MINOPT:

A Modeling Language and Algorithmic Framework for Linear, Mixed-Integer, Nonlinear, Dynamic, and Mixed-Integer Nonlinear Optimization

Version 3.1

October 15, 1998

C. A. Schweiger and C. A. Floudas
Department of Chemical Engineering
Princeton University
Princeton, NJ 08544-5263

Contents

1 Introduction

MINOPT is a comprehensive, powerful, and flexible package for the solution of various types of optimization problems. It features both an *advanced modeling language* for the clear and concise representation of complex mathematical models as well as a *robust algorithmic framework* for the efficient solution of wide variety of mathematical programming problems.

MINOPT is capable of handling a wide variety of models described by the types of variables and mathematical relationships employed in the model. MINOPT handles the following variable types:

- continuous time invariant
- continuous dynamic
- integer

and recognizes the following constraint types:

- linear
- nonlinear
- dynamic
- dynamic path
- dynamic point

MINOPT is capable of handling a wide variety of the following model types:

- Linear Programs (LP)
- Nonlinear Programs (NLP)
- Mixed Integer Linear Programs (MILP)
- Mixed Integer Nonlinear Programs (MINLP)
- Nonlinear Programs with Differential and Algebraic Constraints (NLP/DAE)
- Mixed Integer Nonlinear Programs with Differential and Algebraic Constraints (MINLP/DAE)
- Optimal Control Problems (OCP)
- Mixed Integer Optimal Control Problems (MIOCP)

The MINOPT modeling language allows for the natural representation of mathematical models using an advance modeling architecture. Large, complex models can be expressed in a concise, compact, and understandable form. Since the models are easy to understand, the can be easily debugged, modified, and maintained. The MINOPT modeling language has some important key features:

- Clear and concise representation of complex mathematical models
- Representation of both *Algebraic* and *Dynamic* models
- Support for a broad variety of natural mathematical expressions
- Capability to add, change, or delete the sets, variables, data, and constraints easily
- Capability to accept model information and data provided in separate input files
- Connection to *Chemkin* for kinetic modeling
- Checks of model syntax and consistency

The algorithmic framework withint MINOPT is the channel through which the various models are solved. The model definitions from the modeling language are used to set up the problems and subproblems required to solve the model. MINOPT uses available solvers for the solution of these problems. The MINOPT algorithmic framework also has a number of advantages:

- Efficient solution for *Mixed-Integer Nonlinear Programming* problems
- Efficient solution for problems with *dynamic models*
- Efficient *Integration and Sensitivity Analysis*
- Ability to switch easily among various solvers
- Ability to fine tune the solution algorithms with an extensive list of options

MINOPT has connections to a number of solvers and is able to exploit the features, options, and efficiency of the solvers. MINOPT currently has connections to the following solvers:

Solver	Model Types
CPLEX	LP, MILP
LPSOLVE	LP, MILP
MINOS	LP, NLP
NPSOL	LP, NLP
SNOPT	LP, NLP
DASOLV	Dynamic
DAESSA	Dynamic

MINOPT also provides algorithms for the solution of MINLPs

- Generalized Benders Decomposition (GBD)
- Outer Approximation and variants (OA, OA/ER, OA/ER/AP)
- Generalized Cross Decomposition (GCD)

2 Installation

The MINOPT software, MINOPT model library, and additional information can be found at the MINOPT website at `http://titan.princeton.edu/MINOPT`.

2.1 Platforms

MINOPT is currently ported to the following platforms:

Architecture	Operating System
Sun	SunOS 5.5.1
HP	HP-UX 10.20
IBM	AIX 3.2
SGI	IRIX 5.3
Intel	Windows 95/NT (Available soon)

2.2 System Requirements

In order to install MINOPT on a UNIX system, the following are required:

- C compiler
- The make utility
- The libraries for the solvers you wish to use with MINOPT

2.3 Compiling MINOPT

To compile MINOPT on a UNIX system, follow the following steps:

1. Obtain the MINOPT tar file for the appropriate architecture from the MINOPT website. The architectures and the corresponding tar files are the following:

Computer	OS	tar file
HP 9000 series	HP-UX 10.20	minopt_hppa.tar.gz
SGI	IRIX 5.3	minopt_iris4d.tar.gz
SGI	IRIX 6.4	minopt_iris4d.64.tar.gz
IBM rs6000	AIX 3.2	minopt_rs6000.tar.gz
SUN	SunOS 5.5.1	minopt_sun4.tar.gz

2. Uncompress the tar file using gunzip:
   `%gunzip minopt_hppa.tar.gz`

3. Extract the tar file using the tar command:
   `%tar xvf minopt_hppa.tar`
   The will create the directory `minopt_hppa` and extract the MINOPT into this directory. The new directory will contain the following:

   - COPYRIGHT

- Makefile.dist
- README.install (this file)
- README.license
- README.stanford
- configure
- include/
- lib/
- src/

4. Run the configuration script to create the Makefile for MINOPT:
   `% ./configure`
   This will ask you a series of questions regarding the computer as well as which external solvers you have present and wish to include with your MINOPT executable. When the configuration is complete, a Makefile for compiling MINOPT will be created. (For more information about the solvers currently linked to MINOPT and how to obtain them, consult the MINOPT website.)

5. After the configuration is complete and the Makefile has been created. Use the make command to compile the program:
   `% make` This will create the single MINOPT executable.

2.4 Installing MINOPT

The MINOPT program is a single executable file and all that is needed to run MINOPT is this file. To install the program, copy the executable to an appropriate place on the computer. (Somewhere like `/usr/local/bin` is usually a good choice.) The sample input files in the MINOPT model library are not required, but are recommended. They can be obtained from the MINOPT website.

Now MINOPT is ready to be run. To test that it is compiled and installed correctly, run the program: You should get the header message and copyright information.

```
M I N O P T
Version 3.1, Sep 10 1998
C. A. Schweiger and C. A. Floudas
Department of Chemical Engineering
Princeton University
Princeton, NJ 08544-5263

Copyright (c) 1998 Princeton University
All Rights Reserved

Send bugs, comments, and suggestions to minopt@titan.princeton.edu
```

```
Usage: minopt [ options... ] data file

Type minopt -h for a list of options.
```

2.5 Licensing

You can run MINOPT, but you will not get very far without a valid license. Purchasing and licensing of MINOPT is handled through Princeton University. For information about purchasing and licensing, contact

Jean A. Mahoney `<jean@princeton.edu>`
Director, Copyright and Trademark Licensing
Office of Technology and Trademark Licensing
Princeton University
P.O.36; 5 New South Building
Princeton, NJ 08544
Phone 609-258-3097 Fax 609-258-1159

Once MINOPT has been purchased, you will need to obtain a license key. To get the license key, send email to minopt@titan.princeton.edu. In the email, provide the following information:

- Your Name
- Your Company/Institution/University
- Solvers to be licensed (GBD, OA, DAESSA)
- Number of licenses
- Duration of the license
- Hostname of the computer used to run MINOPT (output of `uname -n`)

After you have provided this information, you will receive a license key for MINOPT for the computer you have identified. The MINOPT license key is a 64 character string which needs to be placed in a file named `.minoptlicense`. This file can be located one of three places:

- the user's home directory (the directory which is in the $HOME environment variable).
- The directory specified by the $MINOPTDIR environment variable
- The directory `/usr/local/etc`

Note that multiple licenses can exist in a single license file. This allows for one license file to be used for several machines. Also note that any line in the license file that is not 64 characters in length is considered a comment and is ignored.

With a valid license key in an appropriate file, MINOPT is now ready to be used.

Table 1: Files used by MINOPT

filename	contents
`<input>.dat`	Problem information
`<input>.log`	Solution progress and information
`<input>.dump`	Explicit problem output
`<input>.pricpx`	primal LP output from CPLEX
`<input>.gbdmascpx`	**GBD** master output from CPLEX
`<input>.oamascpx`	**OA/ER** master output from CPLEX
`<input>.apmascpx`	**OA/ER/AP** master output from CPLEX
`<input>.prilps`	primal LP output from LPSOLVE
`<input>.gbdmaslps`	**GBD** master output from LPSOLVE
`<input>.oamaslps`	**OA/ER** master output from LPSOLVE
`<input>.apmaslps`	**OA/ER/AP** master output from LPSOLVE
`<input>.states1`†	dynamic states for the solution of a dynamic problem

†Note that the state file contains the time as the first column and the values for the first 50 z variables in the following columns. If there are more that 50 z variables, additional states files are used with increasing numbers (`<input>.states2`, `<input>.states3`, etc.).

3 Usage

MINOPT is a standalone executable and is invoked from the command line:

```
prompt> minopt [options ...] <input>.dat [options ...]
```

The file `<input>.dat` is the text input file which contains the model written in the MINOPT modeling language. (The syntax of the modeling language is described in the following section.) All of the necessary problem information can be contained within this file although additional files may be included. MINOPT searches for the input file in the directory from which it was run and writes all of its output to various files in the current directory. All of the output files begin with the basename of the input file `<input>` with various other extensions. The main output file is the log file named `<input>.log` that contains the solutions of the subproblems, and the optimal solution. Other output files may be generated based on the options selected and type of problem being solved. These files are listed in Table 1.

Many of the available options for MINOPT can be specified through the command line. Options specified on the command line take precedence over those specified in the input file. MINOPT recognizes the following command line options:

`-j` Solve first primal problem then exit.

`-w` Have the LP/MILP solver write its problems in lp format to the files `<input>.pricpx`, `<input>.gbdmascpx`, `<input>.oamascpx`, `<input>.apmascpx`, `<input>.prilps`, `<input>.gbdmaslps`, `<input>.oamaslps`, and `<input>.apmaslps` for the respective subproblems.

`-a` Perform an auto-initialization procedure where the continuous relaxation problem is solved to determine a starting point for the $\boldsymbol{y}$ variables. The integrality constraints on the $\boldsymbol{y}$ variables are relaxed and the resulting NLP is solved. The $\boldsymbol{y}$ variables are then rounded to the nearest integer and this is used as the starting point for the MINLP algorithm. Thes is the same as specifying the `AUTOINIT` option.

`-r` Solve the continuous relaxation problem then exit. The integrality constraints on the $\boldsymbol{y}$ variables are relaxed and the entire problem is solved as an NLP. This is the same as specifying the `DORELAX` option.

`-d` Write the problem to `<input>.dump`. This is useful for debugging and seeing how MINOPT actually views the problem.

`-i maxiter` Set maximum number of iterations for the MINLP algorithms to maxiter

`-p level` Set print level (amount of output printed to files):

- `level`=0 no printed output.
- `level`>0 print progress for each iteration to file.
- `level`>1 print solutions of primal problems to file.
- `level`>2 print solutions of master problems to file.
- `level`>3 print bounds to file.
- `level`>4 print subproblem solver output to file (usually fortran file number 9: ftn09, fort.9, etc).

`-s level` Set summary level (amount of output printed to the screen):

- `level`=0 no summary output.
- `level`>0 print iteration number to screen.
- `level`>1 print solution progress to screen.
- `level`>2 print solution bounds to screen.
- `level`>3 print solver output to screen.

`-L LPsolver` Set the LP solver: CPLEX (default), LPSOLVE, MINOS, SNOPT, or NPSOL.

`-I MIPsolver` Set MIP solver: CPLEX (default) or LPSOLVE

`-N NLPsolver` Set NLP solver: MINOS (default), NPSOL, SNOPT.

`-M MINLPsolver` Set MINLP solver: **GBD** (default), **OAER**, **OAERAP**, **GCD**.

`-D DAEsolver` Set the DAE solver (integrator): **DASOLV** (default), **DAESSA**.

`-o optionstring` Set any of the yes/no type options listed in the Options section.

`-V` Print the version number of MINOPT and exit.

`-S` Print the solvers compiled with this version of MINOPT and exit.

`-h` Print out a help message displaying the command-line options.

4 MINOPT: The Modeling Language

The input to MINOPT is provided in the form of an input file. This input file is a standard ASCII text file that can be created and modified using any text editor. (In a UNIX environment, vi or emacs are commonly used editors.) All of the necessary information for the problem and its solution is contained in this input file. The model contained in the input file is written using the MINOPT modeling language that has specific syntax that must be followed in order to provide valid input. When MINOPT is executed to solve a model, the input file is read and its syntax of the model is checked. If the syntax is not valid, the errors that exist in the input file are reported. Once the input file has been successfully read, the model is solved using the appropriate algorithm based on the problem type and given options.

The MINOPT modeling language has been designed to follow the natural forms for writing mathematical notations. Many of the types of constructions that can be written mathematically have an equivalent representation in the MINOPT modeling language. The modeling language is clear in that the representations of the mathematical notations are close to the corresponding representation and there is no ambiguity in the use of the language. The modeling language has also been designed to be concise by employing indices and indexed sets. This allows for large models to be represented with a compact notation. The modeling language is flexible in that the model can be easily adapted to new situations in lights of changing requirements. A model used for one type of problem can be easily manipulated to represent a different situation. Parts of one model can be easily incorporated into other models as may be required. Overall, the modeling language is comprehensive and its power is reflected in the broad class of model types that it can address.

This chapter describes the structure of the MINOPT input file and provides a detailed description of the modeling language. The overall structure of the input file and the basic building elements of the modeling language are presented and simple illustrations are provided to demonstrate the usage of the modeling language.

4.1 Lexical Elements of the MINOPT Modeling Language

All of the model information is written using ASCII text. The general input has a free form in that white space (spaces, tabs, and newline characters) may appear anywhere within the file without affecting the meaning of the input. Specific combinations of characters (letters, numbers, and symbols) represent the lexical elements of the modeling language:

- numbers
- special symbols
- mathematical symbols
- keywords
- identifiers

4.1.1 Numbers

Numbers are formed in a standard manner: an optional sign followed by a sequence of digits that may contain a decimal point (for example 13 or -5.17 or +0.71). The number may also have an optional exponent which begins with either an `e` or an `E` followed by an optional sign followed by digits (for example 3e23 or 6.022E23 or 1.3807e-23). All of the numbers are double precision floating point numbers.

4.1.2 Special Symbols

The special symbols are

`{ } | # $. , : ; & < > [ ] ( )`

These are used to construct the various statements for the MINOPT modeling language.

4.1.3 Mathematical Symbols

The mathematical symbols are

`+ - * / ^! = < > & | E : [ ] ? ( )`

These are used to create the mathematical expressions for the MINOPT modeling language. Note that some symbols occur in both the set of special symbols and the set of mathematical symbols. This is because the same symbol can be used in different constructions and have different meanings.

4.1.4 Keywords

Keywords are strings of letters which represent terms that MINOPT recognizes as having a particular meaning. The meanings are fixed and the keywords cannot be redefined. Keywords also can not be used as identifiers.

A list of the keywords used by MINOPT and their meaning are provided in Table 2. All of the keywords all capital letters and the first four letters of the keyword are significant. Thus, for the keyword `DECLARATION`, the full name, `DECLARE`, or `DECL` are all equivalent.

4.1.5 Identifiers

Identifiers are symbols that the user defines to represent index, parameter, variable, and constraint names. Identifiers are alphanumeric patterns which must begin with a letter and may contain underscores and may end with a single quote. Example of valid identifiers are `x`, `Ca`, `Flow_rate`, and `T1`. The trailing single quote is useful for denoting prime as in i' which could be represented with `i'` in MINOPT. It is also used to denote the time derivative of a dynamic variable such that if z is a dynamic variable defined as `z`, the time derivative, $\frac{dz}{dt}$, is `z'`

Indices or subscripts on an identifier are specified by a comma separated list of integers which are enclosed in parentheses and follow the identifier name. There is no limit to the number of dimensions an identifier can have. For example, the mathematical notation w_1 could be written in MINOPT as `w(1)`. The element in the parentheses can also be another identifier

Table 2: Keywords and their meaning

Keyword	meaning
INCLUDE	specify a file to be included
DECLARATION	indicates the start of the declarations section
MODEL	indicates the start of the model section
OPTIONS	indicates the start of the options section
SET	define an identifier as a set
INDEX	define an identifier as an index
PARAMETER	define an identifier as a parameter
XVARIABLES	define the x-variables to be used in the model
YVARIABLES	define the y-variables to be used in the model
ZVARIABLES	define the z-variables to be used in the model
BINARY	specify variables as binary (0 or 1)
INTEGER	specify variables as taking integer values
POSITIVE	specify variables as being positive
XLBDS	specify the lower bounds on the x-variables
XUBDS	specify the upper bounds on the x-variables
YLBDS	specify the lower bounds on the y-variables
YUBDS	specify the upper bounds on the y-variables
ZLBDS	specify the lower bounds on the z-variables
ZUBDS	specify the upper bounds on the z-variables
LBDS	specify the lower bounds on a variable
UBDS	specify the upper bounds on a variable
XSTP	specify the starting points on the x-variables
YSTP	specify the starting points on the y-variables
ZINC	specify the initial conditions the y-variables
STP	specify the starting points for a variable
ICS	specify the z-variables whose initial conditions depend on x-variables (coupled with ISP)
ICP	specify the x-variables used to set the initial conditions of the z-variables (coupled with ICS)
ISPEC	specify the z-variables which are fixed to determine the initial conditions of the DAE system
SHOW	show the values of a parameter
SAVE	save the vaules of a parameter to a file
DISPLAY	display the values of an expression
TIME	specify the time used for the integration of the dynamic system
CKFILE	specify the name of the file to use for Chemikin data
PRINT	print the values of a variable
MINIMIZE	specify the minimimization of the objective function

(an identifier representing either a set or an index). Thus, the mathematical form Tf_i can be expressed as `Tf(i)`. For multidimensional arrays such as for the mathematical form $v_{1,2,3}$, the MINOPT notation is `v(1,2,3)` or for v_{ijk}, the MINOPT form is `v(i,j,k)`. Names for identifiers can not be duplicated and can not have the same first three letters as any of the keywords.

4.2 Overall Structure of MINOPT Input File

The MINOPT input file consists of three sections:

- Declaration Section: This section includes the definitions for the sets, indices, parameters, variables, variable bounds, variable starting points, and initialization information for the dynamic problems.
- Model Section: This section includes the objective function along with all of the constraints for the mathematical problem.
- Options Section: This section, which may or may not exist, lists all of the options for the given mathematical problem.

Each section has the form:
SECTION_NAME {{
statement;
statement; *statement*;
...
}}
where *SECTION_NAME* is the keyword associated with the section (`DECLARATIONS`, `MODEL`, or `OPTIONS`).

Within each of these sections, keywords, identifiers, numbers and symbols are used to create the statements that define the model. Each statement must be terminated by a semicolon.

4.2.1 Comments

Comments can be used at any point throughout the input file. There are three different styles of comments that can be used.

Single-Line Comments These comments begin with "#" and extend to the end of the current line. Anything that appears after the "#" on the current line is not read by the parser.

Echoed Comments These comments begin with a "$" and extend to the end of the current line. Anything that appears after the "$" is a comment which is echoed to the screen as the input is parsed. This can be used to print the name of the model or other details of the model as the input file is read.

Blocked Comments These comments are C-style comments which are formed using the notation "`/*` ... `*/`". Anything between the "`/*` and `*/` is considered a comment. This type of comment can extend over multiple lines and is useful for commenting large portions of the input file.

4.2.2 Including files

At any point within the input file, another file may be included using the keyword `INCLUDE` or `include` followed by the filename to be included. This feature can be used to break apart the pieces of the input into separate files. In this way, all related data can be contained within a separate file. This also allows for the same information to be used with several input files.

The syntax for including a file is

`INCLUDE` *filename* `;`

where *filename* is the name of the file to be included. The file will be included exactly at the point of the `INCLUDE` statement.

4.3 Basic Building Blocks of the Modeling Language

As described above, the lexical elements of the MINOPT modeling language consist of numbers, symbols, keywords, and indentifiers. By assembling these elements in an appropriate fashion, valid statements which describe the MINOPT model are formed.

The model is developed by using the following building blocks of the modeling language:

- indices
- sets
- parameters
- variables
- expressions

4.3.1 Index, Set, Parameter, Variable: Elementary Definitions

The identifiers in MINOPT are symbolic representations of the various types of numerical values used in MINOPT. These symbols are used to generate a clear concise representaion of the mathematical model. There are four types of symbols used by MINOPT:

- index: a symbol representing any element of a set.
- set: a symbol representing an ordered list of elements which can be numbers or strings
- parameter: a symbol representing a numerical value which does not change in the model definition
- variable: a symbol representing a numerical value which is not fixed and is to be determined through the solution of the model

Each type of identifier is defined in a different way and will have a different meaning when used within the model. All of the identifiers are defined in the declarations section of the input file and can be used at any point after their definition, provided the correct syntax is used.

4.3.2 Expressions

Expressions are combinations of the identifiers along with standard mathematical notation used to represent mathematical formulations. Two different types of expressions are used in the MINOPT input file: arithmetic and logical. Although no distintion between the two needs to be made in the input, the two types of expressions are discussed separately.

Arithmetic Expressions The mathematical operators used by MINOPT are the standard operations of addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), and exponetiation (`^`). All of the mathematical operators and their precedence levels are listed in Table 3. Successive operations of the same precedence group to the left except for exponetiation which groups to the right. The order of operations can be changed by using parentheses.

MINOPT recognizes a number of mathematical functions which can be incorporated into the expressions. These functions consist of the name of the function followed by a bracket-enclosed argument. The argument for the expression is itself an expression. The unary mathematical functions which MINOPT recognizes are listed in Table 4. Note that the arguments of these mathematical functions are enclosed within brackets (`[ ]`) and not within parentheses.

Arithmetic expressions range from the simple expressions such as

```
x1 + x2*x3
```

to complex expressions such as

```
(x1+x2*log[x3+1])/((x2+x4)*(y2-y4))
```

Logical and Conditional Expressions The values of aritmetic expressions can be tested by using comparison operators:

`==` equal to
`!=` not equal to
`<` less than
`<=` less than or equal to
`=>` greater than or equal to
`>` greater than

The result of these tests is either 1 if the test is true or 0 if the test if false. Thus the expression `T<10` is 1 if the value of `T` is less than 10 and 0 if it is greater than or equal to 10. The conditional expressions can be combined with logical operators:

`&&` AND
`||` OR

Thus the expression

```
T>=0 && T<=10
```

is 1 only if `T` is greater than or equal to 0 *and* less than or equal to 10.

The fact that the logical expressions result in a numerical value (1 or 0) allows them to be included with arithmetic expressions. In this usage, they can act as switches to turn another expression either on or off. For example

Table 3: Binary Mathematical Operators and Precedence Levels

Precedence	Operator	Name
1	`||`	or
1	`&&`	and
2	`<`	less than
2	`<=`	less than or equal to
2	`==`	equal to
2	`!=`	not equal to
2	`>=`	greater than or equal to
2	`>`	greater than
3	`?`	binary minimization
4	`+`	addition
4	`-`	subtraction
5	`*`	multiplication
5	`/`	division
6	`+`	unary plus
6	`-`	unary minus
7	`^ **`	exponentiation

```
(T>1)*x1
```

will give the value of `x1` when `T` is greater than 1 and 0 otherwise. This is useful when setting the values of parameters that are based on logical conditions.

Sets and Indices One of the fundamental components of MINOPT is the set. The use of sets allows for compact notation as parameters, variables, constraints, and even sets in a model are often defined over sets. Mathematical expressions can be simplified by using operations such as summation, product, and enumeration over sets. Using sets and indices permits large mathematical problems to be written in a concise form. Any model component (parameters, sets, variables, and constraints) can be indexed over a set. Thus sets themselves may be indexed over a set.

A set is an ordered list of elements which are numbers (integers) or sets themselves. It may contain zero (empty set) or more elements. The elements of a set must be either integers or sets. Therefore, the lowest level elements must always be integers.

A simple set of numbers is a comma separated list of numbers delimited by the vertical bar, `|`. Thus `|1,2,3,4,5|` denotes the set of numbers 1,2,3,4 and 5. To denote a range of numbers easily, a colon, `:`, can be used such as `|1:10|` to denote the numbers 1 through 10 inclusive. In addition, a second colon can be used to indicate the step size for a list. Thus `|10:100:10|` denotes the numbers 10, 20, 30, ... , 100. Finally, an empty set is indicated by using a period as in `|.|`.

An identifier is defined as a set by using the keyword `SET`. The following statement defines `I` as the set of numbers 1 through 10:

Table 4: Mathematical Functions

Name	Function
`sin[x]`	sine ($\sin(x)$)
`cos[x]`	cosine ($\cos(x)$)
`tan[x]`	tangent ($\tan(x)$)
`asin[x]`	inverse sine ($\sin^{-1}(x)$)
`acos[x]`	inverse cosine ($\cos^{-1}(x)$)
`atan[x]`	inverse tangent ($\tan^{-1}(x)$)
`sinh[x]`	hyperbolic sine
`cosh[x]`	hyperbolic cosine
`tanh[x]`	hyperbolic tangent
`exp[x]`	exponential (e^x)
`log[x]`	natural logarithm ($\log_e(x)$)
`log10[x]`	common logarithm ($\log_1 0(x)$)
`sqrt[x]`	square root ($\sqrt{x}$)

```
SET I = |1,2,3,4,5,6,7,8,9,10|
```

An equivalent representation is achieved using the colon:

```
SET I = |1:10|
```

To define J as the set of even numbers from 2 to 10, the statement

```
SET J = |2,4,6,8,10|
```

can be used. The same definition can be obtained using the two colons as in

```
SET J = |2:10:2|
```

To define several sets, each must be defined separately. If sets `J`, `K`, and `L` are to be defined as having elements 1 through 6, 1 through 4 and 1 through 10 respectively, the following definitions must be made:

```
SET J = |1:6|

SET K = |1:4|

SET L = |1:10|
```

Indexing expressions In mathematical notation, sets are used in statements such as

- "forall i in the set P"
- "for $t = 1, \ldots, T$"
- "$\forall j \in J$ such that $c_j > 0$"

In these statements, i, t, and j are indices which are elements of the corresponding set taken one at a time. MINOPT requires that all indices be defined prior to their usage. This is done using the keyword `INDEX` as follows:

```
INDEX {i,j,t}
```

To formulate indexing expressions and indicate set membership, the operator `E` is used. This is equivalent to the mathematical operator $\in$. For example, the mathematical expression $i \in P$ is represented by `i E I` in MINOPT. Some other examples:

math	MINOPT
$j \in J$	`j E J`
$slots \in SP$	`slots E SP`
$k \in K \quad l \in L_k$	`k E K, l E L(k)`

Although single letters have been used for the index and set names, this is not required. Any valid identifier can be used for the set and index name. The following set and index definitions are valid:

```
SET Years = |1990,1991,1993,1995,1996,1998|
INDEX {counter}
```

These can be used together as in `counter E Years`. These types of expressions are used in expressions that occur over a set such as summation and enumeration which will be described below.

The index or indices on an indentifier are specified by a comma separated list which is enclosed in parentheses and follows the identifier name. Typically, indices are used to represent the subscripts of a parameter or variable. For example, the variable $x_{1,2}$ would be represented by `x(1,2)` in MINOPT. The elements of the list enclosed in parentheses can be indices provided that the index is active. An index is activated when the index `E` set expression occurs prior to the use of the index. The elements may also be expressions provided that the expression evalauates to a index for which the identifier is defined. This will become more evident when the summation and enumeration are described below.

Indices on an identifier are specified by a comma separated list of integers enclosed in parentheses. There is no limit to the number of dimensions an identifier can have. For example `w(0)`, `w(1)`, etc. can be used to refer to the elements of variable `w`. For multidimensional arrays, the notation is `v(0,1,2)`, `v(2,3,1)`.

Enumeration, Summation, and Products The mathematical constructions which employ the sets and indices are

- summation
- products
- enumeration

Each of these operations occur over a set and require an index to indicate the current element of the set. They are used along with set and index definitions which must occur earlier in the input file.

Summation The summation is enclosed within `<<` and `>>` and has the following construction:

```
<< index E set | expression >>
```

This is used to sum the value of the *expression* for each value of *index* which is a member of the set *set*. For example, the mathematical expression

$$\sum_{i \in I} p_i w_i$$

is represented in MINOPT by

```
<< i E I | p(i)*w(i) >>
```

There is no limit to the number of summations that can be nested within previous summations. A double summation such as

$$\sum_{i \in I} \sum_{j \in J} p_i x_{i,j}$$

is represented in MINOPT by

```
<< i E I | << j E J | p(i)*x(i,j) >> >>
```

The indices used in the summations are dummy variables that take values over the specified set. Although in the above examples the index is the lowercase letter of the corresponding set, there is required correspondence between the index and set. Any index defined by `INDEX` can be used as the index for a summation provided that it is not already being used by another indexing expression. Although there is no restriction on the index and set correspondence, a good practice is to use the same index with the same set throughout the model. Using an index and set which match, such as `i` for the index and `I` for the set, also helps make the model more clear.

The index used in the summation is activated by the index `E` set construct and it remains active over the scope of the summation. This scope ends with the enclosing `>>` which ends the summation and deactivates the index. Not only can the index be used as a "subscript", but it can be used for its numerical value in an expression. For example

```
<<i E I| i*a(i)>>
```

will sum the values index `i` times the value of `a(i)` for all the values of `i` in the set `I`.

Summations can be used in more complex constructions such as

$$(\sum_{i \in I} \sum_{j \in J} \sum_{k \in K} \sum_{p \in P} c_{i,j,k,p} * x_{i,j,k,p}) + (\sum_{j \in J} f_j z_j) + (\sum_{j \in J} v_j (\sum_{i \in I} \sum_{k \in K} \sum_{p \in P} x_{i,j,k,p})^{2.5})$$

which is written as

```
<<i E I|<<j E J|<<k E K|<<p E P|c(i,j,k,p)*x(i,j,k,p)>>>>>>>>
                   + <<j E J|f(j)*z(j)>>
+ <<j E J|v(j)*<<i E I|<<k E K|<<p E P| x(i,j,k,p)>>>>>>^2.5>>
```

Note that by enclosing the summation within `<<` and `>>`, there is no confusion as to when the summation ends.

Product The product over a set is represented in MINOPT very similarly to the way summation is handled. Instead of the `<<` and `>>` used for the summation, `[[` and `]]` are used to denote the product. The apprpriate construction for the product in MINOPT is

```
[[ index E set | expression ]]
```

This is used to multiply the values of the *expression* for each value of *index* which is a member of the set *set*. For example, the mathematical expression

$$\prod_{i \in I} w_i^{a_i}$$

is represented in MINOPT by

```
[[ i E I | w(i)^a(i) ]]
```

Just as with the summation, the index becomes activated by the index `E` set construct and is deactivated at when the product is terminated by the `]]`. Multiple product expressions can be nested within each other.

Enumeration Enumeration is used to create a list of related expressions. Instead of adding or multiplying the arguments to create expressions as in the summation and product representations, the enumeration creates a list of several expressions. The enumeration is represented in a manner similar to the summation and product notations except that a the single `<` and `>` are used to enclose the enumeration. Thus, the enumeration has the form

```
< index E set | expression >
```

This will generate an expression for each value of *index* which is a member of the set *set*.

The enumeration operator is useful for specifying blocks of related expressions in a single statement. For example, the mathematical expression

$$x_i + w_{ij} * y_j \quad \forall i \in I \quad \forall j \in J$$

would be represented in the MINOPT language as

```
< i E I| <j E J| x(i) + w(i,j)*y(j) > >
```

An important note here is that the delimiters for the enumeration *must* be at the outer limits of the expression. The enumerations can be nested, but they must be the first part of the expression. For example, the following is **NOT** valid input:

```
<i E I| x(i) + <j E J| w(i,j)*y(j) > >
```

The correct expression is

```
<i E I| <j E J| x(i) + w(i,j)*y(j) > >
```

The following is also not correct:

```
2*<i E I| <j E J| w(i,j)*y(j) > >
```

The correct expression is

```
<i E I| <j E J| 2*w(i,j)*y(j) > >
```

By enclosing the enumeration within `<` and `>`, there is no confusion as to the scope of the indices and where the indices are active and inactive.

Table 5: Mathematical Functions over Sets

Name	Function
`min[i E I|x(i)]`	minimum value of x_i
`max[i E I|x(i)]`	maximum value of x_i
`interv[i E I|x(i)]`	interval function[1]

Functions over sets MINOPT has several functions which have multiple operands and operate over a set. These functions, listed in Table 5, are the min function, the max function, and the interval function. Each of these has the form

funname `[` *index* `E` *set* `|` *expression* `]`

The `min` function returns the minimum value of the *expression* evaluated for all values of *index* in the set *set*. Thus, the MINOPT representation of the mathematical expression

$$\min_{i \in I} p_i * x_i$$

is

```
min[i E I| p(i)*x(i)]
```

Similarly, the `max` function returns the maximum value of an expression evaluated for each member of the set.

Cardinality Function MINOPT has a cardinlity function denoted as `card[S]` which will return the cardinality of the set `S`. (The cardinality of the set is defined as the number of elements in the set.) The argument of this function must be a set and any other argument will cause an error.

Logical conditions for set qualification There are times when certain conditions should be imposed on the elements of a set which are to be used for a summation, product or enumeration. For example, the situation may require all elements in set S such that the element is less than 10. In mathematical notation, this might look something like $\forall s \in S$ such that $s < 10$. In MINOPT, to indicate that a logical condition is to follow, a single ampersand, `&` is used. A logical expression follows the `&`, and if this expression is true, then that value of the index is used in what ever follows. (If it is used in a summation, the argument for the summation is computed for that value of the index.) Otherwise, that value of the index is skipped and the index is set to the next element of the set. The set is represented in MINOPT as

```
s E S & s<10
```

Consider the expression

$$\sum_{s \in S, s < 10} x_s$$

This is represented in MINOPT as

```
<<s E S & s<10| x(s)
```

This expression will sum all the values of `x(s)` for the elements in the set `S` whose element value is less than 10. The `&` can be read as "such that" and this is a convenient way to think of the use of this operator. Recall that the index `E` set construct activates the index and thus it can occur in the conditional expression. Any logical expression can be used after the `&`. For example,

```
j E J & c(j)>0 && c(j)<10
```

describes the set of all `J` for which `c(j)` is between 0 and 10. As another example

```
j E J & (cs(j)+ct(j))/2 >= 0 || j<10
```

describes the set of elements of `J` such that the average of `cs(j)` and `ct(j)` is greater than or equal to zero or `j` is less than 10.

Sets of Sets Recall that the elements of a set can be numbers or additionally they can be sets themselves. In other words, a set can be an array of sets. In mathematical models, this type of notation is required, for example, in the following summation:

$$\sum_{i \in I} \sum_{j \in J_i} x_j$$

This means is that the summation is over all j in the set J_i where i is an element of set I. Another example may be in an enumeration such as

$$T_{ij} - H \quad \forall i \in I \quad \forall j \in J_i$$

This describes an expression for all j in the set J_i where i is an element of set I.

With these ideas in mind, there is a need to define a set of sets. This is handled in MINOPT by using the `SET` keyword just as before, but now the set being defined will have a dimension. This dimension is a comma separated list of sets whose definition must occur earlier in the input file. This can be best illustrated through an example. Consider a set `I` that has elements 1 through 4 and the 4 sets `J(i)` where `i` is an element of the set `I`. The four sets `J(i)` are to have elements, 1 through 3, 2 through 5, the empty set, and elements 4 through 6 respectively. To define these in MINOPT, the following construction is used:

```
SET I = |1:4|

SET J(I) = {|1:3|, |2:5|, |.|, |4,6|}
```

Note that in the definition of `J`, the identifier in the parentheses (in this case the `I`) must be an already defined set as is the case here where `I` is defined as a set of numbers 1 through 4. Note also that since the set `J` has four elements, each one must be defined. This is done by providing a list of these elements, which in this case are sets, enclosed in braces ({ }). This listing is something that will arise in other definitions that will be discussed later.

Note that a set `I` is used to indicate the size of the array for `J`. This is one place in MINOPT where a definition has an implied meaning. The implication here is that the `J` is to be defined for each element of `I`. A number can be used in the parentheses for the definition of `J` as in `J(3)`. However, this has no bearing on the size of the array and will only define `J(3)` and not other elements for `J` such as `J(1)` and `J(2)`. Note that this is different from programming languages such as C and FORTRAN where when a variable is defined, the size of the arrays are given and the numbering for the array has an assumed starting point (0 for C and 1 for FORTRAN). With the notation used in MINOPT, there is no confusion as to the size of the array and which index elements are used. These come directly from the set used in the definition.

The mathematical expressions

$$\sum_{i \in I} \sum_{j \in J_i} x_j$$

and

$$T_{ij} - H \forall i \in I, \forall j \in J_i$$

with the set definition described above can now be written in MINOPT as

```
<< i E I| <<j E J(i)| x(j) >> >>
```

and

```
<i E I| <j E J(i)| T(i,j) - H > >
```

Since the index `i` is activated by `i E I`, it can be used as the index for the set `J`.

4.3.3 Parameters

Assembling a mathematical model almost invariably requires numerical values. In order to maintain a concise model representation, a symbolic description of these values can be used. Large amounts of data can be provided while still maintaining a readable model.

The named symbols with fixed numerical values are called parameters in MINOPT. The main channel for specifying data for the model is through parameter definitions. These parameters can range from simple scalar values to vectors and matrices which are indexed over sets. Parameters can be used throughout the model in expressions used to define the objective and constraints, in expressions used to define other parameters, or in expressions used to define logical conditions for sets.

The key purpose of parameters is to represent constant numerical values in symbolic form to make the model as concise as possible.

4.3.4 Variables

A general mathematical model consists of variables and constraints, and the solution of the mathematical model determines the values of the variables. Then named symbols in MINOPT whose values are to be determined by the solution algorithm are called variables.

The variables used in the model are very similar to parameters in that they are symbols the stand for numbers. An identifier that is used for a variable looks similar to one used for a parameter. Both can be used in expressions, and, when the expressions were described earlier, no distinction was made as to whether the symbols being used were parameters or variables. However, Variables do not have an assigned value and their values are determined through the solution of the mathematical model.

Within MINOPT there are three types of variables that can be used. These variable types correspond to the variable partitioning to be used by algorithms which solve the mathematical problem. The description of the model can be kept independent of the algorithm used to solve it except for this variable partitioning. The modeler is required to define the variable partitioning as MINOPT will not determine this automatically.

Variable Types MINOPT recognizes three types of variables which are referred to as x-variables, y-variables, and z-variables. Every mathematical model will have x-variables and this is the variable type that will be used for most of the variables. The y-variables are used to partition variables for the MINLP algorithms (Generalized Benders Decomposition, and Outer Approximation and its variants) and usually, although not necessarily, represent the binary variables for these algorithms. The third type of variable is the z-variable which is used to denote the dynamic variables in the model. Again, this variable partitioning will become evident when the algorithms for solving dynamic problems are described.

4.4 The Declarations Section

The is no restriction to the order in which the information in the declarations section is provided except that identifiers must be defined prior to their usage. No general structure will be generally be good for all models, but some structure will help make the model clearer. Most of the input files provided follow a general structure where the sets and indices are defined, then the parameters are defined, then the variables are declared, and this is followed by the bounds and starting points for the variables.

The declarations section begins with `DECLARATIONS` {{ and ends with }}. The body of the declarations setcion consists of statements which can extend over multiple lines and are terminated with a semicolon `;`. These statements are used to declare and define the various elements that will be used in the model: sets, indices, parameters, and variables. Also, contained within this section are definitions for the bounds and starting points for the variables and also information for the initialization of the dynamic system.

4.4.1 Sets

Since sets are used in parameter definitions and variable definitions, they could appear first in the input file. The sets are defined using the keyword `SET`. The syntax for defining a set is the following:

`SET` *identifier* `=` `|` *number* `,` *number* `,` ... `,` *number* `|`

The identifier will become a set whose elements are the numbers enclosed within the `|` `|`.

To specify a range of numbers for a set, the colon can be used with the following syntax:

`SET` *identifier* `=` `|` *start* `:` *end* `|`

This will specify the identifier as being a set with the elements being *start*, *start*+1, *start*+2, ... , *end*. Two colons can be used to define the starting element, ending element and the step size for the elements of a set. The syntax is as follows:

`SET` *identifier* `=` `|` *start* `:` *end* `:` *step* `|`

This will define the *identifier* as a set whose elements are *start*, *start*+*step*, *start*+2*step*, ... , *end*.

4.4.2 Indices

Through the input file, dummy identifiers will be required to indicate a particular element of a set. These identifiers are refered to as indices and are defined in MINOPT using the keyword `INDEX`. The syntax for defining the parameters is

`INDEX` { *identifier* `,` *identifier*`,` ... `,` *identifier* }

This defines each identifier in the list as an index. The indices are used in conjunction with sets in the *index* `E` *set* constructions that are used in enumerations, summations, and products.

4.4.3 Parameters

Parameters are defined in MINOPT using the keyword `PARAMETER` using the following construction:

`PARAMETER` *identifier* `=` *rhs*

where *rhs* is either a list or an expression. The two principal ways of defining parameters are through lists and through expressions. Lists are created by enclosing numbers, or possibly other parameters, within braces ({ }). An expression is created using the rules described earlier.

Parameter definition by list When assigning the values of a parameter using a list of numbers, the following construction is used:

`PARAMETER` *identifier* `=` { *number*`,` *number*`,` ... `,` *number*}

The simplest parameter definition is for a scalar value. A simple parameter definition for defining `R` as having the value 8.314 is the following:

`PARAMETER R = 8.314`

Technically, MINOPT actually considers 8.314 as an expression in this case. An equivalent representation is

`PARAMETER R = {8.314}`

where the `{8.314}` is considered a list by MINOPT. For scalar parameters, this is trivial and either representation is appropriate. However, for more complex definitions, this is an important consideration.

More often, parameters are to be vectors of numerical values. In order to define the parameter `q(i)` as having the values of 25.0, 15.0, 18.0, 16.0, and 12.0 for values of `i` equal to 1 through 5, the following definitions are made:

```
SET I = |1:5|

PARAMETER q(I) = {25.0, 15.0, 18.0, 16.0, 12.0}
```

This will define `q(1)`, `q(2)`, `q(3)`, `q(4)`, and `q(5)` as having values of 25.0, 15.0, 18.0, 16.0, and 12.0 respectively. Note that in the definition of the parameter `q`, The argument on the parentheses is a set whose size is equal to the number of elements in the list on the right hand side, which is 5 in this case. By specifying the set over which the parameter is to be defined, there is no confusion as to the indices for the parameter. Consider defining a variable in the following manner:

```
SET W = |1978, 1982, 1986, 1990, 1994, 1998|

g(W) = {4,4,5,1,0}
```

This defines `g(1978)`, `g(1982)`, `g(1986)`, `g(1990)`, `g(1994)`, and `g(1998)` as having values of 4,4,5,1, and 0 respectively.

Parameters can be defined with an unlimited number of dimensions. So far the scalar and single dimension cases have been examined. Now consider a two dimesional case where a table of data is to be provided. Consider the following parameter definition in MINOPT:

```
SET I = |1:5|

SET J = |1:6|

PARA t(I,J) = { 6.4, 4.7, 8.3, 3.9, 2.1, 1.2,
                6.8, 6.4, 6.5, 4.4, 2.3, 3.2,
                1.0, 6.3, 5.4, 11.9, 5.7, 6.2,
                3.2, 3.0, 3.5, 3.3, 2.8, 3.4,
                2.1, 2.5, 4.2, 3.6, 3.7, 2.2}
```

The key point to note here is that the listing in the table is row-major. That means that the parameter values are assigned in the order `t(1,1)`, `t(1,2)`, `t(1,3)`, ... , `t(1,5)`, `t(2,1)`, ... , `t(5,5)`, `t(5,6)`. Since this is a two dimensional array, the first dimension corresponds to the rows in the table and the second dimension corresponds to the columns. As can be seen in the above example, the number of rows, 5, corresponds to the size of the set `I` while the number of columns, 6, corresponds to the size of the set `J`.

Higher dimensional arrays can be defined in a similar fashion, keeping in mind that the listing in the right hand side is row major. The following is an example of a four dimensional array:

```
SET I = |1:2|

SET J = |1:3|

SET K = |1:4|
```

```
PARAM c(I,J,I,K) =  {10,15,20,10,
                    5, 5,30,10,

                    25,20,15,20,
                    10,20,40, 5,

                    50,25,10,15,
                    60,75,25,10,


                    15, 5,15,35,
                    10,25,40,55,

                    0,15,30,45,
                    60,45,30,15,

                    25,35,45,55,
                    30,10,10,30}
```

This example also illustrates that a set can be repeated in the definition of the parameter. The definition of the parameter `c` used the set `I` twice in `c(I,J,I,K)`. Note that the spacing and formatting of the data has been done in such a way so as to make it readable and easy to understand the ordering of the elements. Although such formatting is not necessary, it is recommended as it makes the input easier to understand.

All of the values of a parameter must be defined. Thus there can be no vacant spots in the data list used to assign the values. There is no automatic assignment of the values. This also means that the number of elements in the list on the right hand side must match the size of the parameter being defined. In the four dimensional case above, the size of the parameter being defined, `c(I,J,I,L)`, is $2 \times 3 \times 2 \times 4 = 48$, and there are 48 numbers in the data list.

MINOPT does have a feature of repeating the values in the list on the right hand side if it has come to the end of the list and all of the values of the parameters being defined have not been assigned. This can reduce redundancy in the parameter definition by eliminating repeated numbers in the input file. As a simple example, consider an array of parameters whose values are all the same. These can be assigned using the notation:

```
SET T = |1:1000|

PARAMETER a(T) = {110}
```

This will set all the values of the parameter `a` to 110 without having to write the number 110 for each of the 1000 elements. Another example of the automatic repeating of the values in the list is the following:

```
SET I = |1:5|

SET J = |1:6|
```

```
PARAMETER down(I,J) = {10,9,8,7,6,5}
```

In this case, the values of `down(1,1)` through `down(1,6)` are set to 10,9,8,7,6,and 5 respectively, `down(2,1)` through `down(2,6)` are set to 10,9,8,7,6,and 5 respectively, and so on.

Parameter definition by expression A model can be kept simple by computing complex parameter expressions in terms of data parameters. In this case, the right hand side of the parameter definition statement consists of an expression. When using an expression to assign the values of a parameter, the following construction is used:

`PARAMETER` *identifier* `=` *expression*

For a scalar definition, a single expression is used to define the parameter. For example, consider the mathmatical definition of the parameter $R = N_a k$ where N_a and k are parameters with defined values. ($N_a = 6.0221 \times 10^{23}$ and $k = 1.3807 \times 10^{-23}$) These definitions are made in MINOPT as:

```
PARAMETER Na = {6.0221e23}

PARAMETER k = {1.3807e-23}

PARAMETER R = Na*k
```

This will assign the value of 8.3147135 to `R`.

Parameter arrays can also be assigned using expressions by making use of the enumeration described earlier. Consider the following list of mathematical expressions:

$$
\begin{array}{rlll}
lln_j = & 0 & j = 1 \ldots 6 & \\
uln_j = & \log(4) & j = 1 \ldots 6 & \\
llv_j = & \log(300) & j = 1 \ldots 6 & \\
ulv_j = & \log(3000) & j = 1 \ldots 6 & \\
lns_{i,j} = & \log(s_{ij}) & i = 1 \ldots 5 & j = 1 \ldots 6 \\
lnt_{i,j} = & \log(t_{ij}) & i = 1 \ldots 5 & j = 1 \ldots 6
\end{array}
$$

These parameter definitions are represented in MINOPT by using the following enumerations:

```
SET I = |1:5|

SET J = |1:6|

PARA t(I,J) = {  6.4, 4.7, 8.3, 3.9, 2.1, 1.2,
                 6.8, 6.4, 6.5, 4.4, 2.3, 3.2,
                 1.0, 6.3, 5.4, 11.9, 5.7, 6.2,
                 3.2, 3.0, 3.5, 3.3, 2.8, 3.4,
                 2.1, 2.5, 4.2, 3.6, 3.7, 2.2}

PARA s(I,J) = {  7.9, 2.0, 5.2, 4.9, 6.1, 4.2,
                 0.7, 0.8, 0.9, 3.4, 2.1, 2.5,
                 0.7, 2.6, 1.6, 3.6, 3.2, 2.9,
                 4.7, 2.3, 1.6, 2.7, 1.2, 2.5,
                 1.2, 3.6, 2.4, 4.5, 1.6, 2.1}
```

```
PARAM lln(J) = <j E J|0>

PARAM uln(J) = <j E J|log[4]>

PARAM llv(J) = <j E J|log[300]>

PARAM ulv(J) = <j E J|log[3000]>

PARAM lns(I,J) = <i E I|<j E J|log[s(i,j)]>>

PARAM lnt(I,J) = <i E I|<j E J|log[t(i,j)]>>
```

The enumeration on the right hand side generates an expression for each value of the index in the corresponding set. Thus, in the first four parameter definitions, the enumeration generates 6 values (the size of the set `J`). The second two definitions generate 30 values (the size of set `I` times the size of set `J`). In these cases, the order of the enumeration is significant. Since the set `I` is in the outer enumeration and `J` is in the inner, the values are generated for values of `(i,j)` equal to `(1,1)`, `(1,2)`, ... , `(5,5)`, `(5,6)`. Note that this corresponds to the order in which the parameters on the left hand side are defined (row-major). In most situations, this will be the case and the order of the enumerations will match the order of the sets used in the left hand side of the parameter definition. This is exactly the case in the above example where on the left hand side, the sets are `I` then `J` and on the right hand side, the enumerations are over the sets `I` then `J`.

The expressions used for the parameter definition can also explicity involve the index such as required in the expression:

$$lnk_k = \ln(k) \quad k = 1 \ldots 4$$

In MINOPT, this would be represented by

```
SET K = |1:4|

PARAMETER lnk(K) = <k E K|log[k]>
```

The more complex expressions

$$\begin{aligned} llt_i = & \quad \ln(\max_{j=0\ldots5}(\tfrac{t_{ij}}{4})) \quad i = 1\ldots5 \\ ult_i = & \quad \ln(\max_{j=0\ldots5}(t_{ij})) \quad i = 1\ldots5 \\ llb_i = & \quad \ln(q_i \max_{j=0\ldots5}(\tfrac{t_{ij}}{6000\times4})) \quad i = 1\ldots5 \\ ulb_i = & \quad \ln(\min(q(i), \min_{j=0\ldots5}(\tfrac{3000}{s_{ij}}))) \quad i = 1\ldots5 \end{aligned}$$

are represented in MINOPT as

```
PARAM llt(I) = <i E I|log[max[j E J|t(i,j)]/4.0]>

PARAM ult(I) = <i E I|log[max[j E J|t(i,j)]]>

PARAM llb(I) = <i E I|log[(q(i)*max[j E J|t(i,j)])/(6000*4.0)]>
```

```
PARAM ulb(I) = <i E I|log[q(i) ?  min[j E J|3000/s(i,j)]]>
```

There is no limit to the complexity of the expressions which can be used to define parameters. The definitions must follow some simple rules. First, each expression in the parameter definition must evaluate to a numerical value or list of numerical values. This may seem odd as every expression is expected to evaluate to some number, but there are cases where a valid expression may not yield a number. One case is when the expression contains a *variable* which will be described in the next section. Since the variable does not have any value assigned to it, the expression will not evaluate to a numerical value.

The second rule is that the number of numerical values that the right hand side expression generates must be the same as the number of values that are being defined on the left had side. For this reason, using conditional expressions on the set for an enumeration generally will be useful since this will restrict the number of generated values. For example, the following definition:

```
SET I = |1:5|

PARAMETER a(I) = <i E I & i<4| 2*i>
```

will only generate three values from the right hand side expression (when `i` $= 1$, 2, and 3) where the left hand side is expecting 5 values. Since all values of the parameters must be assigned as there is not automatic assignment of values, this will generate a error. This is because `a(4)` and `a(5)` must have the values assigned. One of two possible cases is probably desired here. First, if the values of `a(4)` and `a(5)` are to be zero, then the appropriate construction is

```
SET I = |1:5|

PARAMETER a(I) = <i E I| (i<4)2*i>
```

Recall that the expression `(i<4)` will evaluate to 1 if `i` is less than 4 and 0 otherwise. Using this, the values of `a(1)`, `a(2)`, `a(3)`, `a(4)`, and `a(5)` will be 2, 4, 6, 0, and 0 respectively. If the intention were to define parameters for `a(1)`, `a(2)`, and `a(3)` and assign their values, the following construction could be used:

```
SET I = |1:5|

SET J = |1:3|

PARAMETER a(J) = <i E I (i<4)| 2*i>
```

Now the right hand side will generate the three values which is what the left hand side of the definition expects.

MINOPT also allows parameters to be redefined. When the parameter is first defined, the dimensions and sizes of the dimensions are establised and the inital values are set. Any subsequent parameter definitions for a parameter which has already been defined will reassign the value of the parameter. This is useful for cases where large tables of data contain many entries with the same values and relatively fewer entries which are different. Take for example a vector parameter h_i where $i = 1 \ldots 1000$. For 995 of these parameters, the value is 0 while 5 of them have nonzero values: $h_{23} = 6.74$, $h_{514} = 6.83$, $h_{609} = 2.99$, $h_{878} = 2.75$, and $h_{999} = 9.73$. These parameter values can be assigned in MINOPT by using the following:

```
SET T = |1:1000|

INDEX {t}

PARAMETER h(T) = <t E T|0>

PARAMETER h(23) = 6.74

PARAMETER h(514) = 6.83

PARAMETER h(609) = 2.99

PARAMETER h(878) = 2.75

PARAMETER h(999) = 9.73
```

The parameter definitions which follow the intial parameter definition only assign the value of the parameter for the index specified. The other elements in the array are unaffected.

4.4.4 Variables

The variables for the model are defined using the keywords `XVARIABLES`, `YVARIABLES`, and `ZVARIABLES` to define the x-variables, y-variables, and z-variables respectively. The syntax used for defining x-variables is

`XVAR {` *identifier*, *identifier*, ... , *identifier*`}`

The same syntax with `YVAR` and `ZVAR` replacing `XVAR` is used for defining the y-variables and z-variables. The list of identifiers used to define the variables are enclosed within braces({ }).

For example if the variables f and x are to be defined as x-variables, the following statement would be used in MINOPT:

```
XVAR {f,x}
```

If q is to be defined as a y-variable, this would be done by using

```
YVAR {q}
```

Just as can be done with parameters, variables can be defined over sets. To define the variable v_j for all j in J and b_i for all i in I, the following statements are used

```
SET I = |1:5|

SET J = |1:6|

XVAR {v(J),b(I)}
```

When the array variables are declared, the identifier in the parentheses must be a set indicating the indices for the variable. Note that the sets used in the variable declaration (`I` and `J` in this example) must be defined prior to the variable declarations.

There is no limit to the number of dimensions a variable can have. The declaration

```
XVAR {a(I,J,K,L)}
```

declares `a` as an x-variable with four dimensions whose indices correspond to the elements of the sets `I`, `J`, `K`, and `L` which must be defined prior to the variable declaration.

Some more examples for declaring variables in the MINOPT input file are

```
XVAR {x(I,J), y(I,J), L(I), V(I), T(I), feed(I), r, P1, P2, hl(I), hv(I)}
YVAR {q(I)}
ZVAR {x(I),u}
```

When z-variables are defined, their time derivatives are also defined automatically. The time derivatives of z-variables have the same name as the corresponding z-variable with a prime `'` appended. Thus, declaring z-variables with

```
ZVAR {x(I),u}
```

will also declare the variables `x'(I)` and `u'`. Note that the prime is appended before the indices. These variables are used to represent the time derivatives of the dynamic variables such that

$$\frac{dx_1}{dt}$$

which is represented by `x'(1)`.

When variables with multiple dimensions are defined, they are defined with the indices ordered row-major. This means that the variable definition for `v` represents the variables `v(1,1)`, `v(1,2)`, `v(1,3)`, ..., `v(5,5)`, and `v(5,6)`. (This will become important when the bounds are assigned for the variables.) Multiple declarations can occur for any type of variable. This means that multiple **XVAR**, **YVAR**, or **ZVAR** statements can occur in the input file.

4.4.5 Bounds

As discussed earlier, the values for the variables are not assigned by the modeler and are determined from the solution of the model. Lower and upper bounds on the values that the variables can have can be assigned by the modeler. When the variables are declared, their lower and upper bounds are set automatically to $-\infty$ and $+\infty$ where $-\infty$ is the smallest number the computer can handle and $+\infty$ is the largest number the computer can handle. For most cases, these bounds will be unsatisfactory and new bounds will need to be assigned.

The upper or lower bounds on the variables can be specified in two possible ways. The first way is to assign all of the lower bounds or all of the upper bounds for one of the variable types. The keywords which are used for this are **XLBDS**, **YLBDS**, and **ZLBDS** for assigning the lower bounds for all of the x-variables, y-variables, and z-variables respectively. For assigning the upper bounds, the keywords **XUBDS**, **YUBDS**, and **ZUBDS** are used. The syntax used is the keyword followed by a brace enclosed list of numbers which correspond to the bounds on all of the variables of the particular type. The syntax for each is as follows:

`XLBDS` { *number*, *number*, ... , *number*}

`XUBDS` { *number*, *number*, ... , *number*}

`YLBDS` { *number*, *number*, ... , *number*}

`YUBDS` { *number*, *number*, ... , *number*}

`ZLBDS` { *number*, *number*, ... , *number*}

`ZUBDS` { *number*, *number*, ... , *number*}

For example, if the following x-variables are declared

```
XVAR {x1, x2, x3, x4, x5}
```

then, the lower and upper bounds can be assigned with

```
XLBDS {-10.0, -5.5, -6.7, 0, 3.2}

XUBDS {5.1, 3.7, 9.3, 11.7, 5.4}
```

The list of numbers used to specify the bounds must be the same size as the total number of the variables of the corresponding type. In this example, there are 5 x-variables declared, therefore, there must be 5 numbers in the list for `XLBDS` and `XUBDS`.

Recall that indexed variables are defined with the indices ordered row-major. This must be remembered when setting the bounds on the variables. Consider the following variable declaration:

```
XVAR {x(I,J), r, y(I,J)}
```

The order of the x-variables is `x(1,1)`, `x(1,2)`, ... `x(5,5)`, `x(5,6)`, `r`, `y(1,1)`, `y(1,2)`, ... `y(5,5)`, and `y(5,6)`. Thus the list for `XLBDS` and `XUBDS` must contain 61 numbers and be in the correct order.

Assigning the bounds on all of the variables of a particular type can be confusing for a number of reasons. First, keeping track of the order of the variables can be difficult. Second, the variable ordering may be changed by the modeler which will lead to incorrect bound assignment unless the bounds list is changed accordingly. Third, providing one large list for the bounds can become cumbersome. Because of this, a second way of providing the bounds for the variables is provided. This method assigns the upper and lower bounds on a particular variable thus avoiding any confusion as to which variables are being assigned which bounds. The keywords used to assign the bounds are `LBDS` and `UBDS`. The syntax is identical to that used for the parameter definitions:

`LBDS` *identifier* `=` *rhs*

`UBDS` *identifier* `=` *rhs*

Just as with the parameter definitions, the *rhs* can either be a list or an expression. The same rules for defining the parameters apply for assigning the bounds on the variable. The difference is that the variable must already be defined.

To assign the values of the bounds using a list, the following construction is used:

`LBDS` *variable* `=` { *number*, *number*, ... , *number* }

where the *variable* is already defined. Consider the following variable declarations:

```
SET I = |1:5|

SET J = |1:6|

XVAR {x(I,J), r, y(I,J)}

YVAR {q(I)}
```

The bounds for `x(I,J)` can be assigned by using

```
LBDS x(I,J) = { 6.4, 2.0, 5.2, 3.9, 2.1, 1.2,
                0.7, 0.8, 0.9, 3.4, 2.1, 2.5,
                0.7, 2.6, 1.6, 3.6, 3.2, 2.9,
                3.2, 2.3, 1.6, 2.7, 1.2, 2.5,
                1.2, 2.5, 2.4, 3.6, 1.6, 2.1}
UBDS x(I,J) = { 7.9, 4.7, 8.3, 4.9, 6.1, 4.2,
                6.8, 6.4, 6.5, 4.4, 2.3, 3.2,
                1.0, 5.3, 5.4, 9.9, 5.7, 6.2,
                4.7, 3.0, 3.5, 3.3, 2.8, 3.4,
                2.1, 3.6, 4.2, 4.5, 3.7, 2.2}
```

Just as in the parameter definitions, the dimensions of the variable on the left hand side are sets which are already defined. These indicate the value of the indices used for assigning the bounds. Since the *variable* is already defined, the sets used for the left hand side variable in the bounds specification can be a subsets of the sets which were used to define the variable. This can be useful for specifying bounds on a restricted number of the varaibles. Consider the same variable declarations as before:

```
SET I = |1:5|

SET J = |1:6|

XVAR {x(I,J), r, y(I,J)}
```

If the bounds only need to be specified for `x(i,j)` where `i` is 1 through 5, but `j` is only 3, this is done using

```
LBDS x(I,3) = {5.2, 0.9, 1.6, 1.6, 2.4}

UBDS x(I,3) = {8.3, 6.5, 5.4, 3.5, 4.2}
```

This will set only the bounds in the list and the other bounds will remain unaffected.

The variable bounds can also be assigned using mathematical expressions using the constructions:

`LBDS` *variable* = *expression*

`UBDS` *variable* = *expression*

where the *variable* has already been defined. This will assign the values of the expression to the bounds of the variable. The same type of rules as used for the parameter definitions are followed for the bounds specification.

One way to declare a scalar x-variable t and assign lower and upper bounds of $-10c$ and $10c$ where c is a parameter is to write

```
XVAR {t}
LBDS t = -10*c
UBDS t = 10*c
```

The bounds on a variable array can be assigned using an expression and making use of the enumeration construction. Consider the following variable declarations:

```
SET I = |1:5|
SET J = |1:6|
XVAR {b(I),v(I,J)}
```

The bounds for `b(I)` can be assigned by

```
LBDS b(I) = <i E I| (q(i)*max[j E J| t(i,j) ])/(6000*4.0) >
UBDS b(I) = <i E I| q(i) ?  min[j E J| 3000/s(i,j) ] >
```

where `t(i,j)` and `s(i,j)` are defined parameters with appropriate sizes. The bounds for `v(I,J)` can be specified by

```
LBDS v(I,J) = <i E I| <j E J| -(t(i,j)+s(i,j))/2 > >
UBDS v(I,J) = <i E I| <j E J| (t(i,j)+s(i,j))/2 > >
```

The use of `LBDS` and `UBDS` for specifying the bounds often provides a clearer representation when compared to the use of `XLBDS`, `XUBDS`, etc. Their use will generally help avoid confusion in the model definition.

In many mathematical models, the lower bounds on many of the variables are 0. MINOPT provides the keyword `POSITIVE` for specifying a lower bound of 0 for a list of variables. The syntax for the usage is

`POSITIVE` { *variable*, *variable*, … *variable* }

Thus, if variables are defined with

```
XVAR {b(I),v(I,J)}
```

they can be specified as positive with

```
POSITIVE { b(I),v(I,J)}
```

Binary and integer variables MINOPT also recognizes both binary variables and integer variables. Binary variables are defined as those whose values are either 0 or 1 where as integer variables can take an integer value. To specify variables as binary or integer the keywords `BINARY` and `INTEGER` are used. They have the syntax

`BINARY` { *variable*, *variable*, ... , *variable* }

`INTEGER` { *variable*, *variable*, ... , *variable* }

where the variables in the list are already defined. For example, consider the variable declaration

`YVAR {q(I)}`

The variable `q(I)` is specified as being binary by writing

`BINARY {q(I)}`

Alternatively, `q(I)` is specified as being integer by writing

`INTEGER {q(I)}`

When a variable is specified as binary, its lower and upper bounds are set to 0 and 1 respectively. When a variable is specified as integer, its lower and upper bounds are specified as 0 and 100 respectively.

The user should also be careful when using bounds on the z-variables. The bounds on the z-variables are not bounds used in the optimization. These are bounds passed to the integrator and affect the solution of the dynamic system. For most cases, these bounds should not need to be set. For cases where the dynamic system may have multiple solutions, it may be necessary to set the bounds to ensure that the integrator solves the correct problem. When these bounds are violated, the integrator will detect a discontinuity and most likely will fail.

4.4.6 Starting Points

For some of the solution algorithms, specifying the starting points may help improve the efficiency. When a variable is defined, its default values for the starting points are set to 0. To set the values of the starting points for the variables, MINOPT has the keywords `XSTP`, and `YSTP`, for setting the starting points for the x-variables and y-variables and `ZINC` for setting the initial conditions of the z-variables. The use of these keywords is analogous to the use of keywords for setting the bounds,`XLBD`, `XUBD`, etc. and have the following syntax:

`XSTP` { *number*, *number*, ... , *number* }

`YSTP` { *number*, *number*, ... , *number* }

`ZINC` { *number*, *number*, ... , *number* }

These will assign the starting values or initial conditions for the corresponding variables in the order that they were declared.

In the same way that the bounds can be assigned for a specific variable, the starting point can be assigned for a specific variable using the keyword `STP`. This has the same construction as the parameter and bounds definitions:

`STP` *identifier* `=` *rhs*

where the right hand side can again be either a list or an expression.

To assign the starting values using a list, the following construction is used:

`LBDS` *variable* `=` { *number*, *number*, ... , *number* }

where the *variable* is already defined. If the following declarations are made:

```
SET I = |1:5|

SET J = |1:6|

XVAR {x(I,J), r, y(I,J)}
```

the starting values for `x(I,J)` can be assigned by writing

```
STP x(I,J) = {  7.0, 3.0, 6.2, 3.9, 5.1, 3.2,
                2.3, 1.1, 4.7, 3.6, 2.1, 2.7,
                0.9, 2.9, 3.6, 8.2, 4.3, 3.9,
                3.7, 2.9, 2.7, 2.9, 1.2, 2.7,
                1.3, 3.5, 4.2, 3.9, 3.7, 2.1}
```

To assign the starting values using an expression, the construction

`STP` *variable* `=` *expression*

where the *variable* has already been defined. If the scalar variable t is to be declared and assigned a starting point of $5c$ where c is a parameter, the following notation can be used:

```
XVAR {t}

STP t = 5*c
```

For variable arrays, enumerations can be used with expressions to assign starting points for all the variables in the array. For the variable declaration

```
SET I = |1:5|

SET J = |1:6|

XVAR {b(I),v(I,J)}
```

The starting points for `v(I,J)` can be specified by

```
STP v(I,J) = <i E I| <j E J| t(i,j) ?  s(i,j) > >
```

where `t(i,j)` and `s(i,j)` are defined parameters with appropriate sizes.

All of the statements listed above are concerned with placing restrictions on the values of the variables or with selecting a starting value of the variables. They do not assign values to the variables as this will be handled by the solution algorithm.

4.4.7 Initialization for Dynamic Systems

MINOPT is capable of handling dynamic problems where the dynamic portion of the problem is described by a system of differential and algebraic equations (DAEs). In order to solve this system of DAEs, the initial conditions must be specified. MINOPT has a number of keywords which are used to specify the initial conditions for the DAEs. The use of `ZINC` and `STP` have already been examined as ways to define the initial conditions for the z-variables. However, since the dynamic system is described by DAEs, a set of z-variables must be selected to determine the initial conditions for the DAE system. These are the variables whose values will be fixed in order to solve for the remaining values of the z-variables.

The dynamic model is represented using DAEs:

$$\boldsymbol{f}_1(\dot{\boldsymbol{z}}_1(t), \boldsymbol{z}_1(t), \boldsymbol{z}_2(t), \boldsymbol{x}, \boldsymbol{y}, t) = \boldsymbol{0}$$

$$\boldsymbol{f}_2(\boldsymbol{z}_1(t), \boldsymbol{z}_2(t), \boldsymbol{x}, \boldsymbol{y}, t) = \boldsymbol{0}$$

$$\boldsymbol{z}_1(t_0) = \boldsymbol{z}_1^0$$

$$\boldsymbol{z}_2(t_0) = \boldsymbol{z}_2^0$$

where $\boldsymbol{f}_1$ represents the n differential equations, $\boldsymbol{f}_2$ represents the m dynamic algebraic equations, $\boldsymbol{z}_1(t)$ is a vector of n dynamic variables whose time derivatives, $\dot{\boldsymbol{z}}_1(t)$, appear explicitly, and $\boldsymbol{z}_2(t)$ is a vector of m dynamic variables whose time derivatives do not appear explicitly. The variables $\boldsymbol{x}$ and $\boldsymbol{y}$ are parameters for the DAE system and variables for the optimization where $\boldsymbol{x}$ is a vector of p time invariant continuous variables and $\boldsymbol{y}$ is a vector of q variables. Time t is the independent variable for the DAE system and t_0 is the fixed initial time. The initial condition for the above system is determined by specifying n of the $2n+m$ variables $\boldsymbol{z}_1(t_0), \dot{\boldsymbol{z}}_1(t_0), \boldsymbol{z}_2(t_0)$. For DAE systems with index 0 or 1, the remaining $n+m$ values can be determined.

Thus, although there are $(n+m)$ z-variables, only n initial conditions must be specified. These n variables can be selected from the $2n+m$ variables $\boldsymbol{z}_1(t_0), \dot{\boldsymbol{z}}_1(t_0), \boldsymbol{z}_2(t_0)$.

To specify which variables are to be fixed to determine the initial condition, the keyword `ISPEC` is used in MINOPT. This is used to provide a list of z-variables whose values are fixed to those assigned by `ZINC` or `STP` in order to determine the initial condition. To illustrate consider a DAE system with 2 differential equations and one algebraic equation. This system has 3 dynamic variables (z-variables), 2 whose derivatives appear explicitly in the model. To specify the initial condition, 2 values of the variables must be set. The three variables are $x1$, $x2$, and u and are declared by

```
ZVAR {x1,x2,u}
```

Recall that this also automatically defines the time derivatives of the z-variables by appending a prime `'` to the names of the variables. Thus, `x1'`, `x2'`, and `u'` are declared as variables. If `x1` and `x2` are the two variables whose derivatives appear explicitly in the model (`u'` is never used), then two of the five variables `x1`, `x2`, `u`, `x1'` and `x2'` must be selected to determine the initial condition. The variables that are selected are provided as a list for the `ISPEC` keyword. For example, if `x1` and `x2` are to be used to specify the initial condition, the statement would be

```
ISPE {x1,x2}
```

and to specify the initial conditions, the statement would be

```
ZINC {1,0,0}
```

Even though only two initial conditions are necessary, `ZINC` must provide three values since there are three z-variables. These two statements together indicate that the variables `x1` and `x2` are to be used to determine the initial condition for the DAE system and that their values are to be fixed at 1 and 0 respectively. Solving for the initial condition will determine the remaining values for `u`, `x1'` and `x2'`. The value of 0 provided for `u` as the third element in the `ZINC` statement is not fixed. However, it is used as a starting point for the solution of the initial condition.

If the values of the derivatives `x1'` and `x2'` are to be used to specify the initial condition, then the following statements are used:

```
ISPE {x1',x2'}

ZINC {1,0,0,0,0,0}
```

This states that the values `x1'` and `x2'` are fixed to determine the initial condition of the DAE system. In this case, the list for `ZINC` has six values. The first 3 correspond the the algebraic values of the z-variables `x1`, `x2`, and `u`. The second three correspond to the time derivatives of the z-variables `x1'`, `x2'`, and `u'`. (Note that a value for `u'` is provided even though it will not appear in the model.) Thus, the initial condition will be determined by fixing the values of `x1'` and `x2'` to 0 and solving for the remaining values, `x1`, `x2`, and `u`. (Note that in the model, `u'` could be used to specify the initial condition. Since this is a purely algebraic variable, an error will occur when trying to determine the initial condition.)

In the modeling of dynamic problems, there often occur situations when z-variables used to specify the initial condition depend on the values of the x-variables. In these cases, the initial conditions have the form

$$z(t_0) = x$$

where $z(t_0)$ is the z-variable whose initial condition is to be set to the value of the x-variable x. Since x is a variable, its value will change throughout the solution algorithm and thus the value for $z(t_0)$ can not be specified. In order to address this situation, MINOPT used two keywords, `ICS` and `ICP` to specify the states (z-variables) whose initial conditions depend on x-variables and the corresponding parameters (x-variables). Consider the same dynamic system as before. Now consider that the problem also has the following x-variables

```
XVAR {p1,p2}
```

If the initial conditions of the DAE system are

$$x1 = p1$$

$$x2 = p2$$

this would be written as

`ISPEC {x1,x2}`

`ICS {x1,x2}`

`ICP {p1,p2}`

This indicates that `x1` and `x2` are used to determine the initial condition of the DAE system and that their values are to be fixed to the values of the x-variables `p1` and `p2` to determine the values of the remaining variables (`u`, `x1'` and `x2'`).

There is a one to one correspondence between the list for `ICS` and `ICP` in that the z-variables listed in `ICS` correspond directly to the x-variables listed in `ICP`. Each list must have the same number of elements.

4.5 The Model Section

The model section is where the objective function and constraints for the model are provided. The section begins with `MODEL {{` and ends with `}}` and contains statements which define the mathematical model.

All of the parameters and variables that were defined in the declarations section are now used to assemble the model for the mathematical program. All of the problem constraints are written along with the objective function using the MINOPT modeling language.

The objective function is defined using the keyword `MIN` (or `MAX`) and has the following construction: `MIN:` *expression*; Where the expression is representation of the objective funtion for the model. The `MIN` keyword indicates that the objective is to be minimized, (while the `MAX` keyword indicates that the objective is to be maximized.)

The constraints for the problem have the form:

identifier: *expression relation expression*;

The *identifier* is the name of the constraint which is used to identify the constraint for the marginals and for problem output. The *expressions* are the mathematical representations of the left hand side and right hand side of the constraint. These expressions are formulated using all of the rules for the constraints which have been described earlier. The one exception is that an enumeration enclosed by the single `<` and `>` can not be used in the constraint expressions. A different construction is used to enumerate a list of constraints. An enumerated constraint is formed by using a comma separated list of index in set constructs. This syntax is described by

identifier(*index* `E` *set*, *index* `E` *set*, ... , *index* `E` *set*): *expression relation expression*;

The *index* `E` *set* constructs activate the index for use in the following *index* `E` *set* constructs as well as in the left hand side and right hand side expressions.

The *relation* each expression can be one of $\leq$, $\geq$, or $=$. These are represent in the MINOPT modeling language as `=l=` for less than or equal to ($\leq$), `=e=`, for equality ($=$) and `=g=` for greater than or equal to ($\geq$).

The model section deals with the symbolic representation of the mathematical model using the MINOPT modeling language. The bulk of the work in creating this section deals with converting the mathematical representation of the problem into the MINOPT language.

The mathematical form of the objective function

$$\min a + pb^2 + \exp(c)$$

is represented in the MINOPT language by the statement

```
MIN: a + p*b^2 + exp[c];
```

Consider the following objective function in mathematical form:

$$\min \sum_{i \in I} q_i p_i$$

The corresponding MINOPT statement is

```
MIN: <<i E I| q(i)*p(i) >>;
```

Consider the constraint

$$\sum_{i \in I} p_i \leq 100$$

This is written in MINOPT as

```
constraint:  <<i E I| p(i) =l= 100
```

The name `constraint` is just an identifier used to name the constraint required in any future references.

To illustrate the use of the enumeration for a constraint, consider the following mathematical form of a constraint:

$$q_i p_i \geq 1 \quad \forall i \in I$$

This is written in MINOPT as

```
product(i E I): q(i)*p(i) =g= 1;
```

More complex statements involving both the enumeration and summation can be formed such as

```
linear(i E I): <<k E K| feed(i,k) >> =e= 1;
```

Expressions can appear on both sides of the expression such as in

```
fuel(f E F): z(f) =g= <<g E G| <<k E K| a(g,f,k)*x(g,f)^k >> >>;
```

Multiple enumerations can occur such as in

```
l1(i E I, j E J): v(j) =g= log[s(i,j)] + b(i);
```

Since the *index* E *set* expression activates the index, the index can be used anywhere in the constraint defnintion after it is activated. The scope of the index extends to the end of the constraint. A constraint that makes use of this is

```
ccap1(j E J, i E IJ(j), n E N): b(i,j,n) =l= capmax(j)*wv(i,n);
```

where the index j is used to select the set IJ(j).

Qualifications can be used in the constraint enumerations by following the *index* E *set* expression with an ampersand & as in

```
ccapc1(j E J, i E I & capmax(i,j) != 0.0 && i<=3, n E N): b(i,j,n) =e=
                capmax(i,j)*(tf(i,j,n) - ts(i,j,n));
```

The following is an example of a complete constraint section:

```
MODEL {{
  MIN: 250*<<j E J| exp[n(j)+0.6*v(j)]>>;
  ldvj(i E I, j E J): v(j) - b(i) =g= lns(i,j);
  ldct(i E I, j E J): n(j) + a(i) =g= lnt(i,j);
  nhc: <<i E I| q(i)*exp[a(i)-b(i)]>> =l= 6000;
  ldnp(j E J): <<k E K| lnk(k)*y(k,j)>> - n(j) =e= 0;
  ldsy(j E J): <<k E K| y(k,j)>> =e= 1;
}}
```

Dynamic models The TIME keyword is important for the specification of the point constraints. Point constraints have a specific input which is reflected in the name of the constraint:

```
name[instance]: expression relation rhs;
```

The instance corresponds to the number of the element in the TIME list. For example, if the TIME is specified as

```
TIME {0, 1.5, 3, 5, 10};
```

five instances are specified and numbered 0 through 4. Thus, a constraint of the form

```
con1[3]: z(31) =l= 0.95;
```

declares that the dynamic variable z(31) should be less than 0.95 at time instance 3 (the end of the 3rd time interval). This corresponds to the time 5. Note that the 3 corresponds to a time instance and not an actual time.

The dynamic constraints are defined in the same way as the other constraints. The difference in these constraints is that the derivative of a dynamic variable is needed. The derivative of a dynamic variable is denoted by using a prime or single quote (“' ”). For example the equation

$$\frac{dz_1}{dt} = x_1 z_2 - x_2 z_1$$

would be represented by

```
z'(1) - x(1)*z(2) + x(2)*z(1) =e= 0;
```

Note that the prime comes before the subscript. Also be aware that all of the dynamic constraints must be equality constraints. If a dynamic constraint is not an equality, MINOPT will produce a warning to this effect and proceed with the constraint as equality. Using the relation `=E=` has no effect and is the same as `=e=`.

MINOPT also recognizes t as the variable for time. This allows for time varying systems where time appears explicitly in the problem formulation. Thus, when using dynamic systems, the user should not use the identifier t for any other purpose.

The integration tolerance is specified indirectly through the `FTOL` option. The value of the integration tolerance is set to 100 times the value of `FTOL` so that numerical problems will not arise.

Row Types and Problem Types MINOPT reads in the problem and determines the type of each constraint and ultimately the problem type. It will determine which constraints are linear, which are nonlinear, and which are dynamic. As specified by the user, MINOPT also classifies the dual constraints and the constraints which are not to be relaxed in the infeasible primal. (The default is that all constraints are relaxed.)

After MINOPT has read the entire file, it knows all of the necessary information regarding the variables and constraints and will then determine the problem type. It will determine the form of the primal and master problems, and also determine the separability of the problem. With this information, MINOPT will determine the appropriate algorithm to use and verify that a selected algorithm can be used to solve the problem.

4.6 The Options Section

One of the features of MINOPT is its extensive list of options which allow the user to fine tune the algorithms and set the parameters for the external solvers. The options section itself is optional and does not need to exist in the input file. It can also appear at any point within the input file as long as it is not within the declarations section or the model section.

The option section begins with `OPTIONS` {{ and ends with }}. There are two types of options: "yes/no" options and parameter options. The "yes/no" options are binary options that are off by default and turned on by specifying the option. The statements for these type of options have the form:

option;

Where *option*is the name of the "yes/no" option that is to be used. Parameter options are options which set the value of a specific parameter. The value of the parameter can be either a number or a string enclosed within quotations (`" "`). The parameter options are set by using the syntax:

option = "*string*";

or

option = *number*;

The value of the *string* or *number* depends of the option being set.

The options and their default values are listed below. The default values for many of the double precision options depends on the value of the precision of the machine being used, ϵ. (The value of ϵ can generally be found in the file `/usr/include/float.h` and for many machines is approximately 2.22×10^{-16}.)

The "yes/no" options:
(These can be specified from the command line using the `-o` option followed by the option name.)

- `WRITEINPUT` Copy the input file to the log file.
- `DUMP` Write the parser version of the model to the file `<input>.dump`. (This can be spceified from the command line using the `-d` option.)
- `DUALOPT` Solve the LP problem using the dual simplex method if it is available with the solver (CPLEX).
- `NETOPT` Solve the LP problem using the network optimizer if it is available (CPLEX).
- `WRITELP` Write the LP problems and subproblems to an output file. (This can be specified from the command line using the `-w` option.)
- `WRITEMPS` Write the LP problems and subproblems to an output file in MPS format.
- `JUSTPRIMAL` Solve the first primal problem and then exit. (This can be spcified from the comman line using the `-j` option.)
- `DORELAX` Solve the relaxed problem and exit. The integrality constraints on the y-variables are relaxed and the problem is solved as an NLP. (This can be specified from the command line using the `-r` option.)
- `AUTOINIT` Solve the relaxed problem and use the solution to determine the starting values for the y-variables. If the solution of the relaxed problem yields integer values for the integer variables, the solution is terminated. Otherwise, the integer y-variables are rounded to the nearest integer and used as the starting point for the y-variables. (This can be specified from the command line using the `-a` option.)
- `RANDOMX` Find a random values for the starting values of the x-variables based on the bounds for the variables. (Make sure all of the x-variables are bounded when using this option. Large values of these variables may result causing numerical problems with the NLP solver.)
- `RANDOMY` Find a random values for the starting values of the y-variables based on the bounds for the variables.
- `INTCUT` Incorporate an integer cut into the **GBD** master problem. An integer cut is included in the outer approximation by default. The integer cut incorporated into the

master problem has the form

$$\begin{aligned}
&\sum_{j \in B^k} y_j - \sum_{j \in N^k} y_j \leq |B^k| - 1 \\
&B^k = \{j | y_j^k = 1\} \\
&N^k = \{j | y_j^k = 0\} \\
&k \in \mathbf{F}
\end{aligned}$$

Only the integer combinations of the y-variables which yield feasible primal solutions need to be included because the infeasible **GBD** cuts have already eliminated the previous combinations of the y-variables which were found to yield no feasible solution for the primal problem. The integer cut can only be performed on binary variables and not on integer variables in general. If the problem contains integer variables, the `INTCUT` option will be ignored.

- `IPRIMAL2` Use an alternative form to solve the infeasible primal problem. This form uses a single scalar α rather than the vectors α^+ and α^-. The alternative formulation is the following:

$$\begin{aligned}
\min_{\boldsymbol{x} \in \boldsymbol{x}, \alpha} \quad & \alpha \\
\text{s.t.} \quad & \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}^K) \leq \alpha \\
& \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{y}^K) = \boldsymbol{z} \\
& \alpha \geq \boldsymbol{z}
\end{aligned}$$

  This formulation does not always yield a feasible solution since the equality constraints are not relaxed. This reformulation can, however, give better lower bounds on **GBD** master problems in some cases.

- `XYMASTER` Solve the **GBD** master problem in terms of both the x and y-variables rather than in the y-variables alone. Constraints which are linear and separable in both the x and y-variables are included in the master problem. This may help obtain a tighter lower bound from the master problem.

- `RFIX` Specify that after each infeasible primal problem is solved, the continuous relaxation problem (with only binary variables being fixed) are used to find a feasible solution in the joint x and y-variable space. This is particularly useful when the y-variables consist of both discrete and continuous variables, and is nonconvex in the joint x and y-variable space. This is a heuristic search.

- `RFIXBOTH` Perform the same options as in `RFIX` for both feasible and infeasible primal problems. For feasible primal problem, this may significantly improve the quality of the solutions.

- `UPDATEX` Update the starting values of the x-variables for the primal problems to the solution of the last primal problem.

- `SIMULATE` Simulate the dynamic part of the problem and then exit. This is usful for generating output for certain values of the x and y-variables.

- `RFP` When solving an MINLP problem, replace the first primal iteration by the relaxed problem and use its solution in the master problem.

The integer options:

- `MAXIT` Set the maximum number of primal/master iterations for the MINLP routines. (This can be specified from the command line using the `-i` option followed by an integer.) (default value = 100.)
- `RETRY` Specify the number of times to retry solving the NLPs if an error occurs.
- `PLEVEL` Set the print level for the amount of information printed to the log file. (This can be specified from the command line using the `-p` option followed by an integer.)
- `SLEVEL` Set the summary level for the amount of information printed to the screen. (This can be specified from the command line using the `-s` option followed by an integer.)
- `IPRINT` Set the print level for the solver of the dynamic part of the problem. Should be a value between 0 (no output) and 9 (most output). (default value = 0)
- `NSTEPS` Set the number of steps to be used in the output for the dynamic system. (default value = 500)

The double options:

- `AMIPGAP` Set the absolute MILP gap tolerance.
- `RMIPGAP` Set the relative MILP gap tolerance.
- `INTTOL` Set the integrality tolerance for MILP problems.
- `FTOL` Set the feasibility tolerance for the subproblem solvers. (default value = $\sqrt{\epsilon}$)
- `OTOL` Set the optimality tolerance for the subproblem solvers. (default value = $\sqrt{\epsilon}\epsilon^{0.8}$)
- `ITOL` Set the integration tolerance for the dynamic solvers. (default value = $0.01\times$ `FTOL`.)
- `SSTOL` Set the tolerance level for the initialization of the DAE system. (default value = $\sqrt{\epsilon}$)
- `EVTOL` Set the event tolerance for detecting discontinuities in the dynamic problems. (default value = $\sqrt{\epsilon}$)
- `MAXSTEP` Set the max step size for the integrator. (default: automatic.)
- `INITSTEP` Set the initial step size for the integrator. (defualt: automatic.)
- `EPSR` Set the relative tolerance for the convergence of the MINLP algorithms. (default value = $\sqrt{\epsilon}$)
- `EPSA` Set the absolute tolerance for the convergence of the MINLP algorithms. (default value = $\sqrt{\epsilon}\epsilon^{0.8}$)

- `BALLT` When `RFIX` is specified, this option specifies that only when the solution of the infeasible primal problem is less than this `BALLT` tolerance that the continuous relaxation problem heuristic search mentioned above is invoked. (default value = 1^{20})

- `BALLR` When `RFIX` is specified, this specifies that the problem is solved within the vicinity of the previous solutions from the infeasible primal problem. If X^U and X^L are the original upper and lower bound of an x-variable, which takes a value of $x^\star$ in the solution of the infeasible primal problems, the new bounds are calculated by

$$(X^U)' = min(X^U, \boldsymbol{x}^\star + \mathrm{BALLR}(X^U - X^L))$$
$$(X^L)' = max(X^L, \boldsymbol{x}^\star - \mathrm{BALLR}(X^U - X^L))$$

  (default value = 1.0)

- `BALLS` Since large value of `BALLR` could lead to an infinite loop, when the continuous relaxation search repeatedly finds the same solution. The value of `BALLR` is decreased by a factor of `BALLS` at every iteration. (default value = 1.0)

The string options:

- `LOGFILE` Specify a name for the log file. The default name for the log file is the name of the input file with a ".log" extension instead of a ".dat" extension.

- `DFORM` Specify the precision of the solutions to be printed. The corresponding string gives print format recognized by C language. The default is "%-12.5lg". Here '%' sign and 'lg' are required because the number to be printed is in double precision. '-' denotes left justified output, '12' denotes the width of the number, and '.5' denotes number of decimal points. For more information on the print format, consult any C language manual such as Kernighan and Ritchie (1988).

- `LP` Set the solver for LP problems and subproblems. Valid arguments for this option are `CPLEX`, `MINOS`, and `NPSOL` for the corresponding solvers are CPLEX4.0, MINOS5.4, and LSSOL. (This can be specified from the command line using the `-L` option followed by the name of the solver.)

- `MIP` Set the solver for MILP problems and subproblems. The only valid argument for this option id `CPLEX` since CPLEX4.0 is currently the only available solver for this type of problem. (This can be specified from the command line using the `-I` option followed by the name of the solver.)

- `NLP` Set the solver for NLP problems and subproblems. The valid arguments for this option are `MINOS`, `NPSOL`, and `SNOPT` for the corresponding solvers MINOS5.4, NPSOL4.2, and SNOPT5.0. (This can be specified from the command line using the `-N` option followed by the name of the solver.)

- `MINLP` Set the MINLP solver. Valid arguments for this option are `GBD`, `OAER`, `OAERAP`, and `GCD`. These correspond to the MINLP algorithms. (This can be specified from the command line using the `-M` option followed by the name of the solver.)

- `DYN` Set the integrator for the problem. The available solvers for the dynamic problems are `DASOLV` and `DASSL`. (This can be specified from the command line using the `-D` option followed by the name of the solver.)

External Solver Options The options for the external solvers can be set by using the name of the solver as the option. These options are parameter options where the parameter is a string corresponding to the appropriate option for the solver. For the appropriate solver options, consult the manual for that particular solver.

For MINOS, SNOPT, and NPSOL, the options consist of the name of the option followed by the parameter value. For example, to set the "Iterations Limit" option in MINOS to a value of 100000, the following option statement is used in MINOPT:

```
MINOS = "Iterations Limit 100000";
```

Similarly this option is set in SNOPT by using

```
SNOPT = "Iterations Limit 100000";
```

Note that MINOPT does not attempt to determine the validity of these options and sends them directly to the solver. The solver will then determine whether or not the option is valid.

For CPLEX, the options are specified by using the name of the option followed by the parameter value. The name of the option can be found in the CPLEX manual and the name used for MINOPT is the name of the option without the preceding "CPX_PARAM_". For example, to set the node selection strategy for CPLEX (CPX_PARAM_NODESEL) to be the best-estimate search, the following option statement is used

```
CPLEX = "NODESEL 2";
```

5 Output

As mentioned in the Usage section, MINOPT produces an output file with the `.log` extension. This contains the solution of each of the subproblems and the progress of the algorithm in terms of the bounds. MINOPT also includes options for generating additional file output and output to the screen. This is useful when the user wants to see the progress of the algorithm as it is occurring and for debugging.

The two key options for specifying the amount of output are the `-p` and `-s` options. The `-p` option can be used to set the print levels for the solvers. Specifyin a value of 4 or greater will cause the NLP solvers to write their output to the FORTRAN file number 9 (`fort.9, ftn9`, etc). The amount of output to these files is controller by the NLP solver. The command line option `-s` is similar to `-p` with the exception that the output is sent to the screen instead of to a file. This can be used to see the progress of the solution of a particular subproblem.

Another useful option that provides output useful for debugging is the `-d` option. This will cause MINOPT to print the problem to the file `<input>.dump`. This output contains the problem information as MINOPT views it.

For dynamic problems, additional files, `<input>.states1`, `<input>.states2`, etc. are generated. These files contain the time output of the dynamic variables. Each file has the time as the first column and the values of the states for each time in the following columns. The states file `<input>.states1` contatins the values for the first 50 z-variables`<input>.states2` the values for the second 50, etc. The columns are in the same order as the z-variables are given in the input file. This makes it easy to import the time responses of the dynamic system into spreadsheets or to plot them using using software packages such as GNUPLOT or xmgr.

6 Simple Illustrations

An excellent way to really get a feeling for the MINOPT modeling language is to examine existing input files. In this section, some simple examples will be used to illustrate some of the key points about the modeling language. One simple illustrations will be given for each of the problem types: LP, MILP, NLP, MINLP, NLP/DAE, Optimal Control, and MINLP/DAE. All of these examples are taken from the CACHE Process Design case Studies (Morari and Grossmann, 1991). The filename in parentheses after the problem name is the name if the file in the CACHE directory of the MINOPT model library.

6.1 Linear Program (`plan.dat`)

This example is a model for production planning in multipurpose batch plants. The general idea is to determine the optimal production routes and their duration for each product. The sets in the problem are the products P, the routes R, and the campaigns C. The scalar paramters are the number of products to be produced nc, the total available plant operation time h, the total number of possible campaigns nc. The other parameters are the number of routes for each product nrp_p $\forall p \in P$, The profits for each product profit_p $\forall p \in P$, the batch sizes of routes b_{pr} $\forall p \in P, r \in R$, the imes requires to produce one batch for routes ct_{pr} $\forall p \in P, r \in R$, and the incidence matrix for the product routes in the campaigns rc_{prc} $\forall p \in P, r \in R, c \in C$. The variables are Total production times assigned to routes t_{pr} $\forall p \in P, r \in R$, the total amounts of products produced q_p $\forall p \in P$, and the lengths of campaigns cl_c $\forall c \in C$.

The model for the problem is

$$
\begin{array}{ll}
\max & \sum\limits_{p \in P} \text{profit}_p q_p \\
\text{subject to} & q(p) = \sum\limits_{r=1}^{nrp_r} c_{pr} t_{pr} / ct_{pr} \quad \forall p \in P(p < np) \\
 & \sum\limits_{c=1}^{nc} cl_c < h \\
 & \sum\limits_{c=1}^{nc} rc_{prc} cl_c > t_{pr} \quad \forall p \leq np \quad \forall r \leq nrp_p
\end{array}
$$

The MINOPT input file is shown below. Note that in order to do the constructions such as $\forall p \in P(p < np)$ the qualification symbol `&` is used: `p E P & (p<=np)`.

```
$**********************************************************
$ Production Planning in Multipurpose Batch Plants
$ Iftekhar A. Karimi
$
$ CACHE Process Design Case Studies
$ M. Morari and I.E. Grossmann
$
$ Optimal Solution:  -454.249
$**********************************************************

DECLARATION {{
  INDEX {p,r,rr,c};
```

```
  SET P = |1:3|; # Products
  SET R = |1:3|;  # Routes
  SET C = |1:20|; # Campaigns

  PARA np = {3};   # Number of products to be produced
  PARA h  = {700}; # Total available plant operation time [h]
  PARA nc = {9};   # Total number of possible campaigns

#Number of routes for products
  PARA nrp(P) = {2, 2, 3};

#Profits of products [$/kg]
  PARA profit(P) = {1.0, 0.8, 0.5};

#Batch sizes of routes [Mg]
  PARA b(P,R) = {2.5 , 2.0 , 0,
                 1.67, 1.33, 0,
                 0.5 , 1.0 , 1.5};

#Times requires to produce one batch for routes [h]
  PARA ct(P,R) = {6.2, 5.4, 1e-10,
                  4.8, 4.0, 1e-10,
                  3.2, 4.9, 6.4};

#Incidence matrix for product routes in campaigns
  PARA rc(P,R,C) = <p E P| <r E R| <c E C| 0 > > >;
  PARA rc(1,1,1) = 1;  PARA rc(1,2,1) = 1;  PARA rc(3,1,1) = 1;
  PARA rc(1,1,2) = 1;  PARA rc(2,2,2) = 1;  PARA rc(3,1,2) = 1;
  PARA rc(1,1,3) = 1;  PARA rc(3,1,3) = 1;  PARA rc(3,2,3) = 1;
  PARA rc(1,2,4) = 1;  PARA rc(2,1,4) = 1;  PARA rc(3,1,4) = 1;
  PARA rc(1,2,5) = 1;  PARA rc(3,1,5) = 1;  PARA rc(3,3,5) = 1;
  PARA rc(2,1,6) = 1;  PARA rc(2,2,6) = 1;  PARA rc(3,1,6) = 1;
  PARA rc(2,1,7) = 1;  PARA rc(3,1,7) = 1;  PARA rc(3,2,7) = 1;
  PARA rc(2,2,8) = 1;  PARA rc(3,1,8) = 1;  PARA rc(3,3,8) = 1;
  PARA rc(3,1,9) = 1;  PARA rc(3,2,9) = 1;  PARA rc(3,3,9) = 1;

  XVAR {t(P,R), # Total production times assigned to routes [h]
        q(P),   # Total amounts of products produced [Mg]
        cl(C)   # Lengths of Campaigns [h]
       };

  POSI  {t(P,R), q(P), cl(C)};

  LBDS q(1) = 100; LBDS q(2) = 250; LBDS q(3) = 150;
  UBDS q(1) = 150; UBDS q(2) = 300; UBDS q(3) = 200;
}}
```

```
MODEL {{
  MIN:  <<p E P & (p<=np)| -1*profit(p)*q(p) >>;

  production(p E P & (p<=np)):
      q(p) =e= <<r E R & (r<=nrp(r))| b(p,r)*t(p,r)/ct(p,r) >>;
  sumcl:  <<c E C & (c<=nc)| cl(c) >> =l= h;
  hconst(p E P & (p<=np), r E R & (r<=nrp(p))):
      <<c E C & (c<=nc)| rc(p,r,c)*cl(c) >> =g= t(p,r);
}}
```

When MINOPT solves the problem, it displays the following statistics:

```
CPLEX Solving Primal Problem.
CPLEX status = 1
CPLEX LP optimal solution found.

=======================
Optimal solution found.
=======================
Objective function: -454.249
Computation time:   0.05
Total CPU time:     0.1
```

MINOPT also generates an output file with the optimal values of the variables. The output file named plan.log is the following:

```
===========================================================================

M I N O P T
Version 3.1, Aug 14 1998
C. A. Schweiger and C. A. Floudas
Department of Chemical Engineering
Princeton University
Princeton, NJ 08544-5263

Copyright (c) 1998 Princeton University
All Rights Reserved

Send bugs, comments, and suggestions to minopt@titan.princeton.edu

===========================================================================
System Information
cronus 9000/780 HP-UX B.10.20 A
===========================================================================

===========================================================================
OPTIMAL SOLUTION FOUND.
             PROBLEM:  LP
              SOLVER:  CPLEX
```

```
              STATUS:  Optimal
        SOLVERSTATUS:  1

     Generation time:  0.01
       Solution time:  0.01
          Total time:  0.02
Number of iterations:  15

Objective function value
------------------------
       Objective: -454.2489

Optimal X Variables
-------------------
          t(1,1): 372.0000              t(1,2): 0.000000
          t(1,3): 0.000000              t(2,1): 154.6667
          t(2,2): 700.0000              t(2,3): 0.000000
          t(3,1): 700.0000              t(3,2): 0.000000
          t(3,3): 173.3333                q(1): 150.0000
            q(2): 286.5611                q(3): 150.0000
           cl(1): 0.000000               cl(2): 372.0000
           cl(3): 0.000000               cl(4): 0.000000
           cl(5): 0.000000               cl(6): 154.6667
           cl(7): 0.000000               cl(8): 173.3333
           cl(9): 0.000000              cl(10): 0.000000
          cl(11): 0.000000              cl(12): 0.000000
          cl(13): 0.000000              cl(14): 0.000000
          cl(15): 0.000000              cl(16): 0.000000
          cl(17): 0.000000              cl(18): 0.000000
          cl(19): 0.000000              cl(20): 0.000000

===========================================================================
```

6.2 Mixed-Integer Linear Program (`complex.dat`)

The following example is simple in that no sets or parameters are used. The problem is the design of a chemical complex which can be manufactured using different processes. The continuous variables are

variable	description
PA	purchases of A [tons/hr]
PB	purchases of B [tons/hr]
SC	sales of C [tons/hr]
BI	production rate of B in process I [tons/hr]
BII	consumption rate of B in process II [tons/hr]
$BIII$	consumption rate of B in process III [tons/hr]
CII	production rate of B in process II [tons/hr]
$CIII$	production rate of B in process III [tons/hr]

The binary variables are

variable	description
YI	whether or not process I is selected
YII	whether or not process I is selected
$YIII$	whether or not process I is selected

The mass balance equations for chemicals B and C are

$$B_I + P_B = B_{II} + B_{III};$$

$$C_{II} + C_{III} = S_C$$

The material balances around each process are

$$B_I = 0.9P_A$$

$$C_{II} = 0.82B_{II}$$

$$C_{III} = 0.95B_{III}$$

The logical constraints are

$$P_A \leq 16Y_I$$

$$B_{II} \leq (10/0.82)Y_{II}$$

$$B_{III} \leq (10/0.95)Y_{III}$$

$$Y_{II} + Y_{III} \leq 1$$

The objective is to maximize the profit defined as sales revenu minus the investment and operating costs and the cost of the raw materials:

$$\begin{array}{ll} \max & 1800S_C - (1000Y_I + 250P_A + 1500 * Y_{II} + 400B_{II} + 2000Y_{III} + 550B_{III}) \\ & -500P_A - 950P_B \end{array}$$

The MINOPT input file is the following:

```
$*************************************************************
$ Design of a Chemical Complex
$ Nikolaos V. Sahinidis and Ignacio E. Grossmann
$
$ CACHE Process Design Case Studies
$ M. Morari and I.E. Grossmann
$
```

```
$ Optimal Solution:  -459.35
$*************************************************************
$
DECLARATION {{
  XVAR {PA,   #purchases of A [tons/hr]
        PB,   #purchases of B [tons/hr]
        SC,   #sales of C [tons/hr]
        BI,   #production rate of B in process I [tons/hr]
        BII,  #consumption rate of B in process II [tons/hr]
        BIII, #consumption rate of B in process III [tons/hr]
        CII,  #production rate of B in process II [tons/hr]
        CIII  #production rate of B in process III [tons/hr]
       };
  XVAR {YI,   #whether or not process I is selected
        YII,  #whether or not process I is selected
        YIII  #whether or not process I is selected
       };
  BINA {YI, YII, YIII};
  POSI {PA, PB, SC, BI, BII, BIII, CII, CIII};
  UBDS SC = {10};
}}

MODEL {{
  MIN:  1000*YI   + 250*PA
      + 1500*YII  + 400*BII
      + 2000*YIII + 550*BIII
      +  500*PA + 950*PB
      - 1800*SC;

#mass balance for chemicals B and C
  Bmassbal:  BI + PB - BII - BIII =e= 0;
  Cmassbal:  CII + CIII - SC =e= 0;

#balances around processes I, II, and III
  Ibal:      BI =e= 0.9*PA;
  IIbal:    CII =e= 0.82*BII;
  IIIbal:  CIII =e= 0.95*BIII;

#logical constraints
  log1:    PA =l= 16*YI;
  log2:   BII =l= (10/0.82)*YII;
  log3:  BIII =l= (10/0.95)*YIII;

  log4:  YII + YIII =l= 1;
}}
```

The output file, `complex.log`, is the following:

```
===========================================================================
```

```
M I N O P T
Version 3.1, Aug 14 1998
C. A. Schweiger and C. A. Floudas
Department of Chemical Engineering
Princeton University
Princeton, NJ 08544-5263

Copyright (c) 1998 Princeton University
All Rights Reserved

Send bugs, comments, and suggestions to minopt@titan.princeton.edu

===========================================================================
System Information
cronus 9000/780 HP-UX B.10.20 A
===========================================================================

===========================================================================
OPTIMAL SOLUTION FOUND.
             PROBLEM:  MILP
              SOLVER:  CPLEX
              STATUS:  Optimal
        SOLVERSTATUS:  101

     Generation time:  0.01
       Solution time:  0.01
          Total time:  0.02
Number of iterations:  7
     Number of nodes:  1

Objective function value
------------------------
       Objective: -459.3496

Optimal X Variables
-------------------
              PA: 13.55014                 PB: 0.000000
              SC: 10.00000                 BI: 12.19512
             BII: 12.19512               BIII: 0.000000
             CII: 10.00000               CIII: 0.000000
              YI: 1.000000                YII: 1.000000
            YIII: 0.000000

===========================================================================
```

6.3 Nonlinear Program (`fueloil.dat`)

This example considers the power generation via fuel oil. The problem has 3 sets: the set of power generators, G, the set of fuels, F, and the set for the constants in the fuel consumption equations. The parameters are the coefficients for the fuel consumption equations, $a_{g,f,k}$ and amount of power output required, p_{req}. The variables are

variable	description
$p_g \quad \forall g \in G$	Total power output of the generators [MW]
$x_{g,f} \quad \forall g \in G f \in F$	Power output of the generators from specific fuels
$z_f \quad \forall f \in F$	Total amount of fuel purchased

The equation for the power requirement is

$$\sum_{g \in G} p_g \geq p_{req}$$

The equations for the power generated are

$$p_g = \sum_{f \in F} x_{g,f} \quad \forall g \in G$$

The fuel usage equations are

$$z_f \geq \sum_{g \in G} \sum_{k \in K} a_{g,f,k} x_{g,f}^k \quad \forall f \in F$$

The objective is to minimize the amount of fuel 1 purchased:

$$\min z_1$$

The MINOPT input file is

```
$*************************************************************
$ Power Generation via Fuel Oil
$ I. A. Karimi
$
$ CACHE Process Design Case Studies
$ M. Morari and I.E. Grossmann
$
$ Optimal Solution:  4.68089
$*************************************************************
$
DECLARATION {{
  INDEX {g,f,k};

  SET G = |1:2|; #Power Generators
  SET F = |1:2|; #Fuels
  SET K = |0:2|; #Constants in Fuel Consumption Equations

#Coefficients in the fuel consumption equations
```

```
  PARA a(G,F,K) = {1.4609, 0.15186, 0.001450,
                  1.5742, 0.16310, 0.001358,
                  0.8008, 0.20310, 0.000916,
                  0.7266, 0.22560, 0.000778};

#Total power output required [MW]
  PARA preq = 50.0;

  XVAR {p(G),   #Total power output of the generators [MW]
        x(G,F), #Power output of the generators from specific fuels
        z(F)    #Total amount of fuel purchased
       };

  POSI {p(G), x(G,F), z(F)};

  LBDS p(G) = {18.0, 14.0};
  STP  p(G) = {20.0, 20.0};
  UBDS p(G) = {30.0, 25.0};

  UBDS z(2) = {10.0};
}}

MODEL {{
  MIN: z(1);

#Power Requirement
  power1:  <<g E G| p(g) >> =g= preq;

#Power generated
  power2(g E G): p(g) =e= <<f E F| x(g,f) >>;

#Fuel usage
  fuel(f E F): z(f) =g= <<g E G| <<k E K| a(g,f,k)*x(g,f)^k >> >>;
}}
```

The MINOPT output file, `fueloil.log`, is the following:

```
===========================================================================

M I N O P T
Version 3.0, Jul 16 1998
C. A. Schweiger and C. A. Floudas
Department of Chemical Engineering
Princeton University
Princeton, NJ 08544-5263

Send bugs, comments, and suggestions to carl@titan.princeton.edu

===========================================================================
```

```
System Information
titan 9000/730 HP-UX B.10.20 A
===========================================================================

===========================================================================
OPTIMAL SOLUTION FOUND.
             PROBLEM:  NLP
              SOLVER:  MINOS
              STATUS:  Optimal
        SOLVERSTATUS:  0

     Generation time:  0
       Solution time:  0.04
          Total time:  0.05
Number of iterations:  33

Objective function value
------------------------
       Objective: 4.680890

Optimal X Variables
-------------------
            p(1): 30.00000                    p(2): 20.00000
          x(1,1): 10.11428                  x(1,2): 19.88572
          x(2,1): 3.561229                  x(2,2): 16.43877
            z(1): 4.680890                    z(2): 10.00000

===========================================================================
```

6.4 Mixed-Integer Nonlinear Program (`batdes.dat`)

This example considers the design of multiproduct batch plants. The goal is to determine the sizes of the units required at each processing stage and the number of units that should be operating in parallel to minimize the investment cost.

The sets in the problem are the sets of products, stages, and potential number of parallel units, I, J, and K respectively. The parameters in the problem are

parameter	description
$q_i \quad \forall i \in I$	demand of product i [kg]
$\alpha_j \quad \forall j \in J$	cost coefficient
$\beta_j \quad \forall j \in J$	cost exponent
$s_{i,j} \quad \forall i \in I, j \in J$	size factor of product i in stage j [L/kg]
$t_{i,j} \quad \forall i \in I, j \in J$	processinig time of product i in stage j [h]

The continuous variables in the problem are

variable	description
$v_j \quad \forall j]inJ$	volume of stage j [L]
$b_i \quad \forall i \in I$	batch size of product i [kg]
$tl_i \quad \forall i \in I$	cycle time of product i [h]
$n_j \quad \forall j \in J$	number of unit in parallel stage j

The binary variables in the problem are the existence of a stage represented by $y_{k,j} \quad \forall k \in K, j \in J$

The volume requirement in stage $j \in J$ is

$$v_j \geq \log(s_{i,j}) + b_i \quad \forall i \in I, j \in J$$

The cycle time for each product $i \in I$ is

$$n_j + tl_i \geq log(t_{i,j}) \quad \forall i \in I, j \in J$$

The constraint for the production time is

$$\sum_{i \in I} q_i \exp(tl_i - b_i) \leq 6000$$

The equation relating number of units to the binary variables is

$$n_j = \sum_{kEK} \log(k) y_{k,j} \quad \forall j \in J$$

The requirement that only one choice for parallel units is feasible is

$$\sum_{kEK} y_{k,j} = 1 \quad \forall j \in J$$

The objective is to minimize the investment cost:

$$\min \sum_{jEJ} \alpha_j exp(n_j + \beta_j v_j)$$

```
$*************************************************************
$ Optimal Design of Multiproduct Batch Plants
$ Ignacio E. Grossmann
$
$ CACHE Process Design Case Studies
$ M. Morari and I.E. Grossmann
$
$ Optimal Solution:  167,428
$*************************************************************
$
DECLARATION {{
  INDEX {i,j,k};

  SET I = |1:2|; #products
  SET J = |1:3|; #stages
```

```
  SET K = |1:3|; #potential number of parallel units

  PARA q(I)     = {200000, 150000}; #demand of product i [kg]
  PARA alpha(J) = {250, 500, 340};  #cost coefficient
  PARA beta(J)  = {0.6, 0.6, 0.6};  #cost exponent
  PARA s(I,J)   = {2,3,4,           #size factor of product
                   4,6,3};          #i in stage j [L/kg]
  PARA t(I,J)   = {8,20,4,          #processing time of product
                   10,12,3};        #i in stage j [h]

  XVAR {v(J),  #volume of stage j [L]
        b(I),  #batch size of product i [kg]
        tl(I), #cycle time of product i [h]
        n(J)   #number of unit in parallel stage j
       };

  LBDS v(J) = <j E J|log[250] >;
  UBDS v(J) = <j E J|log[2500] >;

  LBDS b(I) = <i E I|log[(q(i)*max[j E J|t(i,j)])/(3*6000)]>;
  UBDS b(I) = <i E I|log[q(i) ? min[j E J|2500/s(i,j)]]>;

  LBDS tl(I) = <i E I|log[max[j E J|t(i,j)]/3.0]>;
  UBDS tl(I) = <i E I|log[max[j E J|t(i,j)]]>;

  LBDS n(J) = <j E J|0>;
  UBDS n(J) = <j E J|log[3]>;

  YVAR {y(K,J)}; #existence of stage
  BINA {y(K,J)};
}}

MODEL {{
  MIN:  <<j E J| alpha(j)*exp[n(j) + beta(j)*v(j)] >>;

#Volume requirement in stage j
  l1(i E I, j E J): v(j) =g= log[s(i,j)] + b(i);

#Cycle time for each product i
  l2(i E I, j E J): n(j) + tl(i) =g= log[t(i,j)];

#Constraint for production time
  n1:  <<i E I| q(i)*exp[tl(i) - b(i)] >> =l= 6000;

#Relating number of units to 0-1 variables
  l3(j E J):  n(j) =e= <<k E K| log[k]*y(k,j) >>;

#Only one choice for parallel units is feasible
```

```
  l4(j E J): <<k E K| y(k,j) >> =e= 1;
}}
```

When MINOPT runs this problem, the objective values of the primal and master problems and the bounds for each iteration are displayed to the screen. The output file for this problem, `batdes.log`, displays the solution information for for the primal and master for each iteration. The solution requires six iteration of GBD and the output file is the following:

```
============================================================================

M I N O P T
Version 3.1, Aug 14 1998
C. A. Schweiger and C. A. Floudas
Department of Chemical Engineering
Princeton University
Princeton, NJ 08544-5263

Copyright (c) 1998 Princeton University
All Rights Reserved

Send bugs, comments, and suggestions to minopt@titan.princeton.edu

============================================================================
System Information
cronus 9000/780 HP-UX B.10.20 A
============================================================================

============================================================================
ITERATION #1: PRIMAL PROBLEM

             PROBLEM:  Infeasible Primal
              SOLVER:  MINOS
              STATUS:  Optimal
        SOLVERSTATUS:  0
     Generation time:  0
       Solution time:  0.02
          Total time:  0.02
Number of iterations:  93

Objective function value
------------------------
       Objective: 0.9400073

Fixed Values of Y Variables
---------------------------
          y(1,1): 1.000000              y(1,2): 1.000000
          y(1,3): 1.000000              y(2,1): 0.000000
          y(2,2): 0.000000              y(2,3): 0.000000
```

```
         y(3,1): 0.000000              y(3,2): 0.000000
         y(3,3): 0.000000

Optimal Values of X Variables
-----------------------------
           v(1): 7.418581                v(2): 7.824046
           v(3): 7.824046                b(1): 6.437752
           b(2): 6.032287               tl(1): 2.238047
          tl(2): 2.120264                n(1): 0.000000
           n(2): 0.3646431               n(3): 0.000000

Variable Marginals
------------------
           v(1): 0.000000                v(2): 1.000000
           v(3): 1.000000                b(1): 0.000000
           b(2): 0.000000               tl(1): 0.000000
          tl(2): 3.096601e-11            n(1): 0.000000
           n(2): 0.000000                n(3): 0.000000

Constraint Marginals
--------------------
          l1(0): 0.000000               l1(1): 0.000000
          l1(2): -1.000000              l1(3): 0.000000
          l1(4): -1.000000              l1(5): 0.000000
          l2(0): 0.000000               l2(1): -1.000000
          l2(2): 0.000000               l2(3): -1.000000
          l2(4): 0.000000               l2(5): 0.000000
          l3(0): 1.000000               l3(1): 1.000000
          l3(2): 0.000000               n1(0): 0.0003333333

===========================================================================
===========================================================================
ITERATION #1: MASTER PROBLEM

           PROBLEM:  GBD Master
            SOLVER:  CPLEX
            STATUS:  Optimal
      SOLVERSTATUS:  101
   Generation time:  0
     Solution time:  0.01
        Total time:  0.01
Number of iterations:  2

Objective function value
------------------------
       Objective: -1.000000e+10

Optimal values of Y variables
```

```
-----------------------------
          y(1,1): 0.000000             y(1,2): 1.000000
          y(1,3): 1.000000             y(2,1): 0.000000
          y(2,2): 0.000000             y(2,3): 0.000000
          y(3,1): 1.000000             y(3,2): 0.000000
          y(3,3): 0.000000

============================================================================
GBD Iteration #1
GBD Current Objval (-1e+10:0.940007)
GBD Current Bounds (-1e+10:1.79769e+308)
GBD Relative Error  ++
GBD Absolute Error  1.79769e+308

============================================================================
ITERATION #2: PRIMAL PROBLEM

             PROBLEM:  Infeasible Primal
              SOLVER:  MINOS
              STATUS:  Optimal
        SOLVERSTATUS:  0
     Generation time:  0
       Solution time:  0.04
          Total time:  0.04
Number of iterations:  138

Objective function value
------------------------
       Objective: 0.5803517

Fixed Values of Y Variables
---------------------------
          y(1,1): 0.000000             y(1,2): 1.000000
          y(1,3): 1.000000             y(2,1): 0.000000
          y(2,2): 0.000000             y(2,3): 0.000000
          y(3,1): 1.000000             y(3,2): 0.000000
          y(3,3): 0.000000

Optimal Values of X Variables
-----------------------------
            v(1): 7.418581               v(2): 7.824046
            v(3): 7.824046               b(1): 6.437752
            b(2): 6.032287              tl(1): 2.415381
           tl(2): 1.904555               n(1): 1.098612
            n(2): 0.5803517              n(3): 0.000000

Variable Marginals
------------------
```

```
           v(1): 0.000000              v(2): 0.4029851
           v(3): 0.5970149             b(1): 1.657308e-11
           b(2): 1.118761e-11         tl(1): -1.657308e-11
          tl(2): -1.118761e-11         n(1): 0.000000
           n(2): 0.000000              n(3): 0.000000

Constraint Marginals
--------------------
          l1(0): 0.000000             l1(1): 0.000000
          l1(2): -0.5970149           l1(3): 0.000000
          l1(4): -0.4029851           l1(5): 0.000000
          l2(0): 0.000000             l2(1): -0.5970149
          l2(2): 0.000000             l2(3): 0.000000
          l2(4): -0.4029851           l2(5): 0.000000
          l3(0): 0.000000             l3(1): 1.000000
          l3(2): 0.000000             n1(0): 0.0001666667

===========================================================================
===========================================================================
ITERATION #2: MASTER PROBLEM

             PROBLEM:  GBD Master
              SOLVER:  CPLEX
              STATUS:  Optimal
        SOLVERSTATUS:  101
     Generation time:  0
       Solution time:  0
          Total time:  0
Number of iterations:  2

Objective function value
------------------------
       Objective: -1.000000e+10

Optimal values of Y variables
-----------------------------
          y(1,1): 1.000000           y(1,2): 0.000000
          y(1,3): 1.000000           y(2,1): 0.000000
          y(2,2): 0.000000           y(2,3): 0.000000
          y(3,1): 0.000000           y(3,2): 1.000000
          y(3,3): 0.000000

===========================================================================
GBD Iteration #2
GBD Current Objval (-1e+10:0.580352)
GBD Current Bounds (-1e+10:1.79769e+308)
GBD Relative Error  ++
GBD Absolute Error  1.79769e+308
```

```
=============================================================================
ITERATION #3: PRIMAL PROBLEM

             PROBLEM:  Infeasible Primal
              SOLVER:  MINOS
              STATUS:  Optimal
        SOLVERSTATUS:  0
     Generation time:  0
       Solution time:  0.03
          Total time:  0.03
Number of iterations:  112

Objective function value
------------------------
       Objective: 0.02631731

Fixed Values of Y Variables
---------------------------
          y(1,1): 1.000000            y(1,2): 0.000000
          y(1,3): 1.000000            y(2,1): 0.000000
          y(2,2): 0.000000            y(2,3): 0.000000
          y(3,1): 0.000000            y(3,2): 1.000000
          y(3,3): 0.000000

Optimal Values of X Variables
-----------------------------
            v(1): 7.418581              v(2): 7.824046
            v(3): 7.824046              b(1): 6.437752
            b(2): 6.032287             tl(1): 2.053124
           tl(2): 2.276268              n(1): 0.02631731
            n(2): 1.098612              n(3): 0.000000

Variable Marginals
------------------
            v(1): 0.000000              v(2): 0.5844156
            v(3): 0.4155844             b(1): 1.230566e-11
            b(2): 1.730471e-11         tl(1): -1.230560e-11
           tl(2): -1.730471e-11          n(1): 0.000000
            n(2): 0.000000              n(3): 0.000000

Constraint Marginals
--------------------
           l1(0): 0.000000             l1(1): 0.000000
           l1(2): -0.4155844           l1(3): 0.000000
           l1(4): -0.5844156           l1(5): 0.000000
           l2(0): -0.4155844           l2(1): 0.000000
           l2(2): 0.000000             l2(3): -0.5844156
```

```
          l2(4): 0.000000              l2(5): 0.000000
          l3(0): 1.000000              l3(1): 0.000000
          l3(2): 0.000000              n1(0): 0.0001666667

============================================================================
============================================================================
ITERATION #3: MASTER PROBLEM

             PROBLEM:  GBD Master
              SOLVER:  CPLEX
              STATUS:  Optimal
        SOLVERSTATUS:  101
     Generation time:  0
       Solution time:  0.01
          Total time:  0.01
Number of iterations:  4

Objective function value
------------------------
       Objective: -1.000000e+10

Optimal values of Y variables
-----------------------------
          y(1,1): 0.000000              y(1,2): 0.000000
          y(1,3): 1.000000              y(2,1): 0.000000
          y(2,2): 0.000000              y(2,3): 0.000000
          y(3,1): 1.000000              y(3,2): 1.000000
          y(3,3): 0.000000

============================================================================
GBD Iteration #3
GBD Current Objval (-1e+10:0.0263173)
GBD Current Bounds (-1e+10:1.79769e+308)
GBD Relative Error  ++
GBD Absolute Error  1.79769e+308

============================================================================
ITERATION #4: PRIMAL PROBLEM

             PROBLEM:  Primal
              SOLVER:  MINOS
              STATUS:  Optimal
        SOLVERSTATUS:  0
     Generation time:  0
       Solution time:  0
          Total time:  0
Number of iterations:  2
```

```
Objective function value
------------------------
       Objective: 181201.7

Fixed Values of Y Variables
---------------------------
          y(1,1): 0.000000             y(1,2): 0.000000
          y(1,3): 1.000000             y(2,1): 0.000000
          y(2,2): 0.000000             y(2,3): 0.000000
          y(3,1): 1.000000             y(3,2): 1.000000
          y(3,3): 0.000000

Optimal Values of X Variables
-----------------------------
            v(1): 6.738679               v(2): 7.144144
            v(3): 7.431826               b(1): 6.045532
            b(2): 5.352385              tl(1): 1.897120
           tl(2): 1.386294               n(1): 1.098612
            n(2): 1.098612               n(3): 0.000000

Variable Marginals
------------------
            v(1): 0.000000               v(2): 7.275958e-12
            v(3): 0.000000               b(1): 1.055487e-07
            b(2): 9.499854e-08          tl(1): -1.055669e-07
           tl(2): -9.500218e-08           n(1): 0.000000
            n(2): 0.000000               n(3): 0.000000

Constraint Marginals
--------------------
           l1(0): 0.000000              l1(1): -39594.07
           l1(2): -17627.50             l1(3): -25654.00
           l1(4): -25845.42             l1(5): 0.000000
           l2(0): 0.000000              l2(1): -57221.58
           l2(2): 0.000000              l2(3): 0.000000
           l2(4): -51499.42             l2(5): 0.000000
           l3(0): -42756.67             l3(1): -344.8195
           l3(2): -29379.17             n1(0): 18.12017

===========================================================================
===========================================================================
ITERATION #4: MASTER PROBLEM

             PROBLEM:  GBD Master
              SOLVER:  CPLEX
              STATUS:  Optimal
        SOLVERSTATUS:  101
     Generation time:  0
```

```
       Solution time:  0.01
          Total time:  0.01
Number of iterations:  11
     Number of Nodes:  7

Objective function value
------------------------
       Objective: 163725.5

Optimal values of Y variables
-----------------------------
          y(1,1): 0.000000             y(1,2): 0.000000
          y(1,3): 1.000000             y(2,1): 1.000000
          y(2,2): 1.000000             y(2,3): 0.000000
          y(3,1): 0.000000             y(3,2): 0.000000
          y(3,3): 0.000000

============================================================================
GBD Iteration #4
GBD Current Objval (163726:181202)
GBD Current Bounds (163726:181202)
GBD Relative Error  0.101332
GBD Absolute Error  17476.2

============================================================================
ITERATION #5: PRIMAL PROBLEM

             PROBLEM:  Primal
              SOLVER:  MINOS
              STATUS:  Optimal
        SOLVERSTATUS:  0
     Generation time:  0
       Solution time:  0.01
          Total time:  0.01
Number of iterations:  3

Objective function value
------------------------
       Objective: 167427.7

Fixed Values of Y Variables
---------------------------
          y(1,1): 0.000000             y(1,2): 0.000000
          y(1,3): 1.000000             y(2,1): 1.000000
          y(2,2): 1.000000             y(2,3): 0.000000
          y(3,1): 0.000000             y(3,2): 0.000000
          y(3,3): 0.000000
```

```
Optimal Values of X Variables
-----------------------------
           v(1): 7.159070                v(2): 7.564535
           v(3): 7.824046                b(1): 6.437752
           b(2): 5.772775               tl(1): 2.302585
          tl(2): 1.791759                n(1): 0.6931472
           n(2): 0.6931472               n(3): 0.000000

Variable Marginals
------------------
           v(1): 0.000000                v(2): 0.000000
           v(3): 0.000000                b(1): 67011.99
           b(2): 6.548362e-11           tl(1): 0.000000
          tl(2): -7.275958e-11           n(1): 0.000000
           n(2): 0.000000                n(3): 0.000000

Constraint Marginals
--------------------
          l1(0): 0.000000               l1(1): 0.000000
          l1(2): -22304.59              l1(3): -22009.39
          l1(4): -56142.62              l1(5): 0.000000
          l2(0): 0.000000               l2(1): -89316.58
          l2(2): 0.000000               l2(3): 0.000000
          l2(4): -78152.01              l2(5): 0.000000
          l3(0): -36682.31              l3(1): 73897.55
          l3(2): -37174.31              n1(0): 27.91143

============================================================================
============================================================================
ITERATION #5: MASTER PROBLEM

             PROBLEM:  GBD Master
              SOLVER:  CPLEX
              STATUS:  Optimal
        SOLVERSTATUS:  101
     Generation time:  0
       Solution time:  0
          Total time:  0
Number of iterations:  12
     Number of Nodes:  6

Objective function value
------------------------
       Objective: 163865.3

Optimal values of Y variables
-----------------------------
          y(1,1): 0.000000             y(1,2): 0.000000
```

```
         y(1,3): 1.000000            y(2,1): 1.000000
         y(2,2): 0.000000            y(2,3): 0.000000
         y(3,1): 0.000000            y(3,2): 1.000000
         y(3,3): 0.000000

============================================================================
GBD Iteration #5
GBD Current Objval (163865:167428)
GBD Current Bounds (163865:167428)
GBD Relative Error  0.0215056
GBD Absolute Error  3562.33

============================================================================
ITERATION #6: PRIMAL PROBLEM

             PROBLEM:  Primal
              SOLVER:  MINOS
              STATUS:  Optimal
        SOLVERSTATUS:  0
     Generation time:  0
       Solution time:  0
          Total time:  0.01
Number of iterations:  1

Objective function value
------------------------
       Objective: 178545.2

Fixed Values of Y Variables
---------------------------
         y(1,1): 0.000000            y(1,2): 0.000000
         y(1,3): 1.000000            y(2,1): 1.000000
         y(2,2): 0.000000            y(2,3): 0.000000
         y(3,1): 0.000000            y(3,2): 1.000000
         y(3,3): 0.000000

Optimal Values of X Variables
-----------------------------
           v(1): 6.850597              v(2): 7.256062
           v(3): 7.543744              b(1): 6.157450
           b(2): 5.464303             tl(1): 1.897120
          tl(2): 1.609438              n(1): 0.6931472
           n(2): 1.098612              n(3): 0.000000

Variable Marginals
------------------
           v(1): 0.000000              v(2): 0.000000
           v(3): 0.000000              b(1): 9.047679e-06
```

```
           b(2): 1.017866e-05          tl(1): -9.047690e-06
          tl(2): -1.017866e-05           n(1): 0.000000
           n(2): 0.000000               n(3): 0.000000

Constraint Marginals
--------------------
          l1(0): 0.000000              l1(1): -31560.91
          l1(2): -18851.85             l1(3): -18290.56
          l1(4): -38423.79             l1(5): 0.000000
          l2(0): 0.000000              l2(1): -50412.76
          l2(2): 0.000000              l2(3): -56714.36
          l2(4): 0.000000              l2(5): 0.000000
          l3(0): 26230.08              l3(1): -66228.41
          l3(2): -31419.75             n1(0): 17.85452

===========================================================================
===========================================================================
ITERATION #6: MASTER PROBLEM

             PROBLEM:  GBD Master
              SOLVER:  CPLEX
              STATUS:  Optimal
        SOLVERSTATUS:  101
     Generation time:  0
       Solution time:  0.01
          Total time:  0.01
Number of iterations:  11
     Number of Nodes:  6

Objective function value
------------------------
       Objective: 167427.7

Optimal values of Y variables
-----------------------------
         y(1,1): 0.000000             y(1,2): 0.000000
         y(1,3): 1.000000             y(2,1): 1.000000
         y(2,2): 1.000000             y(2,3): 0.000000
         y(3,1): 0.000000             y(3,2): 0.000000
         y(3,3): 0.000000

===========================================================================
GBD Iteration #6
GBD Current Objval (167428:178545)
GBD Current Bounds (167428:167428)
GBD Relative Error  6.95317e-16
GBD Absolute Error  -1.16415e-10
```

```
==============================================================================
OPTIMAL SOLUTION FOUND AT ITERATION #5

             PROBLEM:  MINLP
              SOLVER:  GBD
              STATUS:  Optimal
        SOLVERSTATUS:  0

         PRIMAL time:  0.15
         MASTER time:  0.04
     Generation time:  0
       Solution time:  0.16
          Total time:  0.2
Number of iterations:  6

Objective function value
------------------------
       Objective: 167427.7

Optimal X Variables
-------------------
            v(1): 7.159070                v(2): 7.564535
            v(3): 7.824046                b(1): 6.437752
            b(2): 5.772775               tl(1): 2.302585
           tl(2): 1.791759                n(1): 0.6931472
            n(2): 0.6931472               n(3): 0.000000

Optimal Y Variables
-------------------
          y(1,1): 0.000000              y(1,2): 0.000000
          y(1,3): 1.000000              y(2,1): 1.000000
          y(2,2): 1.000000              y(2,3): 0.000000
          y(3,1): 0.000000              y(3,2): 0.000000
          y(3,3): 0.000000

==============================================================================
```

To switch to OAER, the option `-MOAER` can be added to the command line. To solve the relaxed problem and use this to determing the starting values for the $\boldsymbol{y}$ variables, the option `-a` can be added to the command line.

6.5 Nonlinear Program with Differential and Algebraic Constraints (`car1.dat`)

To demonstrate using MINOPT to solve dynamic problems, a simple car problem is considered. The problem is to minimize the the time required to cover a fixed distance. The model for this problem is

$$\frac{dx_1}{dt} = \tau x_2$$

$$\frac{dx_2}{dt} = \tau u$$

where x_1 is the distance, x_2 is the velocity, u is the acceleration which is the control variable, and τ is a scaling factor for the time. The control is parameterized using a simple quadratic polynomial expression:

$$u = k_1 + k_2 * t + k_3 * t^2$$

where k_1, k_2, and k_3 are the control parameters. The above three equatations are the DAEs for the problem. The limits on the integration are 0 and 1 so that the scale factor τ represents the total time covered by the integration. This allows for the total time required to vary and to be optimized. The total distance to be covered is 300 at which the velocity must be 0. These are represented by point constraints:

$$x_1(t_f) = 300$$

$$x_2(t_f) = 0$$

The control is bounded between -2 and 1:

$$-2 \leq u(t) \leq 1$$

The objective is to minimize the total time:

$$\min \tau$$

The MINOPT inpout file for this problem is the following:

```
$**************************************************************
$ Minimum Time to Cover a Fixed Distance
$ L. T. Biegler and I. B. Tjoa
$
$ CACHE Process Design Case Studies
$ M. Morari and I.E. Grossmann
$
$ Optimal Solution:  34.641
$**************************************************************
$
OPTION {{
  FTOL = "1e-4";
  OTOL = "1e-4";
}}
```

```
DECLARATION {{
  XVAR {k1,k2,k3,tau};
  LBDS tau = {0.1};
  XSTP {1,1,1,20};

  ZVAR {x1,x2,u};
  ISPE {x1,x2};
  ZINC {0,0,0};
  TIME {0,1.0};
}}

MODEL {{
  MIN:  tau;
  dae1: x1' =e= tau*x2;
  dae2: x2' =e= tau*u;
  dae3: u =e= k1 + k2*t + k3*t^2;
  con[0]: u =l= 1;
  con[0]: u =g= -2;
  con[1]: u =l= 1;
  con[1]: u =g= -2;
  con[1]: x1 =e= 300;
  con[1]: x2 =e= 0;
}}
```

The oputput file for this problem, `car1.log`, not only displays the information for the x variables but the values of the z variables at the time instances. Since the solution trajectories for the dynamic problem are often required, these are provided in the file `car1.states1`. The first column of the file is the time and the remaining columns correspond to the z variables. This facilitates using the file in spreadsheets, or plotting programs such as gnuplot or xmgr. For example, the velocity profile can be easily plotted in gnuplot using the command `plot "car1.states" u 1:3 w l`.

```
===========================================================================

M I N O P T
Version 3.1, Aug 14 1998
C. A. Schweiger and C. A. Floudas
Department of Chemical Engineering
Princeton University
Princeton, NJ 08544-5263

Copyright (c) 1998 Princeton University
All Rights Reserved

Send bugs, comments, and suggestions to minopt@titan.princeton.edu

===========================================================================
System Information
```

```
cronus 9000/780 HP-UX B.10.20 A
============================================================================

============================================================================
OPTIMAL SOLUTION FOUND.
             PROBLEM:  NLP
              SOLVER:  SNOPT
              STATUS:  Optimal
        SOLVERSTATUS:  0

     Generation time:  0
       Solution time:  0.15
          Total time:  0.16
Number of iterations:  11

Objective function value
------------------------
       Objective: 34.64102

Optimal X Variables
-------------------
              k1: 1.000000                    k2: 3.875857e-07
              k3: -3.000000                  tau: 34.64102

Optimal Values of Z Variables at time instances
-----------------------------------------------
time instance 0, time=0
------------------------------------
              x1: 0.000000                    x2: 0.000000
               u: 1.000000                   x1': 0.000000
             x2': 34.64102                    u': 0.000000
time instance 1, time=1
------------------------------------
              x1: 300.0000                    x2: -6.729079e-07
               u: -2.000000                  x1': 1.113916e-05
             x2': -69.28203                   u': -6.000000

============================================================================
```

6.6 Optimal Control Problem (car2.dat)

In this example, the same car problem above is considered. However, the control parameterization is changed to be piecewise continuous

$$u = k(i) + (k(i+1) - k(i)) * (t - \bar{t}(i))/(\bar{t}(i+1) - \bar{t}(i))]$$

where $\bar{t}$ are the time instances used for the control parameterization. The length of each control interval is individually scaled using the scale factor τ_i. The model is

$$\frac{dx_1}{dt} = \tau_i x_2$$

$$\frac{dx_2}{dt} = \tau_i u$$

$$u = k_i + (k_{i+1} - k_i)(t - \bar{t}_i)/(\bar{t}_{i+1} - \bar{t}_i)$$

where t is now the scaled time and k_i are the control parameters. The index i is the index for the control interval. The MINOPT interval function (`interv`) is required for the parts of the model that change from one interval to the next. It has the form:

`interv[ i E CI|` *expression* `]`

where CI is the set of control intervals. The expression inside the functions depends on the index i so that is has a different form in each interval. The MINOPT input file is the following:

```
$************************************************************
$ Minimum Time to Cover a Fixed Distance
$ L. T. Biegler and I. B. Tjoa
$
$ CACHE Process Design Case Studies
$ M. Morari and I.E. Grossmann
$
$ Optimal Solution:  30.0
$************************************************************
$
OPTION {{
  EVTOL = "1e-9";
  FTOL = "1e-4";
  OTOL = "1e-4";
}}

DECLARATION {{
  INDEX {i};
  SET CP = |0:5|; #Control Points
  SET CI = |0:4|; #control Intervals

  XVAR {k(CP),   #control parameters
        tau(CI)  #scaling factor for each interval
       };

  LBDS tau(CI) = <i E CI| 0 >;
  STP  tau(CI) = <i E CI| 1 >;
  UBDS tau(CI) = <i E CI| 100 >;
```

```
  LBDS k(CP) = <i E CP| -2 >;
  STP  k(CP) = <i E CP|  1 >;
  UBDS k(CP) = <i E CP|  1 >;

  ZVAR {x1,x2,u};
  ISPE {x1,x2};
  ZINC {0,0,0};

  PARA time(CP) = {0,1,2,3,4,5};
  TIME            {0,1,2,3,4,5};
}}

MODEL {{
  MIN:  <<i E CI| tau(i)>>;
  dae1: x1' =e= interv[i E CI| tau(i)*x2];
  dae2: x2' =e= interv[i E CI| tau(i)*u];
  dae3: u =e= interv[i E CI| k(i) + (k(i+1)
                      - k(i))*(t-time(i))/(time(i+1)-time(i))];
  con5[5]: x1 =e= 300;
  con6[5]: x2 =e= 0;
}}
```

The output file for this problem is

```
===========================================================================

M I N O P T
Version 3.1, Aug 14 1998
C. A. Schweiger and C. A. Floudas
Department of Chemical Engineering
Princeton University
Princeton, NJ 08544-5263

Copyright (c) 1998 Princeton University
All Rights Reserved

Send bugs, comments, and suggestions to minopt@titan.princeton.edu

===========================================================================
System Information
cronus 9000/780 HP-UX B.10.20 A
===========================================================================

===========================================================================
OPTIMAL SOLUTION FOUND.
             PROBLEM:  NLP
              SOLVER:  SNOPT
              STATUS:  Optimal
```

```
        SOLVERSTATUS:  0

     Generation time:  0
       Solution time:  1.83
          Total time:  1.85
Number of iterations:  78

Objective function value
------------------------
       Objective: 30.00002

Optimal X Variables
-------------------
            k(0): 1.000000              k(1): 1.000000
            k(2): 1.000000              k(3): 1.000000
            k(4): -2.000000             k(5): -2.000000
          tau(0): 0.004758614         tau(1): 6.178884
          tau(2): 13.81330            tau(3): 0.006171572
          tau(4): 9.996910

Optimal Values of Z Variables at time instances
-----------------------------------------------
time instance 0, time=0
------------------------------------
              x1: 0.000000                x2: 0.000000
               u: 1.000000               x1': 0.000000
             x2': 0.004758614             u': 0.000000
time instance 1, time=1
------------------------------------
              x1: 1.133106e-05            x2: 0.004758614
               u: 1.000000               x1': 3.490376e-06
             x2': 0.0009372355            u': 0.000000
time instance 2, time=2
------------------------------------
              x1: 19.11872                x2: 6.183643
               u: 1.000000               x1': 1.905808
             x2': 0.3257020               u': 0.000000
time instance 3, time=3
------------------------------------
              x1: 199.9389                x2: 19.99694
               u: 1.000000               x1': 109.5693
             x2': 6.586856                u': 0.000000
time instance 4, time=4
------------------------------------
              x1: 200.0623                x2: 19.99382
               u: -2.000000              x1': 0.06096550
             x2': -0.004458601            u': -1.482132
time instance 5, time=5
```

```
------------------------------------
             x1: 300.0000                      x2: -1.120439e-06
              u: -2.000000                    x1': 44.70986
            x2': -19.99382                     u': 0.000000

============================================================================
```

6.7 Mixed-Integer Nonlinear Program with Differential and Algebraic Constraints (bindis1.dat)

This example considers a simple ideal binary distillation column with no tray hydraulics. The objective is the minimize the Integral of the Squared Error (ISE) between the bottoms and distillate compositions and their respective set-points.

The variables for this problem are

Variable	Desription
x_i	liquid composition on tray i
y_i	vapor composition on tray i
z	feed composition
l_i	liquid flow through tray i
v	vapor boilup in the column
r	reflux to tray i

The parameters for this problem are

Parameter	Description	value
f_i	feed flowrate	1
α	relative volatility	2.5
x_b^*	bottoms composition set-point	0.02
x_d^*	distillate composition set-point	0.98
m_i	liquid holdup on tray i	0.175

For a problem with 30 trays, $N = 30$, there are 64 x-variables, 60 y-variables, 98 z-variables, and 168 constraints. The binary variables p and q denote the position of the feed and the reflux respectively.

The mathematical model for the problem is the following (N is the total number of trays.)

$$
\begin{array}{ll}
\min & \int_{t_0}^{t_f} [(x_0 - x_b^*)^2 + (x_N - x_d^*)^2] dt \\
\text{s.t.} & m_b \frac{dx_0}{dt} + v y_0 + b x_0 = l_1 x_1 \\
& m_i \frac{dx_i}{dt} + l_i x_i + v y_i = l_{i+1} x_{i+1} + v y_{i-1} + y_i^f f z + y_i^r r x_N \quad \forall i < N \\
& m_i \frac{dx_i}{dt} + l_N x_N + v y_N = v y_{N-1} + y_N^f f z + y_N^r r x_N \\
& m_d \frac{dx_N}{dt} + v x_N = v y_{N-1} \\
& v + b = l_1 \\
& l_i = l_{i+1} + y_i^f f + y_i^r r \quad \forall i < N \\
& l_N = y_N^f f + y_N^r r \\
& r + d = v \\
& \alpha x_0 = y_0 (1 + x_0 (\alpha - 1)) \\
& \alpha x_i = y_i (1 + x_i (\alpha - 1)) \quad \forall i \in T \\
& x_0(t_f) \le x_b^s tar \\
& x_N(t_f) \ge x_d^s tar \\
& \sum_{i=1}^{N} y_i^f = 1 \\
& \sum_{i=1}^{N} y_i^r = 1 \\
& \sum_{i=1}^{N} i y_i^r - i y_i^f \ge 0 \\
& z = 0.54 - 0.09 e^{-10*t}
\end{array}
$$

In this problem, the initial condition is that of steady state for the given values of the x-variables. In order to specify this, the keyword ISPE is used and all of the values are the pointers of the dynamic variables whose derivatives appear explicitly plus the number of z-variables. This specifies that all of the derivative variables are 0 at the initial time which is that of steady state. The dynamics in the system are induced by the step disturbance in the feed composition. The expression

$$z = 0.54 - 0.09 e^{-10*t}$$

is used to approximate a step disturbance.

The MINOPT input file is

```
$*********************************************************
$ Dynamic Optimization of a Binary Distillation Column
$ C. A. Schweiger and C. A. Floudas
$
$ ~Step disturbance in feed composition
$
$ Optimal Solution:
$*********************************************************

OPTION {{
  INTCUT;
  FTOL = "1e-6";
```

```
  OTOL = "1e-6";
}}

DECLARATION {{
  INDEX {i};
  SET  T  = |1:30|; #Trays
  SET  T1 = |0:31|; #Reboiler, Trays, and Condenser

  PARA alpha = 2.5;
  PARA N = 30; #possible number of trays
  PARA m = 0.175;
  PARA xdset = {0.98};
  PARA xbset = {0.02};
  PARA f = {1}; # Feed flowrate;

  XVAR {r, #reflux flowrate to each tray
        v,    #vapor boilup,
        mu
       };

  YVAR {yf(T), #existence of feed to each tray
        yr(T)  #existence of reflux to each tray
       };

  ZVAR {l(T), #liquid flowrate from each tray
        x(T1), #liquid composition on each tray
        y(T1), #vapor composition on each tray
        d,     #distillate flowrate
        b,     #bottoms flowrate
        obj    #objective integral
       };

  BINA {yf(T),yr(T)};
  POSI {r, v};

  LBDS r = 0;
  STP  r = 9.5;
  UBDS r = 100;

  LBDS v = {0.1};
  STP  v = {10};
  UBDS v = {100};

  STP yf(T) = <i E T| 0 + (i==15)*1 >;
  STP yr(T) = <i E T| 0 + (i==30)*1 >;

  ISPE {obj, x'(T1)};
```

```
  TIME {0,400};
}}

MODEL {{
  MIN: mu;

#Objective (Integral of the Squared Error)
  obj:   obj' - (x(31)-xdset)^2 - (x(0)-xbset)^2 =e= 0;

#Reboiler Component Balance
  dae1:  10*m*x'(0) + v*y(0) + b*x(0) =e= l(1)*x(1);

#Tray component balances
  dae2(i E T & i<N ):  m*x'(i) + l(i)*x(i) + v*y(i)
                   =e= l(i+1)*x(i+1) + v*y(i-1)
                     + yf(i)*f*(0.54 - 0.09*exp[-10*t])
                     + yr(i)*r*x(31);
  dae2(i E T & i==N):  m*x'(i) + l(i)*x(i) + v*y(i)
                   =e=                 v*y(i-1)
                     + yf(i)*f*(0.54 - 0.09*exp[-10*t])
                     + yr(i)*r*x(31);

#Condenser Component balance
  dae3:  10*m*x'(31) + v*x(31) =e= v*y(30);

#Reboiler Total Balance
  dae4:  v + b =e= l(1);

#Tray total balances
  dae5(i E T & i<N ): l(i) =e= l(i+1) + yf(i)*f + yr(i)*r;
  dae5(i E T & i==N): l(i) =e=          yf(i)*f + yr(i)*r;

#Condenser Total balance
  dae8:  r + d =e= v;

#Reboiler Equilibrium
  dae9:  alpha*x(0) =e= y(0)*(1+x(0)*(alpha - 1));

#Tray equilibrium
  dae10(i E T):  alpha*x(i) =e= y(i)*(1+x(i)*(alpha - 1));

  dae11: y(31) =e= 0;

#Point (end-time) Constraints
  pcon1[1]: mu =g= obj;
  pcon2[1]: x(0) =l= xbset;
  pcon3[1]: x(31) =g= xdset;
```

```
#Logical constraints
  l1:  <<i E T| yf(i) >> =e= 1;
  l2:  <<i E T| yr(i) >> =e= 1;
  l3:  <<i E T| i*yr(i) >> =g= 2;
  l4:  <<i E T| i*yf(i) >> =g= 1;
  l5:  <<i E T| i*yr(i) >> =g= <<i E T| i*yf(i)>>+2;
}}
```

7 Troubleshooting and Errors

When creating and running a MINOPT input file, errors are likely to arise. These can occur for several different reasons: syntax errors in the input file, errors during the solving phase, bugs in the model, or bugs in the MINOPT program.

7.1 Syntax Errors

Most of the errors that are encountered in MINOPT are usually due to an error in the syntax of the input file. MINOPT will report these errors and indicate what the problem is. Note that MINOPT will try to read through as much of the input file as it can before terminating. This is so that it can report as many errors as possible. The error messages have the form:

`Line 28 at or near "...":  <error message>`

All of the error messages are self explanatory except for the catch-all message "`syntax error`" that MINOPT reports if the input does not fit any of the MINOPT constructions. The creation of the MINOPT input file often requires a few iterations of editing and running the problem before the model is syntactially correct.

7.2 Solving Errors

Once the model is syntactically correct, the parser sends the problem to be solved. The types of errors that can occur here range from a solver being available or not being capable of solving a particular type of problem to invalid option specifications to errors that are internal to the specific solver. These can usually be corrected by specifying the appropriate solver or option. Errors within the solver can also occur that lead to the failure of the algorithm to converge. This is likely due to some numerical problems encountered within the solver and usually are related to bugs within the model.

When solver errors are encountered, the manual for the solver should be consulted to determine the appropriate course of action. A special note should be made about the solver error "Current point cannot be improved". (This is an inform value of 9 for MINOS and SNOPT and 6 for NPSOL.) MINOPT treats this situation as an infeasible problem; however, this may not always be the case. This may arise if the optimality tolerance is too tight or from the choice of a poor starting point. These local solvers often need good starting points, especially for highly nonlinear problems.

7.3 Model Bugs

Although a model may be syntactically correct, the solver may converge to a meaningless answer or may fail to converge altogether. This is likely due to some errors within the model or model bugs. These can be simple problems such as using the wrong sign or misplacing parentheses or using the wrong expressions, to more complex numerical problems such as taking the logarithm of values close to zero or trying to solve a problem that is structurally singular. MINOPT can not detect such errors, but can be used to help determine why the errors are occuring. If a problem is found to be infeasible, determining which constraint(s) can be difficult. The solvers such as SNOPT and MINOS often report these infeasibilities in their output files which can be obtained using the `-p5` option from the command line. (Consult the manuals for these solvers on how to interpret their output.) Alternatively, the user can introduce slack variables to the constraints to solve a feasibility problem to determine the problematic constraints.

Dynamic Problems Particular problems may occur specifically with dynamic problems. Bad models can easily be passed to the integrator and the integrator can fail with messages that may be difficult to understand. These errors often occur because the variables used to specify the initial condition have not been set correctly. The user should carefully inspect the variables used in the `ISPEC` listing. Another frequent problem is the failure to initialize the dynamic system. Note that the initialization requires the solution of a nonlinear set of equations which itself can be a particularly challenging problem. One thing to consider is first solving the steady-state problem to validate the model and determine good starting points for the problem. Good stating points for both the $\boldsymbol{x}$ and $\boldsymbol{z}$ variables can help in the initialization of the DAE system.

If problems arise with the convergence of a dynamic optimization problem, a larger feasibilty tolerance may help. Note that the tolerance of the integrator is set to be ten times smaller than the feasibility tolerance of the NLP solver unless otherwise specified.

Although MINOPT provides an advanced modeling language, it does not make the user a good modeler. The solution algorithms implemented within MINOPT are only as good as the models on which they are used. Developiong a good model takes practice, experience, and some trial and error.

7.4 MINOPT Bugs

Although MINOPT has been tested extensively, bugs within the program are likely occur. These can happen in either the parsing or solving phases of the program. In the parsing phase, MINOPT may read a syntax that it thinks it understands and then get to a point in the program where it no longer knows what to do. This is usally preceded by reports of syntax errors that are followed by

```
Internal parser error on line ...  of file ...  at or near ...
Something has gone awry in the source code for the MINOPT parser.
This has occurred at line ...  of file ...
This probably indicates a bug in the MINOPT parser.
You may want to report this.  In the meantime, try to fix any
of the above parsing errors and see if that fixes the problem
```

The error can be reported by emailing the input file to `minopt@titan.princeton.ed` with a brief description of the problem. If there are no syntex error messages prior to the internal error message, the bug should definitely be reported.

Internal errors that occur in the solver result in a message indicating that a bug has probably been found in the MINOPT code. The input file and a brief description of the problem should be emailed to `minopt@titan.princeton.edu`.

References

Kernighan B.W. and Ritchie D.M., 1988, *The C Programming Language*. Prentice Hall.

Morari M. and Grossmann I.E. (eds.), 1991, *CACHE Process Design Case Studies*, Volume 6: Chemical Engineering Optimization Models with GAMS.