Rounding Error Sim

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(ggthemes)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

Simulating errors that come from "reconstruction" timeseries vote deltas for presidential candidates.

For this simulation we assume that vote count deltas between timepoints are drawn from a uniform distribution.

Later, we formally define when exactly these errors will occur.

```
timepoints <- 50000
min_delta <- 0
max_delta <- 1000

time <- seq(from=1, to=timepoints, by=1)

getMonotonicIntegerRandomWalkDeltas <- function(boost=0) {
  diffs <- round(runif(timepoints, min_delta, max_delta+boost),0)
  return(diffs)
}

getReconstructionDeltaFrame <- function() {
  d_deltas = getMonotonicIntegerRandomWalkDeltas(0)
  d = cumsum(d_deltas)

  r_deltas = getMonotonicIntegerRandomWalkDeltas()
  r = cumsum(r_deltas)

  totals <- d+r

  reconstruct_deltas <- function(cum_votes, totals, rounding_digit) {
    rounded_pcts <- round(d / totals, rounding_digit)
```

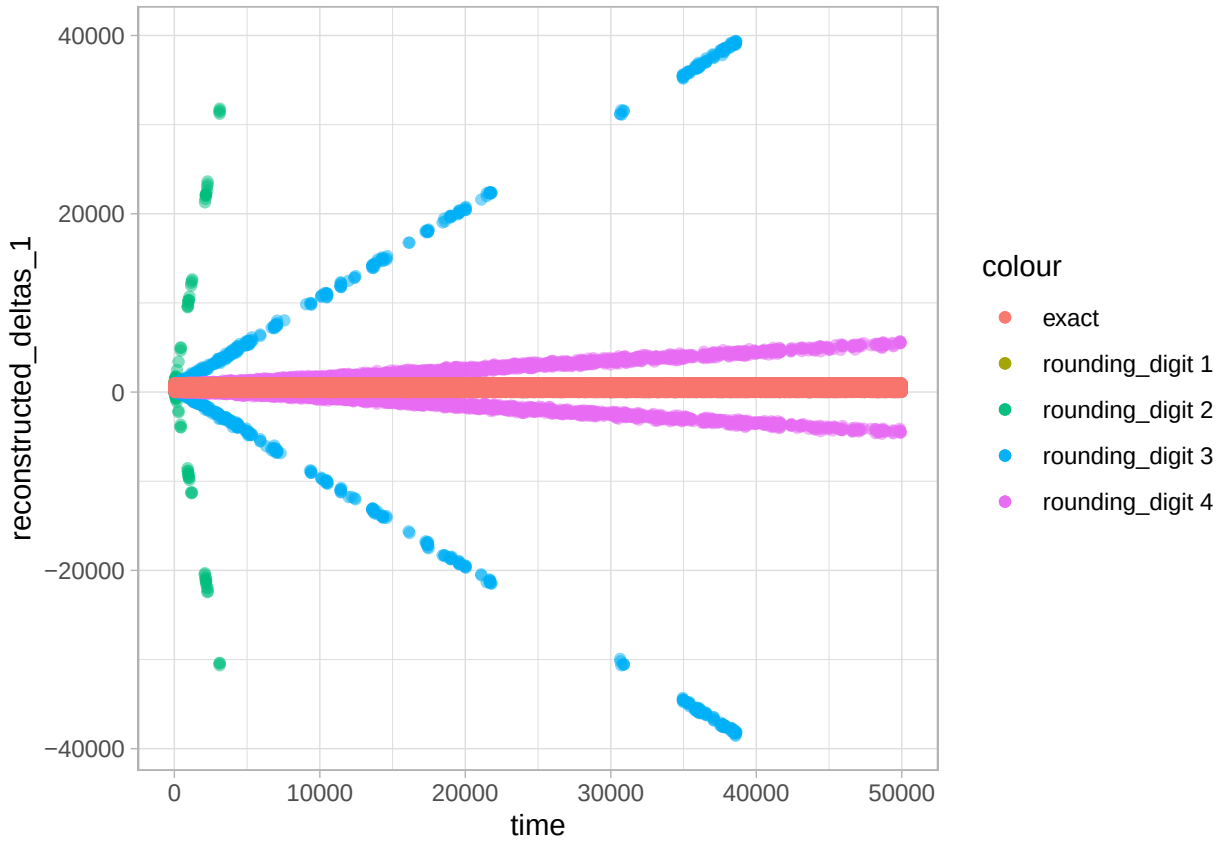
```
    reconstruction <- rounded_pcts * totals
    reconstructed_deltas <- c(reconstruction[1], diff(reconstruction))
    return(reconstructed_deltas)
  }

  df <- data.frame(
    time=time,
    d=d,
    r=r,
    reconstructed_deltas_5 = reconstruct_deltas(d, totals, 5),
    reconstructed_deltas_4 = reconstruct_deltas(d, totals, 4),
    reconstructed_deltas_3 = reconstruct_deltas(d, totals, 3),
    reconstructed_deltas_2 = reconstruct_deltas(d, totals, 2),
    reconstructed_deltas_1 = reconstruct_deltas(d, totals, 1),
    reconstructed_deltas_0 = reconstruct_deltas(d, totals, 0),
    deltas=d_deltas,
    totals=totals
  )
}

df <- getReconstructionDeltaFrame()

alpha <- 0.5
g <- ggplot(df)  +
  geom_point(aes(x=time, y=reconstructed_deltas_1, col='rounding_digit 1'), alpha=alpha) +
  geom_point(aes(x=time, y=reconstructed_deltas_2, col='rounding_digit 2'), alpha=alpha) +
  geom_point(aes(x=time, y=reconstructed_deltas_3, col='rounding_digit 3'), alpha=alpha) +
  geom_point(aes(x=time, y=reconstructed_deltas_4, col='rounding_digit 4'), alpha=alpha) +
  geom_point(aes(x=time, y=deltas, col='exact'), alpha=alpha) +
  theme_light()

g
```



There's a weird pattern that emerges where we see deltas that appear beyond the original distribution we're drawing from.

There must be a rule that governs where these deviations show up!

Let's attack this formally, focusing on negative deltas. When can a reconstructed vote for a candidate slide *backwards*?

Let's define a reconstructed vote as a rounded percentage of vote shares multiplied by the total:

$$reconstruction = R_3(\frac{v}{T})T$$

Where:

- the R_n operation is a rounding to the nth digit
- v is count of votes for one candidate at a timestep
- T is the total number of votes at a timestep

Backslides will happen when:

$$R_3(\frac{v^+}{T^+})T^+ < R_3(\frac{v}{T})T$$

Where the x^+ notation refers to the *next* timestep's quantity.

Let's say we have an operation $D_n(x)$ that tells use the nth digit of x (with subsequent digits in the mantissa). If so, then the operation $R_n(x)$ can be expressed as this piecewise function:

$$R_n(x) = \frac{C(10^n x)}{10^n}; D_{n+1}(x) \geq 5$$

$$R_n(x) = \frac{F(10^n x)}{10^n}; D_{n+1}(x) < 5$$

Where C is the ceiling function and F is the floor function. When their arguments are fractions, these functions can be defined as:

$$F(x) = F(\frac{n}{d}) = \frac{n}{d} - \frac{mod(n,d)}{d}$$

The logic being: the floor function is simply removing the part that cannot be neatly divided from some quotient.

A ceiling, then is:

$$C(x) = F(x) + 1$$

Since a ceiling gets you to the closest next integer, it's equivalent to flooring and then adding 1.

Ok, we now have all the tools we need to figure out when $R_3(\frac{v^+}{T^+})T^+ < R_3(\frac{v}{T})T$.

We'll have to consider every combination of when $D_4(\frac{v^+}{T^+}) \geq 5$ or $D_4(\frac{v}{T}) < 5$ etc. to see how the R_3 function will evaluate, though, which is annoying, but that's what you get when you use piecewise functions I guess.

Condition 1

$D_4(\frac{v^+}{T^+}) \geq 5$ and $D_4(\frac{v}{T}) \geq 5$

In this condition, the statement $R_3(\frac{v^+}{T^+})T^+ < R_3(\frac{v}{T})T$ evaluates to:

$$\frac{C(10^3\frac{v^+}{T^+})}{10^3}T^+ < \frac{C(10^3\frac{v}{T})}{10^3}T$$

Or:

$$C(10^3\frac{v^+}{T^+})T^+ < C(10^3\frac{v}{T})T$$

Which, using the definition of C, becomes:

$$(\frac{10^3 v^+}{T^+} - \frac{mod(10^3 v^+, T^+)}{T^+} + 1)T^+ < (\frac{10^3 v}{T} - \frac{mod(10^3 v, T)}{T} + 1)T$$

Cleaning up to:

$$10^3 v^+ - mod(10^3 v^+, T^+) + T^+ < 10^3 v - mod(10^3 v, T) + T$$

We know that:

$$v^+ = v + \Delta_v$$

So we can write:

$$10^3 v + 10^3 \Delta_v - mod(10^3 v^+, T^+) + T^+ < 10^3 v - mod(10^3 v, T) + T$$

Making our simplified condition:

$$\Delta_v < \frac{mod(10^3 v^+, T^+) - mod(10^3 v, T) - \Delta_T}{10^3}$$

Condition 2

$D_4(\frac{v^+}{T^+}) \geq 5$ and $D_4(\frac{v}{T}) < 5$

With similar logic to the above, this lets us say:

$$\Delta_v < \frac{mod(10^3 v^+, T^+) - mod(10^3 v, T) - T^+}{10^3}$$

Condition 3

$D_4(\frac{v^+}{T^+}) < 5$ and $D_4(\frac{v}{T}) \geq 5$

$$\Delta_v < \frac{mod(10^3 v^+, T^+) - mod(10^3 v, T) + T}{10^3}$$

Condition 4

$D_4(\frac{v^+}{T^+}) < 5$ and $D_4(\frac{v}{T}) < 5$

$$\Delta_v < \frac{mod(10^3 v^+, T^+) - mod(10^3 v, T)}{10^3}$$

```
mod <- function(n, d){
  # Sugar to make mod functions easier to read
  return(n %% d)
}

digit <- function(number, n){
  # Getting the nth digit of a number
  return(number*(10**n) - floor((10**(n-1))*number)*10)
}
```

```
conditionOne <- function(n, vp, tp, v, t) {
  # Applicable when D4(v+) >= 5 and D4(v) >= 5
  Vdeltas <- vp - v
  Tdeltas <- tp - t
  return(Vdeltas < (mod((10**n)*vp, tp) - mod((10**n)*v, t) - Tdeltas) / (10**n))
}

conditionTwo <- function(n, vp, tp, v, t) {
  # Applicable when D4(v+) >= 5 and D4(v) < 5
  Vdeltas <- vp - v
  Tdeltas <- tp - t
  return(Vdeltas < (mod((10**n)*vp, tp) - mod((10**n)*v, t) - tp) / (10**n))
}

conditionThree <- function(n, vp, tp, v, t) {
  # Applicable when D4(v+) < 5 and D4(v) >= 5
  Vdeltas <- vp - v
  Tdeltas <- tp - t
```

```
  return(Vdeltas < (mod((10**n)*vp, tp) - mod((10**n)*v, t) + t) / (10**n))
}

conditionFour <- function(n, vp, tp, v, t) {
  # Applicable when D4(v+) < 5 and D4(v) < 5
  Vdeltas <- vp - v
  Tdeltas <- tp - t
  return(Vdeltas < (mod((10**n)*vp, tp) - mod((10**n)*v, t)) / (10**n))
}

meetsConditions <- function(roundingDigit, votes, totals) {
  # Return a vector where we satisfy a condition for returning a negative delta

  # We're only really concerned with steps where there was a previous timestep, so we set up
  # vp,tp--vectors for the 'next timestamp' in the equations above, and
  # v, t--vectors for the current timestamp
  v <- votes[1:length(votes)-1]
  t <- totals[1:length(totals)-1]
  vp <- votes[2:length(votes)]
  tp <- totals[2:length(totals)]

  shares <- v / t
  sharesP <- vp / tp

  d <- digit(shares, roundingDigit + 1)
  dP <- digit(sharesP, roundingDigit + 1)

  meetsConditions <- logical(length(v))

  idx <- dP >= 5 & d >= 5
  meetsConditions[idx] <- conditionOne(roundingDigit, vp[idx], tp[idx], v[idx], t[idx])
  print(sum(meetsConditions[idx]))


  idx <- dP >= 5 & d < 5
  meetsConditions[idx] <- conditionTwo(roundingDigit, vp[idx], tp[idx], v[idx], t[idx])
  print(sum(meetsConditions[idx]))

  idx <- dP < 5 & d >= 5
  meetsConditions[idx] <- conditionThree(roundingDigit, vp[idx], tp[idx], v[idx], t[idx])
  print(sum(meetsConditions[idx]))

  idx <- dP < 5 & d < 5
  meetsConditions[idx] <- conditionFour(roundingDigit, vp[idx], tp[idx], v[idx], t[idx])
  print(sum(meetsConditions[idx]))

  # pad left
  meetsConditions <- c(FALSE, meetsConditions)

  return(meetsConditions)

}
```

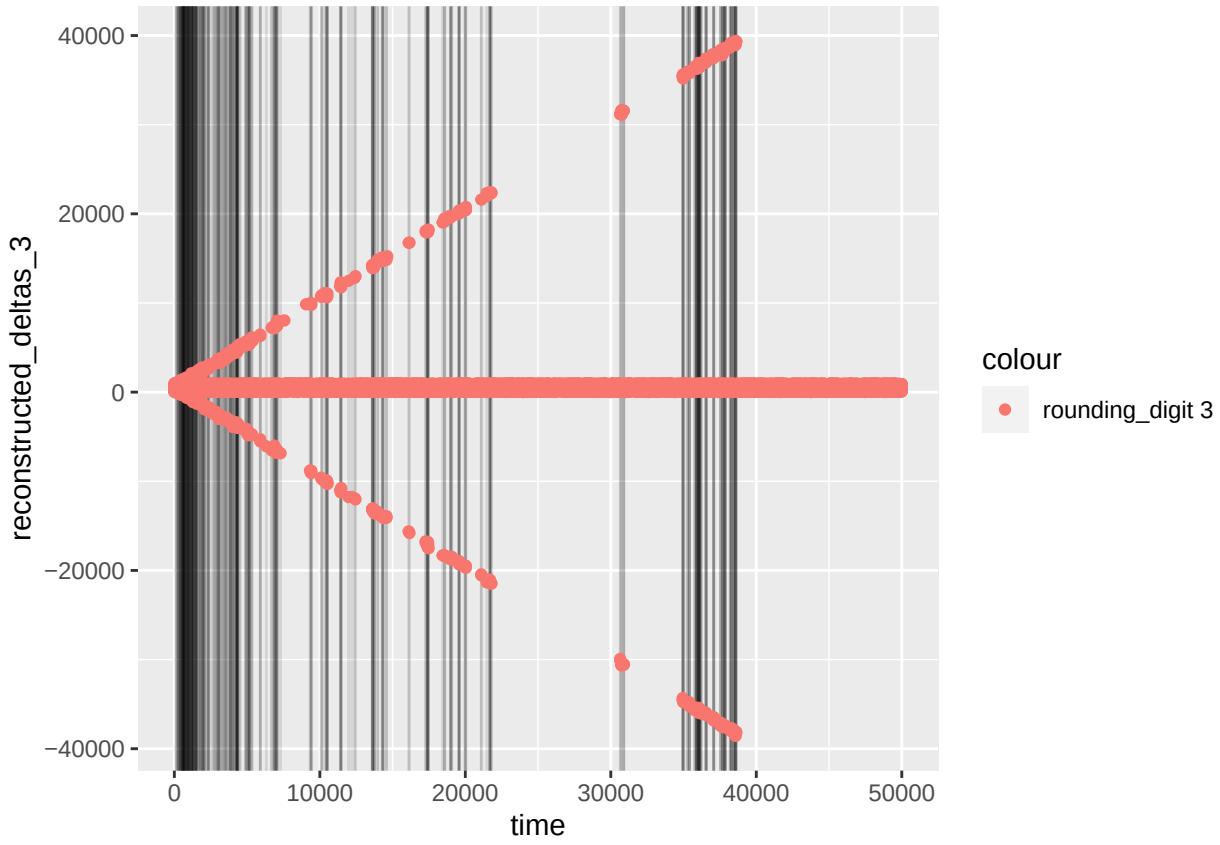
```
meetsConditions <- meetsConditions(3, df$d, df$totals)
```

```
## [1] 3
## [1] 0
## [1] 409
## [1] 8
```

```
locs <- seq(1, length(df$totals), 1)[meetsConditions]
trueLocs <- seq(1, length(df$totals), 1)[df$reconstructed_deltas_3 < 0]

regions <- ggplot(df) +
  geom_vline(xintercept=locs, alpha=0.1) +
   #geom_vline(xintercept=trueLocs, color='red', alpha=0.1) +
  geom_point(aes(x=time, y=reconstructed_deltas_3, col='rounding_digit 3'))

regions
```




```
ggsave(file='rounding-sim-regions.png',regions)
```

```
## Saving 6.5 x 4.5 in image
```

```
sum(meetsConditions)
```

```
## [1] 420
```

```
sum(trueLocs - locs)
```

```
## [1] 0
```