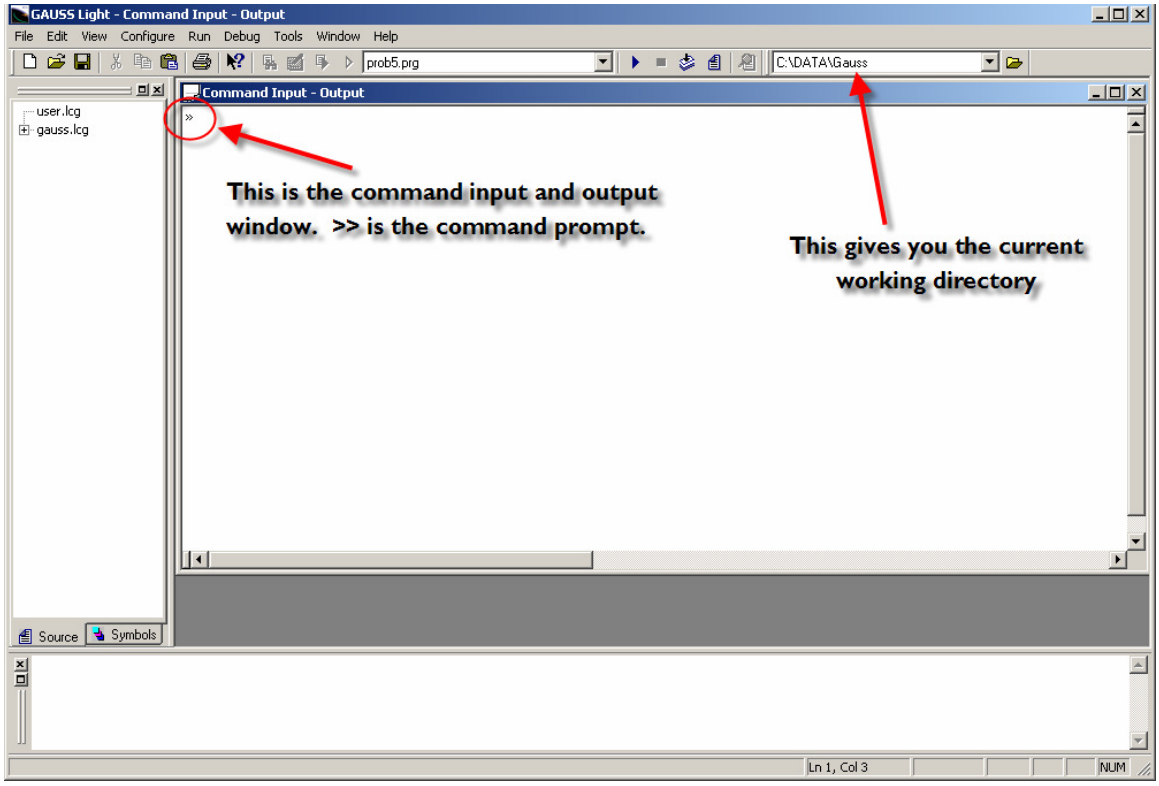
Introduction to Gauss

Gauss is a high level programming language that you can use to learn and do econometrics. This is a brief guide to get you started using GAUSS 8.

First, open the Gauss program. I'll assume you know how to find and open a program in MS Windows. This will open the following window



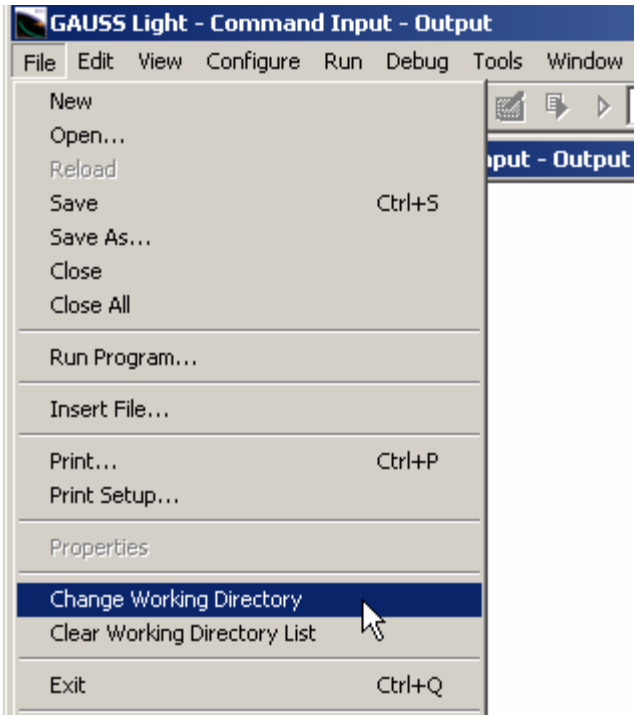
Gauss can be operated in two ways. First, you can use it interactively. That means you can input program statements one line at a time. You can check the results in each step to make sure the program is computing what you want it to compute.

The other way to use Gauss is to collect program statements into a file that can be executed in a batch.

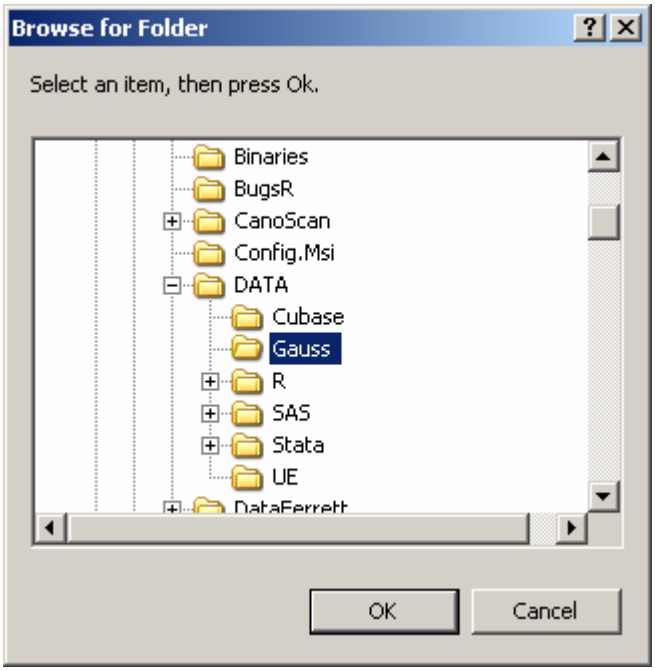
Change the working directory

The first thing you should do is to change Gauss's working directory to some location that is easy to get to. Use something as close to **`C:\`** as you can, using short directory and file names with no spaces. If you use something more complicated like **`C:\Documents and Settings\Lee\Desktop\Gauss\Data\`** then you are asking for trouble. I use

`C:\gauss\programs` or **`C:\data\gauss`**. To change the working directory, go to the menu bar and select File>Change Working Directory as shown



Browse for the desired location



and click OK.

Load Consumption.txt

The first thing you are going to do is to load the data contained in the file consumption.txt. This data in this file are in ascii format. That means that it contains no formatting information other than having spaces between the numbers. There are no variable names in the file. We'll add these separately.

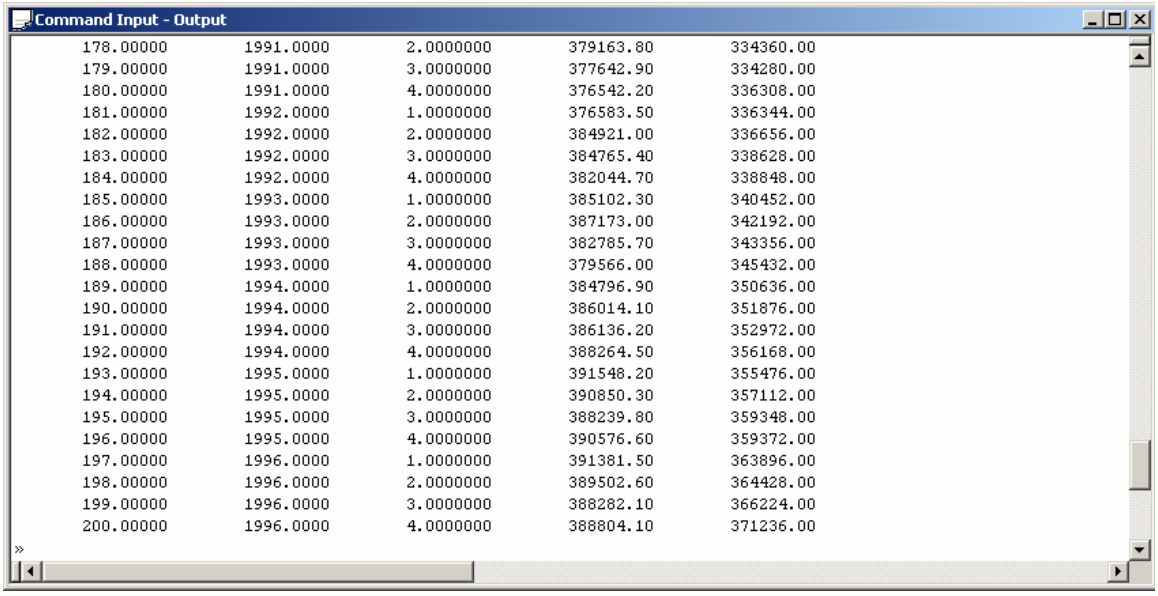
Download the consumption.txt file from my website and put it into your working directory. Then, at the command prompt, type

» load dat[200,5]=consumption.txt;

If this is successful then it will appear as if nothing happened. If it fails then you'll get an error message in the bottom window of the first figure above. To see if this loaded, type

» dat

at the command prompt and press enter. The contents of **`dat`** will be printed to the screen in the output window.

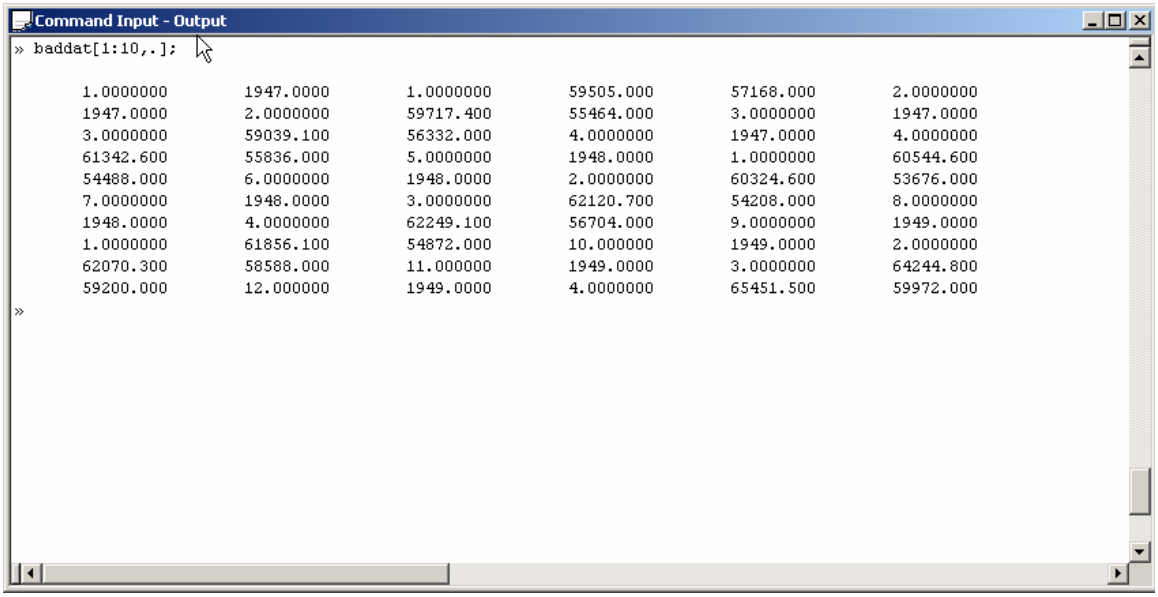


The syntax of the statement is fairly simple. load is the command use to load ascii files. dat[200,5] tells Gauss that you want to create a 200 row by 5 column matrix called dat. The contents of that will be the file consumption.txt. For this to work, the file has to contain 1000 numbers. Gauss will read in these numbers 5 at a time, assigning each to a row. So, if the data are arranged by observation with 5 variables and 200 observations (spaces between each number) then the data will be read in as desired.

Try loading in the data into 6 columns 200 rows and see what happens. Hint: it is not good.

```
» load baddat[200,6]=consumption.txt;
» baddat[1:10,.];
```

The result of this is



That does not look like data arranged in row and column by observation so you have obviously done something wrong.

The statement used to list the contents of the matrix has been modified using Gauss's indexing capabilities. Here I used

```
» baddat[1:10,.];
```

to list rows 1 through 10, with all columns. If I wanted to print the contents of the 3rd column I would use **`baddat[.,3].`** To find the element in the 2nd row and 2nd column use **`baddat[2,2]`** and so on.

Means and standard deviations

Two functions are particularly useful. To get the column means of a matrix **`X`** use **`meanc(X)`** and to get the sample standard deviations use **`stdc(X).`** For the consumption data these are:

```
» meanc(dat);

      100.50000
      1971.5000
      2.5000000
      213507.83
      189998.84
```

```
» stdc(dat);

        57.879185
        14.467083
        1.1208396
        116948.86
        101117.11
```

Also, you can get correlations among the variables in **dat** using **corrx(dat).**

The 4th column of the matrix dat is YD, personal disposable income, 1986 dollars, Canada, SA and column 5 is CE, personal consumption expenditure, 1986 dollars. If you wanted the sample mean of YD you could simple take

```
» avgdat = meanc(dat);
» avgYD = avgdat[4];
» avgYD;
        213507.83
```

First, you found the means of all the columns. Then you use the indexing ability to pick off the 4th element (the one that corresponds to YD). Notice that avgdat is a vector (1 row or column) and so you can index it using a single argument, it position in the vector.

Regression

Once the data are in X and y, regression is a snap. Let's put consumption into y and income and a column of 1's into X.

```
» y=dat[.,5];
» jt = ones(rows(dat),1);
» x = jt ~ dat[.,4];
```

The first line puts all rows (.) of the 5th column of dat into a new variable, **y**. The next line creates a column of ones to put into **x**. It uses the **ones(n,k)** function, where **n** is the number of rows and **k** is the number of columns that you want to take the value of one. We used the **rows(dat)** function to count the number of rows in our **dat** matrix. This column of ones is now called **jt**. In the last line we used the horizontal concatenation operater ~ to join the ones column with the 4th column of **dat** (which contains income). Print out the first 5 rows of **x** to confirm that the data are arranged as you want.

```
» x[1:5,.];

        1.0000000        59505.000
        1.0000000        59717.400
        1.0000000        59039.100
```

```
1.0000000        61342.600
1.0000000        60544.600
```

Now that y and X are defined we want to compute least squares. Recall that algebraically, the least squares estimator is

$$b = (X'X)^{-1}X'y$$

The syntax in Gauss could hardly be simpler (or more intuitive)

```
b = invpd(x'*x)*x'*y
```

This uses the **invpd()** function, which takes the inverse of a positive definite, symmetric matrix; it uses the matrix multiplication operator, *****; and it uses the transpose operator, **'**.

```
» b = invpd(x'*x)*x'*y;
» b;

      6000.2561
     0.86178847
```

These are your least squares estimates of the intercept and slope of the model. Now, it is up to you to keep track of which is which! This is done based on the algebra and on how you built x. Since the column of ones is in the first column of x, the first element of b will correspond to the intercept. Try this, make a new x and reestimate the model.

```
» x1 = dat[.,4]~jt;
» b1 = invpd(x1'*x1)*x1'*y;
» b1;

     0.86178847
      6000.2561
```

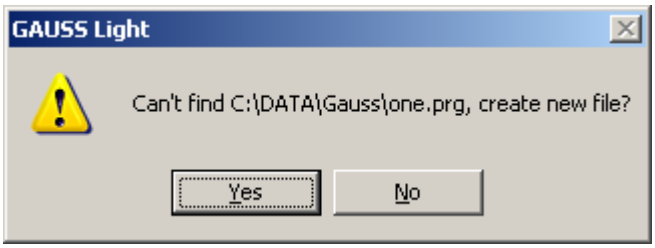
Notice that reversing the columns of x reverses the coefficients.

Put it into a program

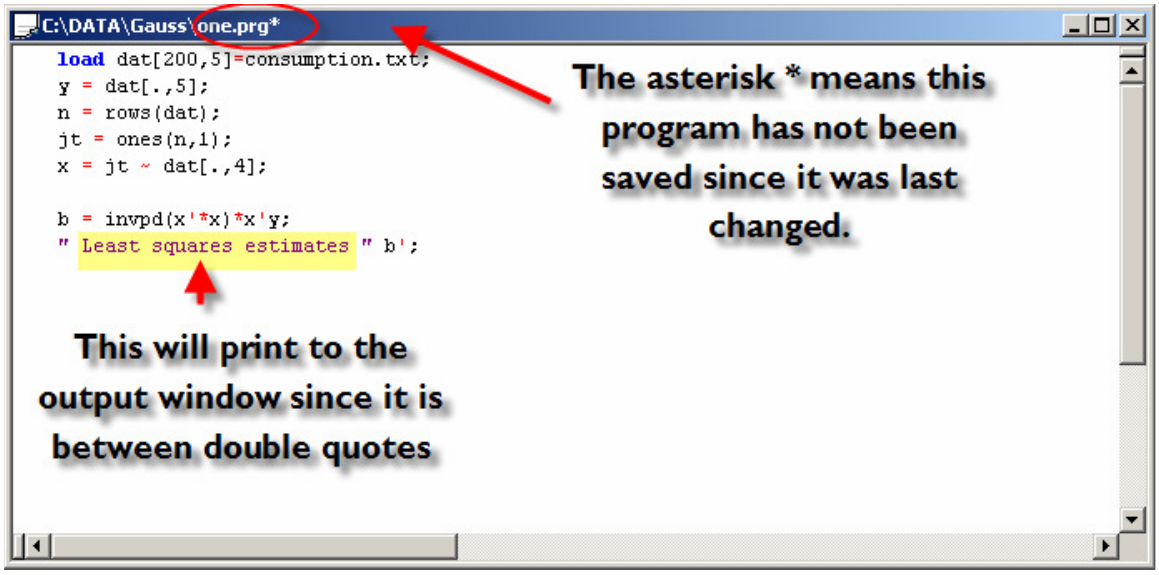
Open a new file called **one.prg** by typing in **edit one.prg** at the command line

```
» edit one.prg;
```

You'll be presented with this dialog asking you whether you really want to create a new file called **one.prg** in your working directory. If **one.prg** already exists, it will simply open without this nag.



Click Yes and a new, empty program will open. Type in the program shown



Basically this program consists of statements we've already used. I created a separate variable for sample size called **n** using the **rows(dat)**. Also, I used a fancier, annotated statement to print the result, which is transposed. Gauss will print whatever you list between double quotation marks, in this case **Least Squares estimates**.

Save the program using **Crtl+S** or from the pull-down menu (**File>Save**). To run the program press **F6**, or **Run>Run Active File** from the pull-down menu.

The results look like this:

```
» run C:\DATA\Gauss\one.prg;
 Least squares estimates        6000.2561       0.86178847
```

To exit Gauss, save your programs or output if desired, and use **Ctrl+Q** or **File>Exit**. You are now a Gauss professional!