Single Precision Storage Default - Is it time to bid farewell?

presentation for Oceania Stata Conference 2025

Jan Kabatek

The University of Melbourne, CentER, IZA, LCC & Netspar

February 5, 2025

Why am I here?

- I have identified a legacy issue that is important enough (IMO) to warrant your attention.
- Last time around, I talked about **inefficiencies in Stata visualization workflows** (twoway/histogram/line/etc.).
- And I introduced my PLOT suite of **graphing commands for large datasets:**

```
ssc install plottabs
```

- Today, I want to highlight another issue and propose a readily-available solution

Illustrative example

Stata code

```
clear
set obs 10
generate x = _n /10
list x
```

Illustrative example

Stata code

```
clear
set obs 10
generate x = _n /10
list x
```

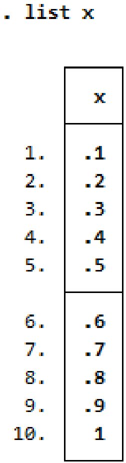
...this produces and lists a variable $x \in \{0.1, 0.2, \ldots, 1\}$

Illustrative example

Stata code

```
clear
set obs 10
generate x = _n /10
list x
```

...this produces and lists a variable $x \in \{0.1, 0.2, \ldots, 1\}$



```
. list x

     +----+
     |  x |
     |----|
  1. | .1 |
  2. | .2 |
  3. | .3 |
  4. | .4 |
  5. | .5 |
     |----|
  6. | .6 |
  7. | .7 |
  8. | .8 |
  9. | .9 |
 10. |  1 |
     +----+
```

Illustrative example

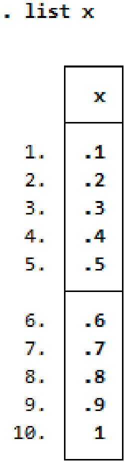
Stata code

```
clear
set obs 10
generate x = _n /10
list x
```

...this produces and lists a variable $x \in \{0.1, 0.2, \ldots, 1\}$

Let's introduce some basic conditionality...

```
list x if x <= .4
```



```
. list x

     +----+
     |  x |
     |----|
  1. | .1 |
  2. | .2 |
  3. | .3 |
  4. | .4 |
  5. | .5 |
     |----|
  6. | .6 |
  7. | .7 |
  8. | .8 |
  9. | .9 |
 10. |  1 |
     +----+
```

Illustrative example

Stata code

```
clear
set obs 10
generate x = _n /10
list x
```

...this produces and lists a variable $x \in \{0.1, 0.2, \ldots, 1\}$

Let's introduce some basic conditionality...

```
list x if x <= .4
```

...what happens now?

Illustrative example

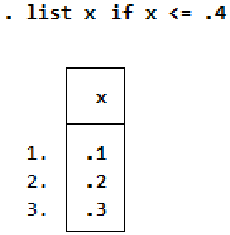
Stata code

```
clear
set obs 10
generate x = _n /10
list x
```

…this produces and lists a variable $x \in \{0.1, 0.2, \ldots, 1\}$

Let's introduce some basic conditionality…

```
list x if x <= .4
```



```
. list x if x <= .4

     +-----+
     |   x |
     |-----|
  1. |  .1 |
  2. |  .2 |
  3. |  .3 |
     +-----+
```

Illustrative example

Stata code

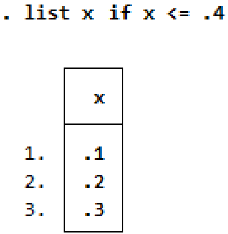
```
clear
set obs 10
generate x = _n /10
list x
```

…this produces and lists a variable $x \in \{0.1, 0.2, \ldots, 1\}$

Let's introduce some basic conditionality…

```
list x if x <= .4
```

…huh? Where's **0.4**???



```
. list x if x <= .4

     +----+
     |  x |
     |----|
  1. | .1 |
  2. | .2 |
  3. | .3 |
     +----+
```

Blast from the past: Floating-point precision issues

- Computer architectures have been known to struggle with **non-integer numbers**, such as fractions, π, ρ, *etc*.

$$^1\!/_3 = 0.3333333333\ldots$$

Blast from the past: Floating-point precision issues

- Computer architectures have been known to struggle with **non-integer numbers**, such as fractions, π, ρ, *etc*.

$$^1\!/_3 = 0.3333333333\ldots$$

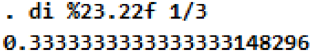
- To evaluate and work with these numbers, we typically resort to approximation (and Stata does so, too).

Blast from the past: Floating-point precision issues

- Computer architectures have been known to struggle with **non-integer numbers**, such as fractions, π, ρ, *etc*.

$$^{1}/_{3} = 0.3333333333\ldots$$

- To evaluate and work with these numbers, we typically resort to approximation (and Stata does so, too).



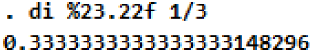
```
. di %23.22f 1/3
0.3333333333333333148296
```

Blast from the past: Floating-point precision issues

- Computer architectures have been known to struggle with **non-integer numbers**, such as fractions, π, ρ, *etc*.

$$^1\!/_3 = 0.3333333333\ldots$$

- To evaluate and work with these numbers, we typically resort to approximation (and Stata does so, too).



```
. di %23.22f 1/3
0.3333333333333333148296
```

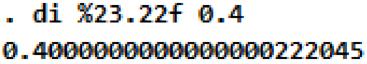
- The exact sequence of numbers **outside the precision range** is IEEE-standardized and replicable across programming languages (on the same hardware).

Back to the example: What happens with 0.4?

- So let's look at the precision handling of **0.4**:

Back to the example: What happens with 0.4?

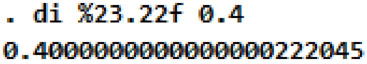
- So let's look at the precision handling of **0.4**:



```
. di %23.22f 0.4
0.4000000000000000222045
```

Back to the example: What happens with 0.4?

- So let's look at the precision handling of **0.4**:

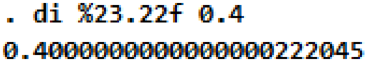


```
. di %23.22f 0.4
0.4000000000000000222045
```

- This number is greater than **0.4**! That is why **0.4** was excluded from the list!!!

Back to the example: What happens with 0.4?

- So let's look at the precision handling of **0.4**:

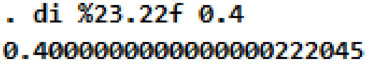


```
. di %23.22f 0.4
0.4000000000000000222045
```

- This number is greater than **0.4**! ~~That is why **0.4** was excluded from the list!!!~~

Back to the example: What happens with 0.4?

- So let's look at the precision handling of **0.4**:

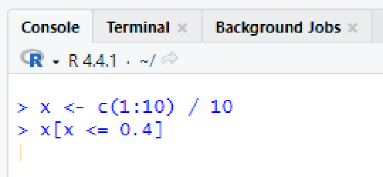


```
. di %23.22f 0.4
0.4000000000000000222045
```

- This number is greater than **0.4**! ~~That is why **0.4** was excluded from the list!!!~~
- Nope, this is not the reason. If it were, we should be able to replicate the same behavior across different programming languages.

Killing joy with R

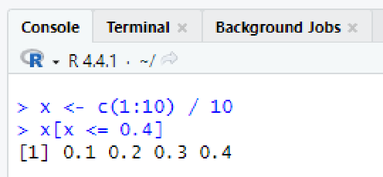
- Let’s produce an equivalent workflow in **R**:



```
> x <- c(1:10) / 10
> x[x <= 0.4]
```

Killing joy with R

- Let's produce an equivalent workflow in **R**:

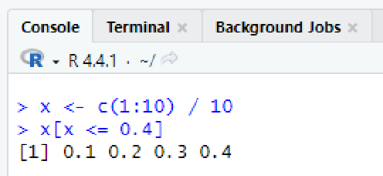


```
> x <- c(1:10) / 10
> x[x <= 0.4]
[1] 0.1 0.2 0.3 0.4
```

- **R** produces the correct result, which means that floating-point arithmetic is not to blame here.

Killing joy with R

- Let's produce an equivalent workflow in **R**:

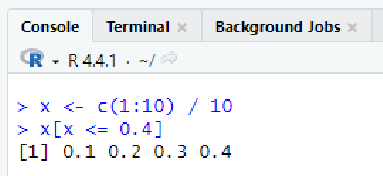


```
> x <- c(1:10) / 10
> x[x <= 0.4]
[1] 0.1 0.2 0.3 0.4
```

- **R** produces the correct result, which means that floating-point arithmetic is not to blame here.
- This makes intuitive sense, since the precision of the numbers **stored in the c() vector**, and the number used in the **conditional statement** use the same standard.

Killing joy with R

- Let's produce an equivalent workflow in **R**:



```
> x <- c(1:10) / 10
> x[x <= 0.4]
[1] 0.1 0.2 0.3 0.4
```

- **R** produces the correct result, which means that floating-point arithmetic is not to blame here.
- This makes intuitive sense, since the precision of the numbers **stored in the c() vector**, and the number used in the **conditional statement** use the same standard.
  - Fundamentally, we are asking whether 0.40000000000000000222045 is smaller or equal than 0.40000000000000000222045, which it is!

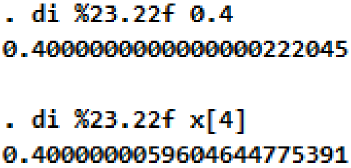
So what is going in Stata???

So what is going in Stata???

- The culprit is **inconsistent storage types**. **By default**, Stata uses **different storage types** for the numbers that are **stored as data points** (`float`), and the numbers that are **used to perform arithmetic operations** (`double`).

So what is going in Stata???

- The culprit is **inconsistent storage types**. **By default**, Stata uses **different storage types** for the numbers that are **stored as data points** (`float`), and the numbers that are **used to perform arithmetic operations** (`double`).

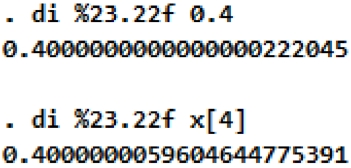


```
. di %23.22f 0.4
0.4000000000000000222045

. di %23.22f x[4]
0.4000000059604644775391
```

So what is going in Stata???

- The culprit is **inconsistent storage types**. **By default**, Stata uses **different storage types** for the numbers that are **stored as data points** (`float`), and the numbers that are **used to perform arithmetic operations** (`double`).



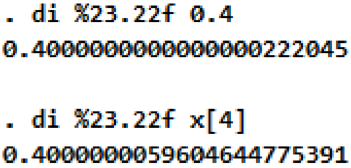
```
. di %23.22f 0.4
0.4000000000000000222045

. di %23.22f x[4]
0.4000000059604644775391
```

- That is why the stored number 0.4 does not satisfy the weak inequality restriction in `list x if x<= 0.4`.

So what is going in Stata???

- The culprit is **inconsistent storage types**. **By default**, Stata uses **different storage types** for the numbers that are **stored as data points** (`float`), and the numbers that are **used to perform arithmetic operations** (`double`).



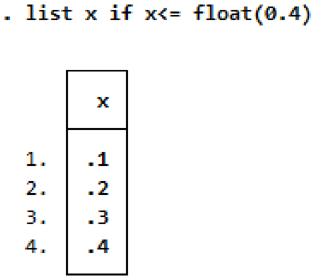
```
. di %23.22f 0.4
0.4000000000000000222045

. di %23.22f x[4]
0.4000000059604644775391
```

- That is why the stored number 0.4 does not satisfy the weak inequality restriction in `list x if x<= 0.4`.
- The stored value is **strictly greater** than the value used in the if-statement.

Quod Erat Demonstrandum

- We get the correct behavior if we force the value in the if-statement to be of the same precision (`float`) as the stored value:



```
. list x if x<= float(0.4)

     +----+
     |  x |
     |----|
  1. | .1 |
  2. | .2 |
  3. | .3 |
  4. | .4 |
     +----+
```

So what?

- The **Stata manual** is not too bothered, stating that:

> *”This is unlikely to affect any calculated result because Stata performs all internal calculations in double precision.”*

So what?

- The **Stata manual** is not too bothered, stating that:

> *”This is unlikely to affect any calculated result because Stata performs all internal calculations in double precision.”*

- Well, I disagree.

So what?

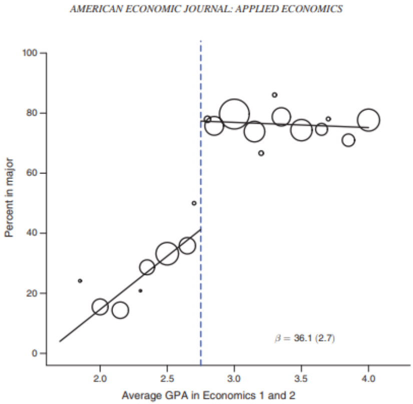
- The **Stata manual** is not too bothered, stating that:

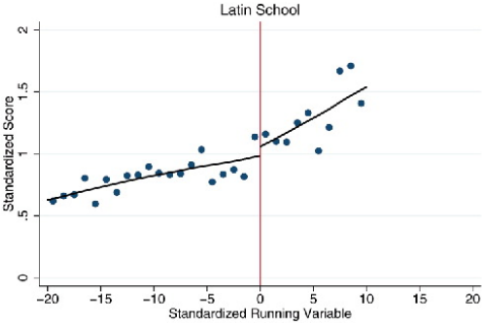
> *"This is unlikely to affect any calculated result because Stata performs all internal calculations in double precision."*

- Well, I disagree.
- The problem is that this behavior is **unexpected**, and it is capable of producing **calculation & data construction errors** that can be **extremely damaging to modern causal inference designs**.

RDD: Regression Discontinuity Debacle

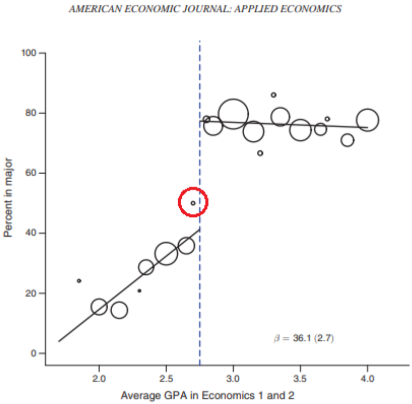
- Many causal designs operate with cut-off points, and a correct classifications of observations in the vicinity of the cutoff point is critical:

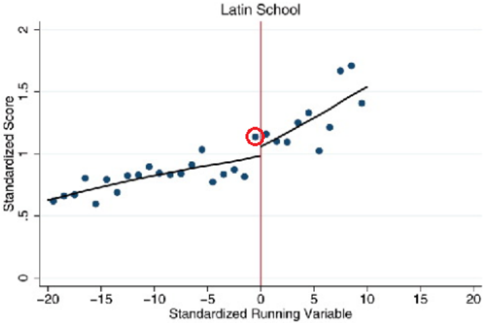




RDD: Regression Discontinuity Debacle

- Many causal designs operate with cut-off points, and a correct classifications of observations in the vicinity of the cutoff value is critical:





Well, that's not great...

- Yup.
- These precision issues introduce another layer of uncertainty that can hamper reliability and replicability of scientific studies.
- IMHO, we should endeavor to eliminate these hidden traps, especially when solutions are readily available.

Solution 1: Everything in double!

- Make Stata 19 use the `double` precision for **both** math operations & data storage **BY DEFAULT**.

Solution 1: Everything in double!

- Make Stata 19 use the `double` precision for **both** math operations & data storage **BY DEFAULT**.
- This will mean that the datasets storing non-integer numbers will become larger, but that's a minor **legacy issue** (considering the capacities of modern hard drives)

Solution 1: Everything in double!

- Make Stata 19 use the `double` precision for **both** math operations & data storage **BY DEFAULT**.

- This will mean that the datasets storing non-integer numbers will become larger, but that's a minor **legacy issue** (considering the capacities of modern hard drives)
- **R** uses the very same default.

Solution 1: Everything in double!

- Make Stata 19 use the `double` precision for **both** math operations & data storage **BY DEFAULT**.

- This will mean that the datasets storing non-integer numbers will become larger, but that’s a minor **legacy issue** (considering the capacities of modern hard drives)
- **R** uses the very same default.

In the meantime, we can set the precision standard manually:

```
set type double
*caution: ’set type float’ will NOT override double for arithmetic ops
clear
set obs 10
generate x = _n/10
list x if x <= 0.4
```

Solution 2: Smart precision handling

- The alternative is to assign the type of values used in mathematical expressions according to the precision of the stored values that are being evaluated:

Solution 2: Smart precision handling

- The alternative is to assign the type of values used in mathematical expressions according to the precision of the stored values that are being evaluated:

Pseudocode:

```
if type(x) = float ---> evaluate: x<= float(0.4)
if type(x) = double --> evaluate: x<= double(0.4)
if type(x) = quad ----> evaluate: x<= quad(0.4)
```

Solution 2: Smart precision handling

- The alternative is to assign the type of values used in mathematical expressions according to the precision of the stored values that are being evaluated:

Pseudocode:

```
if type(x) = float ---> evaluate: x<= float(0.4)
if type(x) = double --> evaluate: x<= double(0.4)
if type(x) = quad ----> evaluate: x<= quad(0.4)
```

- This is more cumbersome (and could run into problems with complex operations where the correct use might be ambiguous), but it would avoid making datasets larger by default.

Takeaways

- Unless the single-precision default has some other justification that supersedes the concerns presented here, **I say that it is time to bid it a heartfelt farewell.**

Takeaways

- Unless the single-precision default has some other justification that supersedes the concerns presented here, **I say that it is time to bid it a heartfelt farewell.**
- The Stata practitioners will thank you for it.

  (or they would if they were aware of this issue to start with)

Thank you for your attention!

Email:	j.kabatek@unimelb.edu.au
Web:	www.jankabatek.com
Git:	github.com/jankabatek
Bluesky:	@jankabatek.com