Obliczanie wschodów i zachodów Słońca

ebvalaim 2015-12-22 1 Comment

Aktualności

haskell

Dziś przesilenie zimowe - najkrótszy dzień w roku, a najdłuższa noc. Nie każdemu jednak znany jest fakt, że mimo iż dni będą już coraz dłuższe, to jednak Słońce nadal będzie wschodziło coraz później. Jest to spowodowane kształtem orbity Ziemi, który nie jest idealnie okrągły, a eliptyczny. Teraz zbliża się moment, w którym Ziemia będzie najbliżej Słońca w ciągu swojej rocznej wędrówki, a przez to również porusza się szybciej niż zwykle. To z kolei opóźnia południe słoneczne każdego dnia o kilkanaście sekund, powodując, że godziny wschodów są wciąż coraz późniejsze.

Kiedy zatem mamy najpóźniejszy wschód i najwcześniejszy zachód, jeśli nie w przesilenie? Mógłbym pewnie sprawdzić gdzieś w internecie, ale po co, skoro mogę policzyć sam przy pomocy komputera ;)

Do realizacji zadania wybrałem Haskella - głównie dlatego, że wciąż słabo go ogarniam, a jest to dla mnie bardzo ciekawy i zmieniający sposób myślenia język. Zdecydowałem więc, że warto go poćwiczyć.

Etap pierwszy - znalezienie odpowiednich równań

Jakkolwiek sporo zależności da się wyprowadzić z podstawowych zasad, wiele wynika z faktów obserwacyjnych, więc bez internetu się nie obejdzie. Np. każdy wie, że równonoc wiosenna wypada 21 marca (co, nawiasem mówiąc, przestało być prawdą ok. roku 2000 - teraz przypada ona 20 marca, wróci na 21 ok. roku 2100), jednak mniej oczywiste jest już, jak daleko od Słońca znajduje się wtedy Ziemia i jak szybko się porusza. Tyle wystarczyłoby, żeby liczyć długość dnia z dokładnością do kilku minut, jednak skoro chcemy liczyć takie szczegóły jak przesuwanie się południa słonecznego, trzeba to zrobić porządniej.

Znalezienie danych nie jest trudne - praktycznie od razu Google odsyła do artykułu na Wikipedii. Jest tam wypisane praktycznie krok po kroku co trzeba policzyć, choć oczywiście warto wiedzieć, o co w tym wszystkim chodzi ;) W szczególności dość ważne są pojęcia *rektascensji* i *deklinacji*. Już dawno temu ludzie wpadli na pomysł, że przydałby im się system opisywania położeń gwiazd i planet na niebie. Problem w tym, że niebo pozornie cały czas się obraca (co, jak wiemy, jest efektem ruchu obrotowego Ziemi). Pojawił się wobec tego pomysł wprowadzenia układu współrzędnych niebieskich niezależnych od tego ruchu, nieruchomych względem gwiazd. Idea jest bardzo prosta.

Po pierwsze, wprowadzamy na niebie odpowiednik ziemskiej szerokości geograficznej, zwany deklinacją. Deklinacja +90° odpowiada północnemu biegunowi niebieskiemu, tj. punktowi, w który celuje oś Ziemi "wychodząc" z bieguna północnego. Deklinacja -90° to południowy biegun niebieski, 0° to rzut równika ziemskiego na sferę niebieską. To załatwia jedną współrzędną.

Druga współrzędna to rektascensja, odpowiednik długości geograficznej. Tak, jak na Ziemi mamy arbitralnie wybrany południk zerowy w Greenwich, tak na niebie rektascensji 0 odpowiada tzw. punkt Barana. Jest to punkt przecięcia ekliptyki (płaszczyzny orbity Ziemi, czy też - względem Ziemi - rocznego toru Słońca po niebie) z równikiem niebieskim. Przez ten punkt Słońce przechodzi w momencie równonocy wiosennej. Po przeciwnej stronie nieba mamy drugie przecięcie ekliptyki z równikiem - punkt Wagi.

Dla odmiany, rektascensję liczy się nie w stopniach, ale w godzinach, minutach i sekundach i przyjmuje ona wartości od 0h do 24h. Ma to swoje uzasadnienie - rektascensja oznacza bowiem,

ile czasu mija, nim dany punkt na równiku znajdzie się w tym samym punkcie nieba, w którym w danym momencie jest punkt Barana.

Jak już wspomniałem, odzwierciedleniem ruchu orbitalnego Ziemi jest pozorny ruch Słońca po ekliptyce. Znając współrzędne niebieskie (rektascensję i deklinację) Słońca, współrzędne geograficzne miejsca obserwacji i wiedząc, który południk niebieski góruje w danym momencie możemy obliczyć, kiedy Słońce wzejdzie, a kiedy zajdzie.

Dokładnie tych danych dostarczają nam równania z Wikipedii. Po przeliczeniu daty i godziny na tzw. datę juliańską (Julian Date), otrzymujemy współrzędne ekliptyczne Słońca (λ i β na Wiki), które następnie przeliczamy na rektascensję (α) i deklinację (δ), korzystając z kąta nachylenia osi ziemskiej do płaszczyzny orbity (ϵ). Niejako za darmo otrzymujemy przy tym odległość Ziemi od Słońca w danym momencie ;)
To, co bardzo się przyda do obliczenia wschodów i zachodów, to wysokość Słońca nad horyzontem (która jest 0 podczas wschodu i zachodu). Po te wartości Wikipedia odsyła nas do artykułów dot. kąta zenitalnego oraz azymutu. Najprościej rzecz ujmując, te dwa kąty tworzą układ współrzędnych sferycznych, w którym to horyzont pełni rolę równika, zenit jest biegunem północnym, nadir - południowym, a kierunek północny wskazuje południk 0.
Oczywiście to wszystko jest nie do obliczenia, jeśli nie znamy orientacji Ziemi w przestrzeni. Pojawiają się tu takie pojęcia, jak kąt godzinny i czas gwiazdowy. Czas gwiazdowy to nic innego, jak rektascensja aktualnie górującego południka - gdy góruje akurat punkt Barana, mamy czas gwiazdowy 0. Mamy więc czas gwiazdowy Greenwich (czyli czas gwiazdowy na południku Greenwich), który można obliczyć na podstawie daty i godziny, oraz lokalny czas gwiazdowy, który jest po prostu czasem Greenwich skorygowanym o przesunięcie wynikające z długości geograficznej punktu obserwacji. Gdy do tego wszystkiego doliczyć rektascensję interesującego nas punktu (w tym wypadku - Słońca), otrzymujemy kąt godzinny dla tego punktu.
Kąt godzinny określa, jak daleko południk przechodzący przez dany obiekt jest od górowania. Znając go i deklinację obiektu, możemy już obliczyć azymut i wysokość (co sprowadza się do przeliczenia kilku obrotów na sferze niebieskiej). Gotowe równania są podane na Wiki (choć w momencie, gdy je znalazłem, był w nich drobny błąd wynikający z pomieszania terminologii - pochwalę się tu, że go wykryłem i poprawiłem ;)).

Mogąc obliczać wysokość Słońca nad horyzontem dla danego momentu i lokalizacji jesteśmy już tylko o krok od wschodów i zachodów - wystarczy znaleźć momenty, w których wysokość jest 0.

Etap drugi - kodowanie

Mając ogarniętą teorię, możemy wziąć się do pisania kodu. Gotowy program znajduje się tutaj.
W pierwszym podejściu zacząłem od utworzenia własnych typów danych do przechowywania daty i godziny. Dobrzy programiści wiedzą jednak, że obsługa dat, obok kryptografii, jest jednym z przysłowiowych trudnych problemów (nieregularność długości miesięcy to tylko pierwsza przeszkoda - gdy wchodzą różne standardy mierzenia czasu, dopiero robi się ciekawie), których nie należy rozwiązywać samodzielnie. Tak więc postanowiłem poszukać gotowych rozwiązań i odnalazłem moduł `Data.Time`.
Pierwszy krok, czyli obliczenie daty juliańskiej, moduł ten załatwia praktycznie za nas. Co prawda oblicza tylko część całkowitą (tj. numer dnia) i w wersji tzw. zmodyfikowanej (liczonej od innego dnia i od północy, zamiast od południa), ale doliczenie reszty jest proste - wystarczy przeliczyć godzinę na sekundy, które minęły od północy i podzielić przez 86400, po czym przesunąć wynik o pół dnia. Dalej należy przeliczyć to na tzw. epokę J2000, co sprowadza się do odjęcia pewnej stałej.

Kolejne funkcje obliczają parametry ruchu Ziemi po orbicie i długość ekliptyczną (dla Słońca ekliptyczna szerokość jest zawsze 0, co trochę ułatwia):

```
1 -- nie mam pojęcia jak to jest po polsku xD średnia długość czy coś
2 meanLongitude :: Double -> Double
```

```
meanLongitude n = 280.46 + 0.9856474*n

-- średnia anomalia w stopniach
meanAnomaly :: Double -> Double
meanAnomaly n = 357.528 + 0.9856003*n

-- długość ekliptyczna Słońca
eclipticLong :: Double -> Double
eclipticLong n =
    let g = deg2rad $ meanAnomaly n
    in meanLongitude n + 1.915 * sin g + 0.02 * sin (2*g)
```

Wszystkie funkcje jako parametr przyjmują datę przeliczoną na epokę J2000.

Teraz jesteśmy gotowi do obliczenia rektascensji i deklinacji Słońca:

```
-- nachylenie osi ziemskiej
obliquity n = 23.439 - 0.0000004 * n

-- Rektascensja Słońca
sunRA :: Double -> Double
sunRA n
    | ra < 0 = ra * 12 / pi + 24
    | otherwise = ra * 12 / pi
    where
        eps = deg2rad $ obliquity n
        l = deg2rad $ eclipticLong n
        x = cos eps * sin l
        y = cos l
        ra = atan2 x y

-- Deklinacja Słońca
sunDec :: Double -> Double
sunDec n =
    let eps = deg2rad $ obliquity n
        l = deg2rad $ eclipticLong n
    in rad2deg $ asin (sin eps * sin l)
```

Ostatnią rzeczą, którą możemy obliczyć nie znając położenia obserwatora, jest czas gwiazdowy Greenwich:

```
-- Greenwich Mean Sidereal Time
gmst :: Double -> Double
gmst n = 18.697374558 + 24.06570982441908 * n
```

Dalej potrzebujemy już lokalizacji, więc przyda się jakiś typ danych, który go będzie przechowywał:

```
data Location = Loc { lat :: Double, lon :: Double }
```

Możemy teraz obliczyć lokalny czas gwiazdowy i kąt godzinny Słońca:

```
-- lokalny czas gwiazdowy
lst :: Location -> Double -> Double
lst l n = gmst n + lon l / 15

-- kąt godzinny Słońca
sunLHA :: Location -> Double -> Double
sunLHA l n = (lst l n - sunRA n) * 15
```

A stąd już prosta droga do kąta zenitalnego (i tym samym wysokości nad horyzontem) oraz azymutu:

```
-- kąt zenitalny
```

```
sunZenith :: Location -> Double -> Double
sunZenith l n =
    let d = deg2rad $ sunDec n
        h = deg2rad $ sunLHA l n
        lat' = deg2rad $ lat l
    in rad2deg $ acos (sin lat' * sin d + cos lat' * cos d * cos h)

-- wysokość
sunElev :: Location -> Double -> Double
sunElev l n = 90 - sunZenith l n

-- azymut
sunAzim :: Location -> Double -> Double
sunAzim l n
    | az < 0 = az + 360
    | otherwise = az
    where
        lat' = deg2rad $ lat l
        dec = deg2rad $ sunDec n
        h = deg2rad $ sunLHA l n
        sinAz = - sin h * cos dec
        cosAz = (sin dec * cos lat') - (cos h * cos dec * sin lat')
        az = rad2deg $ atan2 sinAz cosAz
```

Teraz pytanie, jak sprawdzić, kiedy wysokość wynosi 0.

W pierwszym podejściu zastosowałem naiwny algorytm - dla podanej daty wybieram południe i północ (godziny 0 i 12), po czym sprawdzam, czy w jednym z momentów Słońce jest nad horyzontem, a w drugim pod. Jeśli nie, to stwierdzam dzień/noc polarny/ą, jeśli tak, to dzielę interwał na 2 i wybieram te dwa punkty, dla których Słońce znów jest raz nad, raz pod horyzontem. Powtarzam procedurę dotąd, aż różnica czasu spadnie poniżej zadanego progu i przyjmuję, że jeden z otrzymanych momentów to szukany czas.

Procedura działała całkiem ładnie... dla lokalizacji niezbyt oddalonych od południka 0 i sytuacji bez dnia/nocy polarnego/ej. W innych przypadkach zaczynały się problemy.

Przede wszystkim, długość geograficzna dość mocno wpływa na moment południa słonecznego. Ponieważ wszystko liczymy w czasie UTC, w pobliżu południka Greenwich nie miało to wpływu, ale odpowiednio daleko południe mogło uciec tak, że godziny 0 i 12 obie wypadały w dzień lub w nocy. No i mamy błędnie rozpoznaną sytuację polarną.

To było łatwo skorygować (przesunąć godzinę o długość geograficzną / 15), lecz to nie wszystko. Drugi problem jest mniej oczywisty i ma związek ze wspomnianą na początku eliptycznością orbity. Otóż południe słoneczne nie zawsze wypada o 12 czasu lokalnego; potrafi się wahać o kilkanaście minut do przodu i do tyłu. W efekcie np. w pobliżu nocy polarnej godzina 12 potrafiła wypadać w nocy, mimo że Słońce tego dnia wschodziło. Program rozpoznawał to jednak jako noc polarną.

Na szczęście działające rozwiązanie okazało się niewiele bardziej skomplikowane. Wystarczy najpierw znaleźć południe słoneczne, czyli moment z największą wysokością Słońca. To można zrobić analogicznym algorytmem - zaczynamy np. od lokalnej 11:30 i 12:30 (bo moment południa nie waha się o więcej niż kilkanaście minut, więc nie warto przeszukiwać całej doby i ryzykować, że zahaczymy o południe sąsiedniego dnia). Następnie wybieramy środkowy punkt i zastępujemy nim ten z pary, w którym wysokość Słońca była mniejsza. Powtarzając to aż momenty się zbliżą, osiągniemy południe słoneczne.

Mając południe słoneczne, możemy wybrać momenty oddalone o 12 godzin w obie strony i tym sposobem wrócić do sytuacji z początkowego algorytmu, jednak bez ryzyka, że przegapimy dzień lub noc. W ten sposób znajdziemy wschód i zachód, kiedy to tylko możliwe.

Obliczenie długości dnia to już tylko kwestia odjęcia wschodu od zachodu - nie ma co się nad tym rozpisywać ;)

A oto gotowy algorytm:

```
 1 -- Wschody, zachody
 2 diffThreshold :: (Fractional a) => a
 3 diffThreshold = 0.2
 4
 5 timeDiff :: (Num a) => Location -> a
 6 timeDiff l = fromIntegral (floor (lon l * 240))
 7
 8 findNoon :: Location -> Day -> UTCTime
 9 findNoon l d = findNoonBin l time1 time2
10     where
11         time1 = addUTCTime (41400 - timeDiff l) (UTCTime d 0)
12         time2 = addUTCTime (45000 - timeDiff l) (UTCTime d 0)
13         findNoonBin l t1 t2
14             | diffUTCTime t2 t1 < diffThreshold = t1
15             | elev1 < elev2 = findNoonBin l timeMid t2
16             | otherwise = findNoonBin l t1 timeMid
17             where elev1 = sunElev l (j2000 t1)
18                   elev2 = sunElev l (j2000 t2)
19                   timeMid = addUTCTime (diffUTCTime t2 t1 / 2) t1
20                   elevMid = sunElev l (j2000 timeMid)
21
22 sunBinSearch :: Location -> UTCTime -> UTCTime -> Maybe UTCTime
23 sunBinSearch l time1 time2
24     | elev1 * elev2 > 0 = Nothing
25     | diffUTCTime time2 time1 < diffThreshold = Just time1
26     | elev1 * elevMid <= 0 = sunBinSearch l time1 timeMid
27     | elevMid * elev2 <= 0 = sunBinSearch l timeMid time2
28     | otherwise = Nothing
29     where elev1 = sunElev l (j2000 time1)
30           elev2 = sunElev l (j2000 time2)
31           timeMid = addUTCTime (diffUTCTime time2 time1 / 2) time1
32           elevMid = sunElev l (j2000 timeMid)
33
34 sunrise :: Location -> Day -> Maybe UTCTime
35 sunrise l d = sunBinSearch l midnight noon
36     where
37         noon = findNoon l d
38         midnight = addUTCTime (-43200) noon
39
40 sunset :: Location -> Day -> Maybe UTCTime
41 sunset l d = sunBinSearch l noon midnight
42     where
43         noon = findNoon l d
44         midnight = addUTCTime 43200 noon
45
46 dayLength :: Location -> Day -> Maybe DiffTime
47 dayLength l d = maybeDiff time1 time2
48     where
49         maybeDiff Nothing _ = Nothing
50         maybeDiff _ Nothing = Nothing
51         maybeDiff (Just t1) (Just t2) = Just $ secondsToDiffTime $ floor $ diffUTCTime t1 t2
52         time1 = sunset l d
53         time2 = sunrise l d
```

Etap trzeci - aplikacja i odpowiedź na początkowe pytanie

Pozostało stworzyć prostą aplikację, która pozwoli na wykorzystanie napisanych funkcji. Skorzystałem w tym celu z optparse-applicative - biblioteka ta pozwala na łatwe utworzenie parsera opcji z linii poleceń. Korzystając z gotowej aplikacji, można już odpowiedzieć kiedy Słońce najpóźniej wschodzi i kiedy najwcześniej zachodzi:

```
$ day-length-hs -s 2015-12-10 -e 2016-01-10
Location: Lat 52.2; Lon 20.9
Time zone offset: +0100
2015-12-10: rise 07:40:48, set 15:17:00, length = 7h 36min 12s
2015-12-11: rise 07:41:53, set 15:16:50, length = 7h 34min 56s
2015-12-12: rise 07:42:56, set 15:16:44, length = 7h 33min 47s
2015-12-13: rise 07:43:56, set 15:16:41, length = 7h 32min 44s
2015-12-14: rise 07:44:54, set 15:16:41, length = 7h 31min 47s
2015-12-15: rise 07:45:48, set 15:16:46, length = 7h 30min 58s
2015-12-16: rise 07:46:39, set 15:16:54, length = 7h 30min 15s
2015-12-17: rise 07:47:27, set 15:17:05, length = 7h 29min 38s
2015-12-18: rise 07:48:11, set 15:17:21, length = 7h 29min 9s
2015-12-19: rise 07:48:53, set 15:17:39, length = 7h 28min 46s
2015-12-20: rise 07:49:31, set 15:18:02, length = 7h 28min 31s
2015-12-21: rise 07:50:05, set 15:18:28, length = 7h 28min 22s
2015-12-22: rise 07:50:37, set 15:18:57, length = 7h 28min 20s
2015-12-23: rise 07:51:04, set 15:19:30, length = 7h 28min 25s
2015-12-24: rise 07:51:29, set 15:20:07, length = 7h 28min 37s
2015-12-25: rise 07:51:50, set 15:20:47, length = 7h 28min 56s
2015-12-26: rise 07:52:07, set 15:21:30, length = 7h 29min 22s
2015-12-27: rise 07:52:21, set 15:22:16, length = 7h 29min 55s
2015-12-28: rise 07:52:31, set 15:23:06, length = 7h 30min 35s
2015-12-29: rise 07:52:37, set 15:23:59, length = 7h 31min 21s
2015-12-30: rise 07:52:41, set 15:24:55, length = 7h 32min 14s
2015-12-31: rise 07:52:40, set 15:25:55, length = 7h 33min 14s
2016-01-01: rise 07:52:36, set 15:26:57, length = 7h 34min 21s
2016-01-02: rise 07:52:28, set 15:28:02, length = 7h 35min 33s
2016-01-03: rise 07:52:17, set 15:29:10, length = 7h 36min 53s
2016-01-04: rise 07:52:03, set 15:30:21, length = 7h 38min 18s
2016-01-05: rise 07:51:44, set 15:31:35, length = 7h 39min 50s
2016-01-06: rise 07:51:23, set 15:32:51, length = 7h 41min 28s
2016-01-07: rise 07:50:58, set 15:34:10, length = 7h 43min 12s
2016-01-08: rise 07:50:29, set 15:35:31, length = 7h 45min 1s
2016-01-09: rise 07:49:57, set 15:36:54, length = 7h 46min 56s
2016-01-10: rise 07:49:22, set 15:38:20, length = 7h 48min 57s
```

Dla Warszawy, jak się okazuje, najpóźniejszy wschód przypada na 30 grudnia, a najwcześniejszy zachód - 13/14 grudnia :)