Using R and Excel

7 December 2015

Don’t use Excel for research

What spreadsheets are good for

- Easy data entry
- Many people are familiar with Excel, and the structure is intuitive (at least for data entry)
- Looking over data (if there aren't too many variables)
- ????

Problems with spreadsheets

- Not flexible in selecting data
- Calculations are opaque
- Loops and recursion are impossible (I think)
- Data and algorithms are part of the same data structure
  - Not obvious which cells are calculated and which were entered
  - Running the same algorithm on different data isn't easy.
- Hard to find and correct errors
- Can't do sophisticated statistical analysis

Reinhart-Rogoff: A case study in Excel

Growth in a Time of Debt Carmen M. Reinhart, Kenneth S. Rogoff

NBER Working Paper No. 15639
Issued in January 2010

NBER Program(s): IFM ME We study economic growth and inflation at different levels of government and external debt. Our analysis is based on new data on forty-four countries spanning about two hundred years. The dataset incorporates over 3,700 annual observations covering a wide range of political systems, institutions, exchange rate arrangements, and historic circumstances. Our main findings are: First, the relationship between government debt and real GDP growth is weak for debt/GDP ratios below a threshold of 90 percent of GDP. Above 90 percent, median growth rates fall by one percent, and average growth falls considerably more. We find that the threshold for public debt is similar in advanced and emerging economies. Second, emerging markets face lower thresholds for external debt (public and private)--which is usually denominated in a foreign currency. When external debt reaches 60 percent of GDP, annual growth declines by about two percent; for higher levels, growth rates are roughly cut in half. Third, there is no apparent contemporaneous link between inflation and public debt levels for the advanced countries as a group (some countries, such as the United States, have experienced higher inflation when debt/GDP is high.) The story is entirely different for emerging markets, where inflation rises sharply as debt increases.

Reinhart-Rogoff: A case study in Excel

- George Osborne, Feb 2010: 'As Ken Rogoff himself puts it, "there's no question that the most significant vulnerability as we emerge from recession is the soaring government debt. It's very likely that will trigger the next crisis as governments have been stretched so wide." The latest research suggests that once debt reaches more than about 90% of GDP the risks of a large negative impact on long term growth become highly significant… As I have made clear, our aim will be to eliminate the bulk of the structural current budget deficit over a Parliament.'

- Matt Yglesias: "At one point they set cell L51 equal to AVERAGE(L30:L44) when the correct procedure was AVERAGE(L30:L49). By typing wrong, they accidentally left Denmark, Canada, Belgium, Austria, and Australia out of the average. When you fix the Excel error, a -0.1 percent growth rate turns into 0.2 percent growth." (Based on work by Herndon, Ash, Pollin.)

- R-R reply: "the latest academic kerfuffle should not divert our attention…"

What to do if you have data in an Excel spreadsheet

1. Start Excel.
2. Export your spreadsheet as comma-separated (.csv).
3. Close Excel.
4. Load into R with `read.csv('filename.csv')`.
5. Or, use `read.xls('filename.xls')` (but I can't vouch for this.

Statistical computing with R

What's good about R?

- It's free.
- Open source.
- It's free (of commercial restrictions).
- Basic statistical applications are very straightforward.
- Full-function programming language, relatively easy to write and read
- Very easy to make simple default plots, but not too hard to exert fine control over the graphics.
- Flexible syntax.
- Huge community writing packages and keeping an eye on the basics.
- High-quality random number generators.

What's not so good about R?

- On-board documentation is generally mediocre.
- Slow, particularly for any application involving loops. Such programs need to be written in C++ and embedded in R.
- Not the best for numerical methods.
- Doesn't do algebraic manipulation or high-precision integer calculations.
- Packages are of mixed quality. Outside of core R, *caveat emptor.*

Finding help

- If you know what command you need, typing `?command` brings up a help page.
- If you know approximately what command, or a keyword, `??word` searches for help pages.
- Lots of free online help is available.
  - Short intro: http://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf
  - Short reference card: http://cran.r-project.org/doc/contrib/Short-refcard.pdf
  - Long reference: http://web.udl.es/Biomath/Bioestadistica/R/Manuals/r_in_a_nutshell.pdf
  - Online course (if you like watching video lectures): https://www.coursera.org/course/rprog
- Lots of non-free books are now available
  - General books, e.g., Venables and Ripley, Modern Applied Statistics with S-Plus; Crawley, The R Book and Statistics: An introduction using R.
  - Specific books, with titles like Statistical Method X with R, often published by CRC Press.

Interacting with R

- Console: Type here for immediate action. Output comes here.
- R script: Write programs, longer work that you want to save, modify, etc.
- Graphics windows are generated by R commands.
- Important shortcuts:
  - Up-arrow repeats previous line. Repeating goes back through your history.
  - Ctrl-Enter (Cmd-Enter on Mac) after selecting text in an R-script window runs the selected commands in the console.
- Integrated development environments (IDEs) such as RStudio (available free) help to organise the windows, scripts, variables, etc.

Basic arithmetic in R

```
> 2+3
[1] 5
> 2*3
[1] 6
> 2^3
[1] 8
> 7%%3
[1] 1
> 7%/%3
[1] 2
> x=27
> sqrt(x)
[1] 5.196152
> y<-sqrt(x)
> y^2
[1] 27
```

Boolean operators

```
> y<5
[1] FALSE
> y^2==27
[1] TRUE
> (x/y)==y
[1] TRUE
> (x/y)<=y
[1] TRUE
```

R works with vectors (like MATLAB)

```
> z=seq(0,1,.1)                    or   > z=(0:10)/10
> z
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> z*5
 [1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> z+(0:10)
 [1]  0.0  1.1  2.2  3.3  4.4  5.5  6.6  7.7  8.8  9.9 11.0
> z*(0:10)
 [1]  0.0  0.1  0.4  0.9  1.6  2.5  3.6  4.9  6.4  8.1 10.0
> z^2
 [1] 0.00 0.01 0.04 0.09 0.16 0.25 0.36 0.49 0.64 0.81 1.00
> z^z
 [1] 1.0000000 0.7943282 0.7247797 0.6968453 0.6931448 0.7071068
 [7] 0.7360219 0.7790559 0.8365116 0.9095326 1.0000000
> cos(z)
 [1] 1.0000000 0.9950042 0.9800666 0.9553365 0.9210610 0.8775826
 [7] 0.8253356 0.7648422 0.6967067 0.6216100 0.5403023
```

```
> z
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> z+(1:2)
 [1] 1.0 2.1 1.2 2.3 1.4 2.5 1.6 2.7 1.8 2.9 2.0
Warning message:
In z + (1:2) :
  longer object length is not a multiple of shorter object length
> (1:10)+(1:2)
 [1]  2  4  4  6  6  8  8 10 10 12
> z<.7
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
[11] FALSE
> z<-1
> z
[1] 1
> z< -1
[1] FALSE
> 2^(1:10)
 [1]    2    4    8   16   32   64  128  256  512 1024
```

called “recycling”

```
> z
[1] 1
> z< -1
[1] FALSE
> 2^(1:10)
 [1]    2    4    8   16   32   64  128  256  512 1024



> z=seq(0,1,.1)
> z==.5
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
> z=.5
> seq(0,1,.1)==.5
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
> seq(0,1,.1)=.5
Error in seq(0, 1, 0.1) = 0.5 :
  target of assignment expands to non-language object
```

Indexing

```
> x=c(1,-1,7,7,9,1,0)
> x[3]
[1] 7
> x[3:5]
[1] 7 7 9
> x[-5]
[1]  1 -1  7  7  1  0
> x[x>2]
[1] 7 7 9
> x[8]
[1] NA
> y=(1:3)*2
> x[y]
[1] -1  7  1
> x[y<=4]
[1]  1 -1  7  9  0
```

```
> names=c('Emily','Li Min','Pietro','Jose')
> sex=c('F','F','M','M')
> nation=c('UK','China','Italy','Mex')
> names[sex=='F']
[1] "Emily"  "Li Min"
> names[sex=='M'&!nation=='Mex']
[1] "Pietro"
> names[names>'K' | nation=='UK']
[1] "Emily"  "Li Min" "Pietro"
```

concatenation

```
> c(names,'Zarathustra')
[1] "Emily" "Li Min" "Pietro" "Jose" "Zarathustra"
```

Boolean indices recycle

Functions

- A function has
  - one or more inputs. inputs may be determined by
    - name
    - position
    - default
  - one output (that may have multiple parts)
  - effects such as
    - changing the state of a global variable
    - printing
    - plotting

Mathematical functions

```
> x=seq(0,1,.2)
# scalar functions
> mean(x)
[1] 0.5
> sum(x)
[1] 3
> sd(x)
[1] 0.3741657
> var(x)
[1] 0.14
> min(x)
[1] 0
> max(x)
[1] 1
> range(x)
[1] 0 1
```

```
#vector functions
> sin(x)
[1] 0.0000000 0.1986693 0.3894183 0.5646425 0.7173561 0.8414710
> cospi(x)
[1]  1.000000  0.809017  0.309017 -0.309017 -0.809017 -1.000000
> atan(x)
[1] 0.0000000 0.1973956 0.3805064 0.5404195 0.6747409 0.7853982
> exp(x)
[1] 1.000000 1.221403 1.491825 1.822119 2.225541 2.718282
> log2(x)
[1]  -Inf -2.3219281 -1.3219281 -0.7369656 -0.3219281  0.0000000
> cosh(x)
[1] 1.000000 1.020067 1.081072 1.185465 1.337435 1.543081
> factorial(8:10)
[1]   40320  362880 3628800
> lfactorial(8:10)
[1] 10.60460 12.80183 15.10441
> choose(3:7,3)
[1]  1  4 10 20 35
> round(lfactorial(8:10),2)
[1] 10.6 12.8 15.1
```

```
> sinpi(x)
[1] 0.0000000 0.5877853 0.9510565 0.9510565 0.5877853 0.0000000
> cospi(x)
[1]  1.000000  0.809017  0.309017 -0.309017 -0.809017 -1.000000
> pmax(sinpi(x),cospi(x))
[1] 1.0000000 0.8090170 0.9510565 0.9510565 0.5877853 0.0000000
> pmin(sinpi(x),cospi(x))
[1]  0.0000000  0.5877853  0.3090170 -0.3090170 -0.8090170 -1.0000000
> pmax(sinpi(x),.5)
[1] 0.5000000 0.5877853 0.9510565 0.9510565 0.5877853 0.5000000
> max(sinpi(x),cospi(x))
[1] 1
```

```
> help(seq)
```

seq {base} R Documentation

Sequence Generation

Description

Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

Usage

```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)

seq.int(from, to, by, length.out, along.with, ...)

seq_along(along.with)
seq_len(length.out)
```

Arguments

`...`	arguments passed to or from methods.
`from, to`	the starting and (maximal) end values of the sequence. Of length `1` unless just `from` is supplied as an unnamed argument.
`by`	number: increment of the sequence.
`length.out`	desired length of the sequence. A non-negative number, which for `seq` and `seq.int` will be rounded up if fractional.
`along.with`	take the length from the length of this argument.

```
> seq(from=0,to=1,by=.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5
       0.6 0.7 0.8 0.9 1.0
> seq(0,1)
[1] 0 1
```

```
> seq(0,1,length.out=6)
[1] 0.0 0.2 0.4 0.6 0.8 1.0
> seq(1,0,along.with=c(17,9,-4,0,NA))
[1] 1.00 0.75 0.50 0.25 0.00
```

```
> rep(5,3)
[1] 5 5 5
> rep(4:1,3)
 [1] 4 3 2 1 4 3 2 1 4 3 2 1
> rep(4:1,each=3)
 [1] 4 4 4 3 3 3 2 2 2 1 1 1
> rep(4:1,1:4)
 [1] 4 3 3 2 2 2 1 1 1 1
> rep(-1:3,length.out=10)
 [1] -1  0  1  2  3 -1  0  1  2  3
> rep(-1:2,length.out=10)
 [1] -1  0  1  2 -1  0  1  2 -1  0
```