User's Guide for *MultiSRVN* Version 4.0

Revision: 1.13

January 7, 1997

Contents

List of Figures

List of Tables

1 Introduction

The *MultiSRVN* program provides a method for solving multiple *stochastic rendezvous networks* (SRVN) [1] which allows the software designer to analyze multi-tasking systems and observe trends in their performance.

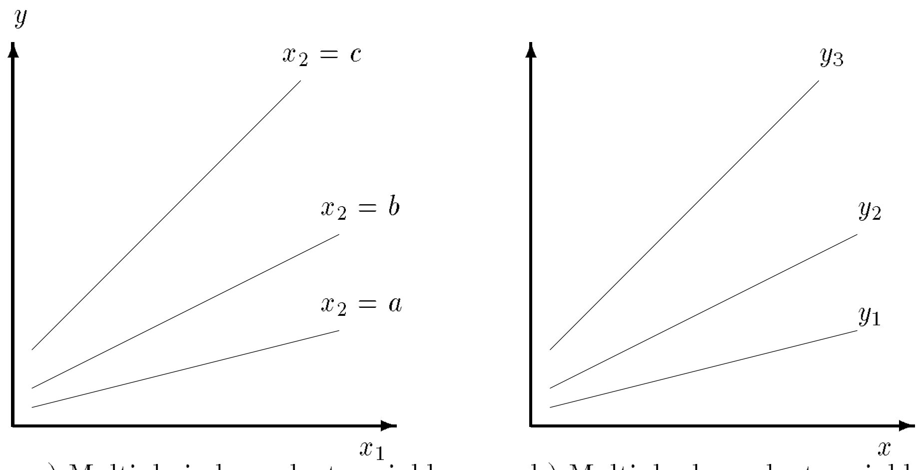
The designer develops an SRVN model to represent the multi-tasking software system which is to be analyzed. One or more parameters of that model are then selected as independent variables of the analysis and a range for each of these variables is determined. Dependent variables representing the properties of the system about which information is desired are also selected. *MultiSRVN* varies the independent variables throughout their ranges, creating a new data set, or *case* for each change. Each case is "solved" using SRVN solution techniques to generate "result values" for the dependent variables. The results for each case are collected into a report and presented to the designer.

A complete set of *cases* is called an *experiment.* A suite of experiments is called a *session.*

Performing experiments provides the software designer with a powerful method for estimating the effects of design changes on the performance of the system under study. For example, software deficiencies such as bottlenecks can be found and solutions proposed.

Currently, experiments are carried out using *srvn* [3] which requires the user to change the independent variables manually each time a new case is to be solved. In addition to handling the cases automatically, *MultiSRVN* provides a way of isolating which of the many possible dependent variables is to be examined. *Srvn* has not got this convenience.

Figure 1 shows two examples of how experiment results can be organized to provide information on how certain performance criteria vary as functions of system parameters.



a) Multiple independent variables   b) Multiple dependent variables

Figure 1: Experiment examples

In Figure 1.a two independent variables, x_1 and x_2, have been controlled while a single dependent variable, y, has been observed. Figure 1.b shows a single independent variable, x, being controlled and three dependent variables, y_1, y_2 and y_3 begin observed.

2 Program Usage

MultiSRVN is invoked from the UNIX command line with such arguments and options as needed for the desired operation. Input and output is handled by *MultiSRVN* through files. This section describes the invocation of *MultiSRVN*, the syntax and meaning of the various arguments and options and a brief description of the input and output files. Section 3 describes the various files in detail.

2.1 Functional Overview

Figure 2 shows the *MultiSRVN* and *srvn* programs, their inputs and outputs, and the relationship between them. The thick-line boxes represent the programs, the thin-line boxes are the data stores. Arrows show the direction of the flow of data.

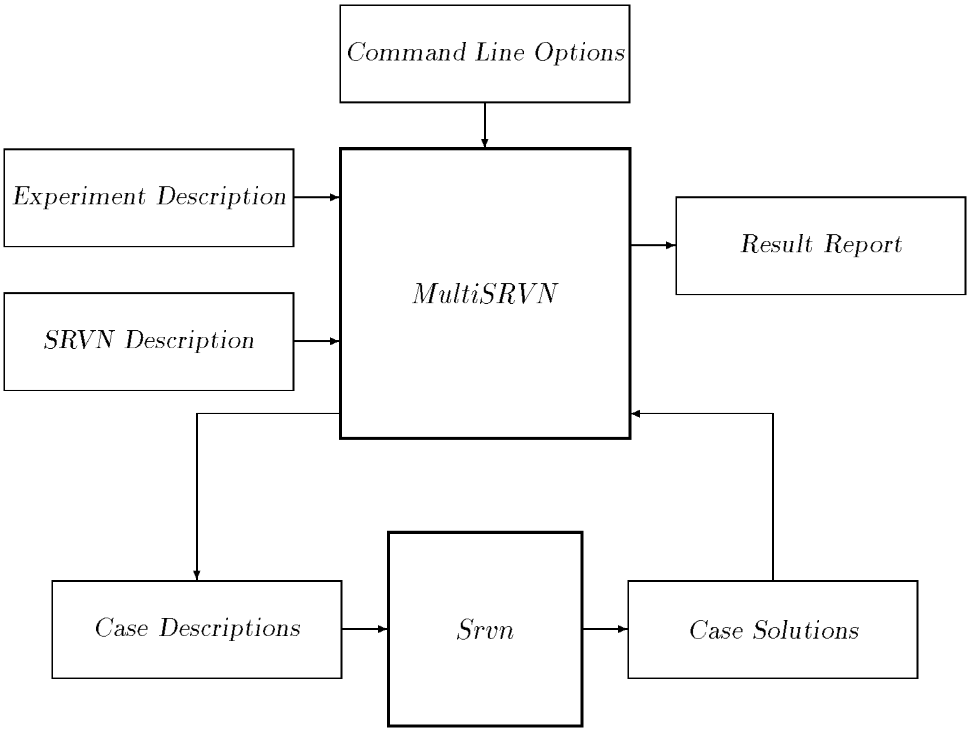


Figure 2: *MultiSRVN* files and data flow

An *SRVN description* file [2], as given in the example in Section 6.2.1, is created by the operator to contain the description of the software system under study. An *experiment description* file, containing the specification of the independent and dependent variables as well as some other data is also created by the operator. The details of this experiment description file are given in section 3 and an example is given in Section 6.2.2.

The operator invokes *MultiSRVN*from the command line, specifying and desired options. The command line syntax is described in section 2.2. *MultiSRVN* reads the experiment description and the SRVN description and generates several *case descriptions* as shown in the diagram. *Srvn* is invoked by *MultiSRVN* to process these cases, generating the *case solutions*. Finally, *MultiSRVN* reads the case solutions and prepares its report which is then put into the *result report* file.

The case files, both descriptions and solutions, are stored in a UNIX subdirectory. If more than one experiment description is given in the experiment description file, then a subdirectory is created for each experiment to store that experiment's case file.

2.2 Invocation Syntax

The UNIX command line syntax for invoking *MultiSRVN* is as summarized below. The syntax (*option1* || *option2*) is used to indicate that *option1* or *option2* may be used, but not both. When entering the *MultiSRVN* command, use one of the options and do not type the parentheses.

MultiSRVN [-**c**] [-**d**] [-**e** *name*] [-**f** *fsep*] [-**F** *format*] [-**I**] ([-**o** *rfile*] || [-**s**]) [-**O** *options*]
[-**R** [retries = ⟨*int*⟩][, uf = ⟨*real*⟩][, if = ⟨*real*⟩]] [-**S** *solver*] [-**v**] [*efile* ...]

MultiSRVN -**r** [-**d**] [-**e** *name*] [-**f** *fsep*] [-**F** *format*] [-**I**] ([-**o** *rfile*] || [-**s**]) [-**v**] [*efile* ...]

MultiSRVN -**n** [-**c**] [-**d**] [-**e** *name*] [-**f** *fsep*] [-**I**] [-**v**] [*efile* ...]

MultiSRVN reads its experiment descriptions from the file *efile*. This file contains the experiments as defined by the experiment file syntax described in the section 3. If no filename is specified, the input is read from *standard input*. The output is written to a file whose name is derived from *filename* by appending a suffix of ".res". Other suffixes are possible depending on the format of the output selected by the operator. More information about this is given in the option descriptions below.

MultiSRVN operates in three main modes. The "reuse" mode, activated by the -**r** option causes the case description files to be reused, preventing new case description files from being generated. The "no results" mode, activated by the -**n** option allows the case description file to be generated but prevents the solver from being run and the case result files from being created. If neither of these modes is invoked, *MultiSRVN* will produce both types of case file. This is the third mode of operation.

The invokation syntax above describes which options may be used in each of the three modes. Options which are not listed for a particular mode are incompatible with that mode and will cause an error if used.

2.3 Options

There are several options available for *MultiSRVN*. The following are descriptions explaining the purpose and usage of each option.

-c Cleanup the experiment directory, if one exists, before performing the experiment. This causes all the old files in the experiment directory to be deleted. The directory whose contents are deleted by this option is determined from the experiment name in the experiment description file. Do not accidently delete the wrong directory.

-d Display debugging information. This option is similar to the **v** option except that more information is displayed. Normally, this option should not be used. Repeated uses of "-d" on the command line increases the amount of debugging information displayed: "-ddd" or "-d -d -d" displays the maximum amount.

-e *ename* This option allows specific experiments to be selected from the experiment file to be executed. The experiment named *ename* (see section 3) is selected. Any number of **-e** options may be used. The default is that all experiment descriptions in the experiment file are processed.

-f *fsep* This option allows the operator to select a string which will be used in the creation of case file names. The default value for this option is "–".

-F *format* This option is used to select the format for the result output. The option takes an argument, *format* which is one of the following: *standard, matlab, gnuplot* or *latex*. *MultiSRVN* will accept certain abbreviations for the format name.

standard This argument produces an output file having the suffix **.res**. The results are provided in the form of tables of numbers. This is the default output format and is automatically selected when the **F** option is omitted. The abbreviation is "sta".

matlab This format produces a file having the suffix **.m**. The file contains scripts which when executed from within the *MatLab* program, provides an interactive graphical "back end". The abbreviation is "mat".

gnuplot This format produces a file having the suffix **.gnu**. The results are provided in columns representing the independent and dependent variables. The user must use the `plot` or `splot` commands within *gnuplot* to make use of the gnuplot format output. The abbreviation is "gnu".

latex This format produces a file with the suffix **.tex**. The file contains LaTeX formatted tables suitable for inclusion in LaTeX documents. The abbreviation is "tex".

-I This option causes the name, date and copyright notice for *MultiSRVN* to be displayed. Program execution stops immediately after the data is displayed, regardless of any other options and arguments used.

-l This option causes *MultiSRVN* to record diagnostics related to solving cases into a case log file. The case log files have the same root file name as the other case files but end with `.log`. The information recorded includes user-level iteration status, solver debugging information and other output. If this option is absent, the information is written to standard output and standard error.

-n This option puts *MultiSRVN* into "no results" mode in which the solver is not used, case results files are not generated and no result report is produced. This mode is useful for having *MultiSRVN* generate the case description files.

-N This option sets the *nice(1)* level that will be used when running subprocesses including all remote processes launched by *MultiSRVN*. Using "-N" sets the nice level to be 10; "-NN" or " -N -N" sets it to 20. The default is 0.

-o *rfile* The filename *results* is used as the file in which all the results are written. If the filename is given as "-" as in "-o-", then the *standard output* device is used for writing results. This option is mutually exclusive with the **-s** option.

-O *options* This option passes along its argument, *options* to the SRVN solver. If white-space or "-" characters are required in the option string to the solver, then the string must be enclosed in quotation marks ("). This prevent *MultiSRVN* from parsing the option string as options to itself. These options are used in addition to any options specified in the experiment description.

-r This option places *MultiSRVN* into "reuse" mode. In this mode, the results from previous computations are reused. The case description files are preserved and reused. A result file base on the results generated before is created. This mode reduces the execution time of *MultiSRVN*.

-R This option controls the retry feature for the SRVN solver. If the solver reports that the particular case did not converge, it may be possible to get convergience by reducing the *underrelaxation coefficient* and increasing the number of *iterations*. This can be acieved using this option. The following are the possible sub-options which may be used in any combination.

retries This suboption controls the number of times the solver is tried on a case which does not converge. The default is 4.

uf This suboption is the underrelaxation factor. The current underrelaxation coefficient (See Section 3 for the experiment is multiplied by this real valued number prior to each retry. The default for this suboption is 1.0.

if This suboption id the iteration factor. The current number of allowed iterations (See Section 3) for the experiment is multiplied by this real valued number and truncated to an integer prior to each retry. The defdault for this suboption is 2.

-s Split the output into separate files. This options forces the results to be output with a unique file for each experiment. This occurs regardless of the configuration of the input file(s). For example if three files are given as input and the first two have two experiments each while the third has one experiment, then five result files will be generated. The output file names are generated using the the experiment name of the current experiment and a suffix. This latter is determined by the format of the output as described below. This option is mutually exclusive with the **-o** option.

-S *solver* This option permits the operator to select a "solver". The option takes an argument which is one of the following: *lqns, parasrvn, petrisrvn, srvn, srvnbpg* or *srvntda*. *MultiSRVN* accepts certain abbreviations for the solver names.

lqns This argument selects the lqns solver for use with the experiments.

parasrvn This argument selects the parasol srvn simulator to solve the cases. The simulator requires more time than the other SRVN solvers and may cause *MultiSRVN* to take a long time to finish. Use this argument with caution. The abbreviation is “para”.

petrisrvn This argument selects the petrisrvn solver for use with the experiments. The abbreviation is “petri”.

srvn When this argument is used, the standard version of the solver is used to solve each of the cases.

srvnbpg This argument selects the srvn to prolog bounds generator solver. The abbreviation is “bpg”.

srvntda This argument causes the newer “task directed aggregation” algorithm to be used. The abbreviation is “tda”.

-v This options turns on verbose mode. While this mode is active, status information is written to the *standard output* device. This information includes the current experiment filename, experiment name and other such data.

3 Experiment File

The BNF description of the Experiment Description File syntax is given in this section. The notation ⟨*non-terminal*⟩ is used to identify the non-terminal tokens of the BNF. Terminal tokens are given in the "typewriter" style font as in `terminal`. When a token is optional, it is marked as such by the use the following notation: optional-token$_{\text{opt}}$.

The "non-terminals", ⟨*identifier*⟩, ⟨*integer*⟩ and ⟨*real*⟩, are common to many sections of this BNF. They are therefor given their own sub-section at the end of the BNF. The "non-terminals" ⟨*parameter-id*⟩ and ⟨*result-id*⟩ are sufficiently complex to warrant their own sub-sections as well.

3.1 Experiment File Syntax

The Experiment Description File syntax consists of optional global definitions section followed by an ⟨*experiment-list*⟩.

⟨*experiment-file*⟩	→	⟨*global-section*⟩$_{\text{opt}}$ ⟨*experiment-list*⟩

3.2 Global Section

The global section is optional and allows various program settings to be assigned default values. The syntax consists of the keyword `.global` followed by a list of ⟨*set-statement*⟩, each terminated by a semi-colon. These statements associate a string value to a ⟨*set-item*⟩. The global section is terminated by the `.end` keyword.

⟨*set-section*⟩	→	`.global` ⟨*set-list*⟩ `.end`
⟨*set-list*⟩	→	⟨*set-statement*⟩
	|	⟨*set-list*⟩ ⟨*set-statement*⟩
⟨*set-statement*⟩	→	⟨*set-item*⟩ `=` ⟨*string*⟩ `;`

Table 1 contains a complete list of keywords which may be used as ⟨*set-item*⟩. The keyword `.directory` is used to specify the UNIX path for the directory into which the various case files placed. The `.template` keyword provides a way to select the SRVN Description File to be used as a template.

`.boundsopts`	`.directory`	`.format`	`.hosts`
`.initexpr`	`.lqnsopts`	`.paraopts`	`.petriopts`
`.sample`	`.seed`	`.solver`	`.srvnopts`
`.tdaopts`	`.template`	`.title`	

Table 1: A list of set items

The `.solver` keyword takes one of `''bpg''`, `''lqns''`, `''petri''`, `''para''`, `''srvn''`, or `''tda''` as a setting and thus selects a particular solver. The `.boundsopts`, `.lqnsopts`, `.paraopts`, `.petriopts`, `.srvnopts`, and `.tdaopts` keywords allow command line options to be specified for the various solvers.

The `.hosts` keyword takes a string made up of a comma separated list of workstation names as a setting. The various cases to be solved are then distributed over the workstations so that each workstation named is used while there are still cases to be solved. Note that if the `/hosts` setting is used, then *only* the named workstations are used – that is, the current workstation on which *MultiSRVN* is running will not be used to solve cases unless its name is included in the `.hosts` setting.

The `.format` keyword allows the format of the experiment output to be selected. The settings for this keyword are `''standard''`, `''latex''`, `''gnuplot''`, `''matlab''`. The keyword `.title` gives a title to the result output.

The `.sample` keyword is used to select a subset of the test cases. It accepts a numeric argument ranging from `0.0` to `1.0`. This number denotes the probability that a test case is generated. The `.seed` keyword is used to seed the random number generator used to select test cases. It accepts an integral argument.

The `.initexpr` is used to initialize *bc(1)* and to declare functions which will be used in control statement expressions. Any valid *bc(1)* statements may be placed within the quotes of the set item string.

Default settings are provided by the tool for settings which are not included from the global section. In addition, certain command line options can override the global settings.

3.3 Experiment List

Each experiment is started with the keyword `.experiment` and followed by an experiment identifier. The body of the experiment consists of several sections, some of which are optional. These sections are each described in detail below. The end of the experiment is marked by the keyword `.end`.

⟨*experiment-list*⟩	→	⟨*experiment*⟩
	|	⟨*experiment*⟩ ⟨*experiment-list*⟩
⟨*experiment*⟩	→	`.experiment` ⟨*identifier*⟩ ⟨*experiment-body*⟩ `.end`
⟨*experiment-body*⟩	→	⟨*set-section*⟩$_{\mathrm{opt}}$ ⟨*initialize-section*⟩$_{\mathrm{opt}}$ ⟨*declare-section*⟩ ⟨*vary-section*⟩ ⟨*control-section*⟩ ⟨*iterate-section*⟩$_{\mathrm{opt}}$ ⟨*observe-section*⟩

3.4 Set Section

The set section is optional and allows various program settings to be altered from their default values. The syntax consists of the keyword `.set` followed by a ⟨*set-list*⟩ as described in the *global* section.

⟨*set-section*⟩	→	`.set` ⟨*set-list*⟩

Table 1 is a list of the keywords which may be used to identify options to be set. The settings made local to an experiment override the defaults set in the *global* section with the exception of the `.initexpr` for which the local settings are appended to the global setting. Some command line options may be used to override the local settings.

3.5 Initialize Section

The initialize section is optional and allows SRVN model parameters to be initialized prior to the experimentation. This section begins with the keyword `.initialize` which is followed by a list of ⟨*initialize-statement*⟩, each terminated by a semi-colon. Each ⟨*initialize-statement*⟩ associates a value with a ⟨*initialize-item*⟩.

⟨*initialize-section*⟩	→	`.initialize` ⟨*initialize-list*⟩

⟨*initialize-list*⟩	→	⟨*initialize-statement*⟩
	|	⟨*initialize-statement*⟩ ⟨*initialize-list*⟩
⟨*initialize-statement*⟩	→	⟨*initialize-item*⟩ `=` ⟨*value*⟩ `;`
⟨*initialize-item*⟩	→	`.cl` | `.il` | `.uc`
	|	⟨*parameter-id*⟩

The keyword `.cl` refers to the "convergence limit" and controls the precision of the SRVN calculations. It takes a ⟨*value*⟩ of type *real*. The keyword `.il` refers to the "iteration-limit" and controls the number of iteration allowed in the SRVN calculations. It takes a ⟨*value*⟩ of type *integer*. The keyword `.uc` is the underrelaxation-coefficient. This *real* value can be used to help the SRVN calculations converge. The ⟨*parameter-id*⟩, described below, allows any model parameter that can be controlled by *MultiSRVN* to be intialized.

3.6 Declare Section

The declare section associates identifiers with lists of values. These identifiers can then be used in the following sections. The section begins with the keyword `.declare` and is followed by a list of ⟨*declare statements*⟩. These declare statements bind ⟨*value-specifications*⟩ to identifiers. Either a list or a range may be used as a value specification.

⟨*declare-section*⟩	→	`.declare` ⟨*declare-list*⟩
⟨*declare-list*⟩	→	⟨*declare-statement*⟩
	|	⟨*declare-statement*⟩ ⟨*declare-list*⟩
⟨*declare-statement*⟩	→	⟨*variable*⟩ `=` ⟨*value-specification*⟩ `;`
	|	⟨*variable*⟩ `=` ⟨*expression*⟩ `;`
	|	⟨*variable*⟩ `=` ⟨*result-id*⟩ `;`
⟨*variable*⟩	→	⟨*identifier*⟩
⟨*value-specification*⟩	→	⟨*value-list*⟩ | ⟨*value-range*⟩
⟨*value-list*⟩	→	⟨*value*⟩
	|	⟨*value*⟩ `,` ⟨*value-list*⟩
⟨*value-range*⟩	→	`[` ⟨*value*⟩ `-` ⟨*value*⟩ `]` `,` ⟨*integer*⟩

A ⟨*value-list*⟩ is simply a comma separated list of values. A ⟨*value-range*⟩ consists of a start value and a stop value, located within brackets and separated by "-" followed by a comma and then the number of values in the range (including the start and stop values). The ⟨*expression*⟩ and ⟨*result-id*⟩ syntax is explained below.

3.7 Vary Section

The vary section identifies how the variables declared in the ⟨*declare-section*⟩ are to be varied. The section is started with the keyword `.vary` and is followed by a ⟨*vary-list*⟩. A vary list consists of one or more ⟨*vary-group*⟩, each group consisting of one or more variables separated by commas and terminated by a semicolon.

The variables in a particular vary group are cycled through their various values together. The first variable in the group is used to determine the length of the cycle. Variables which have fewer values than the cycle length are cycled again starting at their initial value. Variables which have more values than the cycle length have their remaining values ignored.

The vary groups are not varied together; they are varied *across* one another. For each value in the cycle for the first group, the second group is varied through its cycle. Similarly, for each value in the second group, the third group is varied through its cycle. In this way, all the possible combinations of values for all the vary groups are generated.

⟨*vary-section*⟩	→	`.vary` ⟨*vary-list*⟩
⟨*vary-list*⟩	→	⟨*vary-group*⟩
	|	⟨*vary-group*⟩ ⟨*vary-list*⟩
⟨*vary-group*⟩	→	⟨*variable-list*⟩ ;
⟨*variable-list*⟩	→	⟨*variable*⟩
	|	⟨*variable-list*⟩ , ⟨*variable*⟩

3.8 Control Section

The control section associates expressions involving variables with SRVN model parameters. In this way, the independent variables of the experiment are selected and values for them are specified. The section starts with the keyword `.control` followed a list of ⟨*control-statements*⟩. A control statement takes the form of a ⟨*parameter-id*⟩ followed by an equals sign followed by an ⟨*expression*⟩. It describes the mannor in which the value of an srvn parameter is to be controlled.

⟨*control-section*⟩	→	`.control` ⟨*control-list*⟩
⟨*control-list*⟩	→	⟨*control-statement*⟩
	|	⟨*control-statement*⟩ ⟨*control-list*⟩
⟨*control-statement*⟩	→	⟨*parameter-id*⟩ = ⟨*expression*⟩ ;
⟨*expression*⟩	→	{ ⟨*bc-expression*⟩ }
⟨*label*⟩	→	⟨*string*⟩

The ⟨*parameter-id*⟩ is explained below. A ⟨*bc-expression*⟩ may be any expression permitted by the syntax of the *bc(1)* program as described in the manual page for *bc(1)*. Expressions include references to variables declared in the declare section and identified in a particular vary group. Expression may include references to declarations made in the `.initexpr` statements of either the set section or global section.

The optional label following the ⟨*expression*⟩ is used in the presentation of the results in the result report. If this label is missing, a label is formed from the components of the ⟨*parameter-id*⟩.

3.9 Iterate Section

The iterate section identifies the maximum number of user-level iterations permitted as well as the specific model parameters for which convergience tests are to be performed. For each of these, a convergience limit is specified.

⟨*iterate-section*⟩	→	`.iterate` ⟨*max-count*⟩ ⟨*iterate-list*⟩
⟨*max-count*⟩	→	⟨*integer*⟩
⟨*itertate-list*⟩	→	⟨*iterate-statement*⟩
	|	⟨*iterate-list*⟩ ⟨*iterate-statement*⟩
⟨*iterate-statement*⟩	→	⟨*result-id*⟩ ⟨*convergience*⟩ ;
⟨*convergience*⟩	→	⟨*real*⟩

The integer, ⟨*max-count*⟩, gives the maximum number of user-level iterations permitted. The ⟨*result-id*⟩ gives the model output for which convergence is to be tested and ⟨*convergence*⟩ gives the limit of the test. An iteration is considered to be "successful" only if all ⟨*result-ids*⟩ being tested varied less than their respective ⟨*convergences*⟩ when compared to the last iteration.

3.10 Observe Section

The observe section identifies both the independent and dependent model parameters to be observed as results in the output.

⟨*observe-section*⟩	→	`.observe` ⟨*observe-list*⟩
⟨*observe-list*⟩	→	⟨*observe-statement*⟩
	|	⟨*observe-statement*⟩ ⟨*observe-list*⟩
⟨*observe-statement*⟩	→	`.plot` ⟨*expression*⟩ ⟨*label*⟩$_{\mathrm{opt}}$ `;`
	|	`[observe-statement]` ⟨*parameter-id*⟩ ⟨*label*⟩$_{\mathrm{opt}}$ `;`
	|	`[observe-statement]` ⟨*result-id*⟩ ⟨*label*⟩$_{\mathrm{opt}}$ `;`

If there are N vary groups declared in the vary section, then the first N observe statements must correspond, one-to-one, to each vary group. This is how the particular independent variables to be reported in the output are identified. Any single variable or expression may be specified using the `.plot` command. Alternatively, model parameters may be specified explicitly using the ⟨*parameter-id*⟩ syntax. Once the N independent model variables have been given, all subsequent observe statements are taken to refer to dependent quantities.

The optional label following the ⟨*result-id*⟩ is used in the result report. If this label is not specified, a label is generated automatically.

3.11 Parameter Id

The ⟨*parameter-id*⟩ constructions are used to identify SRVN parameters for which new values are to be used. The initialize and control sections both make use of the ⟨*parameter-id*⟩ construction in their syntax.

⟨*parameter-id*⟩	→	⟨*pcode-3*⟩ `(` ⟨*entity-id*⟩ `,` ⟨*entity-id*⟩ `,` ⟨*entity-id*⟩ `)`
	|	⟨*pcode-2p*⟩ `(` ⟨*entity-id*⟩ `,` ⟨*entity-id*⟩ `,` ⟨*phase*⟩ `)`
	|	⟨*pcode-2*⟩ `(` ⟨*entity-id*⟩ `,` ⟨*entity-id*⟩ `)`
	|	⟨*pcode-1p*⟩ `(` ⟨*entity-id*⟩ `,` ⟨*phase*⟩ `)`
	|	⟨*pcode-1*⟩ `(` ⟨*entity-id*⟩ `)`
⟨*pcode-3*⟩	→	`.arr` | `.asr`
⟨*pcode-2p*⟩	→	`.rr` | `.sr` | `.fi` | `.fo`
⟨*pcode-2*⟩	→	`.pf` | `.cd` | `.acv` | `.apt` | `.amst` | `.ast` | `.az`
⟨*pcode-1p*⟩	→	`.cv` | `.pt` | `.st` | `.ez` | `.mst`
⟨*pcode-1*⟩	→	`.ar` | `.pm` | `.pq` | `.pr` | `.ps` | `.sf` | `.tm` | `.tp` | `.tz` | `.ep`
⟨*entity-id*⟩	→	⟨*task-id*⟩ | ⟨*entry-id*⟩ | ⟨*processor-id*⟩ | ⟨*queue-id*⟩
⟨*task-id*⟩	→	⟨*identifier*⟩
⟨*entry-id*⟩	→	⟨*identifier*⟩
⟨*processor-id*⟩	→	⟨*identifier*⟩
⟨*queue-id*⟩	→	⟨*identifier*⟩

⟨*phase*⟩	→	`1` | `2` | `3`

Table 2 summarizes the parameter id specification. It lists all the parameters and identifies their value types, parameter codes, and argument lists. Note that the scheduling flag paramerter takes an integer argument, not a character as would be suggested by the scheduling policy field of the template and case files. The mapping of these integer scheduling policy type codes to the scheduling policy types (and character flags) is given in Table 3.

3.12 Result Id

The ⟨*result-id*⟩ constructions are used to identify SRVN result values for observation. These values are the dependent variables in the experiment. The observe section makes use of the ⟨*result-id*⟩ construction in its syntax.

⟨*result-id*⟩	→	⟨*result-code*⟩ `(` ⟨*category*⟩ `,` ⟨*arg-list*⟩ `)`
⟨*result-id*⟩	→	⟨*plain-rcode*⟩ `(` ⟨*ccode-2*⟩ `,` ⟨*entity-id*⟩ `,` ⟨*entity-id*⟩ ⟨*opt-phase*⟩$_{\mathrm{opt}}$ `)`
	|	⟨*plain-rcode*⟩ `(` ⟨*ccode-1*⟩ `,` ⟨*entity-id*⟩ ⟨*opt-phase*⟩$_{\mathrm{opt}}$ `)`
	|	⟨*confidence-rcode*⟩ `(` ⟨*ccode-2*⟩ `,` ⟨*entity-id*⟩ `,` ⟨*entity-id*⟩ `,` ⟨*confidence-level*⟩ ⟨*opt-phase*⟩$_{\mathrm{opt}}$ `)`
	|	⟨*confidence-rcode*⟩ `(` ⟨*ccode-1*⟩ `,` ⟨*entity-id*⟩ `,` ⟨*confidence-level*⟩ ⟨*opt-phase*⟩$_{\mathrm{opt}}$ `)`
⟨*plain-rcode*⟩	→	`.tb` | `.tbl` | `.tbu` | `.th` | `.ut` | `.wt`
⟨*confidence-rcode*⟩	→	`.tch` | `.utc` | `.wtc`
⟨*ccode-2*⟩	→	`.ee` | `.et` | `.tt`
⟨*ccode-1*⟩	→	`.t` | `.e` | `.p`
⟨*opt-phase*⟩	→	`,` ⟨*phase*⟩ | `0`
⟨*confidence-level*⟩	→	⟨*integer*⟩

Table 4 summarizes the results which may be observed as dependent variables and identifies their result codes, valid categories and argument lists. If an optional phase is given as zero or is omitted the value reported is the sum over all phases.

The categories listed in the BNF for this section as well as Table 4 are used to identify the type of data to be observed. The category `.e` indicates that the result to be observed belongs to an *entry* whereas `.p` and `.t` refer to *processors* and *tasks* respectively. The category `.ee` indicates that the observation is to be made for the data associated with *two* entries. Similarly, `.et` is entry to task and `.tt` is task to task.

3.13 Miscellaneous

The non-terminals ⟨*letter*⟩ and ⟨*decimal-digit*⟩ are not defined in the usual BNF style. Instead, letters are taken to be any letter in the ranges [a-z] or [A-Z]. Decimal digits include the digits in the range [0-9].

3.13.1 Identifiers

⟨*identifier*⟩	→	⟨*letter*⟩
	|	⟨*letter*⟩ ⟨*alpha-numeric-string*⟩
⟨*alpha-numeric-string*⟩	→	⟨*alpha-numeric*⟩

Parameter Name	*Type*	*Code*	*Argument List*
		Processor Parameters	
Processor Delay	Real	`.cd`	⟨*processor-id*⟩, ⟨*processor-id*⟩
Scheduling Flag	Integer, $0-4$	`.sf`	⟨*processor-id*⟩
Processor Multiplicity	Integer	`.pm`	⟨*processor-id*⟩
Processor Quantum	Real	`.pq`	⟨*processor-id*⟩
Processor Replication	Integer	`.pr`	⟨*processor-id*⟩
Processor Speed	Real	`.ps`	⟨*processor-id*⟩
		Task Parameters	
Task Multiplicity	Integer	`.tm`	⟨*task-id*⟩
Task Priority	Integer	`.tp`	⟨*task-id*⟩
Task Replication	Integer	`.tr`	⟨*task-id*⟩
Think Time	Real	`.tz`	⟨*task-id*⟩, ⟨*phase*⟩
		Entry Parameters	
Arrival Rate	Real	`.ar`	⟨*entry-id*⟩
Coefficient of Variation	Real	`.cv`	⟨*entry-id*⟩, ⟨*phase*⟩
Phase Type	Integer	`.pt`	⟨*entry-id*⟩, ⟨*phase*⟩
Max Service Time	Real	`.mst`	⟨*entry-id*⟩, ⟨*phase*⟩
Entry Priority	Integer	`.ep`	⟨*entry-id*⟩
Service Time	Real	`.st`	⟨*entry-id*⟩, ⟨*phase*⟩
Think Time	Real	`.ez`	⟨*entry-id*⟩, ⟨*phase*⟩
		Call Parameters	
Probability of Forwarding	Real	`.pf`	⟨*entry-id*⟩, ⟨*entry-id*⟩
Rendezvous Rate	Real	`.rr`	⟨*entry-id*⟩, ⟨*entry-id*⟩, ⟨*phase*⟩
Send-no-reply Rate	Real	`.sr`	⟨*entry-id*⟩, ⟨*entry-id*⟩, ⟨*phase*⟩
Fanin	Integer	`.fi`	⟨*entry-id*⟩, ⟨*entry-id*⟩, ⟨*phase*⟩
Fanout	Integer	`.fo`	⟨*entry-id*⟩, ⟨*entry-id*⟩, ⟨*phase*⟩
Probability Forwarding	Real	`.pf`	⟨*entry-id*⟩, ⟨*entry-id*⟩
		Activity Parameters	
Coefficient of Variation	Real	`.acv`	⟨*task-id*⟩ ⟨*activity-id*⟩
Phase Type	Integer	`.apt`	⟨*task-id*⟩ ⟨*activity-id*⟩
Max Service Time	Real	`.amst`	⟨*task-id*⟩ ⟨*entry-id*⟩
Service Time	Real	`.ast`	⟨*task-id*⟩ ⟨*entry-id*⟩
Think Time	Real	`.az`	⟨*task-id*⟩ ⟨*entry-id*⟩
		Activity Call Parameters	
Rendezvous Rate	Real	`.arr`	⟨*task-id*⟩, ⟨*activity-id*⟩, ⟨*entry-id*⟩
Send-no-reply Rate	Real	`.asr`	⟨*task-id*⟩, ⟨*activity-id*⟩, ⟨*entry-id*⟩

Table 2: Parameter Id Information

Scheduling Policy	*Type Code*	*Scheduling Flag*
First Come First Served	0	`f`
Head Of the Line	1	`h`
Preemptive Priority	2	`p`
Random	3	`r`
Processor Sharing	4	`s`

Table 3: Mapping of scheduling flag to integer type code

Result Name	*Code*	*Category*	*Argument List*
Throughput Bounds	`.tb`	`.e`	⟨*entry-id*⟩
	`.tbu`	`.e`	⟨*entry-id*⟩
	`.tbl`	`.e`	⟨*entry-id*⟩
Throughput	`.th`	`.t`	⟨*task-id*⟩
		`.e`	⟨*entry-id*⟩
Throughput Confidence	`.thc`	`.e`	⟨*entry-id*⟩
Utilization	`.ut`	`.t`	⟨*task-id*⟩, ⟨*phase*⟩
		`.e`	⟨*entry-id*⟩, ⟨*phase*⟩
		`.p`	⟨*processor-id*⟩
Utilization Confidence	`.utc`	`.e`	⟨*entry-id*⟩, ⟨*phase*⟩
		`.p`	⟨*processor-id*⟩
Waiting Time	`.wt`	`.p`	⟨*processor-id*⟩, ⟨*phase*⟩
		`.ee`	⟨*entry-id*⟩, ⟨*entry-id*⟩, ⟨*phase*⟩
Waiting Time Confidence	`.wtc`	`.p`	⟨*processor-id*⟩, ⟨*phase*⟩

Table 4: Result Id Information

$$
\begin{array}{lcl}
 & | & \langle \mathit{alpha\text{-}numeric} \rangle \ \langle \mathit{alpha\text{-}numeric\text{-}string} \rangle \\
\langle \mathit{alpha\text{-}numeric} \rangle & \rightarrow & \langle \mathit{letter} \rangle \ | \ \langle \mathit{decimal\text{-}digit} \rangle
\end{array}
$$

3.13.2 Integers

$$
\begin{array}{lcl}
\langle \mathit{integer} \rangle & \rightarrow & \langle \mathit{decimal\text{-}digit\text{-}string} \rangle \\
\langle \mathit{decimal\text{-}digit\text{-}string} \rangle & \rightarrow & \langle \mathit{decimal\text{-}digit} \rangle \\
 & | & \langle \mathit{decimal\text{-}digit} \rangle \ \langle \mathit{decimal\text{-}digit\text{-}string} \rangle
\end{array}
$$

3.13.3 Reals

$$
\begin{array}{lcl}
\langle \mathit{real} \rangle & \rightarrow & \langle \mathit{decimal\text{-}digit\text{-}string} \rangle \ \texttt{.} \ \langle \mathit{decimal\text{-}digit\text{-}string} \rangle_{\mathrm{opt}} \\
 & & \langle \mathit{exponent\text{-}string} \rangle_{\mathrm{opt}} \\
 & | & \langle \mathit{decimal\text{-}digit\text{-}string} \rangle_{\mathrm{opt}} \ \texttt{.} \ \langle \mathit{decimal\text{-}digit\text{-}string} \rangle \\
 & & \langle \mathit{exponent\text{-}string} \rangle_{\mathrm{opt}} \\
 & | & \langle \mathit{decimal\text{-}digit\text{-}string} \rangle \ \langle \mathit{exponent\text{-}string} \rangle \\
\langle \mathit{exponent\text{-}string} \rangle & \rightarrow & \langle \mathit{exponent\text{-}flag} \rangle \ \langle \mathit{sign} \rangle_{\mathrm{opt}} \ \langle \mathit{decimal\text{-}digit\text{-}string} \rangle \\
\langle \mathit{exponent\text{-}flag} \rangle & \rightarrow & \texttt{D} \ | \ \texttt{d} \ | \ \texttt{E} \ | \ \texttt{e} \\
\langle \mathit{sign} \rangle & \rightarrow & \texttt{+} \ | \ \texttt{-}
\end{array}
$$

4 Result Report File

This sections describes the format of the result report file. Several different formats are possible for this file. The format used for particular instance of this file depends on the command-line options selected by the operator and/or the experiment settings provided in the experiment description file. See Sections 2.2 and 2.3 as well as Section 3 for more information on these options.

4.1 Standard Format

In this format, the experiment result report is presented as an ASCII file containing tables of numbers and some comments. For each experiment result-set in the report file, a header is generated. This header identifies the experiment name, the SRVN description file name and the number of independent and dependent variables. The following is an example of this header.

```
Experiment Results

Experiment: designc
SRVN description file: cmpdesign.lqn
Independent variables: 1
Dependent variables: 1
```

To make the tables consistent, "standard" identifiers such as x1 and y2 are used to mark the rows and columns. For example, in an experiment with two independent variables, the first independent variable is known as x1 and the second is x2. The mapping of these standard names to the names given in the experiment description file is given for each table. If, for example, the first independent variable is named (by default) to "`.rr ( client, get, 1 )`" then the mapping would be

```
x1 -> .rr ( client, get, 1 )
```

For experiments which have only one independent variable, a single table is generated. In this table, the left-most column contains the values of the independent variable. To the right of this column are the columns of dependent variable values. There is a column for each dependent variable. The following is a sample for this type of result.

```
x -> $N_{users}$
y -> $f_Client1$
      x          y
1           0.070043
2           0.129662
3           0.175577
4           0.208075
5           0.229080
6           0.241031
7           0.246707
8           0.248776
9           0.249405
10          0.249187
```

Experiments which have two independent variables have multiple tables, one for each dependent variable. The left-most column of each table contains the values of the independent variable x1. The top-most row contains the values of the dependent variable x2. The remainder of the table consists of the values of the dependent variable located such that the row and column positions identify the corresponding independent variable values. The following is a typical table for this type of result.

```
x1 -> beta
x2 -> gamma
y1 -> X

                 x2 0.1        .48        .86        1.24       1.62       2
       x1
0.1                 0.419980   0.395980   0.369120   0.342340   0.317600   0.287990
.48                 0.396280   0.369810   0.339680   0.321280   0.293530   0.275020
.86                 0.363360   0.345170   0.313970   0.297450   0.269140   0.255400
1.24                0.336000   0.319530   0.293920   0.273020   0.253550   0.235530
1.62                0.313260   0.292840   0.270080   0.249690   0.234240   0.219760
2                   0.284810   0.269400   0.246430   0.236150   0.217480   0.208290
```

If three or more independent variables with the standard format, many tables are generated. The tables will have a header giving the name and values for $n-1$ of the independent variables. The last independent variable provides the left-most column of the tables. The remainder of the columns are filled by the values of the dependent variables.

4.2 MatLab Format

In this format, the result report consists of a *MatLab* script suitable for execution in *MatLab*. The file consists of a comment header, followed by a series of *MatLab* commands.

The comment header contains information similar to that of the `standard` format: the experiment name and SRVN description file are identified. In addition to this information, a menu consisting of plot options is also given. The following is an example of this portion of the matlab result file.

```
%% Experiment Results
%%
%% Experiment: designc
%% SRVN description file: cmpdesign.lqn
%% Independent variables: 1
%% Dependent variables: 1
%%
% 1)  Plot $X_{Clients}$ against $N_{users}$
% 2)  Set capturing of graphs to files
% 0)  Stop
```

The *MatLab* commands provide *MatLab* with the information it needs to plot the result data. First, a `clear` command is given to erase any previous data stored in *MatLab* memory. Then, the tables of result data are given in matrix form. Along with the numerical data, the labels and titles are given as well. Finally, a short *MatLab* program is given which displays the menu mentioned above and performs the plotting. The following is an example of the matlab commands for a simple table (only one independent variable).

```

clear
x = [ 1 2 3 4 5 6 7 8 9 10  ];
```

```
xl = '$N_{users}$';
y1 = [ 0.070043 0.129662 0.175577 0.208075 0.229080 0.241031 0.246707 0.248776 0.\
249405 0.249187  ];
yl1 = '$X_{Clients}$';

n = 0;
p = 0;
while 1
   clc
   help cmpdesign.m
   n = input( 'Select an option number: ' );
   if ( ( n <= 0 ) | ( n > 2 ) )
      break
   elseif ( n == 1 )
      plot( x,y1, x,y1,'o' )
      title('designc -- $X_{Clients}$ vs $N_{users}$')
      xlabel(xl)
      ylabel(yl1)
      if ( p == 1 )
         print -deps designcg1.eps
      end;
   else
      p = input( 'Capture output to file? ( 1 for yes, 0 for no ): ' );
   end
end
```

In order to run a matlab script for a particular experiment, the script for that experiment must be isolated in a file with a `.m` suffix. This means that result files with multiple experiments should be split, either manually or with the -`s` option such that each experiment is in a separate file. Aternatively, individual experiments may be selecrted using the -`e` option. This option is described in Sections 2.2 and 2.3.

The matlab format should not be used for experiments which have more than two independent variables. The standard format tables are used instead of matlab script if more than two independent variables are used.

An example of the usage of *MatLab* is given in Section 6.2.5.

4.3 Gnuplot Format

This format allows the *gnluplot* program to be used to dislpay the result data in the form of graphs.

A data file having a filename formed by the concatenation of the experiment name and the suffix `.dat` is created for each experiment, regardless of whether or not the -`s` option is selected. The data in the `.dat` file is organized into columns. For simple experiments having only one independent variable, the first column contains the values of the indendent variable. Each subsequent column contains the values for one of the dependent variables. When two independent variables are used, the first two columns correspond to the independent variables and the remaining columns are the dependent variable values. The following is a sample of a `.dat` file for a simple experiment with two dependent variables.

```
0.1 0.404224 0.040422
0.575 0.378937 0.217889
1.05 0.349995 0.367494
1.525 0.319965 0.487947
```

```
2 0.291062 0.582125
```

The usual result report file is generated in addition to the `.dat` files. The result report file has the usual header identifying the experiment name, srvn description filename, and the number of independent and dependent variables.

After the header, the result report file contains gnuplot script commands which cause gnuplot to display the experiment data in the data files as graphs. The following is a gnuplot script for an experiment having one independent variable and two dependent variables.

```
# Experiment Results
#
# Experiment: designs
# SRVN description file: cmpdesign.lqn
# Independent variables: 1
# Dependent variables: 1
#
set nokey

#set output "designs-1.ps"
set nolabel
set title "Plot of $X_{Clients}$ versus $N_{users}$"
set xlabel "$N_{users}$"
set ylabel "$X_{Clients}$"
plot "designs.dat" using 1:2 with points, "designs.dat" using 1:2 with lines
pause -1 "Hit return to continue"
```

4.4 Latex Format

The format provides a result report which is formatted for use with LaTeX. This file consists of a comment header followed by the LaTeX table description. The comment header identifies the experiment name and the SRVN description file name.

For experiments with a single independent variable, only one table is generated. Experiments with two independent variables will have result files containing a table for each dependent variable. These tables are of the same design as the tables described in the `standard` format section above. However, LaTeX macros have been used to allow the tables to be included into a document.

Each table is centered in a `\table` environment. The caption for the table provides the mapping between the experiment description file names for variables and the standard names used in the tables. Each table is given a symbolic label of the form "tab:*enamen*" where *ename* is the experiment name and n is the number of the table. Tables are numbered starting at 1. These labels may be used to reference the tables.

The result file may be included directly into a LaTeX document. An example of this is given in Section 6.2.6.

5 Other Files

This section describes the other files used by *MultiSRVN*. These files include the SRVN description files as well as the case description and case solution files. In particular, the naming convention for case files is discussed.

5.1 SRVN Description File

This file is described completely in [2]. The file may either be created by hand or by a tool such as *lqndef* or *TimeBench*. An example of this type of file is given in Section 6.2.1.

MultiSRVN should function successfully with any srvn description file which can be used with the current version of *srvn*, the task directed aggregation version of *srvn* or the current version of *parasrvn*.

5.2 Case File Names

The filenames for the case description files and for the case solution files are derived from various elements of the experiment description. The file name consists of the root `case`, a component for each independent variable, and the suffix ".in".

The file name components associated with the independent variables are formed by appending `-`*n* to the name for each independent variable, where *n* is the index of the value for that variable. For example, suppose an experiment with two independent variables was being performed where the first variable varies of ten values and the second variable varies over 5 values. The case file name for the case where the first independent variable was on its fifth value and the second independent variable was on its second value would be `case-05-02.in`.

5.3 Case Description File

The case description file is generated by *MultiSRVN* from the SRVN description file. *MultiSRVN* places a comment at the top of the file which identifies exactly the values of the independent variables responsible for the generation of this case.

The case may be solved independantly of all the other case by the invokation of the *srvn* program. This can be acomplished by the syntax such as:

```
lqns -o case-05-02.out case-05-02.in
```

The results can be found in the file "case-05-02.out".

5.4 Case Result File

This file is generated by the solver, for example *lqns*. The file will have the name, following the above examples, "case-05-02.out". This file has the same comment as the corresponding case description file. The comment information identifies the values of the independent variables.

5.5 Case Log File

This file is generated by *MultiSRVN* when the "-l" option is used. The file will have the name, following the above examples, "case-05-02.log". The file will contain any *MultiSRVN* or solver diagnostic messages that otherwise are printed to standard output and standard error. The information reported in the log file includes user-level convergence tests.

6 Examples

In this section, two complete examples are given of sessions using *MultiSRVN*. The first example demonstrates how *MultiSRVN* can be used to help a designer make design decisions to best use available resources. The second example shows how *MultiSRVN* can be used to explore the parameter space of a model.

6.1 Design Decisions

The example consists of a set of clients which make requests to a common application server and to a common file server. The file server is also used directly by the application server. All tasks run on their own processors.

Suppose that during the development process, it was determined that one of the system's functions could be performed either at the client tasks, or at the application server. The clients run input and screen management for the users, and can optionally also do a substantial amount of pre-processing of each request (checking, breakdown into file requests, etc.). Alternatively, the pre-processing could be executed by the application task, which would then continue to process the request and make the file accesses. However, the processors used by the Clients are significantly less powerful (say by a factor of ten) than the processor employed by the common application server. Both configurations are shown in Figure 3 complete with their parameters. The parameters of the pre-processing module are aggregated into the task which executes it. For instance, in the case of the "Server Compute" case, the phase execution times of the "Rqst" entry after aggregation are 1.4, 0.5, and 0. The pre-processing work is done by the module shown as a rectangle which is called from the Client task in the left-hand "Client Compute" case, and by the Application task in the "Server Compute" case.

6.1.1 The Client-Compute SRVN Description File

The following is the SRVN Description File, `cmpdesign.lqn`, for the model described above. The comments included in the file attempt to illuminate some of the syntax. A tool such as *TimeBench* or *lqndef* is useful in generating such files.

```
#                        General Information
G "Design Comparison"
0.000010                 # Convergience limit
50                       # Iteration limit
10                       # Print interval
0.500000                 # Coefficient of variation
-1

#                        Processor Information
P 0
  p Application f        # Processor Application FCFS
  p Clients     f i      # Infinite server processor Clients FCFS
  p FileServer  f        # Processor FileServer FCFS
-1

#                        Task Information
T 0
  t Clients r            # Reference task Clients
    Clients -1           # with entry Clients
```

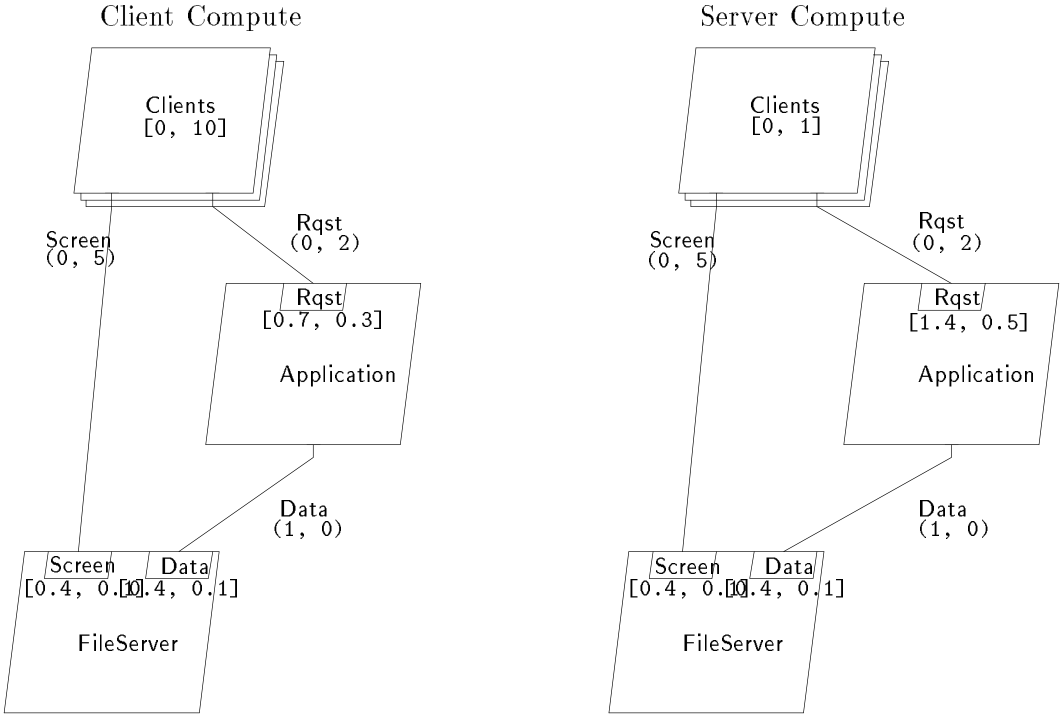


Figure 3: Example to study workload partitioning. The numbers under the entries are the mean phase service times and the numbers on the arcs are the mean visit ratios.

```
     Clients            # on processor Clients
     0 m 4              # having priority 0 and 4 instances

   t Application n      # Non-reference task Application
     Application_Rqst -1 # with entry Application_Rqst
     Application        # on processor Application
     0                  # having priority 0.

   t FileServer n       # Non-reference task FileServer
     FileServer_Screen  # with entry FileServer_Screen
     FileServer_Data -1 # and entry FileServer_Data
     FileServer         # on processor FileServer
     0                  # having priority 0.
-1

#                       Entry Information
E 0
# Phase Service Times   P 1    P 2    P 3

   s Application_Rqst   0.7    0.3    0.0 -1 # Design C
# s Application_Rqst    1.4    0.5    0.0 -1 # Design S
   s Clients            0.0   10.0    0.0 -1 # Design C
# s Clients             0.0    1.0    0.0 -1 # Design S
   s FileServer_Data    0.4    0.1    0.0 -1
   s FileServer_Screen  0.4    0.1    0.0 -1

# Call rates per phase  P 1    P 2    P 3

   y Application_Rqst
     FileServer_Data   1.0    0.0    0.0 -1
   y Clients
     Application_Rqst  0.0    2.0    0.0 -1
   y Clients
     FileServer_Screen 0.0    5.0    0.0 -1
-1
```

6.1.2 The Experiment File

The design comparison requires two *MultiSRVN* experiments, one in which the preprocessing computations are performed by the client and on in which these computations are done by the server. In both cases the number of clients is varied from 1 to 10. The following is the Experiment Description File, `cmpdesign.exp`, for these experiments.

```
.global
    .solver = "lqns";
    .template = "cmpdesign.lqn";
.end
```

Both experiments use the same solver and have the same template file. This is most easily accomplished through settings in the globals section as listed above.

The first experiment places the preprocessing responsibility on the clients. The following is the experiment description. Note that the labels are all given in LaTeX notation. This is so that later, when the *gnuplot* result format is used to produce a LaTeX graph, the labels will appear correctly formatted.

```
.experiment designc
    .declare
        users = [1-10], 10;
    .vary
        users;
    .control
        .tm ( Clients ) = { users };
    .observe
       .plot { users } "$N_{users}$";
        .th ( .t, Clients ) "$X_{Clients}$";
.end
```

The first experiment places the preprocessing responsibility on the clients. The following is the experiment description. Note that in this experiment, there is an initialization section. This section redefines the services times of the entries of the client and server tasks. This is how the preprocessing responsibility is shifted from the client to the server. In the previous experiment description, this section was not required since the template file already had these parameters correctly specified.

```
.experiment designs
    .initialize
# s Application_Rqst    1.4    0.5    0.0 -1 # Design S
        .st ( Application_Rqst, 1 ) = 1.4;
        .st ( Application_Rqst, 2 ) = 0.5;
# s Clients             0.0    1.0    0.0 -1 # Design S
        .st ( Clients, 2 ) = 1.0;
    .declare
        users = [1-10], 10;
    .vary
        users;
    .control
        .tm ( Clients ) = { users };
    .observe
       .plot { users } "$N_{users}$";
        .th ( .t, Clients ) "$X_{Clients}$";
.end
```

6.1.3 Solving The Experiments

The experiments were solved using the following *MultiSRVN* command.

```
MultiSRVN -v -F gnuplot cmpdesign.exp
```

6.1.4 Examining the Results

A gnuplot data file is generated for each experiment in the experiment description file. In this case, the data files are named `designc.dat` and `designs.dat` (the data files are not shown here). A single gnuplot commands file, `cmpdesign.gnu`, is also generated. It contains the gnuplot commands needed to make graphs from the gnuplot data files. The command

```
gnuplot cmpdesign.gnu
```

plots the designc data followed by designs data. Perhaps the best way to really compare these two data sets is to plot the client throughput for both designs on the same graph. To do this, the gnuplot commands files need to be edited by hand. The resulting file, called `cmpdesign2.gnu`, follows:

```
# Experiment Results
#
# Experiment: designc
# SRVN description file: cmpdesign.lqn
# Independent variables: 1
# Dependent variables: 1
#

set title "Plot of $X_{Clients}$ vs $N_{users}$"
set term eepic
set output "cmpdesign-graph.tex"
set xlabel "$N_{users}$"
set ylabel "$X_{Clients}$"
plot "designc.dat" using 1:2 title "Client Compute" with linespoints 1 1,\
     "designs.dat" using 1:2 title "Server Compute" with linespoints 1 2
#pause -1 "Hit return to continue"
```

The eepic gnuplot output option was selected here so that the gnuplot output could be encorporated directly in this document.

The performance characteristics plotted in the graph produced by this script are shown in Figure 4. Clearly, the optimal placement of the pre-processing workload is dependent upon the number of clients, and that with more than three clients, the “Client Compute” alternative is better. This analysis also suggests that some intermediate partitioning of the pre-processing may be optimal for systems with three to six clients.

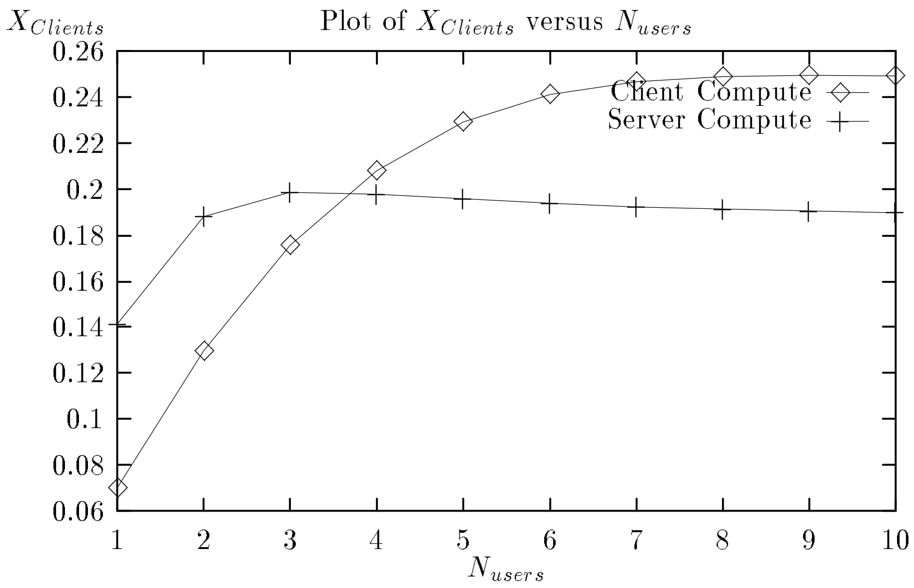


Figure 4: Comparison of throughput versus number of clients where functionality is moved between the clients and the server.

6.2 Exploring a Design

This example session with *MultiSRVN* begins with the presentation of a description of a model for which a performance study is to be done. The model is converted into the a SRVN description file. Next, an experiment description file is created to contain the experiments of the performance study and the experiments are solved. The `standard`, `matlab`, `latex` and *gnuplot* formats are examined in their respective result files and then *MatLab* and *gnuplot* are used to graph the results.

The example model being used here is a split pipeline having two branches. In addition, a server is used by some of the tasks in the pipline. Figure 5 represents the pipeline model.

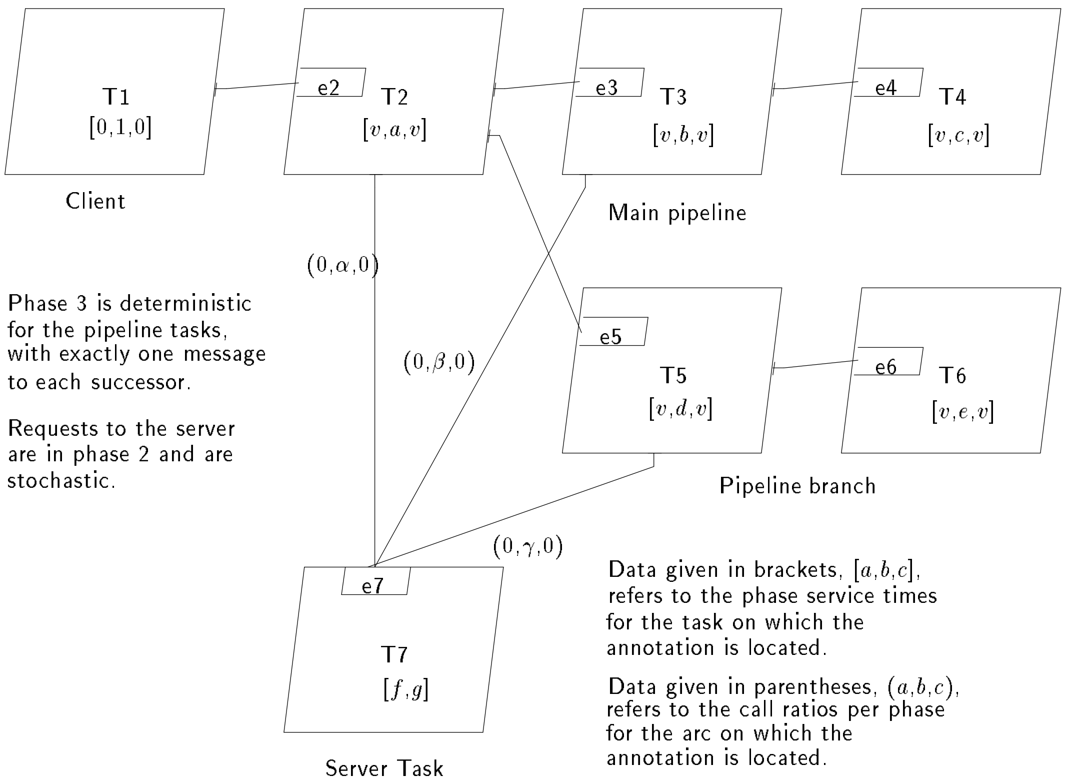


Figure 5: Pipeline example model diagram

The model consists of a reference task, `T1`, which drives a pipeline. The pipeline is a "split" pipeline in that there are two branches. One branch is formed by the tasks `T2`, `T3` and `T4` while the second branch consists of tasks `T2`, `T5` and `T6`. In addition, the tasks `T2`, `T3`, and `T5` each makes use of the server, `T7`.

The SRVN parameters associated with the various tasks and arcs are shown on their respective icons in Figure 5. In order to make the example easier to follow, some of the more important model parameters are named. Let the phase 1 and 3 service times for each of the pipeline tasks, T2 through T6, be represented by v. Let the phase 2 service times for tasks `T2`, `T3`, `T4`, `T5` and `T6` be represented by a, b, c, d, and e respectively. Let the call ratios in phase 2 to the server, `T7`, from `T2`, `T3` and `T5` be α, β, and γ. Let the phase 1 and phase 2 service times for the server, `T7`, be f and g respectively. Finally, let the throughput of task `T1` be X and the utilization for the processor for task `T7` be U.

In this example, three experiments are to be explored. The first experiment has the main branch of the pipeline balanced so that a, b and c are all 1. The server is not used in this experiment and so α, β and γ are all 0. The independent variables in this experiment are d and e which are each to be varied from 0.1 to 10 in 6 steps. The dependent variables are X and U.

In the second experiment, both branches of the pipeline are balanced. Therefore, a through e are all 1. The server is used in this example in such a way that tasks `T3` and `T5` compete for its use. This is accomplished by leaving α as 0, and varying the independent variables, β and γ, from 0.1 to 2 in 6 steps each. The server service times, f and g are both 0.5. The dependent variables are again X and U.

For the third experiment, the pipeline is again balanced with a through e being 1. The call ratios are $\alpha = 0$, β and γ both equal 0.5. Here, the independent variables are the service times of the server, f and g which are varied from 0.1 to 2 in six steps each. As before, X and U are the dependent variables.

6.2.1 The SRVN Description File

The srvn description file for this model may be created using *lqndef*, *TimeBench*, or by hand. The SRVN description file syntax is described in detail in [2]. The following is the contents of the SRVN description file, `pipeline.lqn`, for the pipeline example described above.

```
#                          General Information Section

G "This is a model representing a two-branch pipeline."
0.000010                 # Convergence limit
50                       # Iteration limit
1                        # Print interval
0.500000                 # Coefficient of variation
-1

#                            Processor Information
P 7
p P_T1 f                # Processor P_T1 FCFS
p P_T2 f                # Processor P_T2 FCFS
p P_T3 f                # Processor P_T3 FCFS
p P_T4 f                # Processor P_T4 FCFS
p P_T5 f                # Processor P_T5 FCFS
p P_T6 f                # Processor P_T6 FCFS
p P_T7 f                # Processor P_T7 FCFS
-1

#                               Task Information
T 7
t T1 r T1 -1 P_T1       # Reference task T1, Entry T1, Processor P_T1
t T2 n T2_e2 -1 P_T2    # Non-reference task T2, Entry T2_e2, Processor P_T2
t T3 n T3_e3 -1 P_T3    # Non-reference task T3, Entry T3_e3, Processor P_T3
t T4 n T4_e4 -1 P_T4    # Non-reference task T4, Entry T4_e4, Processor P_T4
t T5 n T5_e5 -1 P_T5    # Non-reference task T5, Entry T5_e5, Processor P_T5
t T6 n T6_e6 -1 P_T6    # Non-reference task T6, Entry T6_e6, Processor P_T6
t T7 n T7_e7 -1 P_T7    # Non-reference task T7, Entry T7_e7, Processor P_T7
-1

#                              Entry Information
E 7
```

```
#                                                 Phase 1   Phase 2   Phase 3
f T2_e2 0 0 1 -1       # Phase type:               sto.      sto.      det.
f T3_e3 0 0 1 -1       # Phase type:               sto.      sto.      det.
f T4_e4 0 0 1 -1       # Phase type:               sto.      sto.      det.
f T5_e5 0 0 1 -1       # Phase type:               sto.      sto.      det.
f T6_e6 0 0 1 -1       # Phase type:               sto.      sto.      det.
s T1 0 1 0 -1          # Phase service times:      0         1         - NA -
s T2_e2 0.05 1 0.05 -1 # Phase service times:      v         a         v
s T3_e3 0.05 1 0.05 -1 # Phase service times:      v         b         v
s T4_e4 0.05 1 0.05 -1 # Phase service times:      v         c         v
s T5_e5 0.05 1 0.05 -1 # Phase service times:      v         d         v
s T6_e6 0.05 1 0.05 -1 # Phase service times:      v         e         v
s T7_e7 0.5 0.5 0 -1   # Phase service times:      f         g         - NA -
y T1 T2_e2 0 1 0 -1    # Calls T1, T2_e2:          0         1         0
y T2_e2 T3_e3 0 0 1 -1 # Calls T2_e2, T3_e3:       0         0         1
y T3_e3 T4_e4 0 0 1 -1 # Calls T3_e3, T4_e4:       0         0         1
y T2_e2 T5_e5 0 0 1 -1 # Calls T2_e2, T5_e5:       0         0         1
y T5_e5 T6_e6 0 0 1 -1 # Calls T5_e5, T6_e6:       0         0         1
y T2_e2 T7_e7 0 0 0 -1 # Calls T2_e2, T7_e7:       0         alpha     0
y T3_e3 T7_e7 0 0 0 -1 # Calls T3_e3, T7_e7:       0         beta      0
y T5_e5 T7_e7 0 0 0 -1 # Calls T6_e6, T7_e7:       0         gamma     0
-1
```

6.2.2 The Experiment File

Once the model has been designed and the SRVN description file created, the experiments must be written into the experiment description file. In this example, the experiment description file is named `pipeline.exp`. The experiment description file consists of a global definitions section followed by a list of one or more experiments. The following is the global definitions section for this example.

```
# Set the MultiSRVN variables which apply to all the experiments to
# be performed.

.global
    .solver = "parasrvn";        # Use the simulator
    .template = "pipeline.lqn";  # Layered queue network description file

# The following is the list of workstations on which the cases
# are to be solved.

    .hosts = "aries, sunset, sunrise, kosmos, tiros, arcturus, voyageur";

.end
```

The following is the experiment description for the first experiment outlined above.

```
# This experiment features a balanced main pipeline where the phase
# two service times of the main branch are all 1.  The service times
# of the extra branch are to be varied.  The server is disconnected.

.experiment pipe1
    .declare
```

```
        d = [0.1 - 10], 5;     # Vary d from 0.1 to 10 in 5 steps.
        e = [0.1 - 10], 5;     # Vary e from 0.1 to 10 in 5 steps.

    .vary
        d;                     # Independent variable
        e;                     # Second independent variable

    .control
        .st ( T5_e5, 2 )  = { d };
        .st ( T6_e6, 2 )  = { e };

    .observe
       .plot { d } "d";
       .plot { e } "e";
        .th ( .t, T1 ) "X";    # Dependent variables
        .ut ( .p, P_T7 ) "U";
.end
```

Note the independent variables in the control section, the service times in phase 2 for tasks `T5` (entry `T5_e5`) and `T6` (entry `T6_e6`). The identifiers *d* and *e* have been associated with these controls.

Similarly, *s* and *t* have been associated with the dependent variables, the throughput for task `T1` and the utilization for processor `T7`.

The experiment description for the second experiment outlined above is as follows:

```
# This experiment has both branches of the pipeline balanced where the
# pipeline members' phase two service times are all 1.  Here, the
# server is introduced.

.experiment pipe2
    .declare
        beta = [0.1 - 2.0], 5; # Vary beta from 0.1 to 2 in 5 steps.
        gamma = [0.1 - 2.0], 5;        # Vary gamma from 0.1 to 2 in 5 steps.

    .vary
        beta;                  # Independent variable
        gamma;                 # Second independent variable

    .control
        .rr ( T2_e2, T7_e7, 2 ) = { beta };
        .rr ( T3_e3, T7_e7, 2 ) = { gamma };

    .observe
       .plot { beta } "beta";
       .plot { gamma } "gamma";
        .th ( .t, T1 ) "X";    # Dependent variables
        .ut ( .p, P_T7 ) "U";
.end
```

Here, the call ratios or "rendezvous rates" for tasks `T2` (entry `T2_e3`) and `T3` (entry `T3_e3`) are the independent variables. These have been named `beta` and `gamma`. The dependent variables are the same as for the previous experiment.

The third experiment defined above can be described as follows:

```
# In this experiment, we vary the service times of the server, keeping
# the rendezvous rates of the pipeline to server links at 0.5

.experiment pipe3
    .initialize
        .rr ( T2_e2, T7_e7, 2 ) = 0.5; # Initialize the call rates for
        .rr ( T3_e3, T7_e7, 2 ) = 0.5;  #   e2 and e3, beta and gamma, also.

    .declare
        f = [0.1 - 2], 5;      # Vary f from 0.1 to 2 in 5 steps.
        g = [0.1 - 2], 5;      # Vary g from 0.1 to 2 in 5 steps.

    .vary
        f;
       g;

    .control
        .st ( T7_e7, 1 ) = { f };
        .st ( T7_e7, 2 ) = { g };

    .observe
       .plot { f } "f";
       .plot { g } "g";
        .th ( .t, T1 ) "X";    # Dependent variables
        .ut ( .p, P_T7 ) "U";
.end
```

This description is similar to the previous two. Note that in this case, the SRVN description file is not quite what's needed for this experiment. The phase 2 call ratios to the server `T7` from tasks `T2` and `T3` must be set to 0.5. This can be accomplished by the first and second lines in the initialization section in this experiment description. The naming of parameters here is the same as with the previous two experiments.

These three experiments may be each placed in a separate experiment description file, or they may all be put into the same file. The latter option was selected for this example and file name given was "pipeline.exp".

6.2.3 Solving The Experiments

Solving the experiments is simply a matter of executing the *MultiSRVN* tool. This can be done with may different options to achieve several different outputs.

First, a standard format result file is generated which contains the results of each of the three experiments. Had each experiment been in a separate file, then three files would have been generated. The syntax for invoking *MultiSRVN* to obtain the desired results is as follows:

```
MultiSRVN -vlNN pipeline.exp
```

The **-v** option is used here so that the progress of the tool can be monitored; the **-l** option causes a log file to be generated for each case, one of which is shown below; the **-NN** option makes the solver run at priority 20. The simulator, *parasrvn*, is used in this example which takes approximately 90 minutes on an HP9000 735. The results may be found in the file "pipeline.res". Here is the log file for the `pipe1/case-01-01` case:

```
        parasrvn -p   -o ./pipe1/case-01-01.out ./pipe1/case-01-01.in
---------------------------[ Solver Output ]----------------------------
------------------------------------------------------------------------
```

Suppose that the results for the second experiment, `pipe2`, are desired in the `matlab` format. Then the following syntax could be used:

```
MultiSRVN -rv -e pipe2 -F matlab pipeline.exp
```

Since the case result files already exist, and no new controls are being manipulated, the **-r** option may be used. This causes *MultiSRVN* to reuse the existing data rather than recalculate it. The **-e** *pipe2* option selects the experiment named "pipe2" to be solved. The other experiments in the experiment file are ignored. Finally, the matlab format is selected using the option **-F** *matlab*. The result outputs are located in the file "pipeline.m".

Suppose further that the results for the third experiment are to be included in a LaTeX document. This can be achieved using the `latex` format as follows:

```
MultiSRVN -rv -e pipe3 -F latex pipeline.exp
```

As above, the **-r** and **-v** options are used to reuse the data and display the tool's progress. Using **-e** *pipe3* causes the third experiment to be solved in isolation. The **-F** *latex* option selects LaTeX output. The result outputs may be found in the file "pipeline.tex".

Now perhaps the results for the `pipe2` experiment would be interesting in a three dimensional plot. The format to use for this is the *gnuplot* format.

```
MultiSRVN -rv -e pipe2 -F gnuplot pipeline.exp
```

As above, the **-r** and **-v** options cause the data to be reused and the status of the run to be monitored. Here, the **-e** *pipe2* option and argument select the `pipe2` experiment for execution. The format is selected with **-F** *gnuplot*. The numeric results are located in the file "pipe2.dat" and the gnuplot commands for displaying the data are put into the file "pipeline.gnu".

6.2.4 Standard Results

Having "solved" the experiments as described above, the result reports may now be analyzed. The following is the result file "pipeline.res" generated by the first of the three invocations of *MultiSRVN* described above.

```
Experiment Results

Experiment: pipe1
SRVN description file: pipeline.lqn
Independent variables: 2
Dependent variables: 2

x1 -> d
x2 -> e
y1 -> X

                x2 0.1          2.575          5.05          7.525         10
      x1
```

```
0.1             0.484710    0.336780    0.190890    0.131630    0.099650
2.575           0.310420    0.241640    0.165600    0.119770    0.094230
5.05            0.179950    0.158510    0.126810    0.101040    0.082710
7.525           0.128010    0.117870    0.101430    0.088210    0.074630
10              0.095950    0.093910    0.084240    0.073680    0.068170

x1 -> d
x2 -> e
y2 -> U

             x2 0.1         2.575       5.05        7.525       10
     x1
0.1             0.000000    0.000000    0.000000    0.000000    0.000000
2.575           0.000000    0.000000    0.000000    0.000000    0.000000
5.05            0.000000    0.000000    0.000000    0.000000    0.000000
7.525           0.000000    0.000000    0.000000    0.000000    0.000000
10              0.000000    0.000000    0.000000    0.000000    0.000000

Experiment Results

Experiment: pipe2
SRVN description file: pipeline.lqn
Independent variables: 2
Dependent variables: 2

x1 -> beta
x2 -> gamma
y1 -> X

             x2 0.1         0.575       1.05        1.525       2
     x1
0.1             0.419480    0.385850    0.358550    0.319180    0.292810
0.575           0.388540    0.351230    0.322050    0.294830    0.267100
1.05            0.347340    0.325420    0.293620    0.267120    0.245800
1.525           0.320190    0.289940    0.266660    0.242420    0.221380
2               0.284040    0.263010    0.241950    0.218560    0.205800

x1 -> beta
x2 -> gamma
y2 -> U

             x2 0.1         0.575       1.05        1.525       2
     x1
0.1             0.084977    0.263226    0.408481    0.526153    0.619321
0.575           0.259408    0.408291    0.529903    0.617047    0.689484
1.05            0.405143    0.523852    0.613504    0.687881    0.742064
1.525           0.515732    0.608446    0.683911    0.743041    0.788012
2               0.600213    0.676060    0.738108    0.785529    0.817346

Experiment Results

Experiment: pipe3
SRVN description file: pipeline.lqn
Independent variables: 2
```

```
Dependent variables: 2

x1 -> f
x2 -> g
y1 -> X

              x2 0.1        0.575     1.05      1.525     2
      x1
0.1              0.419560   0.401660  0.377100  0.338750  0.302860
0.575            0.371660   0.353790  0.328490  0.299690  0.271140
1.05             0.318490   0.310900  0.291300  0.269370  0.241710
1.525            0.286260   0.271120  0.257940  0.238060  0.224190
2                0.253550   0.242880  0.232090  0.213400  0.202370

x1 -> f
x2 -> g
y2 -> U

              x2 0.1        0.575     1.05      1.525     2
      x1
0.1              0.085165   0.271735  0.433073  0.555460  0.645858
0.575            0.251501   0.409977  0.534712  0.630741  0.706023
1.05             0.373792   0.506215  0.603408  0.681603  0.744176
1.525            0.465581   0.573748  0.657660  0.721022  0.772795
2                0.532383   0.628942  0.697990  0.756564  0.794724
```

6.2.5 Matlab Results

Using *MatLab* is easy. To start a session with *MatLab* enter `matlab` at the UNIX command line; a console interface to *MatLab* is started. At the prompt, type in the name of the experiment to start the script. In this case enter "pipeline". The experiment name and a bit of other experiment related information is shown along with a menu of options as follows:

```
% Experiment Results
%
% Experiment: pipe2
% SRVN description file: pipeline.lqn
% Independent variables: 2
% Dependent variables: 2
%
 1)  Plot X vs gamma with beta
 2)  Plot U vs gamma with beta
 3)  Set capturing of graphs to files
 0)  Stop


Select a plot number:
```

The operator is requested to make a selection by entering a number. Enter a "1" to see the first graph, and a "2" to see the second graph. Option "3" controls the destination of the graphs. When this option is selected, the matlab script prompts with

```
Capture output to file? ( 1 for yes, 0 for no ):
```

If "yes" is selected by typing in a "1", the next plots made using menu items "1" and "2" are recorded. The graph files are in the *Encapsulated PostScript* format and are named *name*`g`*n*`.eps` where *name* is the experiment name and *n* is the menu item number, in this case 1 or 2. Figures 6 and 7 show the graphs as produced by *MatLab*.

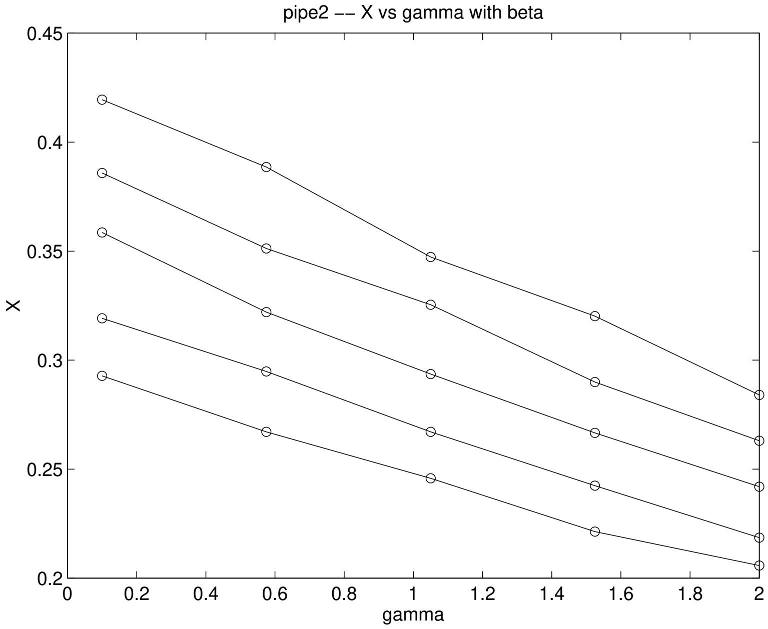


Figure 6: Graph of matlab results menu option 1

6.2.6 LaTeX Results

The third invocation of *MultiSRVN* given above generates the `latex` format result file. The following is the contents of the "pipeline.tex" file.

```
%% Experiment Results
%%
%% Experiment: pipe3
%% SRVN description file: pipeline.lqn
%% Independent variables: 2
%% Dependent variables: 2
%%
\begin{table}[htbp]
 \begin{center}
  \begin{tabular}{|r|rrrrr|} \hline
    & \multicolumn{5}{c|}{g} \\ \cline{2-6}
   \multicolumn{1}{|c|}{f} & 0.1 & 0.575 & 1.05 & 1.525 & 2 \\ \hline
   0.1 & 0.419560 & 0.401660 & 0.377100 & 0.338750 & 0.302860 \\
   0.575 & 0.371660 & 0.353790 & 0.328490 & 0.299690 & 0.271140 \\
   1.05 & 0.318490 & 0.310900 & 0.291300 & 0.269370 & 0.241710 \\
   1.525 & 0.286260 & 0.271120 & 0.257940 & 0.238060 & 0.224190 \\
   2 & 0.253550 & 0.242880 & 0.232090 & 0.213400 & 0.202370 \\
   \hline
  \end{tabular}
 \end{center}
```

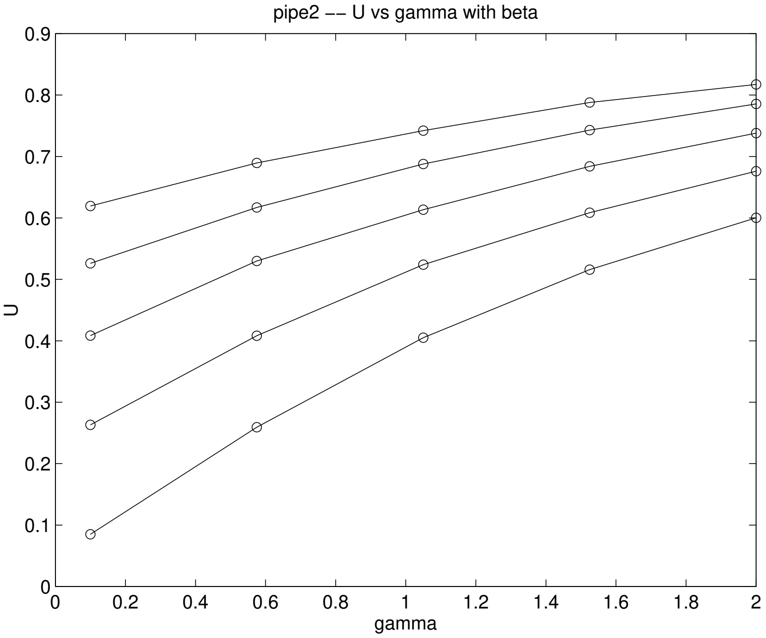


Figure 7: Graph of matlab results menu option 2

```
  \caption[Result table number 1 for experiment pipe3]{\label{tab:pipe31}
  Experiment results for experiment pipe3, observation of X }
\end{table}

\begin{table}[htbp]
 \begin{center}
  \begin{tabular}{|r|rrrrr|} \hline
    & \multicolumn{5}{c|}{g} \\ \cline{2-6}
   \multicolumn{1}{|c|}{f} & 0.1 & 0.575 & 1.05 & 1.525 & 2 \\ \hline
   0.1 & 0.085165 & 0.271735 & 0.433073 & 0.555460 & 0.645858 \\
   0.575 & 0.251501 & 0.409977 & 0.534712 & 0.630741 & 0.706023 \\
   1.05 & 0.373792 & 0.506215 & 0.603408 & 0.681603 & 0.744176 \\
   1.525 & 0.465581 & 0.573748 & 0.657660 & 0.721022 & 0.772795 \\
   2 & 0.532383 & 0.628942 & 0.697990 & 0.756564 & 0.794724 \\
   \hline
  \end{tabular}
 \end{center}
 \caption[Result table number 2 for experiment pipe3]{\label{tab:pipe32}
 Experiment results for experiment pipe3, observation of U }
\end{table}
```

The LaTeX tables in this report file may be inserted into a document or the macro `\input{file}` may be used to include the file. Each table has a label associated with it. The label is formed by the concatenation of the experiment name, in this case, "pipe3", and the number of the table (in this case there is only one table and so the table number is absent). See Tables 5 and 6 for examples of what the tables look like.

f	g				
	0.1	0.575	1.05	1.525	2
0.1	0.419560	0.401660	0.377100	0.338750	0.302860
0.575	0.371660	0.353790	0.328490	0.299690	0.271140
1.05	0.318490	0.310900	0.291300	0.269370	0.241710
1.525	0.286260	0.271120	0.257940	0.238060	0.224190
2	0.253550	0.242880	0.232090	0.213400	0.202370

Table 5: Experiment results for experiment pipe3, observation of X

f	g				
	0.1	0.575	1.05	1.525	2
0.1	0.085165	0.271735	0.433073	0.555460	0.645858
0.575	0.251501	0.409977	0.534712	0.630741	0.706023
1.05	0.373792	0.506215	0.603408	0.681603	0.744176
1.525	0.465581	0.573748	0.657660	0.721022	0.772795
2	0.532383	0.628942	0.697990	0.756564	0.794724

Table 6: Experiment results for experiment pipe3, observation of U

6.2.7 Gnuplot Results

The fourth invocation of *MultiSRVN* given above generates the `gnuplot` format result files. The results consist of a data file for each experiment and a gnuplot command file. In this case, the files are `pipe2.dat` and `pipeline.gnu`. The `pipe2.dat` consists of columns of data, one column for each dependent variable. There are sets of rows separated by a blank line. There is one such set for value of the second independent variable (if there are two independent variables) and one row within each set for each value of the first independent variable. The contents of the "pipeline.gnu" file are commands to gnuplot causing it to graph the data of the `.dat` file(s). The contents of these files are not shown here.

Using *gnuplot* is easy. Type `gnuplot` as a command to a shell as follows:

```
gnuplot pipeline.gnu
```

Each graph is shown in turn as the `Enter` is pressed. The gnuplot commands file, `pipeline.gnu`, may be edited to make gnuplot print the graphs into files. This is done by removeing the comment characters from the lines containing the output statements. In this example, lines 9 and 23 are affected:

```
 9:  #set output "pipe2-1.ps"

23:  #set output "pipe2-2.ps"
```

A three-dimensional plot may be made for experiments with two independent variables. This may be done for the pipeline example as follows. Start gnuplot is a shell and then enter the following commands:

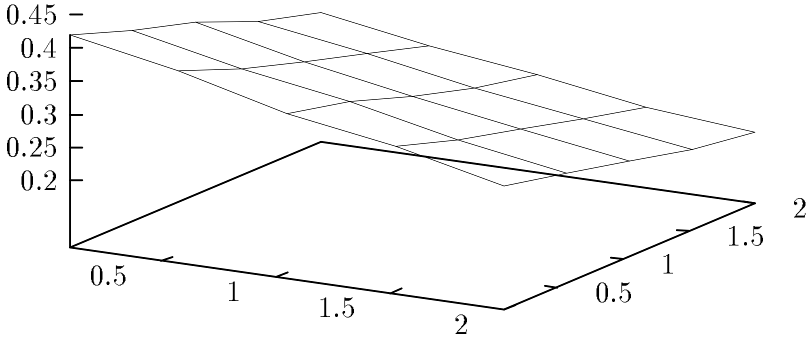


Figure 8: The experiment results in 3-D

```
set parametric

splot "pipe2.dat" with lines
```

If you then wish to have a LaTeX printout of the plot use the following commands:

```
set ouput "pipe2.tex"

set terminal eepic

set nokey

splot "pipe2.dat" with lines
```

Then, in a LaTeX document, use the LaTeX commands

```
\begin{figure}
 \begin{center}
  \input{pipe2}
 \end{center}
\end{figure}
```

The results of these efforts gets you the graph depicted in Figure 8.

7 Error Messages

This section provides listings of the error messages and descriptions of what they mean. There are two basic types of error message in *MultiSRVN*: file input errors and runtime errors.

File input error have the following format:

"*source*": line *num*: *error message text*

The *source* field is the source of the error message. This is the name of the source file, for example the experiment description file, being read when the error occurred. The *num* field is the line number of the line in the file on which the error is located. The *error message text* indicates the cause of the error and may suggest a means of fixing the problem.

Runtime errors have a the following format:

"*MultiSRVN*": (*severity*) *error message text*

The *severity* field indicates, indirectly, the type of error that occurred. Three possible severity levels exist, *warning*, *runtime error*, *fatal error*. The *error message text* explains the nature of the error and in some cases tries to suggest a means of fixing the problem.

7.1 Warnings

Warnings result from minor errors or problems which are not serious enough to warrant the termination of the program. A message is reported to the operator and if necessary, the current operation is terminated. However, pending operations are continued where possible.

`Unsupported data type in srvn description --` *file*

A data type was used in a control specification in the given experiment description file, *file*, which is not currently supported. It is intended that the type eventually be supported but for now, it must be avoided.

`Results may not be valid for srvn description --` *file*

An error may have occurred reading the SRVN description file. As a result, the results for this experiment may be invalid. Normally, if this warning occurs, it will be preceded by other warnings indication the nature of the problem with the SRVN description file.

`Observation request not supported --` *observation description*

The observation specification made in the experiment description file is not currently supported. It is intended that the feature eventually be included but for now it should be avoided. The *observation description* indicates the type of unsupported observation requested.

`Unable to locate experiment -- experiment name ''`*name*`''`

The experiment named *name* specified using the `-e` *ename* option was not located. This means that the experiment was not in the given experiment description file. Check for errors in the spelling of the experiment name.

`Option` *option* `not available`

The option, *option*, specified on the command line is not yet supported. It is intended that it will eventually be available but for now should be avoided.

7.2 Runtime Errors

These errors occur as a result of bad data, errors in the input files, errors in the command line options, etc. An occurrence of this type of error may cause the termination of the program. If recovery from the error is possible, pending operations will be continued.

`Symbol table full`

This error occurs if the SRVN description file is too large. Try to reduce the number of symbols in the SRVN description file is possible.

`Number of` *item* `is outside of program limits of (1,`*n*`)`

The SRVN description file contained a specification in which the number of instances of the entity *item* was outside of the program limits for that quantity. The limits are given as (1,*n*) where *n* is the maximum number allowed for instances of *item*.

`Processor` *proc_id* `for task` *task_id* `has not been defined`

In the SRVN description file, the processor *proc_id* must be defined before it can be used in the task declaration for task *task_id*.

`No entries have been defined for task` *task_id*

In the SRVN description file, a task declaration for task *task_id* was found in which the task had no entries defined. All tasks, reference or non reference must have at least one entry defined.

`Reference task` *task_id* `has more than one entry defined`

In the SRVN description file, the task declaration for task *task_id* was found to have more than one entry defined. Reference tasks may only have one entry.

`Argument to option` *option* `is missing`

The argument for the command line option, *option*, was omitted. To fix the problem, supply the option argument.

`Unable to open file ``*file*`'' --` *UNIX file error message*

The file named *file* could not be opened. The text *UNIX file error message* is the error message supplied by the Unix operating system indicating the reason that the file could not be opened. Usually, the file name was misspelled.

`Invalid srvn description file -- ``*file*`''`

The SRVN description file, *file*, contained an error and was therefore found to be invalid. This message is usually preceded by error messages which indicate the nature of the flaw within the file.

`Errors occurred --` *num* `experiments not completed`

This message indicates that *num* experiments were aborted as a result of one kind of error or another. This message will be preceded by other error messages indicating the nature of the problems with the various experiments.

`Unable to find` *item* `''`*item_name*`''` `to set` *property* `value to` *value*

An error occurred while attempting to set a controlled variable (independent variable) to a particular value. The item *item*, for example processor, having the name *item_name* could not be located in order to set the property *property*, for example scheduling, to the value, *value*.

`Unable to find` *item* `''(`*first_item*`,` *second_item*`)''` `to set` *property* `value to` *value*

This is the same error as the previous one except that in this case, the item, *item*, has two identifiers to specify it. This could be the rendezvous rate, or queue capacity, for example. The names *first_item* and *second_item* could not be found for the given *item*.

`Unable to open directory ''`*name*`'' --` *UNIX error message*

The directory named *name* could not be opened for one reason or another. The message *UNIX error message* is the Unix error message which explains the error.

`Solver ''`*solver*`'' stopped -- signal` *num* `received.`

The solver program named *solver* was stopped, returning the signal *num*.

`Error occurred in solver ''`*solver*`'' --` *error description*

The solver names *solver* stopped with a non-zero exit code. The text *error description* explains the reason for the error.

`Invalid case description file --` *file*

The case description file *file*, generated by *MultiSRVN* was found to be invalid. Typically, other error messages will precede this one to indicate why the file is invalid.

`Unable to locate observation data --` *property* `for` *item* `''`*item_name*`''`

The results for property *property* for item *item* with name *item_name* could not be found. For example, the throughput bound for task "task_id" could not be found. This error can occur if the "item_name" is invalid.

7.3 Fatal Errors

The error messages here are related to errors involving problems encountered with the UNIX library routines, interface errors between components of *MultiSRVN* as well as serious user related errors. These errors cause immediate termination of the *MultiSRVN* program.

7.3.1 General Errors

`Unable to open file ''`*file*`'' --` *UNIX error message*

The file *file* could not be opened. The text *UNIX error message* explains why this error occurred. This error is similar to the "Unable to open file" message in Section 7.2 except that this one is used for cases where *MultiSRVN* cannot complete without the use of *file*.

`Invalid experiment file -- ''`*name*`''`

The experiment file named *name* was found to be invalid. This means that some sort of syntax error was found in the experiment description file. Other error messages will precede this message and will indicate what the error was.

7.3.2 Glist Errors

These next few errors deal with interaction between the various modules of *MultiSRVN* and the *glist* abstract data type.

`Unable to add node to data list ''`*list_name*`''`

This message concerns the addition of data to the abstract data type *glist*. If the addition of data to the glist fails, this message results. The identifier *list_name* indicates the use to which the glist was being put.

`Unable to remove node from data list ''`*list_name*`''`

This message results from a failed attempt to retrieve data from the abstract data type *glist*. The identifier *list_name* indicates the use to which the glist was being put.

`Data node in list is bad -- list ''`*list_name*`'', position` *num*

This message is displayed when a node within an instance of the abstract data type *glist* is discovered to be invalid. The identifier *list_name* indicates which list contained the invalid node and *num* indicates the position of the bad node within the list.

`Index into data list is bad -- list ''`*list_name*`'', position` *num*

This message is displayed when an attempt is made to access a node within an instance of the abstract data type *glist* at a position which is invalid. The identifier *list_name* identifies the list and *num* gives the invalid position.

7.3.3 Code Mismatch Errors

These next few errors deal with mismatches in either initialization codes, control codes or scheduling flag characters between the experiment description file parser and the experiment manager modules.

`Bad initialization code -- code` *code*`, experiment ''`*name*`'', position` *pos*

The code *code* received as an initialization code for experiment *name* at position *pos* was not recognized. This error indicates that a code which passed the parsing phase of the experiment description file was not recognized during the execution of the experiment manager.

`Bad control code -- code` *code*`, experiment ''`*name*`'', position` *pos*

The control code *code*, found in experiment *name* at position *pos* and passed by the parsing phase of the experiment manager, was rejected by the experiment manager. This message indicates a mismatch between the parser and manager modules.

`Bad scheduling flag character -- ''em char''`

The character *char* intended as a scheduling flag in the experiment description file was not a member of the allowed characters for this purpose. However, the parsing phase of the experiment description file was passed so the experiment description file parser and experiment manager are mismatched.

7.3.4 Case File Errors

These next errors occur in connection with the generation of case file names or the case files themselves.

`Position number in filename is too big -- filename ''file'', position` *pos*

This error indicates that the iteration number to to create unique identifiable case filenames was found to be too large. The number of iteration allowed is 999.

`Model comment is too large --` *comment text*

The comment given as *comment text* was too large. This is the comment which is placed in the case description files by *MultiSRVN*. The maximum allowed size is 1024 characters. It is unlikely that this limit will be stretched.

7.3.5 Result Reporting Errors

`Dimension too large for list --` *list_name*

This error is generated in the result report generator module if an attempt is made to write to the list of independent variables values at a dimension position which is larger than the number of independent variables.

7.3.6 System Errors

These errors occur when some vital system resource such as memory is unavailable.

`Library function ''strdup()'' failed`

The strdup() library call returned (char *)0 which means that no memory was available for the duplication of the string.

`No more memory`

An attempt was made to allocate memory which has failed. The cause could be that the requested memory block was unreasonably large or that the system has insufficient memory left as a result of competing processes.

7.4 Experiment File Syntax Errors

When a syntax error occurs during the reading of the experiment description file, a message of a similar format to the format used in the file input error described above in Section 7 is emitted. In addition to displaying the source and line number, these messages also display the expected input text and the actual received text.

It is possible for the error routines to display the line number for the next line after the line on which the error occurred. Since the expected and actual text are displayed, this should not cause too much trouble.

8 Support Programs

MultiSRVN's interface to the operator is through files and command line argument and options. This permits the use of pre and post processing.

For example, it would be possible to create a SunView or OpenLook interface which would provide an interactive way for the operator to create the experiment file. Similarly, the *TimeBench* program could be modified to include an interface to *MultiSRVN*.

References

[1] C.M. Woodside, J.E. Neilson, D.C. Petriu, S. Majumdar, *The Stochastic Rendezvous Network Model for Performance of Synchronous Multi-Tasking Distributed Software*, March 1991.

[2] Dorina Petriu, *SRVN Input File Format*, Real Time and Distributed Systems Group, Carleton University, Ottawa, Canada, March 1988.

[3] J.E. Neilson, C.M. Woodside, D.C. Petriu, J.W. Miernik, *A Short Tutorial on Stochastic Rendezvous Network Models*, Report CSM-7.1m Communication Software-Modelling Project, Carleton University, Ottawa, Canada, May 1988.

A Release Notes

A.1 Bugs and Mis-features

1. The `.ET`, and `.TT` sub-categories are not implemented. The output generated by the solver programs is not complete enough to use these sub-categories as yet.

A.2 Release History

1.0 This was the initial release of the program.

1.1 Some command line options were added.

1.2 The following changes were made for this release:

1. Some options were added.
2. The syntax of experiment description file was changed. Specifically, the queue capacity and queue rate material was added and the range and list specifications changed.

1.3 The following changes were made for this release:

1. The -s option was added to allow the output to be split into a separate file for each experiment.
2. The -c option was added to delete unwanted case files before new case files are generated.
3. The -O option to allow options to be passed to the solver utilities was put into the command line syntax.
4. The queue initial size information was put into the experiment description language.
5. The output generated by the *gnuplot* format was enhanced.
6. Some repairs to the *matlab* format output were made.

Index