

Introduction to Matlab

Lesson 04 — Importing, manipulating, and fitting Data

Rafael Serrano-Quintero

Department of Economics
University of Barcelona

Importing and Manipulating Data

Importing and Manipulating Data

- Download the data from the World Bank in CSV.
 - GDP per capita (constant 2010 US\$)
 - Urban population as a % of total population
- Rename to `urban_pop.csv` and `real_gdp_percapita.csv`
- Forget by now about the metadata files.
- Download in a subdirectory in the working directory as `working_directory/data/`

Importing and Manipulating Data

- Since the data files have *headers* or column names, we explicitly tell the command that.
- Let's import the data as a table using `readtable()`

```
1 close all
2 clear
3 clc
4
5 % Importing data from CSV file
6 gdp = readtable('./data/real_gdp_percapita.csv', '
    ReadVariableNames', true);
7 pop = readtable('./data/urban_pop.csv', 'ReadVariableNames', true
    );
```

Importing and Manipulating Data

- **Tables in Matlab** are a data type created to store data that is column oriented.
- They are not like arrays, we need to extract the numeric columns.
- Let's find out if country codes are the same for both variables.
- To make manipulation easier, transform to a **cell array**.

```
1 % Extract CountryCode for both files
2 ccode_gdp = table2cell(gdp(:,2));
3 ccode_pop = table2cell(pop(:,2));
```

Importing and Manipulating Data

To check if country codes coincide:

1. Compare one by one.

2. Check if all elements are true

```
1 % Compare strings one by one
2 compare_ccode = strcmp(ccode_gdp, ccode_pop);
3 all_true = all(compare_ccode);
4 disp(all_true)
```

Since they are, we can safely merge the two variables.

Importing and Manipulating Data

- Extract the numeric values of GDP and urban population.
- We use `table2array()` to convert into a matrix.
- We transpose so that each row is a year and each column a country.

```
1 % Extract GDP per capita and urban population as a matrices
2 gdp_pc = table2array(gdp(:,5:end))';
3 pop_ub = table2array(pop(:,5:end))';
```

Importing and Manipulating Data

- Explore the rough relationship between $\log(GDP)$ and urban population.
- To do a pooled scatter plot use the `(:)` operator. This converts a matrix into a vector.

```
1 % Explore the relationship between GDPpc and Urban population
2 figure
3 scatter(log(gdp_pc(:)),pop_ub(:),...
4         'filled','MarkerFaceAlpha',0.25)
5 hold on
6 lsline % add a least squares line
7
8 % Rough correlation
9 disp(corrcoef(log(gdp_pc(:)),pop_ub(:),'Rows','complete'))
```


Importing and Manipulating Data

- The correlation for all countries is 0.8290. Let's explore the correlation for each country individually.

```
1 % Correlation by country
2 [T, N] = size(gdp_pc); % Time periods (T) and number of
   countries (N)
3
4 corr_coefs = zeros(N,1);
5 for n = 1:N
6     tmp = corrcoef(log(gdp_pc(:,n)), pop_ub(:,n), 'Rows', '
           Complete');
7     corr_coefs(n,1) = tmp(1,2);
8 end
9
10 mean(corr_coefs, 'omitnan')
```

- The average is 0.54 and the standard deviation 0.60.

Importing and Manipulating Data

Exercise 1

Plot the evolution over time for a particular country. Plot the two series in the same graph with two different y -axes. Also, make sure you include the years in the x -axis.

Importing and Manipulating Data

How has it been for India?

```
1 % Plot the evolution for  
   India  
2 years = 1960:1960+T-1;  
3 india = strcmp(ccode_gdp,'  
   IND');  
4 gdp_india = gdp_pc(:,india);  
5 urb_india = pop_ub(:,india);  
6  
7 figure  
8 plot(years,gdp_india,'-o')  
9 hold on  
10 yyaxis right  
11 plot(years,urb_india,'-s')
```

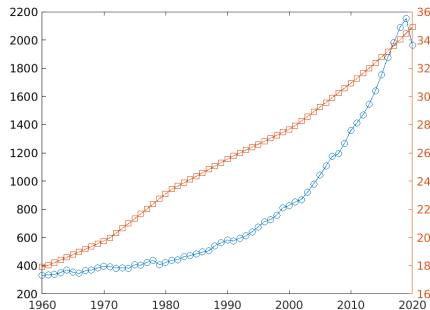


Figure 1: Real GDPpc and Urban Pop

Fitting Data

Linear Regression

Linear Regression

- Let's estimate a linear regression for the relationship between GDP per capita and urban population with a time trend.

$$\log(GDPpc_{t,i}) = \beta_0 + \beta_1 urban_{t,i} + \beta_2 time_t + \varepsilon_{t,i}$$

- The OLS estimator is $\hat{\beta} = (X'X)^{-1}X'y$ where

$$X = \begin{pmatrix} 1 & urban_{1960,1} & 1960 \\ \vdots & \vdots & \vdots \\ 1 & urban_{2020,N} & 2020 \end{pmatrix} \quad y = \begin{pmatrix} \log(GDPpc_{1960,1}) \\ \vdots \\ \log(GDPpc_{2020,N}) \end{pmatrix}$$

Linear Regression

Necessary steps:

1. Reshape matrices into vectors of size $T \times N$ and create $time_t$.

```
1 % Reshape GDPpc matrix into a vector
2 y = reshape(gdp_pc,[T*N,1]);
3 y = log(y);      % Recall our dependent variable is in logs!
4
5 % Reshape pop_ub in the same way
6 urb_vect = reshape(pop_ub,[T*N,1]);
7
8 % Create the time trend
9 years_mat = repmat(years,N,1);
10 years_mat = years_mat';
11 years_vec = reshape(years_mat,[T*N,1]);
```

Linear Regression

Necessary steps:

2. Remove missing values from all variables.

```
1 % Remove missing values from both variables
2 miss_y = isnan(y);
3 miss_u = isnan(urb_vect);
4 tot_miss = logical(miss_y + miss_u);
5
6 y_clean = y(~tot_miss);
7 u_clean = urb_vect(~tot_miss);
8 years_vec = years_vec(~tot_miss);
```


Linear Regression

Necessary steps:

3. Construct X and compute $\hat{\beta} = (X'X)^{-1}X'y$

```
1 % Create matrix X
2 X = [ones(size(u_clean)), u_clean, years_vec];
3
4 % Estimate bhat
5 bhat = ((X')*X)\((X')*y_clean);
```

$$\hat{\beta} = \begin{pmatrix} 11.0201 \\ 0.0509 \\ -0.0027 \end{pmatrix}$$

So, $\Delta_{urban} = 1$ percentage point is associated with $\Delta GDP_{pc} = 5.09\%$. **NOT A CAUSAL RELATIONSHIP!!!**

Linear Regression — Other Ways

- What if we want standard errors, p-values ...?
- Fortunately, not by hand! (`fitlm` and `LinearModel`)
- These are found within the `Statistics and Machine Learning Toolbox`.
- Alternatives:
 - `fit` from the `Curve Fitting Toolbox`
 - `regress` from the `Statistics and Machine Learning Toolbox`
 - `Spatial Econometrics Toolbox` by James P. LeSage. Much more recommended for serious data work are `R`, `Stata`, `Python`

Linear Regression — fitlm and LinearModel

- fitlm creates a LinearModel object, estimates the model, and produces results for that particular model object.
- fitlm accepts data in table format or numeric array format. We will stick to numeric array.
- By default it includes a constant and deals with missing values.

```
1 % Control variables
2 Xfitlm = [urb_vect, years_vec];
3
4 % Model object
5 linmodel = fitlm(Xfitlm,y);
6 disp(linmodel)
```

Linear Regression — fitlm and LinearModel

The default output of the fitlm model includes

- Coefficients, standard errors, t —statistics, p —values ...
- Reassuringly, we get the same coefficients as before.

```
1 >> disp(linmodel)
2
3 Linear regression model:
4   y ~ 1 + x1 + x2
5
6 Estimated Coefficients:
7           Estimate          SE          tStat          pValue
8           - - - - -
9
10  (Intercept)          11.02          0.92273          11.943          1.0777e-32
11    x1              0.050911          0.00032252          157.86              0
12    x2             -0.0026586          0.0004651           -5.7161          1.1155e-08
13
14
15 Number of observations: 12131, Error degrees of freedom: 12128
16 Root Mean Squared Error: 0.821
17 R-squared: 0.688, Adjusted R-Squared: 0.688
18 F-statistic vs. constant model: 1.34e+04, p-value = 0
```

Linear Regression — `fitlm` and `LinearModel`

We can perform some model diagnostics

- `plotResiduals(linmodel)` to show a histogram of residuals.
- `plotResiduals(linmodel, 'fitted')` residuals vs fitted values.
- `plotEffects(linmodel)` to show size of coefficients.

We can access directly several derived objects

- `linmodel.LogLikelihood` the log-likelihood.
- `linmodel.Residuals` the residuals of the fitted model.
- `linmodel.Coefficients` the value of estimated coefficients.

Polynomial Fit and Polynomial Evaluation

Polynomial Fit

Theorem 1 (Weierstrass' Approximation Theorem)

Let $f(x)$ be a continuous real-valued function defined on $[a, b]$. Then, given ε we can find a polynomial $p(x)$ such that

$$\sup |f(x) - p(x)| < \varepsilon$$

Informally: any continuous function on $[a, b]$ can be approximated well by a polynomial function of a sufficient degree.

Polynomial Fit

- Suppose, we have the following relationship between y and x

$$y = \sin(5x^2 + \pi)$$

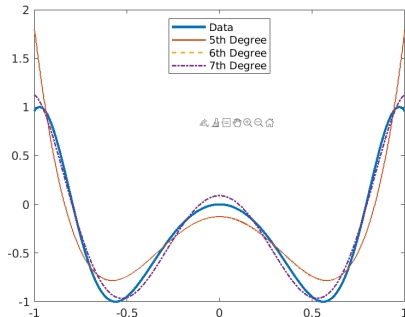
with $x \in \mathbb{R}$. Note that the function is continuous in \mathbb{R} .

- But we do not know the relationship (or it is very expensive to compute). Let us approximate by an n -degree polynomial.
- This is done with the commands `polyfit` and `polyval` to fit and evaluate polynomials respectively. (How exactly are we fitting...?)

Polynomial Fit

- Let's fit the data using 5th, 6th, and 7th degree polynomials.
- Evaluate them and plot them.

```
1 % Simulate variables and  
  their relationship  
2 x = linspace(-1,1,1000);  
3 y = sin(5.*x.^2 + pi);  
4  
5 % Fit 5,6,7th degree  
  polynomials  
6 p5 = polyfit(x,y,5);  
7 p6 = polyfit(x,y,6);  
8 p7 = polyfit(x,y,7);
```



Polynomial Fit

Exercise 2

Generate a vector x in the interval $[-4, 4]$ with 1000 points (`linspace`). Now, create the variable $y = x^2 + \varepsilon$ where ε is white noise (`randn`) with standard deviation 1.5. Fit a second degree polynomial and plot the simulated data and the fitted polynomial on the same plot.

Polynomial Fit

Fitting data with noise.

```
1 X = linspace(-4,4,1000);  
2 Y = X.^2 + 1.5.*randn(1,length(X));  
3 p2 = polyfit(X,Y,2);  
4  
5 figure  
6 scatter(X,Y)  
7 hold on  
8 plot(X,polyval(p2,X),'-','LineWidth',1.35)  
9 title('Polynomial Fit')
```

Nonlinear Least Squares

Nonlinear Least Squares

- Consider a set of data points $(x_1, y_1), \dots, (x_N, y_N)$ and a function $y = f(x, \beta)$ depending on unknown parameters $\beta = (\beta_1, \dots, \beta_m)$ where $N > m$.
- The Nonlinear Least Squares estimator is the set of coefficients β such that

$$\min_{\beta} \sum_{i=1}^N (y_i - f(x_i, \beta))^2$$

- `polyfit` is exactly doing that but when $f(x, \beta)$ is a polynomial of degree n .
- We can extend that to **any** nonlinear function easily.

Nonlinear Least Squares — Functions

- To perform NLLS Matlab provides several alternatives
- We will focus on `lsqcurvefit`. Other alternatives are
 - `lsqnonlin`
 - `nlinfit`
 - `fitnlm` (very similar to `fitlm`)
- `lsqcurvefit` and `lsqnonlin` are **equivalent** since they use the same algorithm.
- We will focus on `lsqcurvefit` provides a convenient way of writing the problem and advances syntax for the lecture on optimization.

Nonlinear Least Squares — `lsqcurvefit`

- Suppose we want to fit a data generating process of the type

$$y = b_1 \exp(b_2 x) + \varepsilon \text{ where } \varepsilon \sim \mathcal{N}(0, 0.15) \text{ and } x \in [0, 5]$$

- Suppose we have N pairs of observations (x_i, y_i) and we want to find parameters b_1 and b_2 that best fit the data.
- Let's simulate the data first

```
1 %Simulate data
2 N = 500;
3 b1 = 2;
4 b2 = -1.5;
5 x = linspace(0,5,N);
6 y = b1*exp(b2*x)+0.15*randn(1,N);
```

Nonlinear Least Squares — lsqcurvefit

- Once the data is simulated, we create an anonymous function of the form

$$y = b_1 \exp(b_2 x)$$

that takes as arguments both the coefficients b_1 and b_2 and the data, x .

```
1 % Declare anonymous function with our model
2 func_fit = @(b,xdata)(b(1)*exp(b(2)*xdata));
```

- Now we can fit the function. However, we **need to provide initial values!**

```
1 % Fit
2 b00 = [1, -1]; % Initial condition
3 [bhat,resnorm,res,exitflag,output] = lsqcurvefit(func_fit,b00,x
    ,y);
```


Nonlinear Least Squares — `lsqcurvefit`

Table 1: NLLS Fit with `lsqcurvefit`

	Estimate	Std Error
b_1	1.9982	0.0362
b_2	-1.5579	0.0402

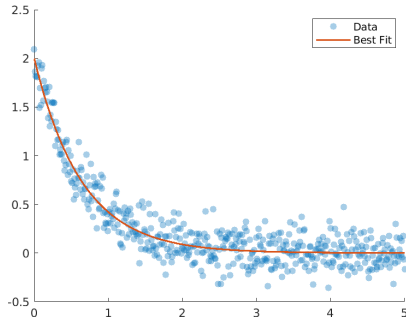


Figure 2: Performance of `lsqcurvefit`