Floating point Numbers

William Gould

President
StataCorp LP

July 2014, Boston

No Report to Users this year

- I have something more interesting.
- I think you'll find it useful.
- The Report to Users will return next year.

Instead, I'm going to talk about ...

Floating Point Numbers

which I subtitle

A Visit Through The Looking Glass

Motivation

- Computers implement Floating Point (FP) numbers.
- FP numbers are similar to, but different from, Real numbers.

This is not unlike the Nutrimatic drinks dispenser,

> The Nutrimatic drinks dispenser produces
> a concoction that is "almost, but not quite,
> entirely unlike tea".
> — Hitchhiker's Guide to the Galaxy

Motivation

- Most researchers (not you) use FP numbers as if they were Real numbers.
- Most researchers (not you) have little understanding of FP numbers.
- Many (not you) couldn't even tell us one way FP and Real numbers differ, much less the most important way.

The purpose of this talk is to ...

Purpose

- Without jargon,
- Without even mentioning the word binary,
- But using the language with which you are familiar,
- Develop your understanding of FP numbers, and
- Develop your intuition about FP numbers so that you can
    - predict when calculations might go awry, and
    - know how to think about the problem, and
    - determine how to fix it.

(Warning: most talks do not live up to the author's billing.)

1 Introduction

FP stands for Floating Point.

- FP numbers are one of a variety of methods that could have been chosen by computer engineers to represent real numbers.
- FP numbers are strange ...
- ... and that strangeness has nothing to do with computers being binary.

Computers don't implement real numbers

Computers don't implement real numbers ...

... computers simulate them

As with all simulations

Some features are preserved ...

.. and others are discarded

Among the discarded features is the defining characteristic of real numbers ...

Missing is the defining characteristic of real numbers

Between any two real numbers,
there is another real number.

That's not true for FP numbers.

Worse, the number of FPs varies between any two FP numbers that are equally far apart!

That's what makes FP numbers strange.

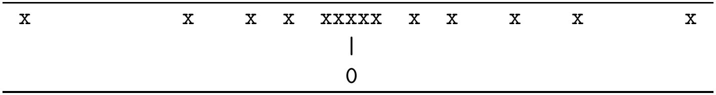
The FP (simulated Real) Number Line

The number line looks like this:

```
x            x    x  x  xxxxx  x  x    x    x        x
                          |
                          0
```

FP numbers' important properties are

- They have finite precision. Only the points shown exist.
- The distance between points increases as you move away from zero.

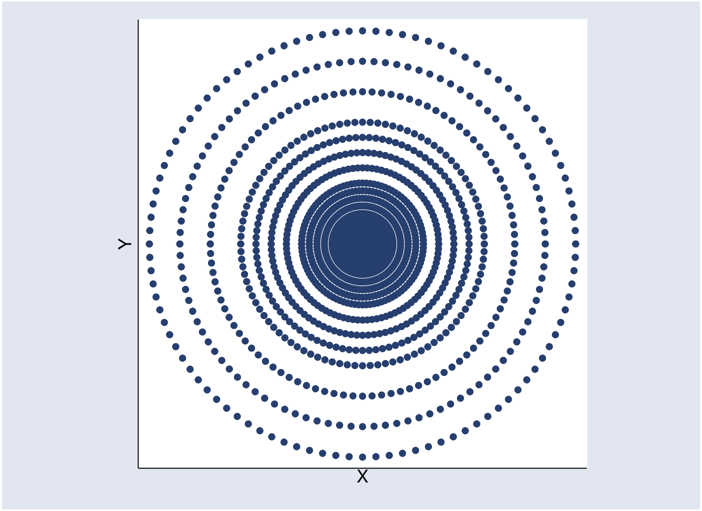


- This number line is not multiplicative or exponential.
- It is linear, punctuated by exponential jumps.

Range	Distance between adjacent numbers
$[1, 10)$	ϵ
$[10, 10^2)$	10ϵ
$[10^2, 10^3)$	$10^2\epsilon$
$[10^{-1}, 1)$	$10^{-1}\epsilon$
$[10^{-2}, 10^{-1})$	$10^{-2}\epsilon$

Let's imagine what the universe would be like if it were based on FP numbers ...

If the Universe were based on floating point numbers ...
... it would have an identifiable center!



- The FP Universe has its highest resolution at its center.
- My pictures don't show a scale. Does the decreasing resolution matter?

Let's measure **resolution** as the minimum distance that two objects can be separated, assuming the separation is greater than 0.

If the Universe had a **resolution** of Planck's distance (1.616199e-35 m) at the center (Earth, of course),

then ...

If the Universe had a **resolution** of Planck's distance (1.616199e-35 m) at the center (Earth), then the resolution on the Moon and Sun would be ...

Decimal digits of accuracy of Universe	Resolution at Moon	Resolution at Sun
1	161,620 km	16,161,990 km
2	16,162 km	1,616,199 km
7	162 m	16,162 m
8	16 m	1,616 m
16	1.616e-07 m	1.616e-05 m

The FP Universe behaves oddly as objects move

Simply moving an object outward from the center would

- cause the object to collapse in on itself, or
- cause the object to grow in size, or
- perhaps we'd have a new uncertainty principle and it would do a little of both.

Which depends on what happens to a particle that occupies a position that disappears. Does the particle move inward or outward?

Or does it vanish? That would violate conservation laws and produce a very different universe than the one we observe.

The Best Answer is

The Universe simply wouldn't work if

it were implemented in FP numbers.

Taking Stock

Where are we?

- Computer engineers created FP numbers as a substitute for real numbers.
- We use computers to perform real-world simulations, simulations based on the Real Universe.
- FP numbers cannot simulate the real universe.

Houston, we have a problem.

Or we might have. It will turn out that we do have a problem.

Why FP Numbers?

Computer engineers chose FP numbers because

- The other alternatives are worse,
- FP numbers have some useful properties, and
- the bad properties can be avoided if you are just a little cautious.

So why didn't anybody ever tell you about this?

FP Numbers

What's an FP Number?

- Real numbers are in no sense natural for digital computers.
- Instead, we simulate real numbers using facilities that are natural for digital computers. The simulation is called FP.
- Integers are natural for computers.
- So we must implement FP numbers using integers only!

Here's how ...

Simulating Real Numbers

Consider real number z. We will write it

$$z = a \times 10^b$$

where a and b are (finite precision) **integers**.

An FP number is the integer pair (a, b).

Real computers use $z = a \times 2^b$, but it will be easier to understand FPs if we use base 10.

Implementing FP Numbers

An FP number is the **integer** pair (a, b) *interpreted* to mean

$$z = a \times 10^b$$

We set limits for a and b,

$$-9999 \leq a \leq 9999$$

$$-99 \leq b \leq 99$$

and we agree on a convention (implied decimal point),

$$(a, b) \;\text{ means }\; z = (a/1000) \times 10^b$$

which will make the slides easier to read.

Examples of FP Numbers

$$(1234, 0) = 1.234$$
$$(1234, 1) = 12.34$$
$$(1234, -1) = 0.1234$$

$$(-1234, 0) = -1.234$$
$$(-1234, 1) = -12.34$$
$$(-1234, -1) = -0.1234$$

(FP integer pair on the left; our **interpretation** of it on the right.)

Normalization of FP Numbers

There are three ways we could write 1.2:

$$(1200, 0) = 1.2$$
$$(0120, 1) = 1.2$$
$$(0012, 2) = 1.2$$

If $|a| \geq 1000$ we say the FP number is **normalized**.

If $|a| < 1000$ we say the FP number is **denormalized**.

Numbers are stored in normalized form.

Adding FP Numbers

Let's add $2.0 + 3.0$:

```
   2.0 = (2000, 0)
+  3.0 = (3000, 0)
------------------
         (5000, 0) = 5.0
```

What made that possible is that the powers were equal.

We just added as integers $2000 + 3000$ to obtain integer 5000.

The result is $(5000, 0)$.

We humans *interpret* $(5000, 0)$ as meaning $5.000 \times 10^0 = 5.000$.

We are still adding numbers with equal powers

Just as easily, we can add $2.981 + 3.425$:

```
     2.981 = (2981, 0)
  +  3.425 = (3422, 0)
  -------------------
             (6403, 0) = 6.403
```

We just added as integers $2981 + 3422$ to obtain integer 6403.

The result is $(6403, 0)$.

We humans *interpret* $(6403, 0)$ as meaning 6.403.

We are still adding numbers with equal powers

Try adding $2.981 + 8.425$, for which the answer is 11.406:

```
     2.981 = (2981, 0)
+    8.425 = (8425, 0)
  --------------------
             (?406, 0)    <-  How can we squeeze 11
                              into one slot?
```

Adding integers $2981 + 8425$ results in 11406.

We can't fit a 5-digit result in a 4-digit accumulator.

Obviously, we need an extra digit in our accumulator.

We need an extra digit in our accumulator ...

So assume it:

```
     2.981 =   2981, 0)
  +  8.425 =   3422, 0)
   --------------------
               (11406, 0)  -> (1140, 1)
               /                    \
accumulator has                      we shift away the extra digit
    extra digit                      (in this case, w/o rounding)
                                     (we could have rounded)
```

The true answer is 11.406, so we lost a digit of accuracy.

We lost accuracy when we normalized the result (removed the extra digit).

A one-digit loss is the worst case.

Adding FP Numbers of Different Powers

We now know how to add $(a_1, b_1) + (a_2, b_2)$ when $b_1 = b_2$.

How do we add when $b_1 \neq b_2$?

Answer: We denormalize the number with the *smaller* power so that the powers are equal. Then we proceed just as we did previously.

Let's add 10.0 + 2.0:

```
                                     powers now equal
                                     |
  10.0    = (1000, 1)  ->  (1000, 1)
+  2.0    = (2000, 0)  ->  (0200, 1)  <- (we shifted right)
 ---------------------------------
                           (1200, 1) = 12.00
```

Adding FP Numbers of Different Powers

Let's add $10 + 2.005$:

```
                                      powers now equal
                                      |
     10.00  = (1000, 1)  ->  (1000, 1)
+     2.005 = (2005, 0)  ->  (0200, 1)  <- (we shifted right)
  -----------------------------------
                             (1200, 1) = 12.00
```

When we shift right to denormalize, we lose digits.

Requested to add $10 + 2.005$, the computer added $10 + 2.00$, and thus it obtained 12.00, not 12.005.

Adding FP Numbers of Different Powers

Just one more example. Let's add $10 + 0.0005$:

```
    10.0   = (1000, 1)  ->  (1000, 1)
+    0.005 = (5000,-3)  ->  (0000, 1)  <- (we lose all digits!)
  ----------------------------------
                            (1000, 1) = 10.0
```

When we shift right, we lose digits.

In this case, we lost *all* the digits. 0.005 became 0.

And thus, $10 + 0.0005 = 10$

Adding FP Numbers: Summary

In calculating $(a_1, b_1) + (a_2, b_2)$, the maximum digit loss is

$$|b_1 - b_2| + 1$$

On binary computers, the same applies, except we lose binary digits.

Binary digits contain less information than decimal digits, but we lose more of them. (It averages out.)

Addition is to be feared if $|b_1 - b_2| \gg 0$

Adding FP Numbers: Example 1

You have an approximation formula

$$S = C(1) + C(2) + C(3) + \dots$$

where $C(\,) > 0$ and $C(i) > C(i+1)$ for all i.

Do not code,

```
real scalar i, result

result = 0
for (i=1; i<=10; i++) result = result + C(i)
```

Do not add the terms from left to right, ...

Adding FP Numbers: Example 1

We are coding the approximation formula

$$S = C(1) + C(2) + C(3) + \dots$$

where $C(\,) > 0$ and $C(i) > C(i+1)$ for all i.

Do code,

```
real scalar i, result
real vector term

result = 0
term   = J(1, 10, .)
for (i=1;  i<=10; i++) term[i] = C(i)
for (i=10; i<=1;  --i) result  = result + term[i]
```

That is, add terms from right to left.

Adding FP Numbers: Example 2

- Imagine a large dataset with the property

$$x[1] \approx x[2] + x[3] + \cdots + X[N]$$

- It is possible that

$$x[1] + x[2] + x[3] + \cdots + x[N] = x[1]$$

- If you are worried about this, sort before adding or use Stata's quad-precision facilities.

Subtracting FP Numbers

At this point, I don't need to lead you through as many examples.

Let's calculate $10.0 - 0.5008$,

```
   10.0   = (1000, 1)  ->  (1000, 1)
-   0.5   = (5008,-1)  ->  (0050, 1)  <- (shift)
 ---------------------------------
                           (0950, 1) = (9500, 0) = 9.5
```

In a mechanical sense, **subtraction is just like addition**.

Except ...

Subtraction is just like addition, **except**

- Whereas adding nearly equal quantities is whoppingly accurate,
- **Subtracting nearly equal quantities can be whoppingly inaccurate.**

Let me show you ...

Let's calculate $10.01 + 10.00$ and calculate $10.01 - 10.00$:

```
    10.01  = (1001, 1)
+   10.00  = (1000, 1)
  -------------------
             (2001, 1) = 20.01         (4 digits of accuracy)


    10.01  = (1001, 1)
-   10.00  = (1000, 1)
  -------------------
             (0001, 1) = (1000, 3)     (1 digit of accuracy)
              ---          ---
               |            \
               |             these digits were made up!
               |
              this is a more honest answer
              we believe every one of these digits
```

Subtraction is always scary

Consider $(a_1, b_1) - (a_2, b_2)$,

- Either $b_1 \neq b_2$, in which case we lose digits
- Or $b_1 = b_2$, in which case we make up digits

Footnote:

- When I say subtraction, I mean subtraction of like-signed values.
- Subtraction of oppositely signed values is addition.

Think about addition and subtraction like this:

When adding $z_1 + z_2$, we do not have to believe that z_1 and z_2 are perfectly accurately recorded.

- Say $z_1 = (1001, 1)$ and $z_2 = (1000, 1)$.
- We do not have to believe that $z_1 = 10.01000$ and $z_2 = 10.00000$.
- Whatever we believe, we obtain $z_1 + z_2$ to advertised accuracy.

With subtraction, however ...

Think about addition and subtraction like this:

When subtracting $z_1 - z_2$, if $z_1 \approx z_2$, we must believe that z_1 and z_2 are perfectly accurately recorded.

- Say $z_1 = (1001, 1)$ and $z_2 = (1000, 1)$.
- We must believe that $z_1 = 10.01000$ and $z_2 = 10.00000$ even though all that's recorded is $z_1 = 10.01$ and $z_2 = 10.00$.
- That is, we must assume enough zeros to justify the made up digits!

Or, if z_1 and z_2 are of different magnitudes, we run into the standard inaccuracies.

For subtraction, nothing beats having extra digits

- It is because of subtraction that modern computers carry "extra digits" in their FP numbers.
- If you would promise never to perform a single subtraction, we wouldn't need them.
- You cannot make that promise, and so we need extra digits in **ALL** the operators.

This way, when you subtract two numbers that are nearly equal, the result contains at least some digits that are accurate.

You use double precision in order to obtain single-precision accuracy!

Sear this onto your brain

When you subtract numbers that are roughly equal,

double precision returns roughly single-precision accuracy

if you are lucky.

Subtracting FP Numbers: Example: Reverse cumulatives

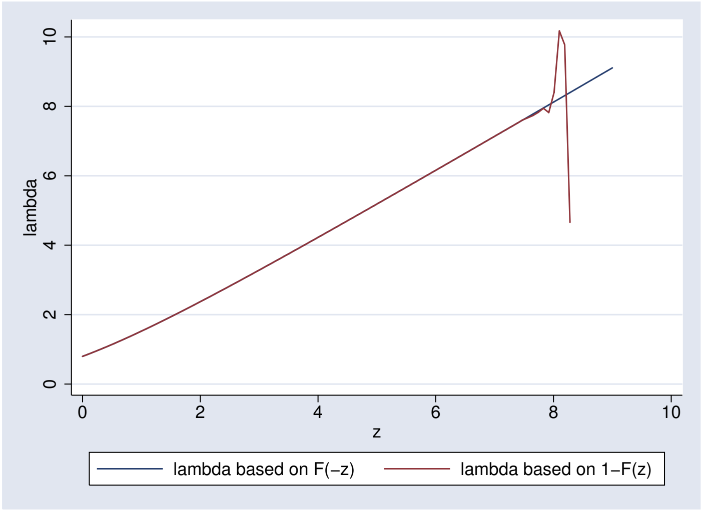
Comparison of $1 - F(z)$ and $G(z) = F(-z)$
F( ) cumulative normal

z	# of digits in common
1	16
3	13
5	9
7	5
8	1
9	0

Regardless of distribution, never calculate $1 - F(\,)$.

Calculate $G(\,)$. (Formulas vary.)

Inverse Mills Ratio



Subtracting FP Numbers: Example 2: Probabilities

The above was not just a cute example.

For $0 \leq p \leq 1$,

The simulated real numbers that computers provide
simply cannot calculate $1 - p$ accurately.

- Restate your formulas so that they do not involve $1 - p$,
- or restrict the range to $p \leq 0.99997$.

Subtracting FP Numbers: Cox Survival Model

- Cox concerns risk pools.
- Subjects enter the risk pools, and leave them.
- We have lots of things we must calculate about the current risk pool.
- When subjects enter, we **add** values to the risk pools.
- When subjects leave, we **subtract** those values back out.
- Run the whole process from start to finish, and the values **ought to be zero**.
- **They are not.**

We are adding to and subtracting from risk pools,

- We evaluated lots of clever normalization and formulae for improving the accuracy of this add-and-subtract process.
- They improved accuracy, but not enough.
- Problem is, despite all of the **double precision code**, because of the **subtractions**, we are carrying only **single-precision accuracy**.
- Concerning subtraction, **nothing beats adding more FP digits**.
- We went to quad precision (and less sophisticated code).
    - Accuracy improved.
    - Run times improved.

Cox Model: Resolution

- We start the problem using double-precision code, yielding single-precision accuracy.
- Drive that to convergence.
- Then we start again, using quad-precision code yielding double-precision accuracy.
- Drive that to convergence.
- We have a fast and accurate result.

If you have an accuracy problem due to subtraction, ...

... consider quad precision.

The one case when subtraction is safe

Subtraction x − c is perfectly safe when

- **x is data** and **not the result of calculation on data**,
- **c is anything**, data or the result of (imperfect) calculation,
- **x − c is closer to the center** ($|x-c| < |x|$).

As a programmer,

- I take the data as given.
- Thus I assume $x = (1001, 2) = 10.01$ really means $x = 10.01000\ldots$
- That's how I get accurate results *conditioned* on the data.

The one case when subtraction is safe

Subtraction to move data toward the center is not only safe, it's a good idea

- The FP number line is denser at the center.
- And because the points are denser, subsequent additions and subtractions performed on $x_i - c$ produce more accurate results than those performed on x.

```
. summarize x

. generate centered_x = x - r(mean)
                              \
                               not just safe, but recommended
```

The FP number line is denser at the center

Move in and lose nothing.

You can move back out without loss.

```
. generate x = 5000.01

. generate closer = x - 5000

. generate original = closer + 5000

. display x-original
0
```

The FP number line is denser at the center

Move out and lose precision.

Move back in and the loss is permanent.

```
. generate x = 5000.01

. generate farther = x + 5000

. generate original = farther - 5000

. display x-original
-.00390625
```

Multiplication and Division

I wish I had time to tell you about multiplication and division,

- It's interesting,
- but it's okay to skip it, because everything I would say is reassuring.
- **Multiplication and division lose at most 1/2 digit.**
- FP was designed for multiplication and division.

Summary of Operators

- **Addition is a red flag**.
- **Subtraction is a red flag with flashing lights**.
    - FP numbers produce a difference that is almost, but not quite, entirely unlike subtraction.
- Multiplication and Division are always accurate.

What this means for Statistical Applications

1. **Subtraction is scary. Be scared.**

2. But **don't ignore addition**.

- Statistics is about data reduction.
- Statisticians' favorite data-reduction operator is addition.
  - We sum data and then divide to get a mean.
  - We sum likelihood contributions to fit models.
  - The list goes on and on.
- And when addition is used, it's used a lot. Addition is major contributor to accuracy problems in statistical calculations.

The best way to sum

When you have to calculate $S = \sum_{i=1}^{n} x_i$ and you're worried about precision,

Use Mata's quad-precision routines

Quad precision works best in my experience.

There's a clever, double-precision alternative. I'll show you ...

To calculate $S = \sum_{i=1}^{n} x_i$, use

$$m_i = m_{i-1} \times \frac{i-1}{i} + \frac{x_i}{i}$$

Start it off the $m_1 = x_1$. Then $S = m_n \times n$.

Code it like this,

```
real vector x
real scalar S, m, i,n

n = length(m)
m = x[1]
for (i=2; i<=n; i++) m = m*((i-1)/i) + x[i]/i
S = m*n
```

Final Advice

When using programs where great care has not been exercised in numerically making the calculations—when when using your own programs or those written by other users,

> **Scale your data so that variables are roughly between 0 and 1000 (or –1000 and 1000).**

Scaled data considerably reduces computational stress.

Final Advice

Even though multiplication is safe, if you use Mata, remember

All matrix operators are problematic, even multiplication.

Matrix multiplication involves scalar multiplication and scalar addition and, if any of the matrix elements are negative, scalar subtraction, too.

Matrix routines, coded without thought and used on unscaled data, are an excellent way of quickly producing inaccurate results.

Conclusion

The problem is

- **FP numbers mostly work like Real numbers**.
- That makes everyone complacent; even us at StataCorp.
- **And then they bite** because, sometimes, FP numbers are not at all like Real numbers.

Conclusion

I wish I had more time

I could show you lots more examples
I could reassure you about many calculations
I could show you workarounds for others

but now you know how to think about the issues

Thank you for you attention.