := (100-x1-x2)*x2

In[2]:=  pr1=ContourPlot[Profit1[x1,x2],{x1,0,50},{x2,0,50},
         ContourShading->False,DisplayFunction->Identity]

Out[2]=  -ContourGraphics-

In[3]:=  pr2=ContourPlot[Profit2[x1,x2],{x1,0,50},{x2,0,50},
         ContourShading->False,DisplayFunction->Identity]

Out[3]=  -ContourGraphics-

In[4]:=  Solve[D[Profit1[x1,x2],x1]==0,x1][[1]]

Out[4]=           100 - x2
         {x1 -> --------}
                     2

In[5]:=  r1=ParametricPlot[{(100-x2)/2,x2},{x2,0,50},
            DisplayFunction->Identity]
         r2=ParametricPlot[{x1,(100-x1)/2},{x1,0,50},
           DisplayFunction->Identity]

In[6]:=  Show[{pr1,pr2,r1,r2},DisplayFunction->$DisplayFunction]
```



```
Out[6]=  -Graphics-
```

■ Teaching

file: math.tex: processed 1/12 at 10:48 a.m.

I have used *Mathematica* to prepare problem sets for both graduate and undergraduate courses. It makes it easy to get the graphs right and to make sure that the calculations come out in round and/or realistic numbers. Lately I have realized that it is silly to compose problems on *Mathematica* and have the students do them by hand: they should have access to the same kinds of tools the professor has access to. In the near future I hope to have some self-contained economic exercises and examples available in *Mathematica*.

I have prepared a set of Notebooks that go through some of the calculations used in my textbook *Microeconomic Analysis*. These are available via MathSource as items 0202-419. I have also experimented with a number of undergraduate exercises. Since many universities are now introducing students to *Mathematica* and other symbolic algebra systems in calculus courses, it should be easy to use these tools in more advanced undergraduate economics courses.

■ References

Belsely, David (1993) ''Econometrics.m: A Package for Doing Econometrics in *Mathematica*,'' in in Varian (1993).

Dickhaut, John and Kaplan, Todd (1993) ''A Program for Finding Nash Equilibria,'' in Varian (1993).

Ley, Eduardo and Mark F. J. Steel (1993), ''Bayesian Econometrics: Conjugate Analysis and Rejection Sampling,'' in Varian (1993).

Stine, Robert A. (1993) ''Time Series Models and Mathematica," in Varian (1993).

Varian, Hal (1974) ''A Bayesian Approach to Real Estate Assessment,'' in S. E. Fienberg and A. Zellner, Studies in Bayesian Econometrics and Statistics, North-Holland Press, Amsterdam.

Varian, Hal (1993) ed. Economic and Financial Modeling with Mathematica, TELOS/Springer-Verlag, New York, 1993.

Zellner, A. (1986)