If a Broken Clock is Right Twice a Day

Arian Nazarian

Figure 1: Damaged Omega Speedmaster 3221.30 — forever at 10:08:00 AM on Saturday, March 11[5].

Abstract

What are the odds of randomly checking a broken time-only 12-hour watch and discovering that it matches exactly with the current local time? This is a relatively straightforward calculation with two successes over the number of Bernoulli trials (in seconds, minutes, or hours) over a single day. What if the watch is an Omega Speedmaster 3221.30 "Day-Date," displaying the time, month, weekday, and date? How often does such a configuration align with reality and what is the probability of a perfect match when checked randomly?

These seemingly trivial questions may suggest inconsequential solutions to the uninitiated, but consider a traveler is moored in time, bound not by will but by a watch standing still. The hands are stuck, the calendar unchanged, yet this broken mechanism serves as their only guide. They may only journey to the rare intersections where time's machinery meets its motionless twin.

The irregular yet cyclical structure of the Gregorian calendar presents a range of possible answers, demanding a deeper examination of its governing mathematics. This paper explores the calendar's underlying mathematical framework, compares algorithmic approaches to computing date recurrences, and provides a comprehensive analysis of when and how often a fixed date-time combination reoccurs. Through a theoretical exploration of probabilistic structures and computational implementation, we establish intuitive and rigorous answers to these critical questions.

1 The Gregorian Calendar

The Gregorian calendar, adopted in 1582 and now formalised in the ISO 8601 standard,[4] was introduced by Pope Gregory XIII as a reform of the Julian calendar.[2;8]

The Julian system assumed

$$365.25 \text{ days},$$

whereas the actual tropical year is approximately

$$365.2422 \text{ days},$$

so the Julian calendar drifted by

$$365.25 - 365.2422 \approx 0.0078 \text{ days/year}.$$

Over the centuries this small discrepancy accumulated to roughly 10 days, displacing the spring equinox from its traditional date around 21 March. The 1582 reform therefore omitted 10 calendar days to realign the seasons.

In addition to this single adjustment, the Gregorian calendar introduced a modified leap-year rule to minimize future drift:[2;8]

- every year divisible by 4 is a leap year;
- except years divisible by 100, which are not leap years;
- unless they are also divisible by 400, in which case they remain leap years.

This rule yields 97 leap years in each 400-year cycle and an average year length of

$$\frac{(303 \times 365) + (97 \times 366)}{400} = 365.2425 \text{ days},$$ [2]

an error of only about one day every 3 300 years. Because a 400-year block contains exactly 146 097 days, it serves as the fundamental period for many calendar algorithms, including Zeller's congruence.[2;8]

2 Zeller's Congruence: Computing the Day of the Week

Zeller's congruence is an efficient algorithm for determining the weekday of any Gregorian date.[9] It is expressed as

$$h = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J \right) \bmod 7, \tag{1}$$

where

- h is the weekday ($0 =$ Saturday, $1 =$ Sunday, $\ldots$, $6 =$ Friday);
- q is the day of the month;
- m is the *modified* month: March $= 3$, $\ldots$, December $= 12$, January $= 13$, February $= 14$ (treated as months of the previous year);
- $K = Y \bmod 100$ (year within the century);
- $J = \lfloor Y/100 \rfloor$ (century).

Handling January and February

Because January and February are counted as months 13 and 14 of the *previous* year, the calendar year must be decremented by one whenever $m > 12$:

$$\text{if } m > 12, \qquad Y \leftarrow Y - 1.$$

This shift automatically aligns leap-year logic and century boundaries.

Alternative Formulation: Starting Months at One

If months use the conventional numbering (January = 1, ..., December = 12), let M be that month index. Zeller's formula becomes[9]

$$h = \left(q + \left\lfloor \frac{13(M+1+12L)}{5} \right\rfloor + (K-L) + \left\lfloor \frac{K-L}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J\right) \bmod 7,$$

where

$$L = \begin{cases} 1, & M \le 2 \text{ (January or February)}, \\ 0, & \text{otherwise}, \end{cases} \qquad K = (Y-L) \bmod 100, \qquad J = \lfloor (Y-L)/100 \rfloor.$$

The adjustment term L explicitly handles the leap-year boundary, albeit at the cost of a slightly more complex expression.

3 Direct Derivation from Gregorian Rules

An alternative approach derives a formula by directly following the Gregorian calendar's definitions. This method computes the total number of days elapsed since a fixed reference date (typically January 1 of year 1 in the proleptic Gregorian calendar) and then determines the day of the week by taking the result modulo 7.[2;8]

Constructing the Formula

Using q to denote the day of the month, the formula for computing the weekday is:

$$h = (1 + q + T(Y) + M_{\text{offset}}(M, Y)) \mod 7,$$

where:

- h represents the day of the week (with a chosen correspondence to weekdays).
- The term 1 accounts for the reference date offset.
- $T(Y)$ represents the total number of days from all full years before Y.
- $M_{\text{offset}}(M, Y)$ represents the total number of days from completed months in year Y.

Counting Days from Full Years

The total number of days contributed by past years up to year Y is:

$$T(Y) = (Y-1) \times 365 + L(Y-1).$$

Here, $L(Y)$ accounts for leap years, calculated as:

$$L(Y) = \lfloor Y/4 \rfloor - \lfloor Y/100 \rfloor + \lfloor Y/400 \rfloor.$$

Accumulating Days from Completed Months

The cumulative number of days contributed by past months in year Y is given by:

$$M_{\text{offset}}(M, Y) = \begin{cases} 0, & M = 1 \text{ (January)} \\ 31, & M = 2 \text{ (February)} \\ 31 + 28 + L(Y), & M = 3 \text{ (March)} \\ 31 + 28 + L(Y) + 31, & M = 4 \text{ (April)} \\ 31 + 28 + L(Y) + 31 + 30, & M = 5 \text{ (May)} \\ 31 + 28 + L(Y) + 31 + 30 + 31, & M = 6 \text{ (June)} \\ 31 + 28 + L(Y) + 31 + 30 + 31 + 30, & M = 7 \text{ (July)} \\ 31 + 28 + L(Y) + 31 + 30 + 31 + 30 + 31, & M = 8 \text{ (August)} \\ 31 + 28 + L(Y) + 31 + 30 + 31 + 30 + 31 + 31, & M = 9 \text{ (September)} \\ 31 + 28 + L(Y) + 31 + 30 + 31 + 30 + 31 + 31 + 30, & M = 10 \text{ (October)} \\ 31 + 28 + L(Y) + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31, & M = 11 \text{ (November)} \\ 31 + 28 + L(Y) + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 30, & M = 12 \text{ (December)} \end{cases}$$

For months beyond February, the function automatically accounts for leap years by adding $L(Y)$. Although the piecewise definition of $M_{\text{offset}}(M, Y)$ appears unwieldy, it clearly enumerates the cumulative days leading up to each month.

Why Modulo 7 Determines the Weekday

Since the formula ultimately relies on taking the result modulo 7, let us examine why this step works:

- The sequence of weekdays follows a repeating cycle every 7 days.
- A normal year contains 365 days, which shifts the weekday forward by $365 \bmod 7 = 1$.
- A leap year contains 366 days, shifting the weekday forward by $366 \bmod 7 = 2$.
- The leap-year function $L(Y)$ adjusts for extra leap days, preventing long-term drift.

Thus, by summing the total days and computing modulo 7, we effectively "wrap around" to obtain the correct weekday.

Comparison with Zeller's Congruence

While both this direct derivation and Zeller's congruence ultimately determine the day of the week, they differ in approach and computational efficiency.

- The direct derivation explicitly follows the Gregorian calendar's structure, computing total elapsed days from a fixed reference date. It provides a conceptual and systematic understanding of how weekdays cycle over time but requires separate handling of year, month, and day contributions.
- Zeller's congruence, on the other hand, is a more compact formula that applies modular arithmetic directly to a given date without explicitly summing total elapsed days. It incorporates a pre-adjusted month numbering system and operates in a form that is computationally efficient for quick lookups.
- The direct derivation is particularly useful for historical and educational purposes, as it breaks down the logic step by step, whereas Zeller's congruence is optimized for direct calculation in programming and algorithmic implementations.

This derivation follows the structure of the Gregorian calendar directly, ensuring an intuitive calculation of the weekday for any given date. It highlights how the combination of a fixed year length and the leap-year adjustments yields a cyclic behavior modulo 7—a fundamental aspect also utilized elegantly in Zeller's congruence.[2;8]

4 Manipulating Zeller's Congruence to Find Matching Years

Zeller's congruence is typically used to compute the day of the week for a given date. However, it can also be rearranged to find all years within a specified range in which a particular weekday, month, and date coincide.[9]

Rewriting the Formula

The standard form of Zeller's congruence for a date (Y, M, q) is:

$$h = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J \right) \bmod 7,$$

where

- h is the day of the week (0 = Saturday, 1 = Sunday, ..., 6 = Friday);
- q is the day of the month;
- m is the modified month (March = 3, ..., December = 12; January = 13, February = 14 of the previous year);

- $K = Y \bmod 100$;
- $J = \lfloor Y/100 \rfloor$.

Given a target weekday h_{target}, we seek all years Y that satisfy

$$h_{\text{target}} = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J \right) \bmod 7.$$

Isolating Year Components

Expressing K and J explicitly as functions of Y:

$$K = Y \bmod 100, \qquad J = \lfloor Y/100 \rfloor,$$

and substituting gives

$$h_{\text{target}} = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + (Y \bmod 100) + \left\lfloor \frac{Y \bmod 100}{4} \right\rfloor + \left\lfloor \frac{\lfloor Y/100 \rfloor}{4} \right\rfloor - 2\lfloor Y/100 \rfloor \right) \bmod 7,$$

which rearranges to the congruence

$$(Y \bmod 100) + \left\lfloor \frac{Y \bmod 100}{4} \right\rfloor + \left\lfloor \frac{\lfloor Y/100 \rfloor}{4} \right\rfloor - 2\lfloor Y/100 \rfloor \equiv h_{\text{target}} - q - \left\lfloor \tfrac{13(m+1)}{5} \right\rfloor \pmod 7.$$

Algorithm for Finding Matching Years

Rather than solving this congruence algebraically, a practical method iterates over a range of years and selects those that satisfy the condition. Observe

- a common year advances the weekday by +1;
- a leap year advances it by +2;

leading to weekday cycles of length 5, 6, or 11 years. A straightforward algorithm is therefore

1. iterate through a specified range of years;
2. for each year compute h via Zeller's congruence;
3. if $h = h_{\text{target}}$, record the year.

This iterative approach efficiently returns every year in which the chosen weekday, month, and date coincide.

5 Comparison of Algorithm Implementations

This section contrasts two approaches for locating every year in which a given weekday, month, and date coincide in the Gregorian calendar: a Python routine based on Zeller's congruence[9] and a JavaScript routine that relies on the built-in `Date` object defined by the ECMAScript specification.[3]

Feature	**Python (Zeller)**	**JavaScript (`Date`)**
Month numbering	March = 3,. . . , December = 12; Jan = 13, Feb = 14 (prev. year)	Jan = 0,. . . , December = 11 (ECMA-262)
Weekday numbering	0 = Saturday, . . . , 6 = Friday	0 = Sunday, . . . , 6 = Saturday (ECMA-262)
Leap-year handling	Implicit in Zeller's arithmetic	Engine follows Gregorian rule
Century rollover	Handled via Jan/Feb shift	Built into `Date` arithmetic
Computation style	Pure modular arithmetic	Object-oriented date ops
Performance	Faster for bulk scans	Simpler for single queries

Table 1: Python implementation of Zeller's congruence vs. ECMAScript `Date`.

Both methods correctly identify, for example, every year in which 11 March falls on a Saturday.

Python Script with Zeller's Congruence

The listing below applies Zeller's congruence (using months 3–14) to scan a year range.[6]

Listing 1: Python implementation of Zeller's congruence for matching years

```python
def find_matching_years(target_weekday, month, day, start_year, end_year):
    def zellers_congruence(day, month, year):
        if month < 3:
            month += 12 # January becomes 13, February becomes 14
            year -= 1 # Adjust the year for these months

        K = year % 100 # Year within the century
        J = year // 100 # Century number

        # Zeller's formula
        h = (day + (13 * (month + 1)) // 5 + K + (K // 4) + (J // 4) - 2 * J) % 7

        return h # 0=Saturday, 1=Sunday, ..., 6=Friday

    matching_years = []
    for year in range(start_year, end_year + 1):
        if zellers_congruence(day, month, year) == target_weekday:
            matching_years.append(year)

    return matching_years

# Find years where March 11 is a Saturday
print(find_matching_years(0, 3, 11, 2001, 2100))
```

- January and February are treated as months 13 and 14 of the previous year.
- Century and leap-year corrections are implicit.
- The script iterates through a range and returns all years whose weekday matches.

JavaScript Using the `Date` Object

The ECMAScript `Date` constructor provides leap-year and century handling out of the box.[3]

Listing 2: JavaScript implementation using the built-in `Date` object

```javascript
function findMatchingYears(targetWeekday, month, day, startYear, endYear) {
    let matchingYears = [];

    for (let year = startYear; year <= endYear; year++) {
        let date = new Date(year, month - 1, day); // JavaScript months are 0-based
        // Note: date.getDay() returns the day of the week for the given date,
        // where 0 represents Sunday, 1 represents Monday, ..., and 6 represents Saturday.
        if (date.getDay() === targetWeekday) {
            matchingYears.push(year);
        }
    }

    return matchingYears;
}

// Example Usage: Find years where March 11 is a Saturday
// Remember: Since getDay() returns 6 for Saturday, we pass 6 as the targetWeekday.
console.log("Years␣where␣March␣11␣is␣a␣Saturday:",
  findMatchingYears(6, 3, 11, 2001, 2100));

// Output: [2006, 2017, 2028, 2034, 2045, 2051, 2056, 2062, 2073, 2079, 2084, 2090]
```

- Relies on the engine's internal Gregorian algorithm.
- Requires only one line of code to obtain the weekday.
- Iterates over a range and collects matching years.

Zeller's modular arithmetic excels at large, offline sweeps, whereas the JavaScript approach is ideal for real-time, browser-based lookups.

6 Probabilistic Analysis and Conclusion

Foundations: Probability Tools for Fixed Date–Time Recurrences

- **Bernoulli trial**. A single 0/1 experiment with success probability p.
- **Geometric distribution**. Waiting time T until the first success: $\Pr(T = t) = (1-p)^{t-1}p,\ E[T] = 1/p,\ \mathrm{Var}[T] = (1-p)/p^2$.[7]
- **Uniform counting**. Once the number of equally likely calendar states is known, $p = \dfrac{\text{favourable}}{\text{total}}$.
- **Zeller's congruence**. Derived in Section 2, it maps (Y, M, D) to a weekday; we reuse it to tally weekday frequencies.[9]

Time-Only Watches (12-Hour Dial)

Let the frozen dial read h:m:s.

Display precision	Successes per day	p
Hours	2	$2/12 = 1/6$
Hours + Minutes	2	$2/(12 \times 60) = 1/360$
Hours + Min + Sec	2	$2/(12 \times 60 \times 60) = 1/21\,600$

The factor "2" is the familiar AM/PM duplication. A 24-hour dial halves each p.

Adding the Calendar One Field at a Time

A full Gregorian cycle contains 146 097 days × 86 400 seconds =

$$N = 12\,582\,912\,800$$

distinct second-level states.

Time + *Date.* All civil dates appear, so the sample space is N.

Time + Date + *Weekday.* Weekday is fixed by the date, so the space is still N.

Time + Date + Weekday + *Month.* Month is implicit in the date; the space remains N. From here onward we focus on *(month, date, weekday)* triples; the granularity of "time of day" merely rescales numerator and denominator by the same factor and leaves the probability unchanged.

Probability of the Sample Display

The Omega Speedmaster in Fig. 1 is frozen at

$$\boxed{\text{Saturday, 11 March, 10:08:00 AM}}.$$

400-year theoretical cycle.

- **Weekday–date matches.** Zeller's congruence shows that 11 March is a Saturday in 56 of the 400 years.[9]
- **Strict match (time included).** Two matching seconds per qualifying date $\Rightarrow s_{\text{time}} = 56 \times 2 = 112$.
- **Calendar-only match (time ignored).** $s_{\text{cal}} = 56 \times 86\,400 = 4\,838\,400$.

$$p_{\text{strict}} = \frac{112}{N}, \qquad p_{\text{calendar}} = \frac{4\,838\,400}{N}, \qquad E[T_{\text{calendar}}] \approx 2.6 \times 10^3 \text{ glances}^*.$$

*This expectation assumes the watch is inspected once per minute without interruption. If the observer sleeps eight hours per day, the calendar-only waiting time increases by roughly 50%. Because the strict date-and-time probability is so small, human scheduling has negligible relative effect.

Civil record 1583–2024 (empirical).

$$\#\big(\text{Saturday, 11 March}\big) = 62, \quad s_{\text{time, emp}} = 124, \quad s_{\text{cal, emp}} = 62 \times 86\,400.$$

Total seconds in that interval:

$$N_{\text{emp}} = 86\,400 \times 161\,438 = 13\,948\,243\,200.$$

$$p_{\text{calendar, emp}} = \frac{s_{\text{cal, emp}}}{N_{\text{emp}}}, \qquad p_{\text{strict, emp}} = \frac{s_{\text{time, emp}}}{N_{\text{emp}}}.$$

Numerically $p_{\text{calendar, emp}} \approx 3.8 \times 10^{-4}$ (≈ 1 in 2 600) and $p_{\text{strict, emp}} \approx 8.9 \times 10^{-9}$ (≈ 1 in 1.1×10^8); both align with the 400-year averages to three significant digits.

Most and Least Common (Weekday, Month, Date) Triples

Ideal 400-year cycle. For any non-leap-day date MD,

$$400 = 57 \times 7 + 1 \implies 57 \text{ or } 58 \text{ occurrences per weekday.}^{1}$$

For 29 February,

$$97 = 13 \times 7 + 6 \implies 13 \text{ or } 14 \text{ per weekday.}$$

$$\boxed{\text{Most frequent triple: } 58/N}, \qquad \boxed{\text{Least frequent valid triple: } 13/N}.$$

Triples with impossible dates (e.g. 31 November) occur zero times.

Civil record 1583–2024. With 442 complete years,

$$442 = 63 \times 7 + 1, \qquad 108 = 15 \times 7 + 3 \text{ (leap-day count).}$$

$$\#_{\max} = 64, \qquad \#_{\min} = 15, \qquad p_{\max,\text{emp}} = \frac{64}{N_{\text{emp}}}, \quad p_{\min,\text{emp}} = \frac{15}{N_{\text{emp}}}.$$

The ranking is unchanged: 1 January triples are most common; 29 February triples are rarest.

Combining the combinatorial structure of the Gregorian calendar with Bernoulli geometry yields transparent, exact probabilities for any frozen watch display. Time-only odds scale predictably with display precision; once calendar fields are included, the likelihood of a perfect match is governed almost entirely by how often a given weekday–date pair can legally occur. For the damaged Omega, the chance of a full date-and-time coincidence is about one in a hundred million per glance.

The numerical results lead to three practical uses. *Forensic analysis.* Section 6 supplies an exact probability for any frozen display; a log that ends on a date-time whose chance is one in 10^8 is unlikely to be accidental, giving investigators an objective likelihood ratio. *Low-power devices.* Because the same weekday–date–month triple can be absent for up to 32 years, an e-ink badge or remote sensor can hibernate until its rare "anniversary" day, waking roughly 14 times per century and stretching a coin-cell's life. *Software testing.* Section 4 shows there are exactly 56 Saturdays on 11 March in every 400-year block; a calendar library that returns 55 or 57 has a leap-year error, so this single query serves as a regression check.

Each application drops straight out of the closed-form counts proved in this paper and needs no extra theory.

Finally, to answer the two questions posed in the abstract: for the damaged Omega Speedmaster display, the calendar fields align with reality 56 times in a 400-year cycle—about once every seven years—and a random glance has a strict match probability of roughly 9×10^{-9}, or one in 1.1×10^8 glances.

References

[1] John H. Conway. The doomsday algorithm. *The Mathematical Intelligencer*, 23(2):64–67, 1997.

[2] Nachum Dershowitz and Edward M. Reingold. *Calendrical Calculations*. Cambridge University Press, Cambridge, UK, 3rd edition, 2008. See p.45 for discussion on the Gregorian calendar reform and leap year rules.

[3] Ecma International. ECMAScript 2024 language specification (15th edition), 2024. URL `https://tc39.es/ecma262/2024/`. Standard.

[4] International Organization for Standardization. ISO 8601-1:2019 — date and time – representations for information interchange, 2019. Standard.

[5] Omega SA. Omega speedmaster date & day date chronograph 40 mm. `https://www.omegawatches.com/en-us/watch-omega-speedmaster-date-day-date-chronograph-40-mm-day-date-32213000`, n.d. Image modified for illustrative purposes. Original image available at `https://www.omegawatches.com/media/catalog/product/o/m/omega-speedmaster-date-_-day-date-chronograph-40-mm-day-date-32213000-55a8b0.png?w=1100`. Omega is a registered trademark of Omega SA.

[6] Python Software Foundation. *datetime — Basic Date and Time Types (Python 3.12 Documentation)*. Python Software Foundation, 2023. URL `https://docs.python.org/3/library/datetime.html`.

[7] Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, 12 edition, 2019. ISBN 9780128143469.

[8] United States Naval Observatory. Introduction to calendars. `https://www.usno.navy.mil/USNO/astronomical-applications/astronomical-information-center/calendar-information`, n.d. Retrieved 9 May 2022.

[9] Christian Zeller. Kalender-formeln. *Acta Mathematica*, 9:131–136, 1886. doi: 10.1007/BF02406733.