Week 10-11 [17+ Nov.] Class Activities

File: week-10-11-MACRO-prog-16nov08.doc
Directory: \\Muserver2\USERS\B\baileraj\Classes\sta402\handouts

From: Chapter 9 - MACRO programming

9. MACRO programming
   0. What is a macro and why would you use it?
   1. Motivation for Macros: numerical integration to determine P(0<Z<1.645)
   2. Macro processing
   3. Macro variables
   4. Conditional execution, looping and macro programs
   5. Debugging macro coding and programming
   6. Saving macros - %include +autocall+stored compiled macros
   7. Functions/routines of potential interest to macro programmers - %index, %length, %eval, symput, symget

Exercises

9.0 What is a macro and why would you use it?

* The macro processor is a text processor that is built into SAS.
Q: Why learn more about this text processor?
A: You are executing the same programs with minor modification with some regularity.

9.1 Motivation for Macros: numerical integration to determine P(0<Z<1.645)

Trapezoidal rule approximates the area under a curve, f(x), over some specified limits of integrations, say “low” and “high”, by summing the areas of a collection of adjacent trapezoids constructed for a collection of points \([x_1,f(x_1)]\), \([x_2,f(x_2)]\), … , \([x_k,f(x_k)]\) where \(x_1=\text{”low”}<x_2<…<x_{k-1}<x_k=\text{”high”}\) (see Burden and Faires (1989) for more description of numerical integration methods).
If \( x_{i+1} - x_i = h \) for \( i=1,\ldots,k-1 \), then the estimated area can be written as

\[
\hat{A} = \frac{1}{2} f(x_1) + f(x_2) + \cdots + f(x_{k-1}) + \frac{1}{2} f(x_k) \times h.
\]

This area estimator is implemented in the program listed in Display 9.1 where \( k=25 \), \( x_1=0 \), \( x_k=1.645 \), \( h=(x_k-x_1)/24 \) and \( f(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2) \).

Display 9.1: Program for estimating \( P(0<Z<1.645) \) using the trapezoidal rule

```sas
/*
Calculate \( P(0 < Z < 1.645) \) using the trapezoidal rule */
data trapper;
retain trapsum 0;
array x_value(25) x1-x25;
array f_value(25) y1-y25;
low = 0;
high = 1.645;
incr = (high-low)/24;
pi = arcos(-1);
do i= 1 to 25;
    x_value[i] = low + incr*(i-1);
    f_value[i] = (1/sqrt(2*pi))*exp(-x_value[i]**2/2);
if i=1 or i=25 then trapsum = trapsum + f_value[i]/2;
else trapsum = trapsum + f_value[i];
end;
area_est = trapsum*incr;
output;
ods rtf bodytitle;
proc print data=trapper;
title "Trapezoidal Rule area estimate for \( P(0 < Z < 1.645) \)"
  var low high incr area_est;
run;
data trapper2;
set trapper;
array x_value(25) x1-x25;
array f_value(25) y1-y25;
do ii=1 to 25;
    xout = x_value[ii];
    yout = f_value[ii];
    output;
end;
proc print data=trapper2;
title "Interpolation points for Trapezoidal Rule"
  var ii low high incr area_est xout yout;
run;
```
arrays used to store the \([x_i, f(x_i)]\) pairs of data

retain used to accumulate the sum of the trapezoidal pieces.

Code generates and outputs the pairs of \([x_i, f(x_i)]\) based on a specification of the limits of integration, calculates the area estimate, prints various elements of this analysis and plots the function values. The first product of this program, the estimated area under the standard normal density, is estimated to be 0.44995 as we see in Display 9.2.

Display 9.2: Output containing the estimated P(0<Z<1.645)

\[\text{Trapezoidal Rule area estimate for } P(0 < Z < 1.645)\]

<table>
<thead>
<tr>
<th>Obs</th>
<th>low</th>
<th>high</th>
<th>incr</th>
<th>area_est</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1.645</td>
<td>0.068542</td>
<td>0.44995</td>
</tr>
</tbody>
</table>

Display 9.3: Output containing the estimated P(0<Z<1.645) along with the function values used in the trapezoidal rule calculation.

\[\text{Interpolation points for Trapezoidal Rule}\]

<table>
<thead>
<tr>
<th>Obs</th>
<th>ii</th>
<th>low</th>
<th>High</th>
<th>incr</th>
<th>area_est</th>
<th>xout</th>
<th>Yout</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1.645</td>
<td>0.068542</td>
<td>0.44995</td>
<td>0.00000</td>
<td>0.39894</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1.645</td>
<td>0.068542</td>
<td>0.44995</td>
<td>0.06854</td>
<td>0.39801</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1.645</td>
<td>0.068542</td>
<td>0.44995</td>
<td>0.13708</td>
<td>0.39521</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1.645</td>
<td>0.068542</td>
<td>0.44995</td>
<td>0.20563</td>
<td>0.39060</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0</td>
<td>1.645</td>
<td>0.068542</td>
<td>0.44995</td>
<td>0.27417</td>
<td>0.38423</td>
</tr>
</tbody>
</table>
[Q] could you modify this program to have vertical lines connecting the density values to the x-axis and connect the density values to display the trapezoids being summed to obtain the estimated area? It would be even cooler to superimpose the true density.

Display 9.4: Output containing the $[x,f(x)]$ values used in the trapezoidal rule estimated of $P(0<Z<1.645)$
A number of questions arise at this point. We have a very specific implementation for a particular problem. What might you want to do now?

Q₁: How hard would it be to modify this code if we wanted \( k = 50 \) points vs. \( k = 25 \) points?

Q₂: Could we generalize this code to specify different limits of integration?

Q₃: Could we set up a “switch” to produce graphics only when requested?

Q₄: Is it possible to package this estimator into a callable “subroutine” or “function”?

Q₅: Could we generalize this code for any arbitrary (corrected coded) function?

(For the impatient, the answers to these questions are: not too hard, easily, easily, yes and yes, respectively. The common denominator to respond to all the questions is that macros in SAS will be employed.)

9.2 Macro processing
Take a quick look at how SAS programs process a program: (Readers with interests in more detailed descriptions are directed to the SAS help documentation “Introduction to SAS Programs and Macro Processing” or Burlew (1998).)

* When program statements are submitted they are sent to the input stack, an area of computer memory.

* The word scanner processes these program statements by converting them to tokens which are directed to an appropriate location for additional processing.

* Appropriate locations include the data step compiler, command processor and macro processor.

* If a program contains any macro statements, then SAS sets up a symbol table for the macro variables (actually SAS defines certain macro variables every time you use it, e.g. the day you run SAS is stored in a macro variables SYSDAY), and needs to resolve any macro requests that are part of a series of commands before these commands are processed.

* So, how does the word scanner “know” to redirect tokens to the macro processor? Any non-blank text that is preceded by a percent sign (\%) or ampersand (&) is a macro trigger. When the word scanner encounters a trigger, it sends the tokens with %/& preceding text to the macro processor

* macro processor will interact with the symbol table to store or to extract values of macro variables as needed, or more generally, interact with the word scanner to resolve any macro statements. The macro processor will substitute the text demanded of it from the macro statements into the input stack.
and this interchange will continue until the input stack has been completely processed. This will be clearer when we look at how code with macro statements and variables are resolved.

9.3 Macro variables, parameters and functions

A macro variable …

* can be referenced anywhere in a SAS program (other than in data lines that are being read into SAS).

* can be assigned text values, and the value of such a variable is stored in either local or global symbol tables.

* is named using a valid SAS name

* is preceded by an & when it is referenced in a SAS program.

* if it follows text, then it is referenced as text&macro-variable.

* if it precedes text, then it is referenced with a period delimiting the value from text, i.e. &macro-variable.text.

* can be “automatic macro variables,” while a program can define others, so-called “user-defined macro variables.”

How will you usually start using MACROS in SAS?

A: most likely use macro variables to define simple substitutions in a program. In Display 9.5, we modify the trapezoidal numerical integration routine to address Q₁ and Q₂ from above -

Q₁: How hard would it be to modify this code if we wanted $k=50$ points vs. $k=25$ points?
Q2: Could we generalize this code to specify different limits of integration?

Use `%LET` to assign values to macro variables `NPTS` (the macro variable for \( k \)), along with `LOW` and `HIGH` (the macro variables that correspond to the limits of integration).

Display 9.5: Replacing the limits of integration and number of points to evaluation in a trapezoidal rule estimation of \( P(low < Z < high) \).

```sas
/* Calculate \( P(low < Z < high) \) using the trapezoidal rule */
%let npts = 50;
%let LOW = -1.645;
%let HIGH = 1.645;

data trapper;
  file "C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est.out" MOD;
  retain trapsum 0;
  array x_value(&npts) x1-x&npts;
  array f_value(&npts) y1-y&npts;

  low = &LOW;
  high = &HIGH;
  incr = (high-low)/( &npts -1);
  pi = arcos(-1);

  do i= 1 to &npts;
    x_value[i] = low + incr*(i-1);
    f_value[i] = (1/sqrt(2*pi))*exp(-x_value[i]*x_value[i]/2);
    if i=1 or i=&npts then trapsum = trapsum + f_value[i]/2;
    else trapsum = trapsum + f_value[i];
  end;
  area_est = trapsum*incr;
  output;

put;
put "est. \( P(\text{LOW} < Z < \text{HIGH}) \) = " area_est "(based on &NPTS points)";
put;
ods rtf bodytitle;
proc print data=trapper;
  title "Trapezoidal Rule area estimate for \( P(\text{LOW} < Z < \text{HIGH}) \)";
  title2 "(based on &NPTS equally spaced points)";
  var low high incr area_est;
run;
```
data trapper2;
  set trapper;
  array x_value(&npts) x1-x&npts;
  array f_value(&npts) y1-y&npts;
  do ii=1 to &npts;
    xout = x_value[ii];
    yout = f_value[ii];
    output;
  end;
end;

proc print data=trapper2;
  title "Interpolation points for Trapezoidal Rule";
  var ii low high incr area_est xout yout;
  run;

proc gplot data=trapper2;
  title "Plot of function values vs. x-values";
  plot yout*xout;
  run;
ods rtf close;

Display 9.6: PROC PRINT output with the numerical integral of \( P(-1.645 < Z < 1.645) \)

Trapezoidal Rule area estimate for \( P(-1.645 < Z < 1.645) \)
(based on 50 equally spaced points)

<table>
<thead>
<tr>
<th>Obs</th>
<th>low</th>
<th>high</th>
<th>incr</th>
<th>area_est</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1.645</td>
<td>1.645</td>
<td>0.067143</td>
<td>0.89990</td>
</tr>
</tbody>
</table>

Display 9.7: Output written to the file “est.out” by 5 submissions of the program in Display 9.5 in which \%LET npts = 50; (or =25, 15, 5, or 3) is specified.

| est. \( P(-1.645 < Z < 1.645) \) =0.8999027424 (based on 50 points) |
| est. \( P(-1.645 < Z < 1.645) \) =0.8994989035 (based on 25 points) |
| est. \( P(-1.645 < Z < 1.645) \) =0.8984685898 (based on 15 points) |
| est. \( P(-1.645 < Z < 1.645) \) =0.8808617859 (based on 5 points) |
| est. \( P(-1.645 < Z < 1.645) \) =0.8258773355 (based on 3 points) |

Display 9.7 was generated by 5 distinct SAS code submissions, each of which involved reassigning a value of \&npts. It would be nice to loop over a range of \&npts values as part of this coding. SAS macro programming allows for this via the macro
equivalent statements of the `DO-END` data step statements, here `%DO` and `%END`.

9.4 Conditional execution, looping and macros

* `%LET` macro statement above can be used anywhere in a program, in so-called “open code.”

* Other macro statements such as `%DO-%END` and `%IF-%THEN`, can only be used in the context of a defined macro.

* A `macro` is a collection of commands that are evaluated by the macro processor when invoked. The general construction of a macro involves the declaration of the macro along with any variables that might be passed to it, the statements that are to be invoked in the macro, and statement declaring the end of the macro. In particular, a macro program looks like the following

```sas
%macro prog-name;
<text line1 >
<text line2 >
<text line3 >
<... >
%mend prog-name;
```

which is invoked by a reference to `%prog-name` omitting the “;” here. The semi-colon is not needed. The `%prog-name` is talking to the macro processor, not to other SAS components.

Parameters can be passed to macro programs. (Parameter has technical meaning to the statistical community and a different meaning to the programming community. Here, a parameter is simply an argument to a macro whose value may influence what a macro does.)

First macro programming example: we explore complicated macro variable constructions. Consider the example in Display 9.8 in which two
macro variables need to be resolved to define a third macro variable. Here, we have variables that are hypothetical values of weights (weight1, weight2) measured on two separate occasions (week1, week2). In this simple macro, we use two macro parameters (variable, obs) to identify the variable and occasion of interest. Our macro is designed to echo the input to the LOG and to print the value of the variable of interest.

Display 9.8: A simple macro to demonstrate resolution of a more complex macro variable name

```sas
%let var1=week;
%let var2=weight;
%let time1=1;
%let time2=2;
%let var1time1 = week1;
%let var1time2 = week2;
%let var2time1 = weight1;
%let var2time2 = weight2;

data tester;
  input week1 weight1 week2 weight2;
  datalines;
  15 70 25 74;
%
%macro showvalue(variable, obs);
  %put Value of '&variable' = &variable;
  %put Value of '&obs' = &obs;
  %put Value of '&&&variable&obs' = &&&variable&obs;
  proc print;
    var &&&variable&obs;
  run;
%mend showvalue;

%showvalue(variable=var1, obs=time1)
%showvalue(variable=var2, obs=time2)
```

One interesting feature of this program is the construct &&&variable&obs which allows us to make a few additional observations about macro variables. First, macro variable names are resolved from left to right. Second, && is resolved to & by the macro processor. In this example, when &variable=var1(=week) and &obs=time1(=1), &&&variable&obs has value var1time1(=week1). The SAS LOG from the first invocation of this macro is given in Display 9.9.

Display 9.9: Output from LOG associated with invoking the macro showvalue

```sas
%showvalue(variable=var1, obs=time1)
Value of '&variable' = var1
Value of '&obs' = time1
Value of '&&&variable&obs' = week1
```

Macro defined below (trap_area_Z):
* we place a macro wrapper around our trapezoidal rule area estimator code.

* Following parameters are defined for this macro – LOW, HIGH, npts_lo, npts_hi, npts_by, fout, print_est, print_pts, display_graph and ODS_on.

* Macro declaration for \texttt{trap_area\_Z} includes the listing of these parameters along with default values for them.

\begin{verbatim}
%macro trap_area_Z(LOW=-1.645,HIGH=1.645,npts_lo=10,npts_hi=10,npts_by=2, fout=C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est3.out, print_est=FALSE, print_pts=FALSE, display_graph=FALSE, ODS_on=FALSE);
\end{verbatim}

These parameters can be passed by \textbf{position}, by \textbf{keyword} or using a mix of the two strategies. For example, the following invocations of this macro are all equivalent:

\textbf{(position)}
\begin{verbatim}
%trap_area_Z(-1.645, 1.645, 10, 10, 2, C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est3.out, FALSE, FALSE, FALSE, FALSE)
\end{verbatim}

\textbf{(keyword)}
\begin{verbatim}
%trap_area_Z(LOW=-1.645, HIGH=1.645, npts_lo=10, npts_hi=10, npts_by=2, fout=C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est3.out, print_est=FALSE, print_pts=FALSE, display_graph=FALSE, ODS_on=FALSE)
\end{verbatim}

\textbf{(mix of position and keyword)}
\begin{verbatim}
%trap_area_Z(-1.645, 1.645, npts_lo=10, npts_hi=10, 2, fout=C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est3.out, FALSE, FALSE, display_graph=FALSE, ODS_on=FALSE)
\end{verbatim}

\textbf{(default)}
\begin{verbatim}
%trap_area()
\end{verbatim}

In the spirit of writing self-documenting code, the “keyword” passing of parameters to the macro is arguably preferred.
As part of the `trap_area_Z` macro construction, we explicitly address the two questions:

Q₃: Could we set up a “switch” to produce graphics only when requested?

Q₄: Is it possible to package this estimator into a callable “subroutine” or “function?”

We start with the naming of this macro along with input parameters. These input parameters are assigned default values in the macro declaration.

* macro comments were added to describe the input parameters to this macro. These comments begin with a %* and end with a semi-colon. (Aside: Now if you used standard comments, e.g. start with an asterisk and end with a semi-colon, these would be displayed on the LOG. The macro comments won’t be displayed until particular options are set.)

* We have already converted hard-coded values into macro variables in Display 9.5. These were assigned values in Display 9.5 using `%LET statements. In the macro below, control of printing, graph generation and the use of `ODS RTF` are all based on parameters of the macro. The `%IF-%THEN` conditional statements are used to check whether these displays are requested. For example, in the macro defined below, the block of code

```sas
%if &display_graph=TRUE %then %do;
    proc gplot data=trapper2;
    title "Plot of function values vs. x-values";
    plot yout*xout;
    run;
%end;
```

causes the lines

```sas
proc gplot data=trapper2;
    title "Plot of function values vs. x-values";
    plot yout*xout;
    run;
```

to be included in the input stack for processing WHEN the macro parameter `display_graph` = TRUE. In other words, the `%IF-%THEN` condition is evaluated by the macro processor. If TRUE, then the code between the `%DO` and `%END statements are processed as part of the input stack. Finally, the macro has the option to evaluate the integral based on a range of values. This is accomplished via the `%do-%to-%by command paired with a `%end statement, here

```sas
%do npts = &npts_lo %to &npts_hi %by &npts_by; *loop over npts values;
```
Putting these ideas together yields the macro that is given in Display 9.10.

Display 9.10: Macro for constructing a trapezoidal rule estimate of the area under a standard normal curve

```sas
%macro trap_area_Z(LOW=-1.645, HIGH=1.645, npts_lo=10, npts_hi=10, npts_by=2, fout=C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est3.out, print_est=FALSE, print_pts=FALSE, display_graph=FALSE, ODS_on=FALSE);
  %*  ========================================================================;
  %* Purpose: estimate P{LOW < Z < HIGH) using the trapezoidal rule;
  %* Macro variables:         ;
  %*  LOW, HIGH: interval of interest;
  %*  NPTS_LO, NPTS_HI, NPTS_BY: # function values evaluated in area calc.;
  %*  FOUT: output data file containing area estimate for each NPTS value;
  %*  PRINT_EST: display PROC PRINT with area estimate;
  %*  PRINT_PTS: display PROC PRINT with
  %*  DISPLAY_GRAPH: generate PROC GPLOT with function values;
  %*  ODS_ON: generate ODS RTF output;
  %*  ========================================================================;

  %do npts = &npts_lo %to &npts_hi %by &npts_by; *loop over npts values;
    data trapper;
      file  "&fout" MOD;
      retain trapsum 0;
      array x_value(&npts) x1-x&npts;
      array f_value(&npts) y1-y&npts;

      low = &LOW;
      high = &HIGH;
      incr = (high-low)/( &npts -1);
      pi = arcos(-1);

      do i= 1 to &npts;
        x_value[i] = low + incr*(i-1);
        f_value[i] = (1/sqrt(2*pi))*exp(-x_value[i]*x_value[i]/2);
        if i=1 or i=&npts then trapsum = trapsum + f_value[i]/2;
        else trapsum = trapsum + f_value[i];
      end;
      area_est = trapsum*incr;
      output;

      put "est. P(&LOW < Z < &HIGH) =" area_est "(based on &NPTS points);"
    %if &ODS_ON=TRUE %then ods rtf bodytitle;
    %if &print_est=TRUE %then %do;
      proc print data=trapper;
        title "Trapezoidal Rule area estimate for P(&LOW < Z < &HIGH)"
        title2 "(based on &NPTS equally spaced points);"
        var low high incr area_est;
      run;
    %end;
  %end;
%
```

14
data trapper2;
  set trapper;
  array x_value(&npts) x1-x&npts;
  array f_value(&npts) y1-y&npts;
  do ii=1 to &npts;
    xout = x_value[ii];
    yout = f_value[ii];
    output;
  end;

%if &print_pts=TRUE %then %do;
  proc print data=trapper2;
  title "Interpolation points for Trapezoidal Rule";
    var ii low high incr area_est xout yout;
  run;
%end;

%if &display_graph=TRUE %then %do;
  proc gplot data=trapper2;
  title "Plot of function values vs. x-values";
    plot yout*xout;
  run;
%end;

%if &ODS_ON=TRUE %then ods rtf close;
%end;   * of loop over npts values;
%mend trap_area_Z;

One last control command that may be of interest causes the macro processor to move to a different location during its execution, namely the %GOTO label; command. A simple example when %GOTO is especially handy is when you do error checking. In the code given in Display 9.11 with associated output presented in Display 9.12, an error message is generated if an invalid argument is passed as a parameter. The SYSEVALF function calculates the greatest integer < &npts which is then compared to 1. If so, error messages are produced and a branch to the end of the macro is executed. Note that the macro variable &npts is enclosed in single quotes; this keeps the macro processor from evaluating this variable. If this macro variable is enclosed in double quotes, then the macro processor will substitute the value of the macro variable in this expression.

**Display 9.11: Macro illustrating %GOTO**

```%macro ncheck(npts);
  /* npts = needs to be a positive variable; */
  %if %sysevalf(&npts, floor)<1 %then %do;
    %put ERROR: '&npts' must exceed 1;
    %put ERROR: value of '&npts' = &npts;
    %goto badend;
  %end;
  %put Value of '&npts' = &npts;
  %badend: %return;
%mend ncheck;

%ncheck(1)
%ncheck(-2)
```
The results from applying the range checking macro is given in Display 9.12.

Display 9.12: LOG displaying %ncheck macro applied to 3 different arguments

%ncheck(1)
Value of '&npts' = 1
%ncheck(-2)
ERROR: '&npts' must exceed 1
ERROR: value of '&npts' = -2
%ncheck(0.5)
ERROR: '&npts' must exceed 1
ERROR: value of '&npts' = 0.5

We started with 5 questions. We have not yet addressed the 5th question: Q5: Could we generalize this code for any arbitrary (corrected coded) function? We will not work through it here; however, here is a hint to try at home. Where in the code is the standard normal density function defined? Could you define a macro variable that would contain this arbitrary function and pass it as a parameter to some general trap_area macro? Would you need to change more than one line in the macro above?

As a final remark, if you were producing this as a macro that would be used in a production environment, then you would need to add checks for valid parameter values. How would you do this in the trap_area_Z macro?

Now, I could claim that this code matches my first attempt at constructing this macro; however, that would be a lie. Alas, the truth is that debugging and error correction was required. The next section touches on some basic strategies for exploring and debugging errors in macro coding.

9.5. Debugging macro coding and programming
The first place to start exploring non-working macro code is to write out the contents or values of macro variables.

* Can write out to the LOG the value of a macro variable that you have defined using the statement `%PUT &macro-var-name`.

* Can display the SAS automatic macro variables (%PUT _automatic_;), the user-defined macros (%PUT _user_;) or both types (%PUT _all_;). For debugging, either specifying particular macro variables or all user-defined macro variables would be more useful.

* As an aside, it is interesting to see what macro variables are defined by SAS (you might be interested in using some of these variables in your programs). The `SYSDATE`, `SYSDAY`, `SYSTIME` are useful for extracting date and time information while `SYSLAST` is useful for defining a default data set use in a macro.

Display 9.13 includes a request to display the values of macro variables that are automatically defined in SAS.

```
656 %put _automatic_;
       AUTOMATIC SYSDATE 06JUL08
       AUTOMATIC SYSDATE9 06JUL2008
       AUTOMATIC SYSDAY Sunday
       AUTOMATIC SYSDSN WORK   TRAPPER2
       AUTOMATIC SYSERR 0
       AUTOMATIC SYSLAST WORK.TRAPPER2
       AUTOMATIC SYSPROCNAME GPLOT
       AUTOMATIC SYSTIME 09:25
       AUTOMATIC SYSVER 9.2
```

There are three main “options” that are useful for constructing and debugging macros, namely, `macrogen` `mprint` `mlogic`.
*symbolgen* shows values of macro variables and can be used to trace the resolution of more complex macro variable names.

*mprint* displays the SAS statements that are generated for execution by the macro processor.

*mlogic* may be the most useful for debugging since it displays a trace of the macro processors execution of a macro program.

The next 3 displays illustrate the LOG output produced by each of the options above.

Display 9.14 presents a portion of the SAS LOG output when the *symbolgen* option is set prior to the invocation of the *trap_area_Z* macro. This option notes the resolution of each reference to a macro variable in the code that follows. The first reference to macro variables in the *trap_area_Z* macro is the *%do* loop:

```sas
%do npts = &npts_lo %to &npts_hi %by &npts_by; *loop over npts values;
```

in which 3 macro variables are defined, namely, *npts_lo*, *npts_hi* and *npts_by*. These are resolved and noted in the first lines starting with *SYMBOLGEN*, here,

| SYMBOLGEN: Macro variable NPTS_LO resolves to 5 |
| SYMBOLGEN: Macro variable NPTS_HI resolves to 25 |
| SYMBOLGEN: Macro variable NPTS_BY resolves to 5 |

Display 9.14: LOG output produced by the *symbolgen* option when using with the macro invocation *

```
options symbolgen nomprint nomlogic;
%trap_area_Z(LOW=0,HIGH=1.96,npts_lo=5,npts_hi=25,npts_by=5, display_graph=TRUE)
```

| SYMBOLGEN: Macro variable NPTS_LO resolves to 5 |
| SYMBOLGEN: Macro variable NPTS_HI resolves to 25 |
| SYMBOLGEN: Macro variable NPTS_BY resolves to 5 |

NOTE: There were 25 observations read from the data set WORK.TRAPPER2.
NOTE: PROCEDURE GPLOT used (Total process time):
  real time 9:21.44
  cpu time 1.49 seconds

| SYMBOLGEN: Macro variable FOUT resolves to C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est3.out |
| SYMBOLGEN: Macro variable NPTS resolves to 5 |
| SYMBOLGEN: Macro variable NPTS resolves to 5 |
| SYMBOLGEN: Macro variable NPTS resolves to 5 |
| SYMBOLGEN: Macro variable NPTS resolves to 5 |
| SYMBOLGEN: Macro variable LOW resolves to 0 |
| SYMBOLGEN: Macro variable HIGH resolves to 1.96 |
Display 9.15 presents a portion of the SAS LOG output when the \texttt{mprint} option is set prior to the invocation of the \texttt{trap_area_Z} macro. This option writes the SAS code resulting from the execution of a macro. After the looping over \texttt{npts} values, the first lines of this macro include

\begin{verbatim}
data trapper;
  file "&fout" MOD;
  retain trapsum 0;
  array x_value(&npts) x1-x&npts;
  array f_value(&npts) y1-y&npts;
  low = &LOW;
\end{verbatim}
high = &HIGH;
incr = (high-low)/( &npts -1);
pi = arcos(-1);

do i= 1 to &npts;
   x_value[i] = low + incr*(i-1);
   f_value[i] = (1/sqrt(2*pi))*exp(-x_value[i]*x_value[i]/2);
   if i=1 or i=&npts then trapsum = trapsum + f_value[i]/2;
   else trapsum = trapsum + f_value[i];
end;

which results in the SAS LOG output

    MPRINT(TRAP_AREA_Z):   data trapper;
    MPRINT(TRAP_AREA_Z):   file "C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est3.out" MOD;
    MPRINT(TRAP_AREA_Z):   retain trapsum 0;
    MPRINT(TRAP_AREA_Z):   array x_value(5) x1-x5;
    MPRINT(TRAP_AREA_Z):   array f_value(5) y1-y5;
    MPRINT(TRAP_AREA_Z):   low = 0;
    MPRINT(TRAP_AREA_Z):   high = 1.96;
    MPRINT(TRAP_AREA_Z):   incr = (high-low)/( 5 -1);
    MPRINT(TRAP_AREA_Z):   pi = arcos(-1);
    MPRINT(TRAP_AREA_Z):   do i= 1 to 5;
    MPRINT(TRAP_AREA_Z):      x_value[i] = low + incr*(i-1);
    MPRINT(TRAP_AREA_Z):      f_value[i] = (1/sqrt(2*pi))*exp(-x_value[i]*x_value[i]/2);
    MPRINT(TRAP_AREA_Z):      if i=1 or i=5 then trapsum = trapsum + f_value[i]/2;
    MPRINT(TRAP_AREA_Z):      else trapsum = trapsum + f_value[i];
    MPRINT(TRAP_AREA_Z):   end;
    MPRINT(TRAP_AREA_Z):   area_est = trapsum*incr;
    MPRINT(TRAP_AREA_Z):   output;

The values of the macro variables are substituted here (e.g. 5 for &npts, 0 for &low, 1.96 for &high, etc.), and this is the SAS code that is now processed and executed. While it doesn’t have the nice indenting for ease of reading, it does show the code that will be executed.

Display 9.15: LOG output produced by the mprint option when using with the macro
invocation %trap_area_Z(LOW=0,HIGH=1.96,npts_lo=5,npts_hi=25,npts_by=5,
display_graph=TRUE)
**Note:** The file "C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est3.out" is:
Filename=C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est3.out,
RECFM=V,LRECL=256, File Size (bytes)=1041,
Last Modified=06Jul2008:10:49:20,
Create Time=02Jul2008:15:41:16

**Note:** The data set WORK.TRAPPER has 1 observations and 17 variables.

**Note:** There were 1 observations read from the data set WORK.TRAPPER.
**Note:** The data set WORK.TRAPPER2 has 5 observations and 20 variables.

**Note:** There were 5 observations read from the data set WORK.TRAPPER2.

**Note:** PROCEDURE GPLOT used (Total process time):
real time 0.40 seconds
cpu time 0.39 seconds
MPRINT(TRAP_AREA_Z): if i=1 or i=10 then trapsum = trapsum + f_value[i]/2;
MPRINT(TRAP_AREA_Z): else trapsum = trapsum + f_value[i];
MPRINT(TRAP_AREA_Z): end;
MPRINT(TRAP_AREA_Z): area_est = trapsum*incr;
MPRINT(TRAP_AREA_Z): output;
MPRINT(TRAP_AREA_Z): put "est. P(0 < Z < 1.96) =" area_est "(based on 10 points);

NOTE: The file "C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est3.out" is:
Filename=C:\Users\baileraj\BAILERAJ\BOOK-stat-prog-may05\ch-09\est3.out,
RECFM=V,LRECL=256,File Size (bytes)=1097,
Last Modified=06Jul2008:10:51:11,
Create Time=02Jul2008:15:41:16

Display 9.16 presents a portion of the SAS LOG output when the mlogic option is set prior to the invocation of the trap_area_Z macro. This option provides more detail on the execution of looping and conditional statements in the context of the execution of a macro. There are a number of optional outputs built into this macro such as requesting ODS RTF or a print out of the estimation data set as we see in the code section that follows

%if &ODS_ON=TRUE %then ods rtf bodytitle;
%if &print_est=TRUE %then %do;
    proc print data=trapper;
Since this macro was called with %trap_area_Z(LOW=0, HIGH=1.96, npts_lo=5, npts_hi=25, npts_by=5, display_graph=TRUE), i.e. ODS ON and print_est are set at their default FALSE values. We see that MLOGIC confirms this
MLOGIC(TRAP_AREA_Z): IF condition &ODS_ON=TRUE is FALSE
MLOGIC(TRAP_AREA_Z): IF condition &print_est=TRUE is FALSE

Display 9.16: LOG output produced by the mlogic option when using with the macro invocation %trap_area_Z(LOW=0,HIGH=1.96,npts_lo=5,npts_hi=25,npts_by=5,
display_graph=TRUE)

NOTE: There were 25 observations read from the data set WORK.TRAPPER2.
NOTE: PROCEDURE GPLOT used (Total process time):
   real time 1:27.23
   cpu time 4.05 seconds
Each of these options is useful although you may not want to have all options simultaneously “switched on” when debugging a macro; setting all three options leads to a difficult to read SAS LOG.

9.6. Saving macros - %include+autocall+stored compiled macros [OMIT]

The examples above all used macros shortly after they are defined. There are often circumstances when you want to save macros for future use. The SAS documentation for “Introduction to storing and reusing macros” provides a solid foundation for learning more about this. Three general strategies for using previously defined macros include: 1) insert the macro code from some external file; 2) use the autocall facility in SAS; and 3) use a stored compiled macro. The %include command inserts the contents of a particular file in the location where the %include command is issued. The autocall strategy collects a set of macros as external text (.sas) files in a particular directory. The macros called via this strategy must be compiled (and stored in WORK.SASMCR) when first called in a session. The option mautosource must be set to use this strategy. The stored compiled macro option saves compiled macro programs in a SAS catalog. The options mstored and sasmstore need to be used here. This may be most preferred if you have macros that are of production quality and are not frequently revised and updated.
9.7. Functions/routines of potential interest to macro programmers - %index, %length, %eval, symput, symget

We used a couple of macro functions in earlier code, e.g. SYSEVALF to calculate the integer “floor” of an argument. This is a general evaluation function. The EVAL function is available for integer arithmetic. There are a host of other functions available for macro programming: %index finds the position of a string in a source string; %length returns the length of an argument (also useful for checking for valid arguments with macro programming); %scan displays a particular word in an argument, %substr extracts a substring from an argument; and %upcase can convert an argument to all upper case characters (useful to apply this before checking an argument since you might want “YES”, “Yes”, “yes” all to lead to the same action in a macro and a conversion to “YES” would guarantee this). Display 9.17 provides a short example to illustrate some of these functions.

Display 9.17: Illustrating built-in macro functions along with the LOG display of results

```
%let summer = June July August;
%let pickmth = 3;
%let mymonth = %scan(&summer, &pickmth); * pickmth word of summer;
%let mymonth3 = %substr(&summer, 11, 3); * start @ position 11 and move 3;
%let upper_month3 = %upcase(&mymonth3);

%put Summer=&summer;
%put Length of '&summer' = %length(&summer);
%put Where is Aug in the '&summer'? = %index(&summer, Aug);
%put Month picked = &pickmth;
%put Which month? = &mymonth;
%put Which month (3 letters)? = &mymonth3;
%put Upper case (3 letters)? = &upper_month3;
```

From SAS LOG . . .
Summer=June July August
Length of '"summer' = 16
Where is Aug in the '"summer'? = 11
Month picked = 3
Which month? = August
Which month (3 letters)? = Aug
Upper case (3 letters)? = AUG

Two other functions / routines, SYMPUT and SYMGET, might be of particular interest to a programmer. These can be used to assign a SAS variable value to a macro variable (SYMPUT) or to assign a macro variable value to a SAS variable (call SYMGET), i.e.

```
SYMGET:  Macro variable value → SAS variable
SYMPUT:  SAS variable value → Macro variable
```
Display 9.18 provide a program in which macro variables &mbrood1, &mbrood2, &mbrood3, and &total are constructed from SAS variables “brood” and an index variable “i.” A %DO loop over these variables is then used to generate separate plots of the number of young vs. concentration for each of the broods.

Display 9.18: MACRO variable SYMPUT manipulations highlighted

```sas
data nitrofen;
  infile '\Muserver2\USERS\B\BAILERAJ\public.www\classes\sta402\data\ch2.dat.txt' firstobs=16 expandtabs missover pad ;
  input @9 animal 2.
    @17 conc 3.
    @25 brood1 2.
    @33 brood2 2.
    @41 brood3 2.
    @49 total 2.;

data test; set nitrofen;
  brood=1; conc=conc; nyoung=brood1; output;
  brood=2; conc=conc; nyoung=brood2; output;
  brood=3; conc=conc; nyoung=brood3; output;

%macro threeregs;
  proc sort data=test out=test;
    by brood;
  run;

  data _null_;
    set test;
    by brood;
    if first.brood then do;
      i+1;
      ii = left(put(i,2.));
      call symput('mbrood'||ii,trim(left(brood)));
      call symput('total',ii);
    end;
  run;
%do ibrood = 1 %to &total;
  proc gplot data=test;
    where brood=&mbrood&ibrood;
    plot nyoung*conc;
    symbol1 interpol=rq
    value=diamond;
%end;
```
Display 9.19: Plots produced by the %threeregs macro (edited)

### 9.8 Bonus material: processing multiple data sets

A common challenge in statistical programming is processing a large number of data sets. For example, you might have data from individual subjects that is stored in separate files such as spreadsheets. Another example is separate data files for different dates. While this usually involves processing a number of external files, we close with a smaller example that we hope captures the essence of this problem.

The general problem addressed here is the merging of a set of data files that contain time-temperature data where a different data set is available for each date. The construction of three time-temperature data sets is presented in Display 9.20. These hypothetical data allow us to use the automatic macro variable &SYSDSN to add the name of the data set as a variable. We name the datasets aug03, aug05 and aug17 assuming that they are three observation dates in
August. The temperature variable is assigned a value that increases until a late afternoon maximum and decreases from that point onward. The macro variable &SYSDSN has two elements, WORK AUGxx, and we select the AUGxx piece using the %scan macro function. This piece is assigned to a user-defined macro variable &myd whose value is assigned to a SAS variable mydate via the symget function.

Display 9.20: Constructing 3 example data sets for later use

```sas
/*
   construct 3 example data set with time temperature data
*/
data aug03;
  do time=1 to 24;
    temp = 74 - abs(time-16) + 0.5*rannor(0);
    output;
  end;
* extract data set name for use as a variable in each dataset;
data aug03; set aug03;
  %let myd = %scan(&sysdsn,2);
  mydate= symget('myd');
data aug05;
  do time=1 to 24;
    temp = 78 - abs(time-16) + 0.5*rannor(0);
    output;
  end;
* extract data set name for use as a variable in each dataset;
data aug05; set aug05;
  %let myd = %scan(&sysdsn,2);
  mydate= symget('myd');
data aug17;
  do time=1 to 24;
    temp = 90 - abs(time-16) + 0.5*rannor(0);
    output;
  end;
* extract data set name for use as a variable in each dataset;
data aug17; set aug17;
  %let myd = %scan(&sysdsn,2);
  mydate= symget('myd');

/* DEBUGGING BLOCK TO CHECK DATA SET CONSTRUCTION
data tester; set aug03 aug05 aug17;
proc print data=tester;
  id time; run;
proc gplot data=tester;
  plot temp*time=mydate;
  symbol interpol=join;
run;
*/
```

Assuming that we want to read an arbitrary number of arbitrarily named data set requires some care. In Display 9.21, we read the names of the data sets we ultimately want to combine to form a single data set, and assign these dataset names as the values of macro variables &dsn1, &dsn2, etc. using symput in a manner similar to the brood constructions in the last example. A count of the number of datasets processed is also stored as a macro variable.
Display 9.21: Reading data set names and constructing corresponding macro variables

```sas
/*
  * create macro variable names corresponding to each data set
*/
data _null_;  
* read data sets and create macro variable with name of each;
retain counter 0;
input times $ @@;
counter = counter + 1;
put times counter;
* create a macro variable with each dataset name;
* create macro variable name with total number of DSNs;
call symput('dsn'||trim(left(counter)), times);
call symput('num_data_sets', counter);
datalines;
aug03 aug05 aug17;
%put _user_;
```

The values of user-defined macro variables that exist after executing Display 9.21 code is requested by the `%put _user_;` and these variables and associated values are:

- **GLOBAL NUM_DATA_SETS**: 3
- **GLOBAL DSN1**: aug03
- **GLOBAL DSN2**: aug05
- **GLOBAL DSN3**: aug17

Note that these macro variables are all global since they were defined in open code, i.e. not defined internal in a macro.

Display 9.22 contains the macro program which builds a data step that concatenates the data sets whose names are values of the macro variables `&dsn1`, `&dsn2`, ..., `&dsn&num_data_sets` from executing the code in Display 9.21. This display invokes this macro program naming the combined data set, `all3`, prints this combined data set and generates a plot of the data.

Display 9.22: Defining and running macro program to combine data sets

```sas
/*
  * construct macro to concatenate the data sets
*/
%macro concatenator(combine);
data &combine;
  set 
    %do ii = 1 %to &num_data_sets;
      &dsn&ii
    %end;
%mend concatenator;

/*
  * concatenate the three data sets and print results
*/
options mprint mlogic;
%concatenator(combine=all3)
ods rtf bodytitle;
```
proc print data=all3;
run;

proc gplot data=all3;
plot temp*time=mydate /
  haxis=1 to 24 by 1
  vaxis=50 to 100 by 5;
%macro syms();
  %do idata=1 %to &num_data_sets;
  symbol&idata interpol=join line=&idata ci=black;
%end;
%mend syms;
%syms();
run;
ods rtf close;

Setting the mlogic and mprint option provides a nice trace of the execution of this macro from which we see that code

data all3;
  set aug03 aug05 aug17 ;

is the result of %concatenator(all3). Notice that another small macro %syms is defined to build the symbol statement to accompany GLOT. The results of macro options set before the macro invocation are given below

MLOGIC(CONCATENATOR): Beginning execution.
MLOGIC(CONCATENATOR): Parameter COMBINE has value all3
MPRINT(CONCATENATOR): data all3;
MLOGIC(CONCATENATOR): %DO loop beginning; index variable II; start value is 1; stop value is 3; by value is 1.
MLOGIC(CONCATENATOR): %DO loop index variable II is now 2; loop will iterate again.
MLOGIC(CONCATENATOR): %DO loop index variable II is now 3; loop will iterate again.
MLOGIC(CONCATENATOR): %DO loop index variable II is now 4; loop will not iterate again.
MPRINT(CONCATENATOR): set aug03 aug05 aug17 ;
MLOGIC(CONCATENATOR): Ending execution.

The print out of the combined data set is presented in Display 9.23.

Display 9.23: Output containing (edited) print of the combined dataset

<table>
<thead>
<tr>
<th>Obs</th>
<th>Time</th>
<th>temp</th>
<th>mydate</th>
<th>Obs</th>
<th>time</th>
<th>temp</th>
<th>mydate</th>
<th>Obs</th>
<th>time</th>
<th>temp</th>
<th>mydate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>58.3697</td>
<td>AUG03</td>
<td>25</td>
<td>1</td>
<td>62.7558</td>
<td>AUG05</td>
<td>49</td>
<td>1</td>
<td>75.3026</td>
<td>AUG17</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>59.8335</td>
<td>AUG03</td>
<td>26</td>
<td>2</td>
<td>63.4330</td>
<td>AUG05</td>
<td>50</td>
<td>2</td>
<td>75.7583</td>
<td>AUG17</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>61.3913</td>
<td>AUG03</td>
<td>27</td>
<td>3</td>
<td>65.2758</td>
<td>AUG05</td>
<td>51</td>
<td>3</td>
<td>76.9702</td>
<td>AUG17</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>61.5808</td>
<td>AUG03</td>
<td>28</td>
<td>4</td>
<td>64.8585</td>
<td>AUG05</td>
<td>52</td>
<td>4</td>
<td>78.1073</td>
<td>AUG17</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>62.3649</td>
<td>AUG03</td>
<td>29</td>
<td>5</td>
<td>67.5989</td>
<td>AUG05</td>
<td>53</td>
<td>5</td>
<td>78.1268</td>
<td>AUG17</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>63.1941</td>
<td>AUG03</td>
<td>30</td>
<td>6</td>
<td>67.7543</td>
<td>AUG05</td>
<td>54</td>
<td>6</td>
<td>80.2560</td>
<td>AUG17</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>65.1616</td>
<td>AUG03</td>
<td>31</td>
<td>7</td>
<td>68.3969</td>
<td>AUG05</td>
<td>55</td>
<td>7</td>
<td>80.6597</td>
<td>AUG17</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>66.0599</td>
<td>AUG03</td>
<td>32</td>
<td>8</td>
<td>70.7063</td>
<td>AUG05</td>
<td>56</td>
<td>8</td>
<td>81.7086</td>
<td>AUG17</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>66.9734</td>
<td>AUG03</td>
<td>33</td>
<td>9</td>
<td>71.4712</td>
<td>AUG05</td>
<td>57</td>
<td>9</td>
<td>82.2032</td>
<td>AUG17</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>68.2219</td>
<td>AUG03</td>
<td>34</td>
<td>10</td>
<td>71.7806</td>
<td>AUG05</td>
<td>58</td>
<td>10</td>
<td>84.3405</td>
<td>AUG17</td>
</tr>
</tbody>
</table>
Statistical Programming in SAS

<table>
<thead>
<tr>
<th>Obs</th>
<th>Time</th>
<th>temp</th>
<th>mydate</th>
<th>Obs</th>
<th>time</th>
<th>temp</th>
<th>mydate</th>
<th>Obs</th>
<th>time</th>
<th>temp</th>
<th>mydate</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>11</td>
<td>68.3961</td>
<td>AUG03</td>
<td>35</td>
<td>11</td>
<td>72.3482</td>
<td>AUG05</td>
<td>59</td>
<td>11</td>
<td>84.9583</td>
<td>AUG17</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>69.9454</td>
<td>AUG03</td>
<td>36</td>
<td>12</td>
<td>74.7662</td>
<td>AUG05</td>
<td>60</td>
<td>12</td>
<td>85.8839</td>
<td>AUG17</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>70.4325</td>
<td>AUG03</td>
<td>37</td>
<td>13</td>
<td>74.0328</td>
<td>AUG05</td>
<td>61</td>
<td>13</td>
<td>87.4010</td>
<td>AUG17</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>72.3879</td>
<td>AUG03</td>
<td>38</td>
<td>14</td>
<td>76.1863</td>
<td>AUG05</td>
<td>62</td>
<td>14</td>
<td>88.5098</td>
<td>AUG17</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>73.5111</td>
<td>AUG03</td>
<td>39</td>
<td>15</td>
<td>77.0349</td>
<td>AUG05</td>
<td>63</td>
<td>15</td>
<td>88.7737</td>
<td>AUG17</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>73.2513</td>
<td>AUG03</td>
<td>40</td>
<td>16</td>
<td>77.7682</td>
<td>AUG05</td>
<td>64</td>
<td>16</td>
<td>90.1271</td>
<td>AUG17</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>73.2294</td>
<td>AUG03</td>
<td>41</td>
<td>17</td>
<td>76.6733</td>
<td>AUG05</td>
<td>65</td>
<td>17</td>
<td>88.4658</td>
<td>AUG17</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>71.7736</td>
<td>AUG03</td>
<td>42</td>
<td>18</td>
<td>75.4574</td>
<td>AUG05</td>
<td>66</td>
<td>18</td>
<td>87.6286</td>
<td>AUG17</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>70.9356</td>
<td>AUG03</td>
<td>43</td>
<td>19</td>
<td>74.6361</td>
<td>AUG05</td>
<td>67</td>
<td>19</td>
<td>87.2270</td>
<td>AUG17</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>69.5076</td>
<td>AUG03</td>
<td>44</td>
<td>20</td>
<td>73.8843</td>
<td>AUG05</td>
<td>68</td>
<td>20</td>
<td>86.8665</td>
<td>AUG17</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>69.1718</td>
<td>AUG03</td>
<td>45</td>
<td>21</td>
<td>73.8843</td>
<td>AUG05</td>
<td>69</td>
<td>21</td>
<td>85.3271</td>
<td>AUG17</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>68.1008</td>
<td>AUG03</td>
<td>46</td>
<td>22</td>
<td>71.5441</td>
<td>AUG05</td>
<td>70</td>
<td>22</td>
<td>83.2352</td>
<td>AUG17</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>67.2909</td>
<td>AUG03</td>
<td>47</td>
<td>23</td>
<td>70.5921</td>
<td>AUG05</td>
<td>71</td>
<td>23</td>
<td>82.2487</td>
<td>AUG17</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>66.8331</td>
<td>AUG03</td>
<td>48</td>
<td>24</td>
<td>70.7164</td>
<td>AUG05</td>
<td>72</td>
<td>24</td>
<td>81.5246</td>
<td>AUG17</td>
</tr>
</tbody>
</table>

The plot of these temperature-time profiles for the three dates is presented in Display 9.24. A separate profile is displayed separately for the three different dates.

Display 9.24: Output containing the plot the combined dataset

Summary

Repeated use of code is facilitated by purposeful generalizations of code. Macro programming in SAS is a natural way to think about implementing such generalizations. We
started this chapter with a trapezoidal rule code example that was first generalized by replacing hard-coded values by macro variables. This code was then nested in a macro to allow for more flexible invocations of this program. The use of a parameters being passed to the macro along with looping and conditional execution of code as a result of these parameters was also discussed at this point. Macro functions were introduced and illustrated with a final example illustrating the automatic processing of a large number of input data sets providing a showcase for a number of these functions.

References:

