Paper AD03

That's the Signpost Up Ahead— Your Next Stop, SAS IML

Gary Allen Senior Database Analyst – Wunderman Brands

Introduction

SAS IML is a mathematics programming language within the SAS system. SAS IML is a separate language within SAS that enables custom procedures that use matrix algebra, non linear optimization, and more. Getting started in IML begins with the “proc iml ;” statement and ends with the “quit” statement. To use iml data needs to be imported and reports need to be produced, and data needs to be exported. This paper will cover the steps to do these necessary tasks.

Direct Entry

The first and most direct way to establish a matrix is by assigning values to a matrix name. For example:

my_matrix = {1 0, 0 1} ;

sets my_matrix equal to a 2 x 2 identity matrix. There are a few things to note here. The values of the matrix are specified between { } brackets, these are the only bracket style that can be used for this. The values are separated by spaces and rows are separated by commas. The technical name for a matrix entered in this way is a *literal.* Matrix values can be numbers or text. A matrix may be all numbers or all text but not both.

Common Operators

The statements covered in this paper have common functionality called operators. The var, where and labeling operators will be covered in this section.

Var Operator

The var operator lets you specify a set of dataset variables to use. The var operator has a single argument which may be one of the following:

Argument Type	Description	Example

Argument Type	Description	Example
A literal containing variable names		`var {time1 time5 time9}`
An expression in parentheses yielding variable names		`var('time1':'time9')`
One of the keywords _ALL_ , _NUM_ , _CHAR_	Use _ALL_ for all variables, _NUM_ for all numeric variables and _CHAR_ for text variables.	`var _all_`
The name of a matrix containing variable names		`var time`

Where Operator

The WHERE operator selects observations from datasets based on the conditional statement in its argument, The syntax is ;

WHERE (*conditional statement*)

The condition statement is made up of a *variable*, a *comparison-operator* and *a operand,* which is a literal value, a matrix name, or an expression in parentheses. The comparison operators are:

Operator	Function
<	less than
<=	less than or equal to
=	equal to
>	greater than
>=	greater than or equal to
^=	not equal to
?	contains a given string
^?	does not contain a given string
=:	begins with a given string
=*	sounds like or is spelled similar to a given string

Arguments in WHERE operators can be matrices. A note about comparisons using Matrices: A “true” is returned with ^= ^? < <= > >= if *all* the elements in the matrix satisfy the condition. Whereas the = ? =: =* operators return a true if *any* of the elements in the matrix satisfy the condition.

The logical expressions AND (&) and OR (|) can be specified within the WHERE clause. Only the symbols “&” and “|” can be used.

Labeling Operators: Row Name – Column Name Statements

The ROWNAME= and COLNAME= specify matrices to be used as labels in reports or variable names in SAS dataset output. The COLNAME= matrix specifies the name of a matrix that will label the columns. When importing data, the variables names are kept in the COLNAME matrix, when exporting data the text values in the COLNAME matrix name the variables. The COLNAME matrix is actually a row vector of text values.

The ROWNAME matrix is used to label observations. When importing data the values of the specified variable is kept and used to label reports. When data is exported the information in the ROWNAME matrix is placed in the dataset under a variable of the same name. The ROWNAME matrix is a column matrix with as many text values as observations.

Importing Data

SAS Datasets

SAS datasets are imported using a two statement process. The first statement points to a dataset to use, and the second statement reads the data into one or more matrices.

Use Statement

The USE statement points to a SAS dataset whose information will be imported. The syntax for the statement is:

USE *SAS-data-set* *optional* ***VAR*** *operator*
optional ***WHERE****(argument) operator*
optional ***NOBS*** *name* **;**

The arguments in USE statement are a SAS dataset. The dataset can be a one word or two word name. Also, the dataset can be specified with dataset options (i.e. keep =, drop = etc.). The USE statement opens the data set for read access and sets it as the current dataset.

Read Statement

The READ statement imports data from the current dataset into one or more matrices. The READ statement has two forms:

READ *range*
VAR *operator*
optional ***WHERE****(argument) operator*

and :

READ *range*
optional ***VAR*** *operator*
optional ***WHERE****(argument) operator*
optional ***INTO*** *matrix – name* ***[ROWNAME =*** *row-name matrix*
COLNAME = *column-name matrix****] ;***

The READ statement selects variables or records from the current SAS data set into column matrices or into a single matrix. The first form of the READ statement will produce a column matrix for each variable in the VAR operator. The second form will read all text or numerical variables specified by the VAR operator into the matrix named in the INTO argument. If you do not specify a VAR clause, the default variables read into the INTO matrix are all the numeric variables. To read all character variables into the INTO matrix use VAR _CHAR_.

Sample Code

```
use inner ;

read all var _all_ into ind ;
read all var _char_ into lvlabl ;

close inner ;
```

This code selects the local SAS dataset inner as the current dataset with the USE statement. The next two lines import data with the READ statement. The first READ statement imports all the numeric variables into the matrix “ind”. Note that *all* the observations are selected. The second imports all text variables into the matrix “lvlabl”.

CSV Files

The process for reading flat files into IML is parallel to the same process in base SAS. The process has an infile and an input statement. They are presented here because they are different and have limitations.

Infile Statement

The INFILE statement sets the flat file to use for input. The INFILE statement has the syntax:

INFILE *file indicator options* **;**

The *file indicator* can be a literal in quotes, a keyword set with a filename statement or an expression in parentheses. It should be noted that with IML the file path and filename cannot be longer than 64 characters. The options that can be used are a subset of those in the BASE SAS version, and have addition limitations. The options are :

LENGTH=variable specifies a variable that is length of a record.
RECFM=N specifies that the file is to be read in as a pure binary file rather than as a file with record separators. The byte operand (<) on the INPUT statement is used to get new records rather than using separate input statements or the new line (/) operator.
To control how IML handles reading past the end a record, the option familiar in BASE SAS are used:
FLOWOVER allows the INPUT statement to go to the next record to obtain values for the variables. **MISSOVER** assigns missing values to variables read past the end of the record and **STOPOVER** treats going past the end of a record as an error condition, and triggers an end-of-file condition. The default is STOPOVER.

Input Statement

The IML INPUT is also parallel to the same statement in BASE SAS. The INPUT statement reads data from the current flat file into matrices. The syntax is :

I**NPUT** *variables optional informats*
optional record-directives
optional positionals ***;***

The variables argument specifies the variables you want to read from the current position in the record. Each variable can be followed immediately by an input informat specification, using the *informats* argument. Standard BASE SAS informats are available. If the width is unspecified, the informat uses list-input

rules to determine the length by searching for a blank (or comma) delimiter. The special format $RECORD can be used to read the rest of the record into one text variable.

The *record-directives* are used to advance to a new record. The *Record-directives* available are the holding @ sign is used at the end of an INPUT statement to instruct IML to hold the current record so that you can continue to read from the record with later INPUT statements. If the @ is not used IML automatically goes to the next record for the next INPUT statement. The / sign advances to the next record.

The > *operand* and the < *operand* are used with the RECFM = n option in the infile statement for reading in file by byte position. The < specifies that the next record to be read starts at the indicated byte position in the file. The > instructs IML to read the indicated number of bytes as the next record.

The *positionals* instruct PROC IML to go to a specific column on the record. The *positionals* operators available are: @ *operand* which goes to the indicated column and the + *operand* which the indicated number of columns. The operands can be a literal number, a variable name, or an expression in parentheses.

Sample Code

```
proc iml ;

loop = 1 ;

infile "D:\my_folder\iml_test_data.csv"  ;

do data ;
  input name $ number1 number2 ;

  if loop = 1 then do ;
    name_    = name    ;
    number1_ = number1 ;
    number2_ = number2 ;
  end ;
  if loop > 1 then do ;
    name_    = name_    // name    ;
    number1_ = number1_ // number1 ;
    number2_ = number2_ // number2 ;
  end ;

  loop = loop + 1 ;
```

```
end ;

closefile “D:\my_folder\iml_test_data.csv” ;
matrix1 = number1_ || number2_ ;

quit ;
```

This sample code starts with a straight forward INFILE statement. Next it executes a loop with the special operand “data” that goes from start of file to end of file. The interior of the loop concatenates variables newly read in to column matrixes. After the loop the column matrixes are concatenated into a single matrix.

Exporting Data

SAS Datasets

Exporting data to SAS datasets parallels importing in that it has a two statement syntax where the first sets the current SAS dataset and the second assigns matrix values to the records. To set the dataset a CREATE statement is used and to assign records an APPEND statement is used.

Create Statement

The CREATE statement has two syntaxes:

> **CREATE** *SAS dataset optional Var Operator* **;**
>
> **CREATE** *SAS dataset* ***FROM*** *matrix*
>
> > *Optional label operator* ***[COLNAME=*** *column label matrix*
> > ***ROWNAME =*** *observation label matrix****]*** **:**

The CREATE statement sets a new current SAS data set and makes the dataset available fro both read and write. The variables in the new SAS data set are either the variables specified with the VAR operator or from the columns of the matrix in the FROM statement. The FROM clause and the VAR clause should not be used together.

The *SAS dataset* can be a one-word name or two-word name. The ma*trix in* the FROM clause is a existing matrix with the data you want to export. Although it must be noted the actual exporting is not done here, but with the subsequent APPEND statement.

The text values in the *column label matrix* become the names of the variables in the new SAS dataset. And if an *observation label matrix* is specified, the same ROWNAME= matrix must also be used on subsequent APPEND statements.

The variable types and lengths are inherited from the current attributes of the matrices specified in the VAR operator or the matrix used in the FROM clause. If no variables are specified all the matrixes columns are used.

Append Statement

The APPEND statement does the actual exporting to the current dataset opened by the CREATE statement. The APEND statement also has two forms:

APPEND *optional* ***VAR*** *operator* ;

APPEND *optional* ***FROM*** *matrix*

optional ***[ROWNAME =*** *observation label matrix****]*** ;

The form of the APPEND statement used must be the same as the CREATE statement used. The APPEND statement to adds data to the end of the current output data set. The FROM clause and the VAR clause should not be specified together.

Sample Code

```
create outdat.fedata from feffall[colname = varlabl rowname = obslabl] ;
append from feffall [rowname = obslabl] ;

close outdat.fedata ;
```

This code creates a SAS dataset, outdat.fedata from the matrix feffall. The variables in the dataset will have the names of the text values in varlabl. So these must be valid SAS names (no spaces or non alphanumeric characters, etc.). Also the will be a variable called obslabl with the values in the column matrix obslabl.

CSV Files

File Statement

Like importing data from a flat file, exporting data involves statements with the same names that are used in BASE SAS. First use a FILE statement to set an open flat file and a PUT statement is used to write values to the flat file.

The syntax for the FILE statement is :

FILE *file indicator options* ;

The *file indicator* can be a literal in quotes, a keyword set with a filename statement or an expression in parentheses. It should be noted that with IML the file path and filename cannot be longer than 64 characters. The options that can be used are a subset of those in the BASE SAS version, and have addition limitations. The options are :

RECFM=N specifies that the file is to be written as a pure binary file without record-separator characters. LRECL=*operand* specifies the record length of the output file. The default record length is 512.

Two special filenames that are recognized by IML: LOG and PRINT. These refer to the standard output streams for all SAS sessions.

Put Statement

PUT *variables optional formats*
optional record-directives
optional positionals ;

The variables argument specifies the variables you want to read from the current position in the record. Each variable can be followed immediately by a format specification, using the *formats* argument. Standard BASE SAS formats are available.

The *record-directives* are used to advance to a new record. The *Record-directives* available are the holding @ sign is used at the end of an INPUT statement to instruct IML to hold the current record so that you can continue to read from the record with later INPUT statements. If the @ is not used IML automatically goes to the next record for the next INPUT statement. The / sign advances to the next record.

The > *operand* and the < *operand* are used with the RECFM = n option in the infile statement for reading in file by byte position. The < specifies that the next record to be read starts at the indicated byte position in the file. The > instructs IML to read the indicated number of bytes as the next record.

The *positionals* instruct PROC IML to go to a specific column on the record. The *positionals* operators available are: @ *operand* which goes to the indicated column and the + *operand* which the indicated number of columns. The operands can be a literal number, a variable name, or an expression in parentheses

Sample Code

```
proc iml ;

filename csvout "D:\MY_FOLDER\sample_csv_out.csv"  ;


a = { 1 2 3, 4 50 66, 70 88 9.9 } ;

file csvout ;

do i=1 to nrow(a);
   do j=1 to ncol(a);
      put (a[i,j]) best3. ',' @ ;
   end;
   put;
end;

closefile csvout;
quit ;

endsas ;
```

In this code sample, the rows and columns of matrix a are looped through and output to the flat flie “sample_csv_out.csv”. The “@” position holding operator is used to stay on the same row until every column is outputted.

Conclusion

This paper has outlined the methods of getting data into and out of SAS IML. With this information and some practice at putting it to use, you can get started with programming in IML. I should say that programming in IML has been the most fun and rewarding SAS programming I have done.