How fast do you really fall?

Minecraft wiki listed two equations for calculating how fast you fall, and they were both wrong.
https://minecraft.gamepedia.com/Transportation#Vertical_transportation

tl;dr:

Replace $d(t) = 3.92 * \left(99 - 49.5 * \left(0.98^{t} + 1\right) - t\right)$**with**
$d(t) = 196 - 3.92 * t - 194.04 * 0.98^{t-0.5}$

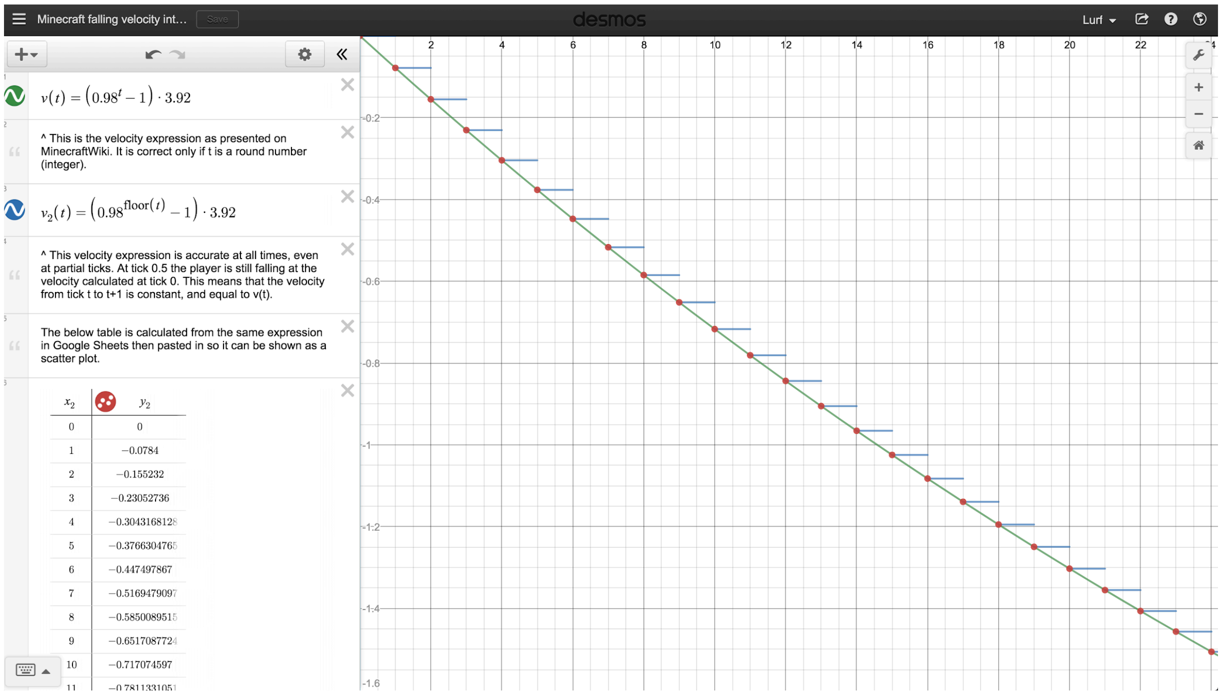
Here's why.

First, it has $v(t) = \left(0.98^{t} - 1\right) * 3.92$
This gives you your velocity (in units of blocks per tick), from how many ticks you've been falling for.
This equation is correct… but **only** when t is an integer. If t is, for example, 3.5, it gives -0.2676 while it should really be -0.2305.
This is because **Minecraft only updates your velocity once a tick**. So from t=3 to t=4, the velocity is v(3)=-0.2305 the entire time. Then, at t=4 it changes suddenly to -0.3043.
The correct equation is $v(t) = \left(0.98^{floor(t)} - 1\right) * 3.92$

This wouldn't normally be an issue, but the next equation integrates v(t) to get d(t).

$$d(t) = \int v(t)dt = \int\left(0.98^{t} - 1\right) * 3.92\, dt = 3.92 * \left(99 - 49.5 * \left(0.98^{t} + 1\right) - t\right)$$

That last equation is presented in the wiki as simply
$d(t) = 3.92 * \left(99 - 49.5 * \left(0.98^{t} + 1\right) - t\right)$
However, this equation is incorrect since it interprets the velocity as a continuous increase instead of a stepwise increase. I have created a visualization of this difference here
https://www.desmos.com/calculator/8tji72kivp



$v(t) = \left(0.98^{t} - 1\right) \cdot 3.92$

^ This is the velocity expression as presented on MinecraftWiki. It is correct only if t is a round number (integer).

$v_2(t) = \left(0.98^{\text{floor}(t)} - 1\right) \cdot 3.92$

^ This velocity expression is accurate at all times, even at partial ticks. At tick 0.5 the player is still falling at the velocity calculated at tick 0. This means that the velocity from tick t to t+1 is constant, and equal to v(t).

The below table is calculated from the same expression in Google Sheets then pasted in so it can be shown as a scatter plot.

x_2	y_2
0	0
1	−0.0784
2	−0.155232
3	−0.23052736
4	−0.3043168128
5	−0.3766304765
6	−0.447497867
7	−0.5169479097
8	−0.5850089515
9	−0.6517087724
10	−0.717074597
11	−0.7811331051

Desmos does not plot floor functions with the vertical connectors correctly, so the blue step function is missing some connections. In that desmos chart, the red dots represent integral tick values, the green curve represents the velocity estimate from the wiki equation, and the blue step function is the correct velocity over time.

The correct equation is $d(t) = \int \left(0.98^{floor(t)} - 1\right) * 3.92\, dt$

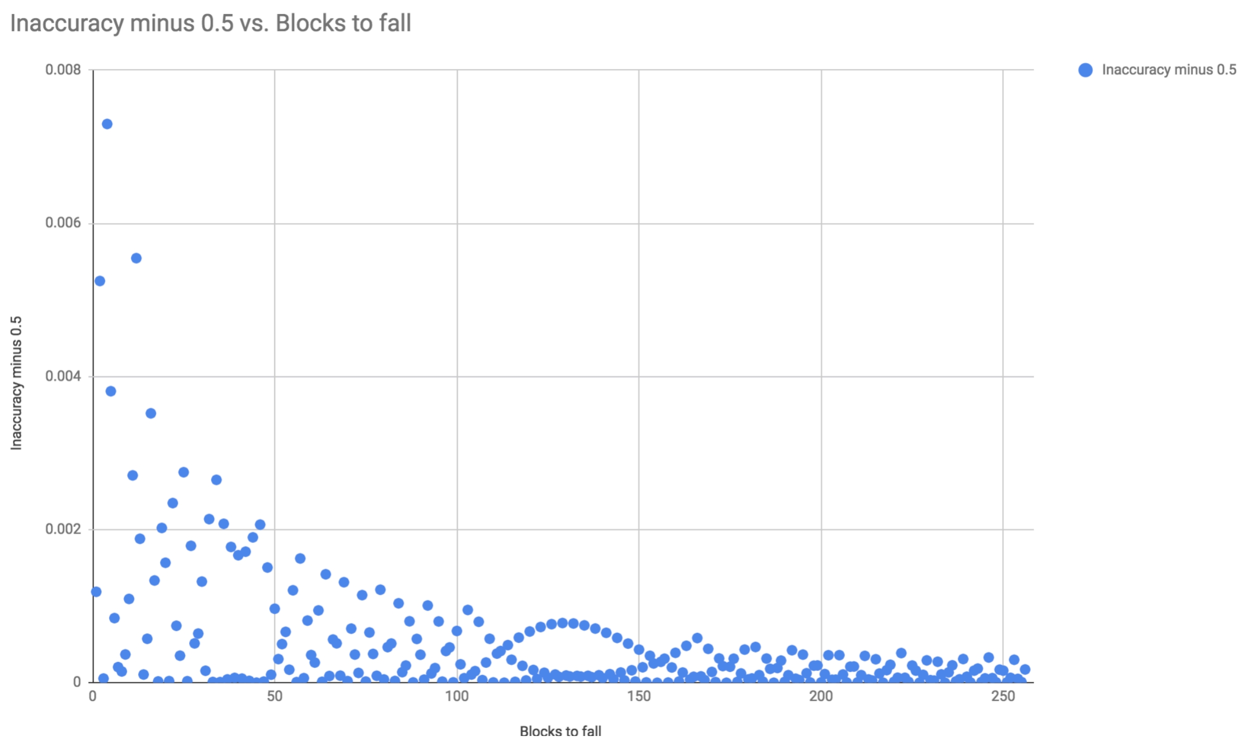
However, this is hard to calculate and harder to invert. **Is there a close estimate?**

In order to get there, we have to look at how big of an error the original wiki equation caused. **The tl;dr is it underestimates how many ticks it takes to fall by between 0.5 and 0.50729.** The error is shockingly close to 0.5 consistently, this is because the function is very close to linear, since the base of the exponent 0.98 is very close to 1.
For example, the original formula states it would take 8.968 ticks to fall three blocks, while it would actually take 9.468. In actual game time, that means it would take 10 ticks instead of 9, making the existing formula an **underestimate by 1/20th of a second**.

I have calculated the underestimate amounts here for every possible block distance to fall between 1 and 256:
https://docs.google.com/spreadsheets/d/1gDqU7kWZPAujbi02Dmkjnb5Ng-XiLRgYq1I3BBOHPHI/edit
114 of those fall distances have their durations underestimated by a tick in the existing formula! The other 142 are correct (but only when you round up to the nearest tick).

Those integration functions go from a tick count to a fall distance, in order to go the other way I numerically solved them for t in Mathematica, and verified the table values in Java (code at the bottom).

The errors are very consistently close to 0.5 or just above.
As you can see in the above spreadsheet, the differences are fascinating when plotted:
In this graph, the Y axis is the difference between the real formula and the estimate from the wiki, minus 0.5, and the X axis is the number of blocks to fall.



I have no explanation for why a pattern suddenly emerges from the chaos at around 100 blocks of falling.
However, that is the inversion. The difference being around 0.5 in the inversion means that offsetting the input by 0.5 should result in very similar output.

Therefore, we would expect

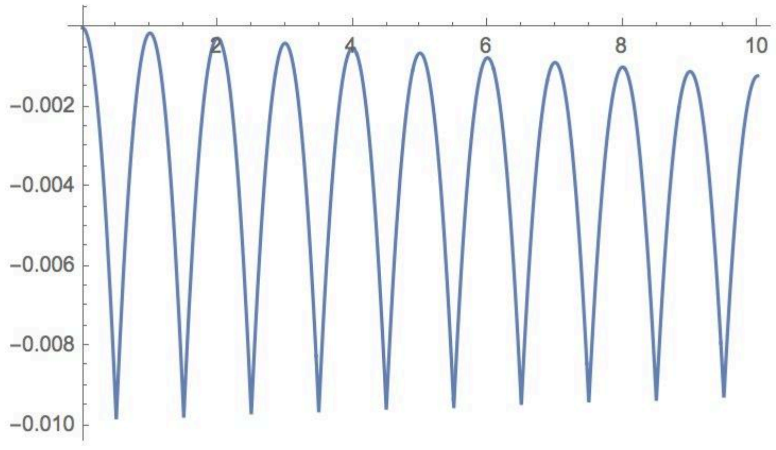
$$\int\left(0.98^{t-0.5} - 1\right) * 3.92\, dt$$

To be very similar to

$$\int\left(0.98^{floor(t)} - 1\right) * 3.92\, dt$$

And when plotted, the difference is consistently less than 0.01! The error seems to peak at every half integer tick count, which is fascinating.

```
Plot[NIntegrate[(0.98 ^ x - 1) * 3.92, {x, 0, t}] -
  NIntegrate[(0.98 ^ Floor[(x + 0.5)] - 1) * 3.92, {x, 0, t}], {t, 0, 10}]
```



Now, taking our improved estimate $d(t) = \int \left(0.98^{t-0.5} - 1\right) * 3.92\, dt$ and evaluating the integral, we get our final improved formula:
$d(t) = 196 - 3.92 * t - 194.04 * 0.98^{t-0.5}$

I verify the equivalency of those two in the spreadsheet, and that the tick count when rounded up is identical to this improved estimate for every integer block fall amount from 1 to 256 (see cell M259).

Assume these definitions:

```
  static double velocity(int ticks) {
    return (Math.pow(0.98, ticks) - 1) * -3.92;
  }

  static double oldFormulaTicksToBlocks(double ticks) {
    return -3.92 * (99 - 49.5 * (Math.pow(0.98, ticks) + 1) - ticks);
  }
```

This is my correct implementation:

```
static double blocksToTicks(double distance) {
    if (distance == 0) {
      return 0; // Avoid 0/0 NaN
    }
```

```
        int tickCount = 0;
        while (true) {
            double fallDistance = velocity(tickCount);
            if (distance <= fallDistance) {
                return tickCount + distance / fallDistance;
            }
            distance -= fallDistance;
            tickCount++;
        }
    }

public double ticksToBlocks(double ticks) {
        double fallDistance = 0;
        int integralComponent = (int) Math.floor(ticks);
        for (int tick = 0; tick < integralComponent; tick++) {
            fallDistance += velocity(tick);
        }
        double partialTickComponent = ticks - Math.floor(ticks);
        double finalPartialTickVelocity = velocity(integralComponent);
        double finalPartialTickDistance = finalPartialTickVelocity * partialTickComponent;
        fallDistance += finalPartialTickDistance;
        return fallDistance;
    }
```