

Principles of Programming in Econometrics

Introduction, structure, and advanced programming techniques

Charles S. Bos

Vrije Universiteit Amsterdam
Tinbergen Institute

`c.s.bos@vu.nl`

August 2020 – Version Python

Separate lecture slides

Compilation: August 6, 2020

Overview

Principles of Programming in Econometrics

D0: Syntax, example 2⁸

D1: Structure, scope

D2: Numerics, packages

D3: Optimisation, speed

Day 2: Numerics and flow

9.30 Numbers and representation

- ▶ Steps, flow and structure
- ▶ Floating point numbers
- ▶ Practical Do's and Don'ts
- ▶ Packages
- ▶ Graphics

13.30 Practical

- ▶ Cleaning OLS program
- ▶ Loops
- ▶ Bootstrap OLS estimation
- ▶ Handling data: Inflation

Precision

Not all numbers are made equal...

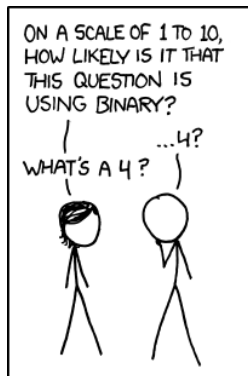
Example: What is $1/3 + 1/3 + 1/3 + \dots$?

Listing 1: precision/onethird.py

```
def main():  
    # Magic numbers  
    dD= 1/3  
  
    # Estimation  
    print ("i j sum diff");  
    dSum= 0.0  
    for i in range(10):  
        for j in range(3):  
            print (i, j, dSum, (dSum-i))  
            dSum+= dD          # Successively add a third
```

See outcome: It starts going wrong after 16 digits...

Decimal or Binary



1-to-10 (Source: XKCD, <http://xkcd.com/953/>)

Representation: Int

In many languages...

- ▶ Integers are represented exactly using 4 bytes/32 bits (or more, depending on system)
- ▶ 1 bit is for sign, usually 31 for number
- ▶ Hence range is $[-2^{31}, 2^{31}-1]$

Q: Afterwards, when $i = 2^{31}-1 + 1$, what happens?

Representation: Int

In many languages...

- ▶ Integers are represented exactly using 4 bytes/32 bits (or more, depending on system)
- ▶ 1 bit is for sign, usually 31 for number
- ▶ Hence range is $[-2^{31}, 2^{31}-1]$

Q: Afterwards, when $i = 2^{31}-1 + 1$, what happens? Answer:

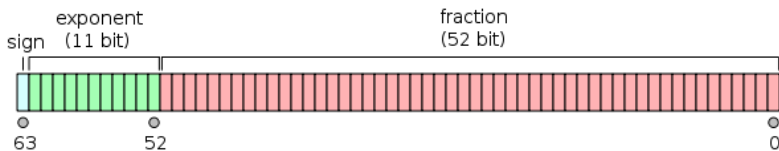
- ▶ Ox: Circles around to a negative integer, without warning...
- ▶ Matlab: Gets stuck at $2^{31}-1$...
- ▶ Python2: Uses 8 bytes, 64 bits. After $2^{63} - 1$, moves to *long* type, without limit
- ▶ Python3: *long* is the standard integer type, without any limit!

See `precision/intmax.py`

Representation: Double

- ▶ Doubles are represented in 64 bits. This gives a total of $2^{64} \approx 1.84467 \times 10^{19}$ different numbers that can be represented.

How?



Double floating point format (Graph source: Wikipedia)

Split double in

- ▶ Sign (one bit)
- ▶ Exponent (11 bits)
- ▶ Fraction or mantissa (52 bits)

Representation: Double II

$$x = \begin{cases} (-1)^{\text{sign}} \times 2^{\text{exponent}-1023} \times \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) & \text{Generally} \\ (-1)^{\text{sign}} \times 2^{1-1023} \times 0.\text{mantissa} & \text{if exp}=0\text{x}.000 \\ (-1)^{\text{sign}} \times \infty & \text{if exp}=0\text{x}.7\text{ff}, \text{ mant} = 0 \\ \text{NaN} & \text{if exp} = 0\text{x}.7\text{ff}, \text{ mant} \neq 0 \end{cases}$$

Note: Base-2 arithmetic

Sign	Expon	Mantissa	Result
0	0x.3ff	0000 0000 0000 ₁₆	$-1^0 \times 2^{(1023-1023)} \times 0.0$ = 0
0	0x.3ff	0000 0000 0001 ₁₆	$-1^0 \times 2^{(1023-1023)} \times 1.0000000000000000222$ = 1.0000000000000000222
0	0x.400	0000 0000 0000 ₁₆	$-1^0 \times 2^{(1024-1023)} \times 1.0$ = 2
0	0x.400	0000 0000 0001 ₁₆	$-1^0 \times 2^{(1024-1023)} \times 1.0000000000000000222$ = 2.0000000000000000444

Bit weird

Consequence: Addition

Let's work in Base-10 arithmetic, assuming 4 significant digits:

Sign	Exponent	Mantissa	Result	x
+	4	0.1234	0.1234×10^4	1234
+	3	0.5670	0.5670×10^3	567

What is the sum?

Consequence: Addition

Let's work in Base-10 arithmetic, assuming 4 significant digits:

Sign	Exponent	Mantissa	Result	x
+	4	0.1234	0.1234×10^4	1234
+	3	0.5670	0.5670×10^3	567

What is the sum?

Sign	Exponent	Mantissa	Result	x
+	4	0.1234	0.1234×10^4	1234
+	4	0.0567	0.0567×10^4	567
+	4	0.1801	0.1801×10^4	1801

Shift to same exponent, add mantissas, perfect

Consequence: Addition II

Let's use dissimilar numbers:

Sign	Exponent	Mantissa	Result	\times
+	4	0.1234	0.1234×10^4	1234
+	1	0.5670	0.5670×10^1	5.67

What is the sum?

Consequence: Addition II

Let's use dissimilar numbers:

Sign	Exponent	Mantissa	Result	x
+	4	0.1234	0.1234×10^4	1234
+	1	0.5670	0.5670×10^1	5.67

What is the sum?

Sign	Exponent	Mantissa	Result	x
+	4	0.1234	0.1234×10^4	1234
+	4	0.000567	0.0005×10^4	5
+	4	0.1239	0.1239×10^4	1239

Shift to same exponent, add mantissas, lose precision...

Further consequence:

Add numbers of similar size together, preferably!

In Python/Ox/C/Java/Matlab/Octave/Gauss: 16 digits (≈ 52 bits) available instead of 4 here

Consequence: Addition III

Check what happens in practice:

Listing 2: precision/accuracy.py

```
def main():  
    dA= 0.123456 * 10**0  
    dB= 0.471132 * 10**15  
    dC= -dB  
  
    print ("a: ", dA, ", b: ", dB, ", c: ", dC)  
    print ("a + b + c: ", dA+dB+dC)  
    print ("a + (b + c): ", dA+(dB+dC))  
    print ("(a + b) + c: ", (dA+dB)+dC)
```

Consequence: Addition III

Check what happens in practice:

Listing 3: precision/accuracy.py

```
def main():  
    dA= 0.123456 * 10**0  
    dB= 0.471132 * 10**15  
    dC= -dB  
  
    print ("a: ", dA, ", b: ", dB, ", c: ", dC)  
    print ("a + b + c: ", dA+dB+dC)  
    print ("a + (b + c): ", dA+(dB+dC))  
    print ("(a + b) + c: ", (dA+dB)+dC)
```

results in

```
a: 0.123456 , b: 471132000000000.0 , c: -471132000000000.0  
a + b + c: 0.125  
a + (b + c): 0.123456  
(a + b) + c: 0.125
```

Other hints

- ▶ Adding/subtracting tends to be better than multiplying
- ▶ Hence, log-likelihood $\sum \log \mathcal{L}_i$ is better than likelihood $\prod \mathcal{L}_i$
- ▶ Use true integers when possible
- ▶ Simplify your equations, minimize number of operations
- ▶ Don't do $x = \exp(\log(z))$ if you can escape it

Other hints

- ▶ Adding/subtracting tends to be better than multiplying
- ▶ Hence, log-likelihood $\sum \log \mathcal{L}_i$ is better than likelihood $\prod \mathcal{L}_i$
- ▶ Use true integers when possible
- ▶ Simplify your equations, minimize number of operations
- ▶ Don't do $x = \exp(\log(z))$ if you can escape it

(Now forget this list... use your brains, just remember that a computer is not exact...)