Principles of Programming in Econometrics

Introduction, structure, and advanced programming techniques

Charles S. Bos

VU University Amsterdam
Tinbergen Institute

c.s.bos@vu.nl

August 2019 – Version Python
Exercises
Compilation: August 23, 2019

Afternoon session

Practical at
VU University
Main building, HG 0B08 (EOR/QRM), 0B16 (TI-MPhil)
13.30-16.00h

Topics:

- Checking variables and functions
- Implementing Backsubstitution
- Secret message (if time permits, should be easy)

Get started

- Log in using your vunet-ID (or use your laptop, with Anaconda installed)
- Create a directory for this course on the network drive, e.g. `h:\ppectr\`
- Unpack the files from `lists_py.zip` from Canvas into your `h:\ppectr\lists_py`
- Create a directory for this session, `h:\ppectr\pp0b`, and within it one for the first exercise, `h:\ppectr\pp0b\assign`
- Copy a version of `h:\ppectr\lists\empty.py` to e.g. `h:\ppectr\pp0b\assign\vars.py`, and edit it to ... start testing variables

Get Started: `vars.py`

Open (your newly created) `vars.py` from Spyder, such that you can

- Assign/print a string

  Hint: `sS= 'Hello'; print ("My string is sS=", sS)`
- Assign/print a double/integer/boolean
- Assign/print a one/two-dimensional list
- Assign the list to a numpy ndarray Hint: mX= np.array(lX)
- Assign/print a function

PS: You might find it easy to first try things in IPython, before typing the commands into the program `vars.py`

Get started: `func.py`

Edit a new file `func.py`, such that you can start testing functions:

- Create a function to print an argument

  Hint: `sS= 'Hello'; PrintMe (sS)`
- Create a function to assign one value through a `return` statement
- Same thing, with two values: Can you ‘catch’ the two values from the calling function?

Get started: `argument.py`

Edit a new file `argument.py`, such that you can start testing functions changing arguments.
Create a main and a function.

1. Pass a double to the function, return the square

   Hint: `return math.pow(dX, 2)`. What is the difference with `return dX ** 2`?
2. Try to change the argument *itself* within the function, squaring it. Does this work? (Answer: No... Why not?)
3. Pass a list with a single double (`lX= [5.5]`) to the function, pass the square back through changing the argument.

   Hint: `lX= [5.5]; SquareMe(lX)`

Get started: `argument.py` II

4. Pass a string, e.g. `sX= 'Aargus';` to the function. Can you change only the "g" to a "h"? (Answer: No... Why not?)
5. Then pass a list with a single string `lsX= ['Aargus']` to the function. Now you should be able to change letter `lsX[0][3]`, how?
6. Pass the list `lX= ['Aargus', 5, [2.4, 4.6]]` to the function, change the 5 to a 7, the 4.6 to its square, and the "g" to a "h".

Ensure you *fully* understand the list/mutable thing here... Talk to the tutor if not.

BS: Print a matrix

Write a Python program which

- contains all necessary explanations
- declares a matrix and a vector, giving them the values

$$A = \begin{pmatrix} 6.0 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{pmatrix}, \qquad b = \begin{pmatrix} 16.0 \\ -6 \\ -9 \\ -3 \end{pmatrix}$$

- prints them, with output on screen as to which is which
- prints the maximum element of `A`, and the minimum of `b`.
- you save as `bs0.py`.

BS: Backsubstitution

Solve the system $Ax = b$ for the matrices you defined before. As a hint, the way to solve it is

$$x_n = b_n/a_{nn}, \quad x_i = \left(b_i - \sum_{j>i} a_{ij} x_j \right) /a_{ii}, \qquad i = n-1, .., 1$$

Think about it before you begin: It might be easier to first define

$$s = \sum_{j>i} a_{ij} x_j$$

and for all $x_n, \ldots, x_1$ use the same formula for solving.

BS: BS function

1. For this purpose, maybe start with a simple `bs1for.py` where you show you can count backwards using a for-loop.
2. Initialise x as a vector of the correct size of zeros (see `np.zeros((iR, iC))`, note the *tuple* in parentheses indicating the size).
   Write `bs2solve.py`, showing the solution for x. How can you/the program check that your solution is correct?
3. Take the program `e0_elim.py`, and add a function `vX= Backsubstition(mA, vB)`. Make sure the function is working correctly. How can you test? Save as `bs3elim.py`.

BS: Python elements to use?

Useful might be

- `iK= mA.shape[0]`: Never use '4', but read off the row-size of `mA` instead
- Matrix multiplication using `NumPy` arrays is performed using e.g. `mA @ vX`
- Calculate s smartly. I can see four different options, where the simplest uses a simple loop. What are the options using matrix multiplications?
- Print your outcome in matrix format using a DataFrame,
  `import pandas as pd; print (pd.DataFrame(mRes, columns=["A", "B", "C"]))`

Secret

(on purpose, exercise is a bit confuse...)

> *You are surrounded by spies, and you want to pass the secret message "This is a secret message" to your compatriots. The deal you made with them is that you would add 3 to the ASCII code of each letter, so that 'A' becomes 'D'. What is the message you send to them?*

Secret inputs

Inputs:

- `empty.py` (copy to your personal directory, give it another name)
- Check out a `for` loop (details will follow):

```
for <element> in <some list/array/string>:
```

- Look up manual at
  `https://docs.python.org/3.7/`
  for functions `ord()` and `chr()`
- Strings can be concatenated using the + symbol, `sS= 'a'+'b'`

Secret outputs

- In groups of two (optionally)
- Keep a log-file: What are you trying? (not optionally...)
- Intermediate versions of your programs, every serious change, save a file with extension indicating the time (for instance `myfile_hhmm.py`).
- Clean out final version

Biggest mistake: Try to work on the exercise at once...

Big bonuspoints: Try to think of simpler exercises, how to test tiny steps first, eventually combining to the outcome

Biggest bonuspoints: Clean log-file, purposeful search of info, small tests (with corresponding tiny programs) and clean final version with sufficient (not too much, not too little either) commenting.

Hand-in

Handin for today:

- Nothing...

Discuss results with tutors, make sure you understand what you do/do not understand!

Afternoon session

Practical at
VU University
Main building, HG 0B08 (EOR/QRM), 0B16 (TI-MPhil)
13.30-16.00h

Topics:

- Regression: Simulate data
- Regression: Estimate model

Exercise: OlsGen

Target of this exercise is to set up a program for a slightly larger task. The task itself is not hard, but the idea is to do it in a structured, extensible way.

Target:

- Generate 20 observations from $y = X\beta + \sigma\epsilon$, with $\beta = [1; 2; 3], X = [1\ u_1\ u_2], u_i \sim U(0, 1), \epsilon \sim \mathcal{N}(0, 1), \sigma = 0.25$
- Estimate OLS on the model. Initially, estimate only $\hat{\beta} = (X'X)^{-1}X'y$
- Provide interesting output

Exercise: OlsGen II

Step 1, analyse the exercise:

1. What variables do I need for initial settings (put them close together, as magic numbers, in `main()`);
2. what separate tasks do I have;
3. hence, what routines could I use;
4. what are inputs and outputs to those routines;
5. what is the final output.

Write, *on paper*, an indication of the plan for your program!
Check the plan, *and especially the magic numbers*, with a TA.

Exercise: OlsGen III

Step 2, start the programming, but in steps:

1. First write `olsgen0.py`, containing only the outline of the program,
2. then `olsgen1.py` which does the initialisation,
3. when it works move to `olsgen2.py` which takes an extra step, etc.
4. ...

Exercise: OlsGen IV

For the initialisation, you will need commands like

- `np.shape()`, `np.size()` for checking how large β is;
- `np.random.rand()` for draws from the uniform random distribution;
- `np.random.randn()` for draws from the $\mathcal{N}(0,1)$ distribution. How do you transform to get variance σ^2?
- Matrix multiplication `mX @ vB`: What shape would the result be, if X is an $(n \times k)$ matrix, and β an $(k \times 1)$? What if β is a one dimensional vector, of shape $(k,)$?

Exercise: OlsGen V

In Econometrics, the basic estimation method is indeed OLS. Its main equation comes from

$$\begin{aligned} y &= X\beta + u, \\ \Leftrightarrow X'y &= X'X\beta + X'u \\ \Leftrightarrow \frac{1}{n}X'y &= \frac{1}{n}X'X\beta + \frac{1}{n}X'u \equiv \frac{1}{n}X'X\hat\beta + 0 \\ \Leftrightarrow \hat\beta &= \left(\frac{1}{n}X'X\right)^{-1}\frac{1}{n}X'y = (X'X)^{-1}X'y \end{aligned}$$

where the switch to $\hat\beta$ follows from the assumption that X and u are unrelated, hence $\frac{1}{n}X'u \approx 0$ when $n \to \infty$.

Exercise: OlsGen VI

To estimate β in your program, you have (at least) three options:

1. using direct matrix multiplication;
2. using your elimination + backsubstitution, noting that

$$b \equiv X'y = X'X\hat{\beta} \equiv Ax.$$

   Of course, use the routines from the `elim0` exercise, and yesterdays *backsubstitution*, here;
3. using a prepackaged function, (see `np.linalg.lstsq()`).

Write three routines `EstimateMM()`, `EstimateEB()`, `EstimatePF()`, which implement the three options, and check that the results indeed are the same.

Exercise: OlsGen VII

Eventually we might also be interested in

$$e = y - X\hat{\beta}, \qquad \hat{\sigma}^2 = \frac{1}{n-k} e'e = \frac{1}{n-k} \sum e_i^2,$$

$$\hat{\Sigma} = \hat{\sigma}^2 (X'X)^{-1}, \qquad s(\hat{\beta}) = \text{diag}(\hat{\Sigma})^{1/2},$$

with n and k the size of y and β, respectively. Also the t-statistics, $t = \hat{\beta}_i / s(\hat{\beta}_i)$, could be of interest.

- Build a version of your program which also computes $s(\hat{\beta})$ and the t-value, and outputs this together with $\hat{\beta}$.
- Try to obtain a nice output routine, using formatted printing.
  Hint:

```
mRes= np.hstack([vB, vS, vT])      # Or: mRes= np.vstack([vB, vS, vT]).T ?
print ('Estimation results: ')
print (pd.DataFrame(mRes, columns=['b', 's(b)', 't']))
```

Exercise: OlsGen VI

Useful tricks:

- Use `dSSR= vE.T@vE` for computing the sum of squared residuals $e'e$
- To get a list with the (square roots of) the diagonal elements of the covariance matrix Σ, take a *list comprehension*, or (simpler), use `np.diagonal()`
- Other functions you might need: `np.linalg.inv()`, `np.linalg.lstsq()`, `np.sqrt()`.

Q, optional: The exercise effectively guides you to use two-dimensional vectors. Can you create a new version of your program where `vY, vB, vE` are one-dimensional vectors instead? What changes?

Afternoon session

Practical at
VU University
Main building, HG 0B08 (EOR/QRM), 0B16 (TI-MPhil)
13.30-16.00h

Topics:

- Cleaning OLS program
- Loops
- Bootstrap OLS estimation
- Handling data

Exercise: Fill

Target of this exercise is to get used to writing functions, to working with matrices and indexes in a smart manner

Goal:

Fill a matrix X such that

$$X_{ij} = i \times j, \qquad i = 1, \ldots, n, j = 1, \ldots, k$$

0. Create `mX` in `main()`, and fill it here as well
1. Work out a function `RetXij(iN, iK)`, which *returns* `mX`
2. Create a matrix of zeros in `main()`, pass it along to `FillXij(mX)`, and have it filled there
3. (extra) Create `mX` in `main()`, using a list comprehension. Can you get the final matrix in a single line?

Exercise: Fill II

Hints:

- You'll need the `zeros((iN, iK))` function from `numpy`. Note that it needs an argument `shape`, which must be a `tuple` (as in `(iN, iK)`) of rows and columns, hence the double parentheses.
- A for-loop looks like

```
for i in range(iN):
    dosomething(i)
```

- In a function, you may indeed alter the *contents* of existing arrays (or lists, or other *mutable* types), but you cannot change the full variable. (Think hard, what does this indeed imply? Discuss with TAs?
- See the List Comprehensions. Remaining question: How can you get a double index? How can you move from a list to an array? How can you reshape into the right size?

Exercise: Fill III

Exercise:

- Download `fill.zip` from Canvas – Files
- Fill in `fill0.py`, . . . , `fill3.py`

and discuss doubts you have left...

OLSGen revisited

As a starter: Take a renewed look at your code of yesterday

- Do you indeed split out magic numbers, initialisation, estimation, output, in separate routines
- Do the routines have minimal input/output
- Is the output of the program clear
- Does the program have sufficient commenting?
- Do you consistently use Hungarian notation?
- Can you move the routines (except for `main()`) to `lib/incols.py`, for clarity? See also `stack/stackols3.py`.

Finish this, ask a TA to check, discuss what might be done better.

OLS SA0

The file `sa0_180827.csv` contains monthly data over the period 1920-2018 on the consumer price index of the US (source: http://data.bls.gov/timeseries/cuur0000sa0).
With this file

1. Read the data, splitting into a vector `vDT` with the time period as `datetime` object, and a vector with the price index, `vP`
2. Calculate the percentage inflation
$y_t = 100(\log(P_t) - \log(P_{t-1}))$
3. Use only data from 1958 onwards
4. Prepare regressors X, containing a constant, 11 dummies for months Jan-Nov, and dummies taking on the value 1 from date 1973:7, 1976:7, 1979:1, 1982:7 resp. 1990:1 onwards.
5. Run a regression of y on X
6. Plot the inflation y_t together with the prediction $\hat{y}_t = X_t\hat{\beta}$ against time.

OLS SA0 II

As always:

- Think hard on division of tasks in smaller steps
- Work in groups of two; use division in smaller steps to try out things separately
- How do you organize data?

OLS SA0 output

```
OLS results over 727 observations, average y= 0.299731:
          BetaHat
Const    0.287291
M2       0.052188
M3       0.088552
M4       0.042134
M5      -0.023780
M6       0.025291
M7      -0.081820
M8      -0.086803
M9      -0.018356
M10     -0.101586
M11     -0.262592
M12     -0.286661
1973/7   0.463860
1976/7  -0.094496
1979/1   0.241878
1982/7  -0.546886
1990/1  -0.096746
```

OLS SA0 output II

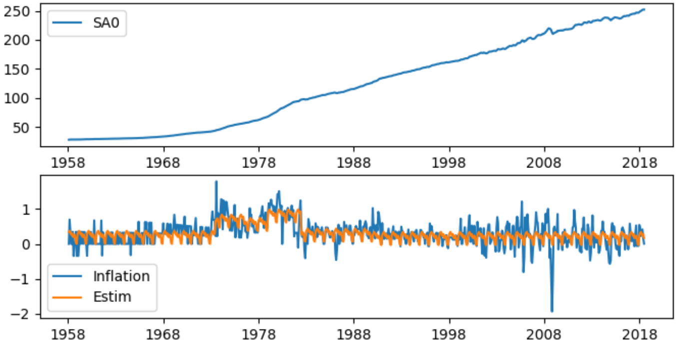

Figure: US Core inflation and prediction, 1958-2018

OLS SA0 hints

Some hints:

- Read the `csv` file into a Pandas DataFrame, with `pd.read_csv()`
- The column `vPer= df["Period"].values` then contains strings, in format `"1920/1"`
- Those strings can be pushed into `datetime` format, using pd.to_datetime(vPer)
- The advantage of the datetime format, is that you can compare a date-time with a string, `vI= vDT >= "1958"`, resulting in an vector of booleans
- You can then index another vector by these booleans, to extract a subset of a vector/matrix, see topic **Boolean index**.
- ...

OLS SA0 hints II

Some further hints:

- ...
- Or you can selectively set ones, `vD[vI]= 1`, to create a set of dummies
- For seasonal dummies, indexing with a step may be convenient. E.g. start with a vector of zeros, then fill `vD[i1::iSeas]= 1` every `iSeas`'th element with a one, starting at period `i1`
- You can join matrices together using `np.hstack([m1, m2])`, which horizontally concatenates the matrices `m1, m2` in the list `[m1, m2]`.
- Use `np.linalg.lstsq(mX, vY, rcond=None)` for OLS (or some other option)

Afternoon session

Practical at
VU University
Main building, HG 0B08 (EOR/QRM), 0B16 (TI-MPhil)
13.30-16.00h

Topics:

- Regression: Maximize likelihood
- GARCH-M: Intro and likelihood

ML estimation of regression

Take the regression model,

$$y = X\beta + \epsilon \qquad\qquad \epsilon \sim \mathcal{N}(0, \sigma^2 I).$$

The likelihood of an observation of the data, for a specific vector of parameters $\theta = (\sigma, \beta)$, is

$$e_t \equiv y_t - X_t\beta$$
$$l(y_t; X_t, \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{e_t^2}{2\sigma^2}\right),$$

or in logarithms

$$\log l(y_t; X_t, \theta) = -\frac{1}{2}\left(\log 2\pi + \log\sigma^2 + \frac{e_t^2}{\sigma^2}\right).$$

ML estimation of regression II

The loglikelihood of all observations is

$$\log l(Y; X, \theta) = \sum \log l(y_t; X_t, \theta).$$

Theory (to be explored in later courses) describes that

$$\hat\theta = \text{argmax}_\theta \log l(Y; X, \theta),$$

$$\Sigma(\hat\theta) = \left(-H(\hat\theta)\right)^{-1} \qquad H(\hat\theta) = \left.\frac{\partial^2 \log l(Y; X, \theta)}{\partial\theta\partial\theta'}\right|_{\theta=\hat\theta}$$

are the *Maximum Likelihood* estimators of the model at hand, the covariance matrix (if the model is correctly specified).
Work on this in steps...

ML Estimation: Steps

Perform, in steps, for instance

1. Prepare data, simulate as before
2. Get the outline of your loglikelihood function. Call it from main, with a valid vector of parameters, and set the likelihood value equal to the average of your y's.
3. Extract β and σ from the vector of parameters. Print them separately from the loglikelihood function.
4. Check the value of σ. If negative, maybe set `LL=-math.inf`, and get out?
5. Construct a vector `vLL` of $\log l(y_t; X_t, \theta)$'s. Does this work?

ML Estimation: Steps II

...

6. Construct full loglikelihood function. Does the value seem 'logical'?
7. Write a wrapper function for `minimize`, where the wrapper function will return the *negative average* loglikelihood
8. Run `minimize()`. What is the result `res`? Can you extract the parameters? How do the parameters relate to the OLS estimators?

ML Estimation: Steps III

...

6. Now combine your code with the SA0 data of yesterday: Can you obtain the same results as OLS, when linking inflation to your constant, seasonal dummies, and step functions?

ML: Standard errors

For the standard errors, you had to find

$$\Sigma(\hat{\theta}) = -H(\hat{\theta})^{-1}$$

$$H(\hat{\theta}) = \left. \frac{\delta^2 l(Y;\theta)}{\delta\theta\delta\theta'} \right|_{\theta=\hat{\theta}}$$

Some standard code could look like

```
res= opt.minimize(AvgNLnLRegrXY, vP0, args=(vY, mX), method="BFGS")
vP= np.copy(res.x)
mH= hessian_2sided(AvgNLnLRegrXY, vP, vY, mX)
mS2= np.linalg.inv(mH)/iN
vS= np.sqrt(np.diag(mS2))
```

9. Get the standard errors with it. How do they change if you only use $N = 10$ observations?
10. Beautify the output: Get a nice print with the maximum likelihood you find, the type of convergence, the parameters, standard errors and t-values

ML estimation GARCH-M

Extend the model to

$$\begin{aligned} y_t &= X_t\beta + a_t & a_t &\sim \mathcal{N}(0, \sigma_t^2), \\ \sigma_{t+1}^2 &= \omega + \alpha a_t^2 + \delta\sigma_t^2, & t &= 1, \ldots, T-1, \\ \sigma_1^2 &\equiv \frac{\omega}{1-\alpha-\delta}. \end{aligned}$$

Note that loglikelihood now changes to

$$\log l(Y; X, \theta) = \sum \log l(y_t; X_t, \theta) = -\frac{1}{2}\sum\left(\log 2\pi + \log \sigma_t^2 + \frac{a_t^2}{\sigma_t^2}\right).$$

ML estimation GARCH-M

Possible steps:

1. Generate data (y_t, X_t, σ_t^2) from the GARCH-M model, using e.g. $\theta = (1, .05, .05, .9)$, using a single constant in X.
2. Create a function `GetGARCH()`, which constructs the vector of variances, given the parameters $\theta = (\beta', \omega, \alpha, \delta)'$ and the data (y, X). Can it reconstruct (exactly) the `vS2` that was generated?
3. Build a new `AvgLnLiklGARCHM()`, using old code for the regression, and your `GetGARCH()`, to construct `vLL` and the average loglikelihood.
4. Optimise... Maybe compare outcomes of optimisation of regression only, or of GARCH-M?
5. ...

ML estimation GARCH-M II

Possible steps:

5. Go back to SA0 data; make a plot of inflation, and of $\sigma_t, t =$ 1958:1–2017:7.
6. Extra: Compare the number of function evaluations needed for each standard model without GARCH, and for model with GARCH

Possible output

To be added...

Closing thoughts

And so, the course comes to an end...

Please

- keep concepts, principles of programming, in mind
- structure your programs wisely

On a voluntary basis:

- in groups of max 2
- before Monday September 30, 9.00AM
- hand in a solution to
  1. GARCH-ML problem (similar to OLS exercise, minor extensions)
  2. BinTree problem (relevant to QRM students, nice setting for others)

(see Canvas for details)