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[2].

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 cyclical yet irregular 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 was introduced by Pope Gregory XIII in 1582 as a reform of the Julian calendar[1;3]. The Julian calendar assumed a year length of

$$365.25 \text{ days},$$

but the actual tropical year is approximately

$$365.2422 \text{ days},$$

resulting in an annual drift of

$$365.25 - 365.2422 \approx 0.0078 \text{ days/year}^{1}.$$

Over the centuries, this small error accumulated to a drift of roughly 10 days, which caused the spring equinox to shift significantly from its traditional date around March 21[1]. To realign the calendar with the astronomical seasons, the Gregorian reform omitted 10 days in October 1582[1].

In addition to this one-time correction, the Gregorian calendar introduced a modified leap year system to minimize future drift[1;3]. Under this system, a year is a leap year if:

- It is divisible by 4;
- Except if it is divisible by 100 in which case it is not a leap year, unless;
- It is also divisible by 400, in which case it remains a leap year.

This rule produces 97 leap years in every 400-year cycle, yielding an average year length of

$$\frac{(303 \times 365) + (97 \times 366)}{400} = 365.2425 \text{ days}^{1}.$$

This average is extremely close to the tropical year, resulting in an error of only about one day every 3,300 years[1]. The 400-year cycle, comprising exactly 146097 days, is a fundamental feature that ensures the calendar's long-term consistency—a fact that underpins many calendar algorithms, including Zeller's congruence[1;3].

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

Zeller's congruence is an efficient algorithm for determining the day of the week for any given date in the Gregorian calendar[4]. It is expressed as:

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

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, defined as follows:
  - March = 3, April = 4, ..., December = 12.
  - January = 13 and February = 14 (treated as months of the previous year).
- $K = Y \mod 100$ (the year within the century).
- $J = \lfloor Y/100 \rfloor$ (the century number).

2.1 Handling January and February

Since January and February are treated as months 13 and 14 of the previous year, the year must be decremented by one for these cases[4]:

$$\text{If } m > 12, \quad Y \leftarrow Y - 1. \tag{2}$$

This adjustment ensures that leap years and century transitions are correctly accounted for in the calculation[4].

2.2 Alternative Formulation: Starting Months at One

If we define months using the conventional numbering system (i.e., January = 1, February = 2, ..., December = 12), we must explicitly adjust how J and K are defined. In this case, let M denote the conventional month number. The formula becomes:

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

where

$$L = \begin{cases} 1, & \text{if } M \leq 2 \text{ (i.e., January or February)} \\ 0, & \text{otherwise,} \end{cases}$$

and the adjusted values of K and J are given by:

$$K = (Y - L) \mod 100, \quad J = \lfloor (Y-L)/100 \rfloor.$$

This version explicitly handles leap years by introducing the adjustment term L, though it is computationally more complex[4].

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[1;3].

3.1 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[1].
- $T(Y)$ 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.

3.2 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:

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

This function follows the Gregorian leap year rules, ensuring long-term accuracy[1].

3.3 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 $L(Y)$[1]. Although the piecewise definition of $M_{\text{offset}}(M,Y)$ appears unwieldy, it clearly enumerates the cumulative days leading up to each month.

3.4 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 $365 \mod 7 = 1$.
- A leap year contains 366 days, shifting the weekday forward by $366 \mod 7 = 2$.
- 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.

3.5 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[1;3].

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 where a particular weekday, month, and day coincide[4].

4.1 Rewriting the Formula

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

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

Here,

- 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),
- $K = Y \mod 100$, and
- $J = \lfloor Y/100 \rfloor$.

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

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

This formulation follows directly from Zeller's original derivation[4].

4.2 Isolating Year Components

Expressing K and J explicitly as functions of Y:

$$K = Y \mod 100, \quad J = \lfloor Y/100 \rfloor,$$

we substitute these into the equation to obtain:

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

Rearranging, we express the condition on Y as:

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

This rearrangement, following Zeller's methodology, provides the basis for an algorithm to search for matching years[4].

4.3 Algorithm for Finding Matching Years

Rather than solving the above congruence algebraically, a practical method is to iterate over a range of years and select those that satisfy the condition[4]. Note that:

- In a normal year, the day of the week advances by 1.
- In a leap year, the day of the week advances by 2.

This leads to weekday cycles of 5, 6, or 11 years. The algorithm is as follows:

1. Iterate through a specified range of years.
2. For each year, compute h using Zeller's congruence[4].
3. If h matches the target weekday h_{target}, include that year in the results.

This iterative approach provides an efficient method for determining all years in which the specified date falls on the desired weekday[4].

5 Comparison of Algorithm Implementations

This section compares two different approaches for computing the years in which a given weekday, month, and date coincide in the Gregorian calendar[4].

Feature	**Python (Zeller's congruence)**[4]	**JavaScript (Date Object)**
Month Numbering	March = 3, ..., December = 12; Jan = 13, Feb = 14 (Previous Year)	Jan = 0, ..., December = 11 (Standard)
Weekday Numbering	0 = Saturday, 1 = Sunday, ..., 6 = Friday	0 = Sunday, 1 = Monday, ..., 6 = Saturday
Leap Year Handling	Implicitly handled	Handled by JavaScript engine
Century Calculation	Adjusts automatically via shifting Jan/Feb	Handled by built-in `Date()` method
Computation	Modular arithmetic	JavaScript object-based date operations
Performance	Faster for bulk calculations	Convenient for single-date calculations

Table 1: Comparison of Zeller's congruence vs. JavaScript Date Method

Both methods find all matching years in a given range where, for example, March 11 falls on a Saturday[4].

5.1 Python Script with Zeller's Congruence

This Python implementation of Zeller's congruence uses the convention of [March = 3, ..., December = 12] and [January = 13, February = 14 (previous year)] to find all years where a given weekday matches the given month and date[4].

Listing 1: Python Implementation of Zeller's congruence for Matching Years

```
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 shifted to months 13 and 14 of the previous year.
- The algorithm maintains the correct century and leap year calculations.
- It iterates through a range of years and returns all years where the specified date matches the specified weekday.

5.2 JavaScript Using Date Object

This JavaScript implementation calculates all years within a stated range where a given weekday matches the given month and date using the built-in `Date()` object. The Date object is part of the ECMAScript standard and provides a convenient method for single-date calculations.

Listing 2: JavaScript Implementation Using `Date()` Object

```
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]
```

- Uses the JavaScript `Date()` object to handle date computations.
- Automatically accounts for leap years and century calculations.
- Iterates through a range of years and returns all matching years.

Although Zeller's congruence is optimized for bulk calculations due to its modular arithmetic properties[4], the JavaScript method provides a convenient alternative for real-time applications. For efficiency, the Python approach is better suited for analyzing large datasets of dates, while JavaScript is preferable for web-based applications.

References

[1] 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.

[2] 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.

[3] 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.

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