TUTORIALS ON S-PLUS AND MATLAB

Nagaraj K. Neerchal, Ph.D
Department of Mathematics and Statistics
University of Maryland Baltimore County
Baltimore, MD 21228

Acknowledgement: This material grew out of interaction between Nagaraj K. Neerchal, STAT 601 instructor and some diligent students of the Fall 1989 class. The hard-working students involved in this project are Amit Bhattacharyayy, Kalyan Ghosh, and Soma Sengupta.

Splus Tutorial - I

Basics

- Connect to MATH13 or SGI1 or UMBC8

- Once you are able to log into the computer and have received the UNIX prompt, type Splus < *Ret* > [Only "S" is capitalized]

  This puts you into a Splus session and will return with the prompt ">".

- > q() < *Ret* >

  You are now quitting the Splus session. The machine will return your Unix prompt.

Splus Tutorial - II

- Assignments :-

  Assignment operator is "$\leftarrow$". (It means "gets"). It consists of a "<" and a "-" sign, e.g.,

  > x $\leftarrow$ 1.4 < *Ret* >

  means that $x = 1.4$

- Long expressions can be continued over any number of lines, so long as it is clear that the end of expression cannot have occurred yet, e.g.,

  > z $\leftarrow$ x+3*(y- < *Ret* >

  + mean(y)) < *Ret* >

  The first line is doubly sure of being continued since it ended with the operator "$-$" and also has an unmatched left parenthesis. As shown above, Splus prompts with "+" for continuation lines.

- > # This prints the comments written on the right of "#" but does execute it.

- To see x, type

  > x < *Ret* >

  [1] 1.4

  [1] means that the number at the beginning of the line is the first element of the vector.

- Object Names :

  Legal Splus names include letters, numbers and periods(.) and cannot start with numbers. It is very important to note that Splus distinguishes capital letters from the lowercase letters.

- Creating Vectors :

  "c" function combines numbers into a vector. e.g.,

  1. > y ← c(15.1,11.3,7.8) < *Ret* >

     will give you

     > y < *Ret* >

     [1] 15.1 11.3 7.8

     "c" function accepts both numbers and vectors simultaneously, e.g.,

  2. > c(1,2,c(3,4)) < *Ret* >

     [1] 1 2 3 4

- Subscripting vectors :

  1. In the above examples, to get the second element of y, type

     > y[2] < *Ret* >

     [1] 11.3

  2. To form subvectors,

     > y[-1] < *Ret* >

     [1] 11.3 7.8

     and

     > y[-3] < *Ret* >

     [1] 15.1 11.3

- Making Sequence :

  1. ":" operator creates sequence between any two numbers in steps of 1 or -1. For example,

     > x ← 3:8 < *Ret* >

     > x < *Ret* >

[1] 3 4 5 6 7 8

and

> y ← 3:-4 < *Ret* >

> y < *Ret* >

[1] 3 2 1 0 −1 −2 −3 −4

2. "seq" function creates sequences with desired increments. For example,

   > z ← seq(-10,6,by=2) < *Ret* >

   > z < *Ret* >

   [1] −10 −8 −6 −4 −2 0 2 4 6

   and

   > l ← seq(10,2,by=-2) < *Ret* >

   > l < *Ret* >

   [1] 10 8 6 4 2

- Creating Matrix :

  To enter a matrix, you can do the following :

  > x1 ← matrix(c(2,4,6,8,9,3),3,2) < *Ret* >

  will give you a 3 × 2 matrix as shown below :

  > x1 < *Ret* >

	[,1]	[,2]
[1,]	2	8
[2,]	4	9
[3,]	6	3

  To enter the elements rowwise;

  > x2 ← matrix(c(2,4,6,8,9,3),3,2,byrow=T) < *Ret* >

  > x2 < *Ret* >

	[,1]	[,2]
[1,]	2	4
[2,]	6	8
[3,]	9	3

Another example :

> y ←matrix(1:12, ncol=4,byrow=T) < *Ret* >

> y < *Ret* >

	[,1]	[,2]	[,3]	[,4]
[1,]	1	2	3	4
[2,]	5	6	7	8
[3,]	9	10	11	12

- Another way to enter numbers in vector or matrix is by using "scan" function:

  1. Creating vector :

     > vec1 ← scan() < *Ret* >

     1 : 2 4 6 8 10 < *Ret* >

     6 : 12 14 16 < *Ret* >

     8 : 18 < *Ret* >

     9 : < *Ret* >

     > *vec*1 < *Ret* >

     [1] 2 4 6 8 10 12 14 16 18

     When you call "scan" function with no arguments , it reads numbers separated by spaces, tabs or return until you hit return on a line all by itself. Each time you start a new line scan prints the index number of the next value it will read.

  2. Creating Matrix :

     > mat1 ← matrix(scan(), byrow=T, ncol=4) < *Ret* >

     1 : 1 2 3 4 < *Ret* >

5 : 5 6 7 8 $< Ret >$

9 : 9 10 11 12 $< Ret >$

13 : $< Ret >$

$>$ mat1 $< Ret >$

	[,1]	[,2]	[,3]	[,4]
[1,]	1	2	3	4
[2,]	5	6	7	8
[3,]	9	10	11	12

Note: To get specifically a row or a column vector, you may conveniently use matrix command, though "c" function also creates vectors.

- Subscripting the matrix :

1. $x[1, 2]$ gives $(1, 2)$ th element of the matrix x, e.g.,

   $>$ mat1[2,3] $< Ret >$

   [1] 7

2. $>$ mat1[c(1,3),c(1,4)] $< Ret >$

	[,1]	[,2]
[1,]	1	4
[2,]	9	12

   This new matrix is formed deleting the second row and the second and third column of mat1.

3. $>$ mat1[1:3,c(1,4)] $< Ret >$

	[,1]	[,2]
[1,]	1	4
[2,]	5	8
[3,]	9	12

4. $>$ mat1[,c(1:3)] $< Ret >$

	[,1]	[,2]	[1,3]
[1,]	1	2	3
[2,]	5	6	7
[3,]	9	10	11

- If you want to change (1,2)th element of mat1 to 0, write

  $>$ mat1[1,2] $\leftarrow$ 0 $<$ *Ret* $>$

- If you want to change the entire first column of mat1 to $(6, 7, 8, 9)$,say, write

  $>$ mat1[,1] $\leftarrow$ c(6,7,8,9) $<$ *Ret* $>$

  For changing the 1st row, use mat1[1,].

Splus Tutorial - III

- Usual arithmetic operators :

  "+" : addition
  "-" : subtraction
  "*" : multiplication
  "/" : division
  "^" or "**" : exponentiation
  "5e3" : 5×10^3

  1. One can do addition or subtraction with two matrices of the same order by "+" or "-".
  2. Usual matrix product (provided their orders conform) is given by "%*%". Note : "*" computes elementwise multiplication, e.g.,

$$\begin{pmatrix} 2 & 3 & 5 \\ 6 & 1 & 2 \end{pmatrix} * \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 & 3 & 5 \\ 12 & 1 & 2 \end{pmatrix}$$

     whereas

$$\begin{pmatrix} 2 & 3 & 5 \\ 6 & 1 & 2 \end{pmatrix} \%*\% \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 12 \\ 15 \end{pmatrix}$$

  3. "/" gives elementwise division for matrices of the same order.

- > solve(A,b) < *Ret* >

  gives the solution to the system of equation $Ax = b$. "A" must be non-singular. If "b" is missing, A^{-1} is returned, *i.e.*,

  > solve(A) < *Ret* >

  returns A^{-1}.

- $>$ t(A) $<$ *Ret* $>$

  gives the transpose of the matrix "A".

- Note : $>$ $A \hat{} r$ $<$ *Ret* $>$

  doesn't compute "A" (a square matrix) multiplied by itself r (a scalar) times. It only does elementwise multiplication of "A" r times.

Splus Tutorial - IV

- Generating Special Matrices :

1. > m ← matrix(k,3,4) < *Ret* >

   gives you a 3×4 matrix with all the elements being k (k is any number)

   Note : When k = 0, the above command gives you a **zero matrix**

2. **Diagonal matrix** :

   (*i*) > n ← diag(c(1,2,3,4,5)) < *Ret* >

   gives you

   > n < *Ret* >

   1 0 0 0 0

   0 2 0 0 0

   0 0 3 0 0

   0 0 0 4 0

   0 0 0 0 5

   (*ii*) > m ← diag(k,3) < *Ret* >

   gives you

   > m < *Ret* >

   k 0 0

   0 k 0

   0 0 k

   Note : k = 1 gives you the **identity matrix**.

Another simple way to form an identity matrix is :

(*ii*) > m ← diag(3) < *Ret* >

> m < *Ret* >

1 0 0

0 1 0

0 0 1

- Generation of Random Numbers :

  > runif(n) < *Ret* >

  generates n random numbers from uniform(0,1) distribution.

  Splus contains random number generates for a dozen distribution. They all have names consisting of the letter "r" followed by a code for the distribution, such as "norm" for the normal, "unif" for the uniform, "t" for Student's t, etc. The first argument is the sample size desired. Other arguments, if any, are the parameters of the desired distribution.

  Some of the popular distributions with codes & default value of the corresponding parameters are given in the following table :

Distribution	Code	Parameters	Defaults
Beta	beta	shape1, shape2	- -
Cauchy	cauchy	location, scale	0 , 1
Chisquare	chisq	df	-
Exponential	exp	-	-
F	f	df1, df2	- -
Gamma	gamma	shape	-
Log-normal	lnorm	mean, sd (of log)	0 , 1
Logistic	logis	location, scale	0 , 1
Normal	norm	mean, sd	0 , 1
Student's t	t	df	-
Uniform	unif	min., max.	0 , 1

  Random generators keep giving new sequence of random numbers at different times. To get rid of it, we can follow the following steps :

> save.seed ← .Random.seed < *Ret* >

Now generate your desired random numbers of size n.

> .Random.seed ← save.seed < *Ret* >

Now, if you generate the random numbers again, those will match with the previous ones.

Another way of getting the same random numbers again and again is :

> set.seed(i) < *Ret* >

Here "i" should be an integer.

- In addition to Random number generation for the distributions mentioned before, you can compute left-tail probability, quantile and density functions by placing the letters 'p', 'q', or 'd' respectively before the distribution code, e.g.,

  1. > pnorm(2.5) < *Ret* >

     gives you P[X < 2.5], where X ∼ N(0,1).

  2. > qt(.95,7) < *Ret* >

     gives you the 95 th quantile of the t-distribution with 7 df.

  3. > df(1.5,3,4) < *Ret* >

     gives the pdf of F(3,4) distribution at F=1.5

Splus Tutorial - V

- Some special functions:

  Let x is a vector containing some elements.

  1. > min(x) < *Ret* >

     gives the minimum.

  2. > max(x) < *Ret* >

     gives the maximum.

  3. > mean(x) < *Ret* >

     gives the mean of x.

  4. > var(x) < *Ret* >

     gives the variance of x. (with divisor length$(x) - 1$)

     To compute s.d., one has to use

  5. > sqrt(var(x)) < *Ret* >

  6. > median(x) < *Ret* >

     gives the median of x.

  7. > sort(x) < *Ret* >

     arranges x in the ascending order.

  8. > rev(sort(x)) < *Ret* >

     arranges x in the descending order.

  9. > order(x) < *Ret* >

     gives the vector containing the positions of the element of x from the least to the greatest.

  10. > rank(x) < *Ret* >

      ranks the elements of x from the highest to the least,

11. > quantile(x, .95) < *Ret* >

    gives the 95th quantile of x.

12. > quantile(x,c(.75,.25)) < *Ret* >

    gives the 75th and 25th percentile.

    Given two vectors of observations x and y of the same length,

13. > var(x,y) < *Ret* >

    computes the covariance between x and y.

14. > cor(x,y) < *Ret* >

    computes the correlation between x and y.

    If x and y are matrices, the above two results are covariance matrix and correlation matrix respectively, treating the columns of x and y as variables.

15. > range(x) < *Ret* >

    computes $c(min(x), max(x))$.

16. > range(x,y) < *Ret* >

    computes $c(min(x, y), max(x, y))$.

17. > sum(x) < *Ret* >

    computes the sum of all the elements in the vector x.

18. > prod(x) < *Ret* >

    computes the product of all the elements of x.

19. > seq(along=x) < *Ret* >

    returns the vector $1 : length(x)$ that goes along with x.

20. > abs(x-3) < *Ret* >

    returns $|x - 3|$.

21. > rep(1:3,2) < *Ret* >

    gives [1] 1 2 3 1 2 3.

- There are comparison operators which compare values and produce true or false. These are :

$<$	: less than
$>$	: greater than
$<=$	: less than or equal to
$>=$	: greater than or equal to
$==$	: equal to
$!=$	: not equal to

- There are also logical operators.

&	: and
|	: or
!	: not

- There are two control operators.

  && expect a single value for its 1st operand and if this value is true, then && evaluates and returns as its value the 2nd operand. Otherwise it returns false without evaluating its 2nd operand.

  || expect a single value for its 1st operand. If this value is false, then the 2nd argument is evaluated and returned. Otherwise it returns true without evaluating its 2nd operand.

  eg. if (all(x>0) && sum(log(x)>100)) big $\leftarrow$ T $< Ret >$

  would not evaluate the logarithms unless all the values in x were positive.

- Here you can create a function of your own, e.g.,

  1. > square ← function(x) x^2 < *Ret* >

     creates a function object named "square". You can now use the function "square" just like any supplied function as :

  2. > square(1:5) < *Ret* >

     [1] 1 4 9 16 25

- You can look at the definition of any function by typing its name :

  > square < *Ret* >

  function(x) x^2 < *Ret* >

  But

  > square(···) < *Ret* >

  calls the function and execute it using the values of the arguments given.

  Note : A function-call must always include the parentheses.

- If you want to make some changes in the definition of "square", you must use the *vi* screen editor to revise the function "square", e.g.,

  > newsquare ← vi(square) < *Ret* >

  [See any standard book on "UNIX" system for the "vi" editor.]

  The "vi" function puts you in "vi" text editor, working on a file that contains the definition of "square". When the changes are finished, write the file and quit from the editor. This means that a new function named "newsquare" with the new version of the older function "square".

  Note : The older function square will be unaltered. So, if you call the "square" function, you'll get the older version and calling the "newsquare" function, you'll

get the newer version. Hence, it is important to assign a new name to the object returned by the "vi" function. You can remove the older version from your list by using the "rm("square")" command. Now, if you type ls(), you will find all variable-objects as well as your file-objects, but the file-object "square" will not be there.

- A function in Splus can have any number of arguments and the computation that is done by it can be anything expressible in Splus, e.g.,

- > test← function(){ < *Ret* >

  x← c(1,2,3,4) < *Ret* >

  y← c(2,3,4,5) < *Ret* >

  p← x-y < *Ret* >

  q← x+y < *Ret* >

  } < *Ret* >

  Now,

  > test() < *Ret* >

  [1] 3 5 7 9

  which means that the function "test" will return only the value of the last expression (*i.e.* q) between the braces. If you want to see all of them, you have to form the function as :

- > test1← function(){ < *Ret* >

  x← c(1,2,3,4) < *Ret* >

  y← c(2,3,4,5) < *Ret* >

  p← x-y < *Ret* >

  print x < *Ret* >

  print y < *Ret* >

print p < *Ret* >

q← x+y < *Ret* >

} < *Ret* >

Now,

> test1()

[1] 1 2 3 4

[1] 2 3 4 5

[1] −1 −1 −1 −1

[1] 3 5 7 9

Splus Tutorial - VII

The steps useful for the purpose of data analysis.

- To create a file named "people" with a unix text editor(e.g. "vi" editor) and put the following information into it.

Alex	120	M	15
Barnie	162	M	42
Beth	137	F	27
Bob	191	M	31
Barbara	135	F	32
David	171	M	60
Fifi	111	F	23

- Start S-plus.

  Now to read in the above unix file (using the function "scan" with the argument "what") do the following steps.

  > people.what ← list(name="", weight=0, gender="",age=0) < *Ret* >

  > people ← scan("people", what=people.what) < *Ret* >

  > people < *Ret* >

  $ name:

  [1] "Alex" "Barnie" "Beth" "Bob"

  [5] "Barbara" "David" "Fifi"

  $ weight:

  [1] 120 162 137 191 135 171 111

  $ gender:

  [1] "M" "M" "F" "M" "F" "M" "F"

  $ age:

```
[1] 15 42 27 31 32 60 23
```

Splus Tutorial - VIII
Hardcopy and Graphics

- Making a Hard copy :

  After logging on to Splus, type > pscript() < *Ret* >

  followed by the other Splus statements. When these statements are executed, tell Splus that you want to print all these results by :

  > ! lpr -r -h Postscript.out < *Ret* >

- Plotting and making Hard copy plots :

  Type

  > X11() < *Ret* >

  The X11 function creates an X11 graphics window on your screen with a control panel at the top. Now that you have started an Splus graphics device, you can issue any Splus plotting command, e.g.,

  1. > plot(aut.weight,aut.price) < *Ret* >

     to see a scatter plot of the object named "aut.weight" (along the x-axis) versus the object named "aut.price" (along the y-axis).

  2. > plot(aut.weight,aut.price,type = "l") < *Ret* >

     to the points in the plot joined by a line. To use the other options for the type of the plot, consult the manual. The plot function uses the names of its arguments as the default axis labels. You may use the commands xlab="$\cdots$" and/or ylab="$\cdots$" to add labels for the horizontal and/or the vertical axes of your plot, e.g.,

  3. > plot(aut.weight,aut.price,xlab="Auto weights",ylab="Auto prices")

     < *Ret* >

You can also use the "Title" function to add titles to your plots. The main argument adds a title above the plot, and the sub argument puts one below the plot, e.g.,

4. > title(main="Automobile Prices versus Weights") < *Ret* >

   You can use the "text" function to put labels in the plotting region. The first argument is the x-coordinate where you want the text, the second argument is the y-coordinate, and the third is the text (in quotes as usual). By default , the "text" function centers your text at the location given by the first two arguments.

   Clicking *Button 1* (usually the left move button) on the "Print Graph" entry in the control panel makes a hard copy of the current plot. If you are using a laser printer for making hard copies, set "Print Method" entry in the control panel to "Laser Jet" [Note : Clicking *Button 1* on the "Print Method", you can toggle the hardcopy output between "Laser Jet' ' and "Post Script"]. You can also toggle the orientation of the hardcopy plot between "Landscape" (which puts the x-axis on the long side of the paper) and "Portrait" (which puts the x-axis on the short side of the paper) by clicking *Button 1* in the "Print Orientation" entry.

MATLAB TUTORIALS

TUTORIAL 1

1. Log on to MATH13 or SGI1 or UMBC8

2. $ matlab_def < *return* >

3. $ matlab < *return* >

   The machine enters you into a matlab session and will return with the matlab prompt >>.

4. >> exit < *return* >

   You are now quitting the matlab session. The machine will return $ prompt.

TUTORIAL 2

Objective: Entering a column vector / a row vector / a matrix

Begin a matlab session by proceeding as in steps 1, 2 and 3 of TUTORIAL 1.

- To enter a row vector named x:

$$>> x = [1234]; \quad < return >$$

  Now the vector is entered and is available for use.

- To enter a column vector named y:

$$>> y = [1; 2; 3; 4]; \quad < return >$$

- To enter a matrix a:

$$>> a = \begin{matrix}[& 1111; \\ & 1234; \\ & 13610; \\ & 141020]; \end{matrix} \quad < return >$$

Remarks:

1. If the semicolon preceeding the ¡ *Return* ¿ are not given MATLAB will display the results of the most recent operation.

2. $>> a = [1111; 1234; 13610; 141020];$

   will have the same result as the command described earlier.

3. You may alter the values of any entry by a reassignment such as

$$>> a(:, 1) = [0; 0; 0; 0];$$

4. You may change the values of all elements of a row by

$$>> a(4,:) = [3567];$$

5. $>> alf = [3];$
   and
   $>> alf = 3;$
   have the same result, a 1×1 matrix.

6. You may name your matrices with any twenty or fewer character strings as long as it starts with a letter.

TUTORIAL 3

Objective: Simple matrix operations

Suppose you have matrixes A, B, C, D already entered.

1. $>> X = A + B; < Return >$

   Usual matrix sum of A and B will be stored in X. If the matrices A and B are not of same dimension then an error message will be printed.

2. $>> X = A * C; < Return >$

   Usual matrix product of A and C will be stored in X. If A and C do not confirm, an error message will be displayed.

3. $>> X = A\hat{}C; < Return >$ Where r is scalar and A is square matrix. The result is

   (a) $\underbrace{A \ldots A \ldots\ldots A}_{r}$ when r is a positive integer.

   (b) A^{-1} = inverse of A when $r = -1$ and A is non-singular.

   In general,

   (c) $P\Lambda^r P'$, where $P\Lambda P'$ is the spectral decomposition of A, wherever this makes sense.

4. $>> X = A \,/\, C; < Return >$

   gives X = a solution to $A = XB$,

   $>> X = A \setminus C; < Return >$

   gives X = a solution to $AX = B$.

5. When any of the symbols $*$, $/$, $\setminus$, or $\hat{}$ is preceeded by a $\cdot$, then the operation will be attempted element wise. That is

   $>> C = A \ldots * B; < Return >$

will result in $C(i,j) = A(i,j) * B(i,j)$, if A and B are of the same size.

$>> C = A \ldots \setminus B; < Return >$

will result in $C(i,j) = \frac{A(i,j)}{B(i,j)}$

6. $>> B = A'; < Return >$

   will give the conjugate transpose.

7. $>> C = \mathrm{kron}(A,B); < Return >$

   will result in $C = A \otimes B = ((a_{i,j}B))$: Kronecker product.

TUTORIAL 4

Objective: Generating vectors and matrices

1. $x =$ start value: increment : end value; $< Return >$

   will create a row vector

   $$x = (\text{ start value, start value } + \text{ increment}, \ldots)$$

   The last value will be a number of the start value $+ j.$(increment), not exceeding the end value.

   e.g.

   $$>> x = 1 : 1 : 3; \quad \text{gives } x = [123]$$

   $$>> x = 1 : 2 : 10; \quad \text{gives } x = [13579]$$

   Remarks:

   1. The default value of increment is 1.
   2. The increment may be negative.
   3. Increment $= 0$ will generate an empty vector.

2. $A =$ eye $(m, n); < Return >$

   creates A such that $A(i, j) = \begin{cases} 1 & \iff i = j \\ 0 & \iff i \neq j \end{cases}$

   $A =$ zeros $(m, n); < Return >$

   creates a $m \times n$ matrix of zeros.

   $A =$ ones $(m, n); < Return >$

   creates a $m \times n$ matrix of ones.

   Remarks:

   When $m = n$, we may use $A =$ eye (m) etc.

3. $A = \text{rand } (m, n); < Return >$

creates a matrix $A_{m \times n}$ whose elements are randomly chosen from either $U[0, 1]$ distribution or $N(0, 1)$ distribution. You can choose either of these distributions by

$$\text{rand ('uniform'); or rand ('normal');}$$

Example 1:

$>>$ rand ('uniform'); $< Return >$

$>>$ rand $(5, 10); < Return >$

A will now have a random sample of 50 from $U(0, 1)$ arranged in a 5×10 array.

Example 2:

$>>$ rand ('normal'); $< Return >$

$>>$ rand $(100, 1); < Return >$

$>> B = 5. * \text{ones } (100, 1) + 2. * A;$

B contains a random sample of size 100 from $N(5, 4)$.

Remarks:

$>>$ rand ('seed', 2501);

sets the seed for random number generation to 2501. This is useful if you want to reproduce a sample.

TUTORIAL 5

Objective: To get a printout of the session (or a part of it)

>> diary filename 1

>> diary on

.....
.....
..... } the matlab computation you care to print.
.....
.....

>> diary off

>> diary filename 2

>> diary on

.....
.....
..... } the matlab computation you care to print.
.....
.....

>> diary off

>> exit

$ print $/q = mp1 \ldots filename1 \quad filename2$

TUTORIAL 6

Objective: To read data from a VMS file called ex1.dat

Matlab can not read .dat files. Matlab can read data from PROMATLAB files. The filetype for such files is .mat. There is a program which can transform your .dat file into a .mat file. It is an interactive program which asks you questions to get the information required to run., when involved. The utility is called "translate" and can be involved either at $ prompt or within matlab. An example session is given below:

$ translate < *Return* >

Your screen will be refreshed and

Prompt	Respond
Input file name:	ex1.dat

TUTORIAL 7

Objective: To read a .mat file

>> load filename < *Return* >

the entire file will be loaded as a matrix.

>> save filename < *Return* >

the contents of all variables will be saved in filename.mat for future use. This is the way to carry over your calculations from one matlab session to another. Do not try to print .mat files. use "translate" to change it to .dat first.

Also >> filename list of variables