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 2015 US$)
- Urban population as a % of total population
- We have this data in two CSV files under `code/class-04/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()`

```
close all
clear
clc

% Importing data from CSV file
gdp = readtable('./data/real_gdp_percapita.csv','ReadVariableNames',true);
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.

```
% Extract CountryCode for both files
ccode_gdp = table2cell(gdp(:,4));
ccode_pop = table2cell(pop(:,4));
```

Importing and Manipulating Data

To check if country codes coincide:

1. Compare one by one.

2. Check if all elements are `true`

```
% Compare strings one by one
compare_ccode = strcmp(ccode_gdp,ccode_pop);
all_true = all(compare_ccode);
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.

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

```
% Explore the relationship between GDPpc and Urban population
figure
scatter(log(gdp_pc(:)),pop_ub(:),...
      'filled','MarkerFaceAlpha',0.25)
hold on
lsline % add a least squares line

% Rough correlation
disp(corrcoef(log(gdp_pc(:)),pop_ub(:),'Rows','complete'))
```

Importing and Manipulating Data

- The correlation for all countries is 0.8214. Let's explore the correlation for each country individually.

```
% Correlation by country
[T, N] = size(gdp_pc); % Time periods (T) and number of
    countries (N)

corr_coefs = zeros(N,1);
for n = 1:N
    tmp = corrcoef(log(gdp_pc(:,n)),pop_ub(:,n),'Rows','
        Complete');
    corr_coefs(n,1) = tmp(1,2);
end

mean(corr_coefs,'omitnan')
```

- The average is 0.55 and the standard deviation 0.58.

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?

```
% Plot the evolution for India
years = 1960:1960+T-1;
india = strcmp(ccode_gdp,'IND');
gdp_india = gdp_pc(:,india);
urb_india = pop_ub(:,india);

figure
plot(years,gdp_india,'-o')
hold on
yyaxis right
plot(years,urb_india,'-s')
```

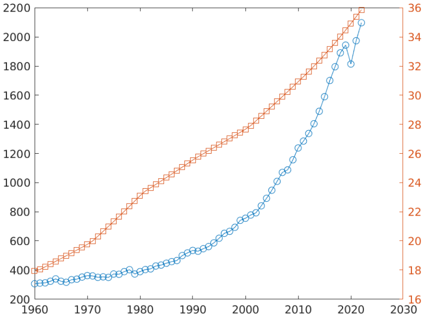


Figure 1: Real GDPpc and Urban Pop

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

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

Linear Regression

Necessary steps:

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

```
% Reshape GDPpc matrix into a vector
y = reshape(gdp_pc,[T*N,1]);
y = log(y);      % Recall our dependent variable is in logs!

% Reshape pop_ub in the same way
urb_vect = reshape(pop_ub,[T*N,1]);

% Create the time trend
years_mat = repmat(years,N,1);
years_mat = years_mat';
years_vec = reshape(years_mat,[T*N,1]);
```

Linear Regression

Necessary steps:

2. Remove missing values from all variables.

```
% Remove missing values from both variables
miss_y = isnan(y);
miss_u = isnan(urb_vect);
tot_miss = logical(miss_y + miss_u);

y_clean = y(~tot_miss);
u_clean = urb_vect(~tot_miss);
years_vec = years_vec(~tot_miss);
```

Linear Regression

Necessary steps:

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

```
% Create matrix X
X = [ones(size(u_clean)), u_clean, years_vec];

% Estimate bhat
bhat = ((X')*X)\((X')*y_clean);
```

$$\hat{\beta} = \begin{pmatrix} 10.4414 \\ 0.0495 \\ -0.0023 \end{pmatrix}$$

So, $\Delta urban = 1$ percentage point is associated with $\Delta GDPpc = 4.95\%$. **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.

```
% Control variables
Xfitlm = [urb_vect, years_vec];

% Model object
linmodel = fitlm(Xfitlm,y);
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.

```
>> disp(linmodel)

Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                    Estimate          SE          tStat        pValue
                   __________    __________    ________    __________
    (Intercept)        10.441       0.83872      12.449    2.1911e-35
    x1               0.049541    0.00030372      163.11             0
    x2             -0.0023286     0.0004228     -5.5076     3.704e-08

Number of observations: 13719, Error degrees of freedom: 13716
Root Mean Squared Error: 0.83
R-squared: 0.675,  Adjusted R-Squared: 0.675
F-statistic vs. constant model: 1.43e+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.

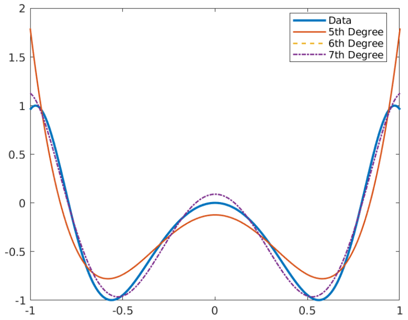
- This is done with the commands `polyfit` and `polyval` to fit and evaluate polynomials respectively. (How exactly are we fitting...?)

Polynomial Fit

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

```
% Simulate variables and their relationship
x = linspace(-1,1,1000);
y = sin(5.*x.^2 + pi);

% Fit 5,6,7th degree polynomials
p5 = polyfit(x,y,5);
p6 = polyfit(x,y,6);
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.

Nonlinear Least Squares

Nonlinear Least Squares

- Consider a set of data points $(x_1, y_1), \ldots, (x_N, y_N)$ and a function $y = f(x, \beta)$ depending on unknown parameters $\beta = (\beta_1, \ldots, \beta_m)$ 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

```
%Simulate data
N = 500;
b1 = 2;
b2 = -1.5;
x = linspace(0,5,N);
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.

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

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

```
% Fit
b00 = [1, -1]; % Initial condition
[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	2.0515	0.0362
b_2	-1.5294	0.0384

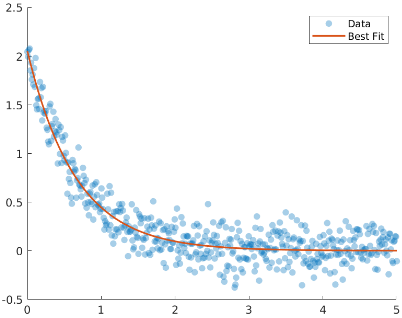


Figure 2: Performance of `lsqcurvefit`