Using the R interface in SAS ® to Call R Functions and Transfer Data

Bruce Gilsen

Federal Reserve Board

May 2021

Using the R interface in SAS ® to Call R Functions and Transfer Data

Bruce Gilsen has worked at the Federal Reserve Board for 37 ½ years. He spends most of his time as an in-house SAS consultant.

His Bruce Force fantasy baseball team will battle for its 6th league title in 32 years in 2021.

Summary of topics (1)

- Introduction
- Potential users
- Possible uses
- System Requirements
- Configure SAS Interface to R On Windows, Linux
- Syntax (informal)
- Syntax (informal) for non-IML users
- Current working directory

Summary of topics (2)

- Persistence in R environment
- Save entire workspace in SAS interface to R and restore it in R: caution
- Windows file path limitation
- Data transfer details: missing values, date/datetime/time values, variable names

Summary of topics (3)

- Examples
  - Call R function, no data transfer
  - Copy SAS data set to R data frame
  - Copy SAS data set to R data frame, run regression, return results to SAS
  - Copy IML matrix to R, call R package and R graphics
  - Use a source file
  - Save or read one or multiple R objects or an entire workspace to/from a file
- Conclusion

Introduction

- Linux and Windows SAS, starting in SAS 9.3
- Call R functions from PROC IML, return results
- Transfer data between R and SAS with SAS subroutines, between
  - SAS data set and R data frame
  - SAS/IML matrix and R matrix
- Copy R data into R interface with R functions like readRDS(), load()
- Save R data from R interface to files with R functions like saveRDS(), save(), save.image()
- Call R from SAS, not the reverse

Potential users

- PROC IML users
- Non-PROC IML users: can use PROC IML just as a wrapper to
  - Transfer data between SAS and R
  - Call R functions, return results

- Focus of presentation: non-PROC IML users who want to take advantage of R interface

Possible uses

- Use R data as input to SAS program
- Export SAS results to R for further processing
- Use new econometric or statistical techniques
  - R easily extendible by users
  - New techniques sometimes in R shortly after publication in econometric journal or elsewhere
  - SAS requires lengthy process (development, testing, etc.) to implement new techniques
  - Quickly, easily use new technique in SAS by calling R instead of waiting for SAS implementation

System Requirements

- SAS 9.3 or later on Linux or Windows
- SAS/IML software. To check, submit SAS statement:

```
proc setinit;run;
```

- R

Configure SAS Interface To R On Windows

- RLANG system option
  - Determines if can call R from SAS
  - Must be set to RLANG, not NORLANG
  - SAS statement to see if RLANG enabled:

    ```
    proc options option=rlang;run;
    ```

  - Can only set at invocation, not during SAS session
  - Add to SAS configuration file if not already there (might require administrator privileges):

    ```
    -rlang
    ```

  - SAS statement to locate SAS configuration file:

    ```
    proc options option=config;run;
    ```

Configure SAS Interface To R On Linux

- RLANG system option
  - Determines if can call R from SAS
  - Must be set to RLANG, not NORLANG
  - SAS statement to see if RLANG enabled:

    ```
    proc options option=rlang;run;
    ```

  - Can only set at invocation, not during SAS session
  - Add to SAS configuration file if not already there:

    ```
    -rlang
    ```

  - SAS statement to locate SAS configuration files:

    ```
    proc options option=config;run;
    ```

Configure SAS Interface To R On Linux

- RLANG system option
  - Typically have system + personal configuration files
  - E.g., SAS 9.4TS1M4 at Fed Reserve, 3 system files, 1 personal file:

```
CONFIG=(/sftwr/sas/SASHome/SASFoundation/9.4/sasv9.cfg
/sftwr/sas/SASHome/SASFoundation/9.4/nls/en/sasv9.cfg
/sftwr/sas/SASHome/SASFoundation/9.4/sasv9_local.cfg
/it/support/it/m1bfg00/sasv9.cfg )
```

  - Recommend: have network admin add –rlang in /sftwr/sas/SASHome/SASFoundation/9.4/sasv9_local.cfg
  - Alternative if can't add –rlang to a configuration file: invocation option, e.g.,

    SAS Windowing environment: sas –rlang &

    Batch mode: sas –rlang myprog1.sas

Configure SAS Interface To R On Linux

- R_HOME Environment Variable
  - Must be set to R home directory
  - Probably = value of R_HOME_DIR environment variable in script that invokes R
  - To locate R script, from Linux prompt: *which R*
  - SAS statements to check if R_HOME set in SAS:

```
data _null_;  var1 = sysget("R_HOME");
put var1=;    run;
```

  - Recommend: have network admin add to file bin/sasenv_local in SASROOT directory (primary directory w/SAS system software):

```
export R_HOME=name-of-R-home-directory
```

  - SAS statement to locate SASROOT directory:

```
proc options option=config;run;
```

Configure SAS Interface To R On Windows or Linux

- Could vary at your site
- Contact local SAS support staff or SAS Technical Support for assistance if needed

Syntax (informal)

```
proc iml;
  (IML statements)

  call ExportDataSetToR("sasdataset", "R-frame" );
  call ExportMatrixToR(IML-matrix, "R-matrix");

  submit / R;        /* Start submitting statements to R */

    (R statements)

  endsubmit;         /* Stop submitting statements to R */

  call ImportDataSetFromR("sasdataset", "R-frame");
  call ImportMatrixFromR(IML-matrix, "R-matrix");

 (IML statements and submit blocks as necessary)
quit; /* end IML */
```

Syntax (informal) for non-IML users: just use PROC IML as wrapper

```
proc iml;
  (IML statements)

  call ExportDataSetToR("sasdataset", "R-frame" );
  call ExportMatrixToR(IML-matrix, "R-matrix");

  submit / R;        /* Start submitting statements to R */

    (R statements)

  endsubmit;         /* Stop submitting statements to R */

  call ImportDataSetFromR("sasdataset", "R-frame");
  call ImportMatrixFromR(IML-matrix, "R-matrix");

 (IML statements)
quit; /* end IML */
```

Syntax (informal) for non-IML users: just use PROC IML as wrapper

```
proc iml;

  call ExportDataSetToR("sasdataset", "R-frame" );


  submit / R;        /* Start submitting statements to R */

    (R statements)

  endsubmit;         /* Stop submitting statements to R */


 call ImportDataSetFromR("sasdataset", "R-frame");


quit; /* end IML */
```

Current working directory in R environment

- Initially, is location of temporary SAS WORK library
  - SAS code to determine location of WORK:

```
proc options option=work;run;
```

  - WORK library deleted at end of SAS session, not good place to store R data
- Modify current working directory in submit block w/R function setwd():

```
setwd("/u:/m1xxx00/sasandr")
```

- Persists across submit blocks until SAS/IML session ends

Persistence in R environment

- R environment created in submit block(s) persists until SAS/IML session ends
  - Data copied to or created in R in a submit block persists across submit blocks until SAS/IML session ends
  - Plots created in R display until close plot window or end SAS/IML session
  - Current working directory persists across submit blocks until SAS/IML session ends
- To preserve R data for after SAS/IML session ends
  - SAS ImportDataSetFromR, ImportMatrixFromR subroutines copy R data to SAS data set or IML matrix (outside submit block)
  - Save one or multiple R objects or entire workspace to a file with R functions saveRDS(), save(), or save.image() (in submit block)

Save entire workspace in SAS interface to R and restore it in R: caution

- Save entire workspace to a file in SAS interface with save.image() function, then restore it in R with load() function
- SAS interface defines R function .Last() to prevent accidental R exit inside SAS/IML
- .Last() saved and restored, when try to end R session with quit(), quit("no"), or quit("yes"), get error:
  - Error in .Last() : You cannot exit R when it is running inside SAS

Save entire workspace in SAS interface to R and restore it in R: caution

- Three ways to prevent error
  - In SAS interface, before save.image(), remove .Last() definition:

    ```
    rm(.Last)
    ```

  - In R, after using load() function to restore workspace, remove .Last() definition:

    ```
    rm(.Last)
    ```

  - When terminate R session, set parameter "runLast" of quit() function to FALSE:

    ```
    quit("no", runLast = FALSE)
    ```

Windows file path limitation

- Backslashes (\) in file path generate error, use forward slashes (/) or double backslashes (\\) instead
  - Generates error:

    ```
    pdf("u:\m1xxx00\sasandr\hist1.pdf")
    ```

  - Use one of these instead:

    ```
    pdf("u:/m1xxx00/sasandr/hist1.pdf")
    pdf("u:\\m1xxx00\\sasandr\\hist1.pdf")
    ```

Data transfer details

- Missing values
- Date/datetime/time values
- Variable names

Data transfer details: Missing values

- ExportDataSetToR copies SAS data set to R data frame
  - Converts . and .A–.Z and ._ to R missing value NA
- ImportDataSetFromR copies R data frame to SAS data set
  - Converts R missing value NA to .

Data transfer details: Date/datetime/time values

- SAS date, time, datetime values: numeric variables that represent data
  - SAS date values: # of days before or after January 1, 1960
  - SAS time values: # of seconds since midnight on a given day
  - SAS datetime values: # seconds before or after January 1, 1960
- Typically have date, time, or datetime format for meaningful display
  - e.g., PUT DATE1;, DATE1 is 5/8/2018 SAS date

DATE1 format	Text displayed
none	21312
DATE9.	08MAY2018
YYMMDDN8.	20180508

Data transfer details: Date/datetime/time values

- In R, classes represent dates and datetimes
- When copy SAS data set to R
  - Variables w/SAS date, time, datetime values need appropriate format to ensure proper transfer
  - Format determines class of variable in R

Format family in SAS	Class in R
Date	Date
Time and Datetime	POSIXct and pseudo-class POSIXt

  - SAS time values converted to datetime values w/date component January 1, 1960

Data transfer details: Date/datetime/time values

- When copy R data frame to SAS, variable format based on class in R

Class in R	Format in SAS
Date	DATE9.
POSIXt	DATETIME19.
All other cases	no format is assigned

Data transfer details: Variable names

- Valid R variable name invalid in SAS if
  - contains a period (.)
  - longer than 32 characters
- ImportDataSetFromR converts invalid R variable names to valid SAS variable names
  - Converts periods to underscores
  - Truncates long names to 32 characters
  - Converts duplicate variable names generated by above steps to unique names by
    - **appending number**
    - **if necessary to limit length to 32 characters, truncate name before appending**

Example 1: Call R function, no data transfer

- Simple example, no data transfer
- Mean in SAS, then R, compare

Example 1: Call R function, no data transfer

```
  /* Create data, take
     mean in SAS */
data one;
  input x;
  datalines;
2
4
6
9
11
100
;run;
proc means data=one mean;
  var x;
run;
```

```
  /* Create data, take
     mean in R */
proc iml;
  submit / R;
    x<-c(2,4,6,9,11,100)
    meanx<-mean(x)
    print(meanx)
  endsubmit;
quit; /* end IML */
```

Example 1: Output

```
The MEANS Procedure

Analysis Variable : x

            Mean
------------
  22.0000000
------------
```

← SAS PROC MEANS output

```
[1] 22
```

← R output

Example 2: Copy SAS data set to R data frame

- Create SAS data set
- Copy data set to R with ExportDataSetToR subroutine
  - Standard SAS missing value (.) converted to R missing value NA
  - DATE1=SAS date, class of DATE in R
- Add column to data frame, isn't valid SAS variable name
- Display R data frame info
  - Variable names and values
  - Column value
  - Class of date value copied from SAS
  - Data frame structure
  - Data summary

Example 2: Copy SAS data set to R data frame

- Copy R data frame to SAS data set with ImportDataSetFromR subroutine, display values
  - Convert R missing value NA to standard SAS missing value (.)
  - Convert invalid SAS variable name net.income to valid SAS variable name net_income - change period to underscore
  - DATE1 has class of DATE in R, gets DATE9 format in SAS
- Interleaved output in slides

Example 2 code:
Copy SAS data set to R data frame

```
  /* Create data set in SAS */

data two;
  input tax income date1 yymmdd8.;
     format date1 yymmddn8.;
  datalines;
1 2 20180301
3 4 20180401
5 6 20180501
7 8 20180601
. 10 20180701
;run;
```

Example 2:
Copy SAS data set to R data frame

```
proc iml;
        /* Copy SAS data set TWO to R data frame IMLTWO */
  call ExportDataSetToR("two", "imltwo");
  submit / R;
    imltwo$net.income<-imltwo$income-imltwo$tax

    names(imltwo)
    [1] "tax"        "income"     "date1"      "net.income"

    print(imltwo)
      tax income      date1 net.income
    1   1      2 2018-03-01          1
    2   3      4 2018-04-01          1
    3   5      6 2018-05-01          1
    4   7      8 2018-06-01          1
    5  NA     10 2018-07-01         NA
```

Example 2:
Copy SAS data set to R data frame

```
print(imltwo$income)
[1]  2  4  6  8 10

class(imltwo$date1)
[1] "Date"

str(imltwo)
'data.frame': 5 obs. of  4 variables:
   $ tax   : num  1 3 5 7 9
   $ income: num  2 4 6 8 10
   $ date1: Date, format: "2018-03-01" "2018-04-01" ..
   $ net.income: num  1 1 1 1 NA
```

Example 2:
Copy SAS data set to R data frame

```
    summary(imltwo)
tax              income          date1                net.income
Min.   :1.0   Min.   : 2   Min.   :2018-03-01   Min.   :1
1st Qu.:2.5   1st Qu.: 4   1st Qu.:2018-04-01   1st Qu.:1
Median :4.0   Median : 6   Median :2018-05-01   Median :1
Mean   :4.0   Mean   : 6   Mean   :2018-05-01   Mean   :1
3rd Qu.:5.5   3rd Qu.: 8   3rd Qu.:2018-06-01   3rd Qu.:1
Max.   :7.0   Max.   :10   Max.   :2018-07-01   Max.   :1
NA's   :1                                       NA's   :1

  endsubmit;

   /*Copy R data frame IMLTWO to SAS data set FROMIMLTWO*/
  call ImportDataSetFromR("fromimltwo", "imltwo");
quit;  /* end PROC IML */
```

Example 2:
Copy SAS data set to R data frame

```
proc print data=fromimltwo;
run;
```

Obs	tax	income	date1	net_income
1	1	2	01MAR2018	1
2	3	4	01APR2018	1
3	5	6	01MAY2018	1
4	7	8	01JUN2018	1
5	.	10	01JUL2018	.

Example 2: Actual Output

```
[1] "tax"        "income"     "date1"      "net.income"

  tax income      date1 net.income

1   1      2 2018-03-01          1
2   3      4 2018-04-01          1
3   5      6 2018-05-01          1
4   7      8 2018-06-01          1
5  NA     10 2018-07-01         NA

[1]  2  4  6  8 10

[1] "Date"
```

Example 2: Actual Output

```
'data.frame': 5 obs. of  4 variables:
       $ tax   : num  1 3 5 7 9
       $ income: num  2 4 6 8 10
       $ date1: Date, format: "2018-03-01" "2018-04-01" ..
       $ net.income: num  1 1 1 1 NA
```

tax	income	date1	net.income
Min. :1.0	Min. : 2	Min. :2018-03-01	Min. :1
1st Qu.:2.5	1st Qu.: 4	1st Qu.:2018-04-01	1st Qu.:1
Median :4.0	Median : 6	Median :2018-05-01	Median :1
Mean :4.0	Mean : 6	Mean :2018-05-01	Mean :1
3rd Qu.:5.5	3rd Qu.: 8	3rd Qu.:2018-06-01	3rd Qu.:1
Max. :7.0	Max. :10	Max. :2018-07-01	Max. :1
NA's :1			NA's :1

Example 2: Actual Output

Obs	tax	income	date1	net_income
1	1	2	01MAR2018	1
2	3	4	01APR2018	1
3	5	6	01MAY2018	1
4	7	8	01JUN2018	1
5	.	10	01JUL2018	.

Example 3: Run regression in R, return results to SAS

- Copy SAS data set to R data frame
- Run regression in SAS with PROC REG to compare
- Run regression in R
- Return R regression results to SAS, use to calculate values

Example 3: Run regression in R, return results to SAS

```
data Class;                                          /* Create data set in SAS */
   length Name $8;
   input Name $ Height Weight;
datalines;
Alfred   69.0 112.5
Alice    56.5  84.0
Barbara  65.3  98.0
Carol    62.8 102.5
Henry    63.5 102.5
James    57.3  83.0
Jane     59.8  84.5
Janet    62.5 112.5
Jeffrey  62.5  84.0
John     59.0  99.5
Joyce    51.3  50.5
Judy     64.3  90.0
Louise   56.3  77.0
Mary     66.5 112.0
Philip   72.0 150.0
Robert   64.8 128.0
Ronald   67.0 133.0
Thomas   57.5  85.0
William  66.5 112.0
;run;
```

Example 3: Run regression in R, return results to SAS

```
proc reg data=Class;   /* Regression in SAS to compare */
   model Weight = Height;
run;

proc iml;

    /* Copy SAS data set Class to R data frame Class1 */
  call ExportDataSetToR("Class", "Class1");

    /* Call R */
  submit / R;

      # Do regression
    Model <- lm(Weight ~ Height, data=Class1,
      na.action="na.exclude")
```

Example 3: Run regression in R, return results to SAS

```
# Create/display R data w/coefficients, fitted values, residuals
    ParamEst <- data.frame(Intercept = coef(Model)[1],
            Height_Parameter = coef(Model)[2])
    Pred     <- fitted(Model)
    Resid    <- residuals(Model)
    print(ParamEst)
                Intercept Height_Parameter
    (Intercept) -143.0269          3.89903
    print(Pred)
          1         2         3         4         5
    126.00617  77.26829 111.57976 101.83218 104.56150
          6         7         8         9        10
     80.38752  90.13509 100.66247 100.66247  87.01587
         11        12        13        14        15
     56.99333 107.68073  76.48849 116.25859 137.70326
         16        17        18        19
    109.63024 118.20811  81.16732 116.25859
```

Example 3: Run regression in R, return results to SAS

```
  endsubmit;  /* End R submit block, return to SAS/IML */

    /* Retrieve parameter estimates from R */
  call ImportDataSetFromR("pe", "ParamEst");

quit; /* end PROC IML */

title Parameter estimates copied from R to SAS ;
proc print data=pe; /* Print parameter estimates from R */
run; title;
```

Parameter estimates copied from R to SAS

Obs	Intercept	Height_ Parameter
1	-143.027	3.89903

Example 3: Run regression in R, return results to SAS

```
  /* Use parameter estimates calculated in R to
     compute predicted values for some heights */
data predictweightfromheight;
   set pe;
   keep height predicted_weight;
   do height = 55 to 70 by 5;
     predicted_weight = intercept + height *
       Height_Parameter;
     output;
   end;
run;

title Predicted weight from height;
proc print data=predictweightfromheight;
run;
```

Example 3: Run regression in R, return results to SAS

Predicted weight from height

Obs	height	predicted_ weight
1	55	71.420
2	60	90.915
3	65	110.410
4	70	129.905

Example 3: Actual Output from R

```
             Intercept Height_Parameter
(Intercept) -143.0269           3.89903

      1         2         3         4         5
126.00617  77.26829 111.57976 101.83218 104.56150
      6         7         8         9        10
 80.38752  90.13509 100.66247 100.66247  87.01587
     11        12        13        14        15
 56.99333 107.68073  76.48849 116.25859 137.70326
     16        17        18        19
109.63024 118.20811  81.16732 116.25859
```

Example 3: Actual Output from SAS

Parameter estimates copied from R to SAS

Obs	Intercept	Height_ Parameter
1	-143.027	3.89903

Predicted weight from height

Obs	height	predicted_ weight
1	55	71.420
2	60	90.915
3	65	110.410
4	70	129.905

Example 4: Copy IML matrix to R, call R package and R graphics

- Define data, transfer it to R
- Pass a parameter to R
- Use R LIBRARY function to load an R package
- Call R functions to analyze data
- Copy R analysis results to SAS/IML vectors, analyze
- Call R to display plot, plot window stays open until we delete it or end SAS/IML with QUIT;
- Create PDF file with plot
  - On Windows, use / or \\ instead of \ in file path

Example 4

```
/*I. Create IML vector q, transfer it to R as vector rq */

proc iml;
  q = {3.7, 7.1, 2, 4.2, 5.3, 6.4, 8, 5.7, 3.1, 6.1, 4.4,
       5.4, 9.5, 11.2};
  RVar = "rq";
  call ExportMatrixToR( q, RVar );

/* II. Pass name of matrix to R as parameter.  Load
  KernSmooth package, compute kernel density estimate. */

submit RVar / R;
   library(KernSmooth)
   idx <-which(!is.na(&RVar))     # exclude missing values
   p <- &RVar[idx]              # from KernSmooth functions
   h = dpik(p)          # Sheather-Jones plug-in bandwidth
   est <- bkde(p, bandwidth=h)          # est has 2 columns
endsubmit;
```

Example 4

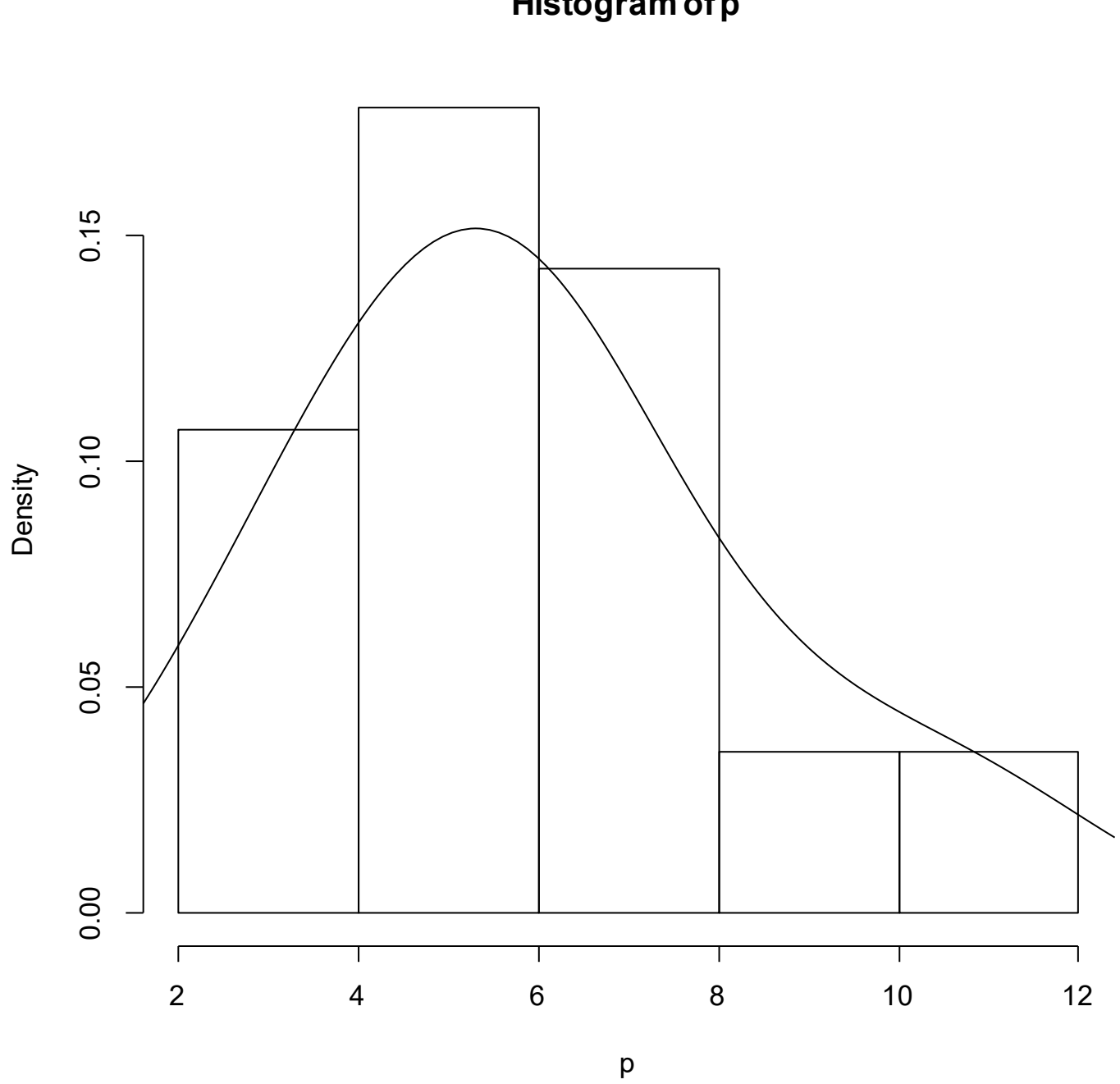
```
/* III Copy results to IML matrix, do more computations */

call ImportMatrixFromR( m, "est" );
                          /* estimate density for q >= 8 */
x = m[,1];                          /* x values for density */
idx = loc( x>=8 );                    /* find values x >= 8 */
y = m[idx, 2];  /* extract corresponding density values */

/* Use trapezoidal rule to estimate area under density
  curve.   Area of trapezoid with base w and heights h1
  and h2 is w*(h1+h2)/2. */
w = m[2,1] - m[1,1];
h1 = y[1:nrow(y)-1];
h2 = y[2:nrow(y)];
Area = w * sum(h1+h2) / 2;
print Area;
```

Example 4

```
/* IV. Display plot, save plot as PDF */

submit / R;
   hist(p, freq=FALSE) # display histogram
   lines(est)          # kde overlay

     # Linux
   pdf("/my/home/m1xxx00/sasandr/hist1.pdf")
     #Windows
   # pdf("u:/m1xxx00/sasandr/hist1.pdf")
   hist(p, freq=FALSE) # write histogram to PDF file
   lines(est)          # kde overlay
   dev.off()

endsubmit;
quit; /* end PROC IML */
```

Example 4: Output print Area in Part III

```
      Area
0.2118117
```


Histogram of p
Density
0.00
0.05
0.10
0.15
2
4
6
8
10
12
p

Example 5: Use a Source file

```
proc iml;

     /* optionally copy data from SAS to R */

  submit / R;

     # Linux file with R statements
    source("/my/home/directory/rprogs/rcode1.R")

     # Windows file with R statements
    # source("u:/m1xxx00/rprogs/rcode1.R")
  endsubmit;

     /* optionally copy data from R to SAS */

quit; /* end IML */
```

Example 6: Save/read one or multiple R objects or entire workspace to/from a file

First, in R:

- Change current working directory with setwd()
- Create two objects (variables) and a data frame
- Save R data to files
  - Single object (variable) to a file w/saveRDS()
  - Two objects (variables) to a file w/save()
  - Entire workspace to a file w/save.image() (Not recommended if very large data files or many intermediate objects in R session)

Then, in SAS:

- Read each saved object, print, modify data, save R data to files w/same three functions

Example 6: in R

```
setwd("~m1xxx00/rstuff") # set current working directory
object1<-1:5             # create objects, assign values
object2<-11:15
   # create data.frame w/2 objects, assign each a new name
df<- data.frame(column1=object1, column2=object2)
                           # save object1 to external file
saveRDS(object1, file = "mydata_object1.rds")
object1[3:5] = -1*object1[3:5]     # modify 3rd-5th values
                          # save 2 objects to external file
save(object1, object2, file = "objects_1_and_2.RData")
          # save entire workspace (do with caution if big)
save.image(file = "my_entire_work_space1.RData")
         # exit R session without saving current workspace
         # to a .Rdata in current working directory
quit("no")
```

Example 6: in SAS interface to R

```
proc iml;
submit / R;
  setwd("~m1xxx00/rstuff") # set current working directory
    # read, print object created with:
    #   object1<-1:5  saveRDS(object1,file= "mydata_object1.rds")
  object1_from_r= readRDS(file = "mydata_object1.rds")
  print(object1_from_r)
  [1] 1 2 3 4 5

  # read, print objects created with:
  #   object1<-1:5
  #   object2<-11:15
  #   object1[3:5] = -1*object1[3:5]
  #   save(object1,object2,file="objects_1_and_2.RData")
  load("objects_1_and_2.RData")
  print(object1)
  [1]  1  2 -3 -4 -5
  print(object2)
  [1] 11 12 13 14 15
```

Example 6: in SAS interface to R

```
# restore workspace and print dataframe created with:
#   object1<-1:5
#   object2<-11:15
#   df<- data.frame(column1=object1, column2=object2)
#   save.image(file = "my_entire_work_space1.RData")
load(file = "my_entire_work_space1.RData")
print(df)
   column1 column2
 1       1      11
 2       2      12
 3       3      13
 4       4      14
 5       5      15
```

Example 6: in SAS interface to R

```
library(dplyr)                  # need for mutate()

                                # add column to data frame, print
df <- mutate(df, column3 = ifelse(column2>12,
   log(column2), column2))
print(df)
  column1 column2   column3
1       1      11 11.000000
2       2      12 12.000000
3       3      13  2.564949
4       4      14  2.639057
5       5      15  2.708050
```

Example 6: in SAS interface to R

```
  # save object1_from_r to new external file
  saveRDS(object1_from_r,file="mydata_object1_from_r.rds")

   # save 2 objects to external file, overwrite prior file
  save(object1, object2, file = "objects_1_and_2.RData")

        # Save entire workspace, overwrite prior file.
        # Workspace available in R, future SAS/R sessions.
        # To prevent error in R, remove .Last definition.
  rm(.Last)
  save.image(file = "my_entire_work_space1.RData")
endsubmit;
quit;
```

Example 6: Actual output

```
[1] 1 2 3 4 5
[1]  1  2 -3 -4 -5
[1] 11 12 13 14 15
  column1 column2
1       1      11
2       2      12
3       3      13
4       4      14
5       5      15

  column1 column2   column3
1       1      11 11.000000
2       2      12 12.000000
3       3      13  2.564949
4       4      14  2.639057
5       5      15  2.708050
```

Conclusion

- R interface, Linux and Windows SAS, starting in SAS 9.3
  - Call R functions from PROC IML, return results
  - Transfer data between R and SAS
  - Call R from SAS, not the reverse
- Potential users
  - PROC IML users
  - Non-PROC IML users: can use PROC IML just as wrapper to
    - **Transfer data between SAS and R**
    - **Call R functions, return results**
    - **This was presentation focus**

For more information, please contact
Bruce Gilsen
Federal Reserve Board, mail stop N-122
202-452-2494
bruce.gilsen@frb.gov