Closed-Form Calendar Deltas

Richard Todd

ABSTRACT

The task at hand is to count the number of days between arbitrary dates, which can span centuries, as part of a Discordian Calendar lua module. Code I had found on the internet just looped across years, checking for leap years on each iteration, which seemed overly wasteful. I wanted to explore the possibility of a more closed-form solution—or at least I hoped to avoid so much iteration—using the Gregorian system for leap years. A derivation of a simple formula is outlined in this report.

Today's exploration is: how many days, Δ, are between two arbitrary dates? In writing a fun lua module for the Discordian Calendar, I had to write a function to calculate the number of days until the "X-Day", on July 5th, 8661 (which is 1998 upside-down!). The code of the common linux utility **ddate** just loops through the years, checking for a leap year on every iteration, and counting the days. This is fast enough for one-off uses, but it made me wonder how hard it would be to find a more closed-form solution.

This report derives a nice formula, step-by-step, which performs much faster than the original code.

Formula Derivation

I denote dates by pairs (y, d), where y is the year and d is the day of the year (from 0 to 364, or 365 on leap years). So, given a start date (y_s, d_s) and an end date (y_e, d_e), what is the fastest way to compute the number of days in the interval?

For a first estimate, I forget leap years, and also assume the days are always Jan 1st on both ends. Then, the obvious solution is:

$$\Delta = 365(y_e - y_s) \tag{1}$$

Equation (1) gives a count that is too large by the number of days that have already passed in y_s, and too small by the count of days that were not counted in y_e. Therefore, it's easy to add in those terms to correct for the partial start and end years:

$$\Delta = 365(y_e - y_s) + d_e - d_s \tag{2}$$

Equation (2) is accurate except that it doesn't account for the extra day given to leap years. Different calendars have used different schemes for leap years, but I'll be using the Gregorian scheme here. A leap year is a year that matches one of these two rules:

- the year is divisible by 4, but not by 100
- the year is divisible by 400

So, I need to augment (2) with a correction term, c, which is the number of leap years in the interval $[y_s, y_e)$. The exception for years divisible by 100 gets in the way of a simple formula, so to avoid `if/then` logic, I reorganized the leap-year conditions into three counts which I can combine to get the right number:

- number of years divisible by 4 in the interval (call it c_4)
- number of years divisible by 100 in the interval (call it c_{100})
- number of years divisible by 400 in the interval (call it c_{400})

It's easy to see that the number of Gregorian leap years, c, can be given by $c_4 - c_{100} + c_{400}$. In other words, if I can find a closed-form way to compute those terms, the complete formula will be:

$$\Delta = 365(y_e - y_s) + d_e - d_s + c_4 - c_{100} + c_{400} \tag{3}$$

I started with the c_4 term, which is the number of years divisible by 4 on the interval $[y_s, y_e)$. Obviously one can get close with integer division by 4, but that would be short by one in many cases (for example, 2020 to 2025 is a span of 5 years, but contains two leap years). The first formula I could find was:

$$c_4 = \left\lfloor \frac{y_e - y_s}{4} \right\rfloor + \left\lceil \frac{(y_e - y_s) \bmod 4 - (-y_s \bmod 4)}{4} \right\rceil \tag{4}$$

The term on the left is integer division, and the term on the right corrects for the edge cases. Happily, it turns out that this formula can be drastically simplified. First, I expanded the modulus operators into yet more floor terms:

$$c_4 = \left\lfloor \frac{y_e - y_s}{4} \right\rfloor + \left\lceil \frac{y_e - y_s - 4\left\lfloor \frac{y_e - y_s}{4} \right\rfloor + y_s + 4\left\lfloor \frac{-y_s}{4} \right\rfloor}{4} \right\rceil$$

Then, I pulled the first term into the bigger ceiling computation:

$$c_4 = \left\lceil \frac{4\left\lfloor \frac{y_e - y_s}{4} \right\rfloor + y_e - y_s - 4\left\lfloor \frac{y_e - y_s}{4} \right\rfloor + y_s + 4\left\lfloor \frac{-y_s}{4} \right\rfloor}{4} \right\rceil$$

Now, I cancel out terms:

$$c_4 = \left\lceil \frac{y_e + 4\left\lfloor \frac{-y_s}{4} \right\rfloor}{4} \right\rceil$$

Split into two fractions:

$$c_4 = \left\lceil \frac{y_e}{4} + \left\lfloor \frac{-y_s}{4} \right\rfloor \right\rceil$$

Finally, I pulled apart the two terms and converted them both to ceiling form for symmetry:

$$c_4 = \left\lceil \frac{y_e}{4} \right\rceil - \left\lceil \frac{y_s}{4} \right\rceil \tag{5}$$

That's unbelievably simple compared to the original equation (4)! So much so, that I wrote a test program to compare their results over a wide range of inputs. Fortunately, they match. Even more fortunately, c_{100} and c_{400} have exactly the same form, so I could jump directly to the short form for those:

$$c_{100} = \left\lceil \frac{y_e}{100} \right\rceil - \left\lceil \frac{y_s}{100} \right\rceil \tag{6}$$

$$c_{400} = \left\lceil \frac{y_e}{400} \right\rceil - \left\lceil \frac{y_s}{400} \right\rceil \tag{7}$$

The code I wrote (in lua, in this case), follows from equation (3) directly, with the c-terms expanded out:

```
local ceil = math.ceil
local delta = (365 * (year_e - year_s) + day_e - day_s
        + ceil(year_e/4)   - ceil(year_s/4)
        - ceil(year_e/100) + ceil(year_s/100)
        + ceil(year_e/400) - ceil(year_s/400))
```

This was a very satisfying exercise, and now I can compute results spanning millennia with no loops in sight.