Published in Level Up Coding

John Clark Craig Follow

Apr 7, 2021 · 8 min read ★ · Listen

Python Sun Position for Solar Energy and Research

Here's a concise Python program you can use to calculate the Sun's position to 0.01 degree accuracy.

The author at the Carissa Plains Power Generation Facility

Here's my favorite solar joke… "I stayed up all night trying to figure out where the Sun went. It finally dawned on me."

Carissa Plains project shown above, and sharing a core Python algorithm to calculate the Sun's position feels like the right thing to do. I hope it helps you, whether on a hobby project, or on some of the world's largest solar projects. More power to you!

The program presented here is short, fast, and accurate to about 0.01 degrees for any date in the year range 1901 to 2099, and at any location on the Earth. To understand this accuracy better, the black dot near the center of this photo of the Sun's disk is about 1/50th the diameter of the Sun, or about 0.01 degrees across.

For aiming sun trackers or heliostats for energy harvest, this accuracy is quite sufficient. For extremely precise scientific purposes such as tracking sun spots or other details, the VSOP87 algorithm is better, but it is so much more complicated.

. . .

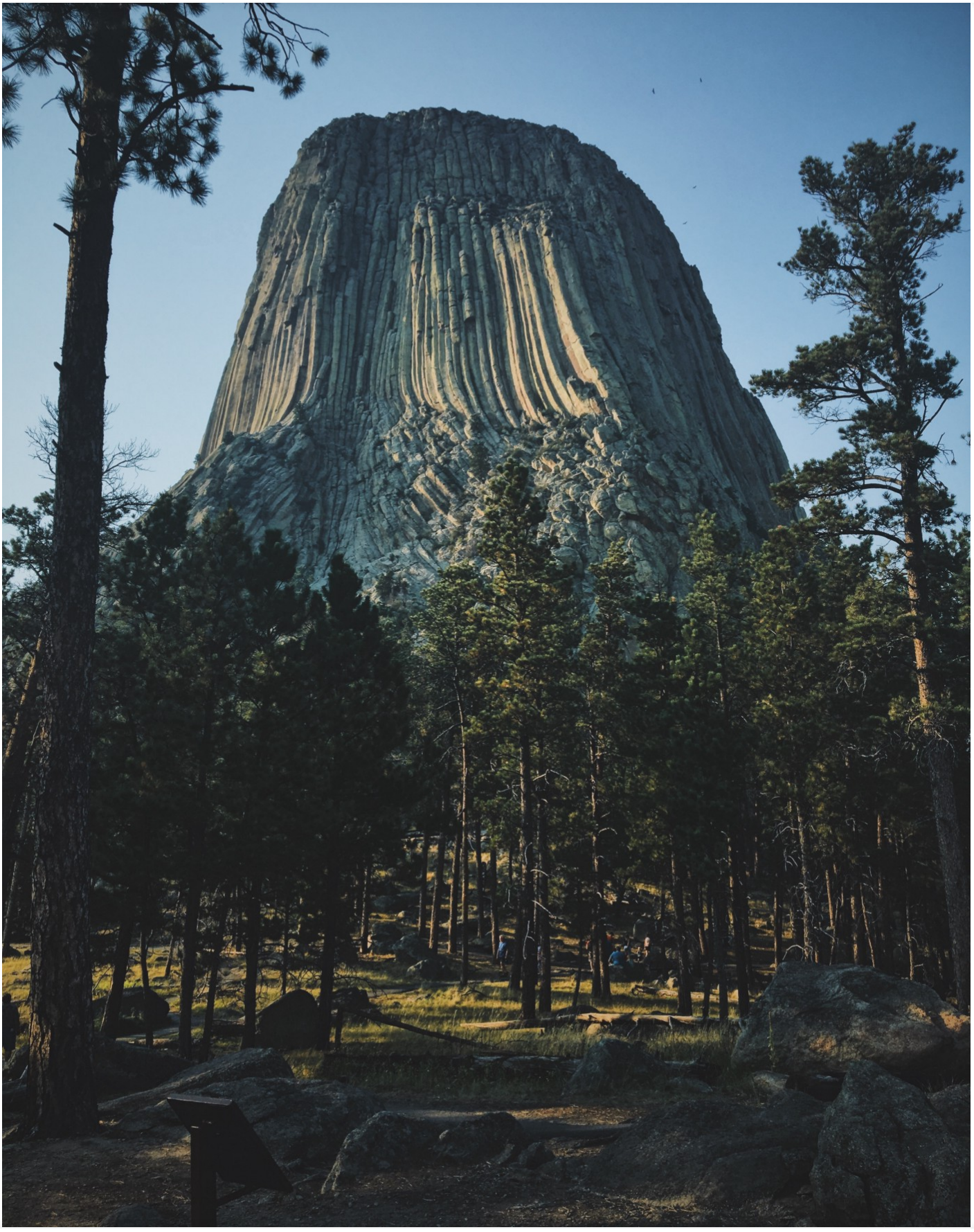
Note: My passion is Python, but if you need this algorithm in a different common programming language, check out my book, "Sun Position: Astronomical Algorithm in 9 Common Programming Languages"

any specific moment in time:

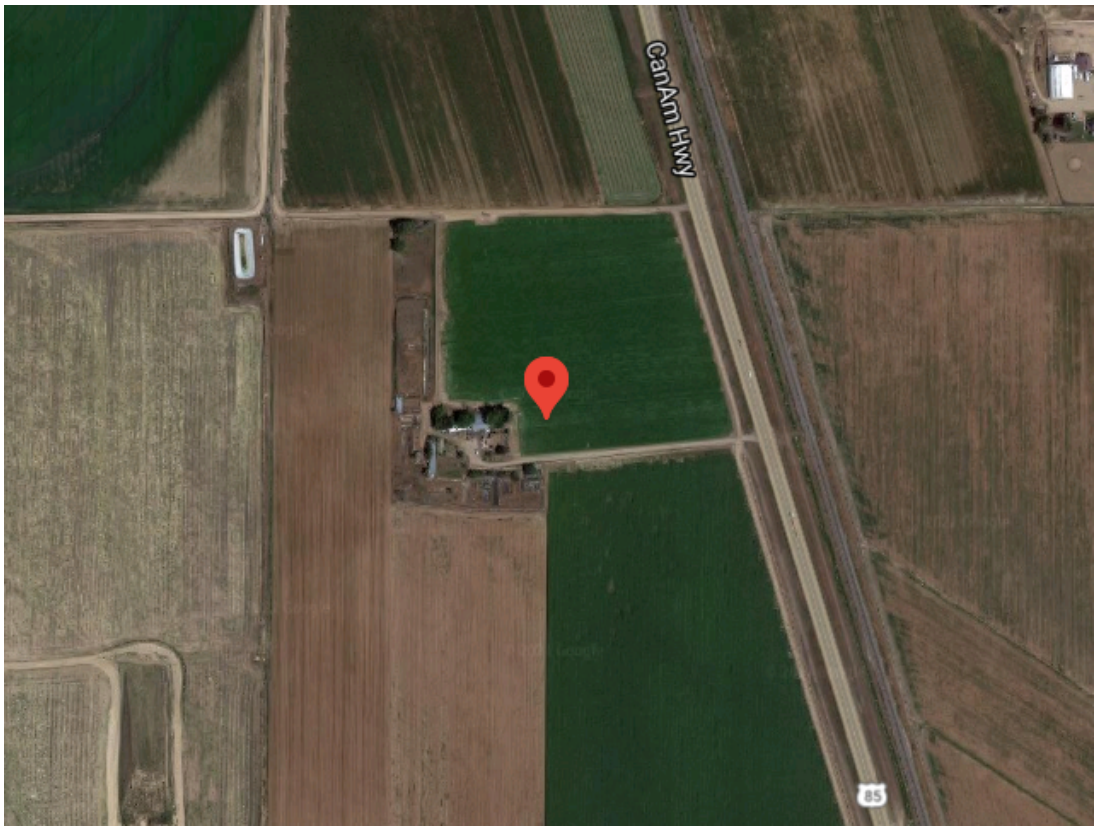
```
# sunpos.py

import math

def sunpos(when, location, refraction):

# Extract the passed data
    year, month, day, hour, minute, second, timezone = when
    latitude, longitude = location

# Math typing shortcuts
    rad, deg = math.radians, math.degrees
    sin, cos, tan = math.sin, math.cos, math.tan
    asin, atan2 = math.asin, math.atan2

# Convert latitude and longitude to radians
    rlat = rad(latitude)
    rlon = rad(longitude)

# Decimal hour of the day at Greenwich
    greenwichtime = hour - timezone + minute / 60 + second / 3600

# Days from J2000, accurate from 1901 to 2099
    daynum = (
        367 * year
        - 7 * (year + (month + 9) // 12) // 4
        + 275 * month // 9
        + day
        - 730531.5
        + greenwichtime / 24
    )

# Mean longitude of the sun
    mean_long = daynum * 0.01720279239 + 4.894967873

# Mean anomaly of the Sun
    mean_anom = daynum * 0.01720197034 + 6.240040768

# Ecliptic longitude of the sun
    eclip_long = (
        mean_long
        + 0.03342305518 * sin(mean_anom)
        + 0.0003490658504 * sin(2 * mean_anom)
    )

# Obliquity of the ecliptic
    obliquity = 0.4090877234 - 0.000000006981317008 * daynum

# Right ascension of the sun
```

```
# Local sidereal time
    sidereal = 4.894961213 + 6.300388099 * daynum + rlon

# Hour angle of the sun
    hour_ang = sidereal - rasc

# Local elevation of the sun
    elevation = asin(sin(decl) * sin(rlat) + cos(decl) * cos(rlat) * cos(hour_ang))

# Local azimuth of the sun
    azimuth = atan2(
        -cos(decl) * cos(rlat) * sin(hour_ang),
        sin(decl) - sin(rlat) * sin(elevation),
    )

# Convert azimuth and elevation to degrees
    azimuth = into_range(deg(azimuth), 0, 360)
    elevation = into_range(deg(elevation), -180, 180)

# Refraction correction (optional)
    if refraction:
        targ = rad((elevation + (10.3 / (elevation + 5.11))))
        elevation += (1.02 / tan(targ)) / 60

# Return azimuth and elevation in degrees
    return (round(azimuth, 2), round(elevation, 2))

def into_range(x, range_min, range_max):
    shiftedx = x - range_min
    delta = range_max - range_min
    return (((shiftedx % delta) + delta) % delta) + range_min

if __name__ == "__main__":

# Close Encounters latitude, longitude
    location = (40.602778, -104.741667)

# Fourth of July, 2022 at 11:20 am MDT (-6 hours)
    when = (2022, 7, 4, 11, 20, 0, -6)

# Get the Sun's apparent location in the sky
    azimuth, elevation = sunpos(when, location, True)

# Output the results
    print("\nWhen: ", when)
    print("Where: ", location)
    print("Azimuth: ", azimuth)
    print("Elevation: ", elevation)

# When:  (2022, 7, 4, 11, 20, 0, -6)
# Where:  (40.602778, -104.741667)
# Azimuth:  121.38
# Elevation:  61.91
```

sunpos() as needed. For now, notice that it's easy to change the location and time parameters near the end of the listing to generate the Sun's position for any location at a single moment in time.



signaled the location of Devil's Tower in Wyoming to meet with the authorities, but the latitude and longitude in the movie actually points to a field near a farmhouse just north of Ault, Colorado. It's not as dramatic of a site, but it would be great for installing some solar trackers!



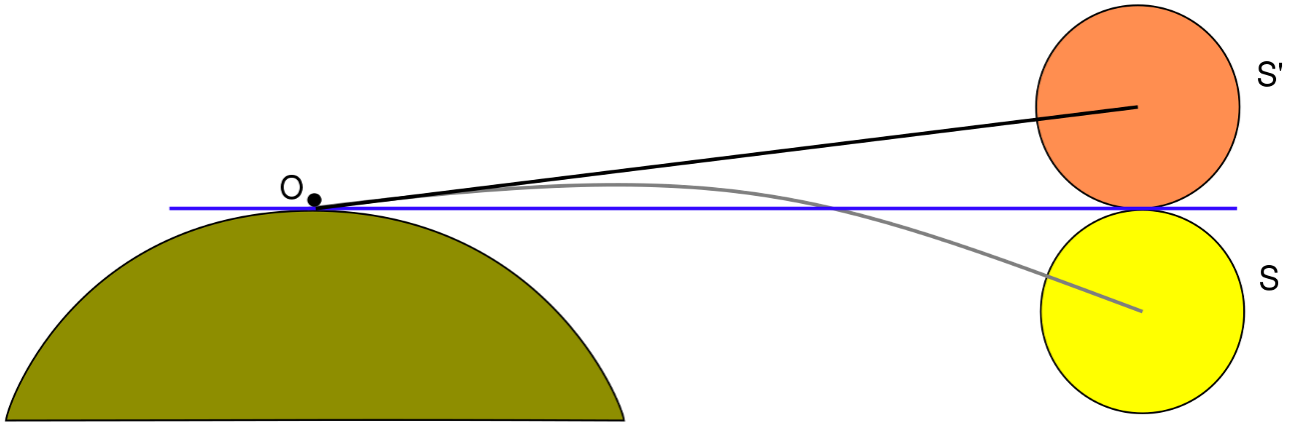
Courtesy Google Maps

For a moment in time I chose 11:20 am on July 4, 2022. Feel free to change these parameters as desired, but I suggest using them to verify the code. Your output should match what's shown in the code listing presented here.

The location and time parameters are set up as lists or tuples. The time includes the timezone, or the number of hours difference from Greenwich, which varies by location, and time of year. There are many places online where you can double check this value.

Refraction Correction

Normally, for solar energy calculations, the Sun's apparent position in the sky is what you want. But there are times, such as when calculating sunrise and sunset, when you don't want to correct for refraction. Notice the True or False parameter that is passed to include or ignore the refraction correction. Let me explain what the refraction correction means.



Courtesy Francisco Javier and Blanco González

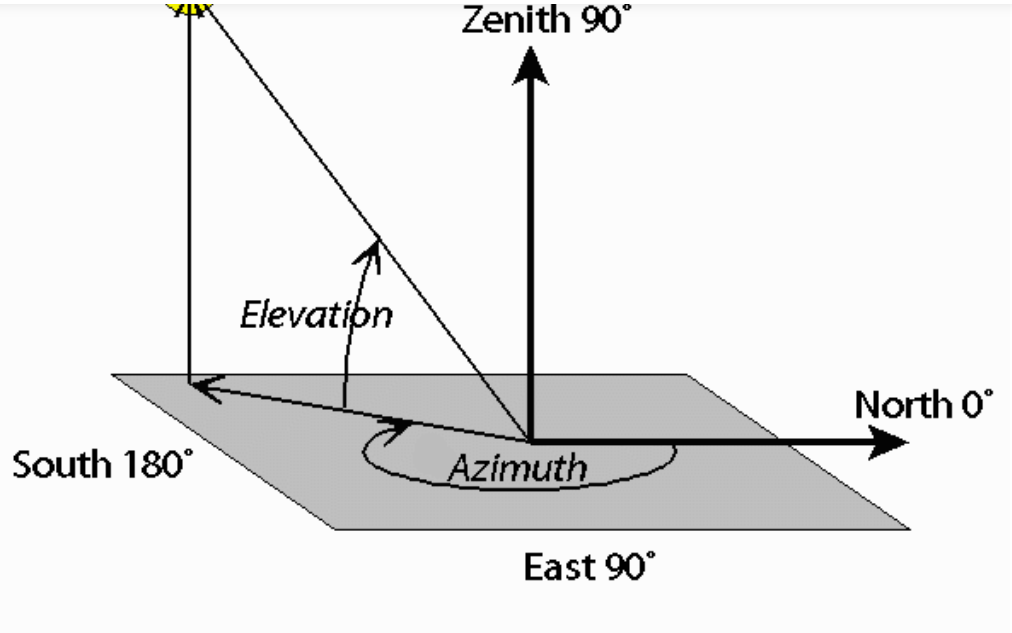
Radians or Degrees

Trigonometric functions in most programming languages require arguments expressed in radians instead of degrees. Most of the equations presented here can be found in astronomical sources expressed in terms of degrees. To speed up the code, I converted these equation terms to radians, did all the complex calculations using radians, and at the end converted back to degrees. It helped speed it up a bit.

Azimuth and Elevation

The location of the Sun is expressed in terms of azimuth and elevation. For clarity, let me define exactly what this means. Azimuth is the number of degrees around on the horizon rotating east from north. So, if the Sun were located exactly above the east horizon, its azimuth would be 90 degrees. At solar noon, when the Sun crosses the sky due south of an observer (northern hemisphere case), its azimuth is 180 degrees. The azimuth is expressed as a positive number of degrees ranging from 0 to 360.

Elevation is the number of degrees up from the horizon to the location of the center of the Sun's disk. When crossing the horizon, the Sun is at 0 degrees elevation, and when the Sun is straight up, overhead at what's called the zenith, its elevation angle is 90 degrees.



Courtesy Celestis.com

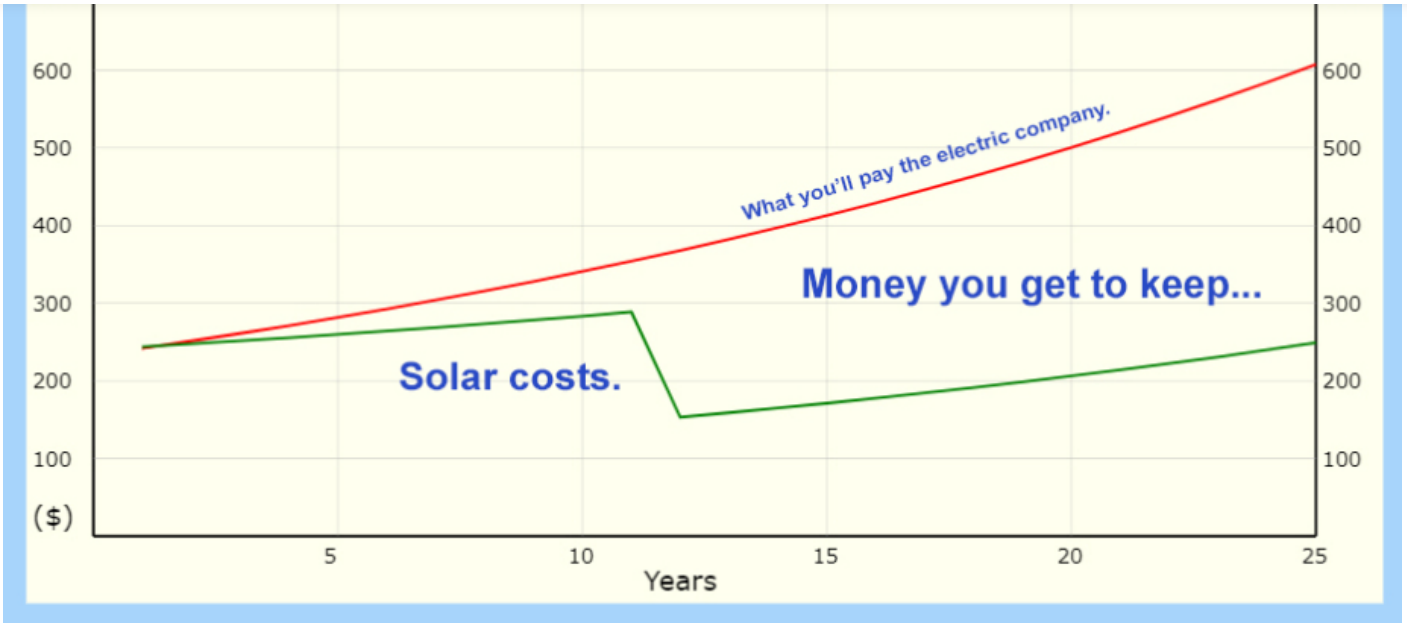
Obliquity of the Ecliptic and Other Strange Terms

You'll notice in the code several calculations for astronomical details that you might not know about. These details handle the factors having to do with the complicated path the Sun takes across our skies due to the tilt of the Earth, the gravitational nudges by the Moon, the Earth's slightly elliptical orbit, and so on. You don't need to understand these factors to use the program, but if interested I suggested searching around on the Internet, or perhaps buying my book mentioned above, as it goes into more explanation.

Creative Applications of the Sun's Position

The Sun's position is useful for aiming sun trackers of all types, such as two-axis heliostats, or various types of single-axis tracking photovoltaic panels. It's also quite useful for non-tracking applications. For example, my next article will show how to calculate overall annual efficiency for panels placed on a roof facing any azimuth, with any standard pitch, and located at some latitude on the Earth.

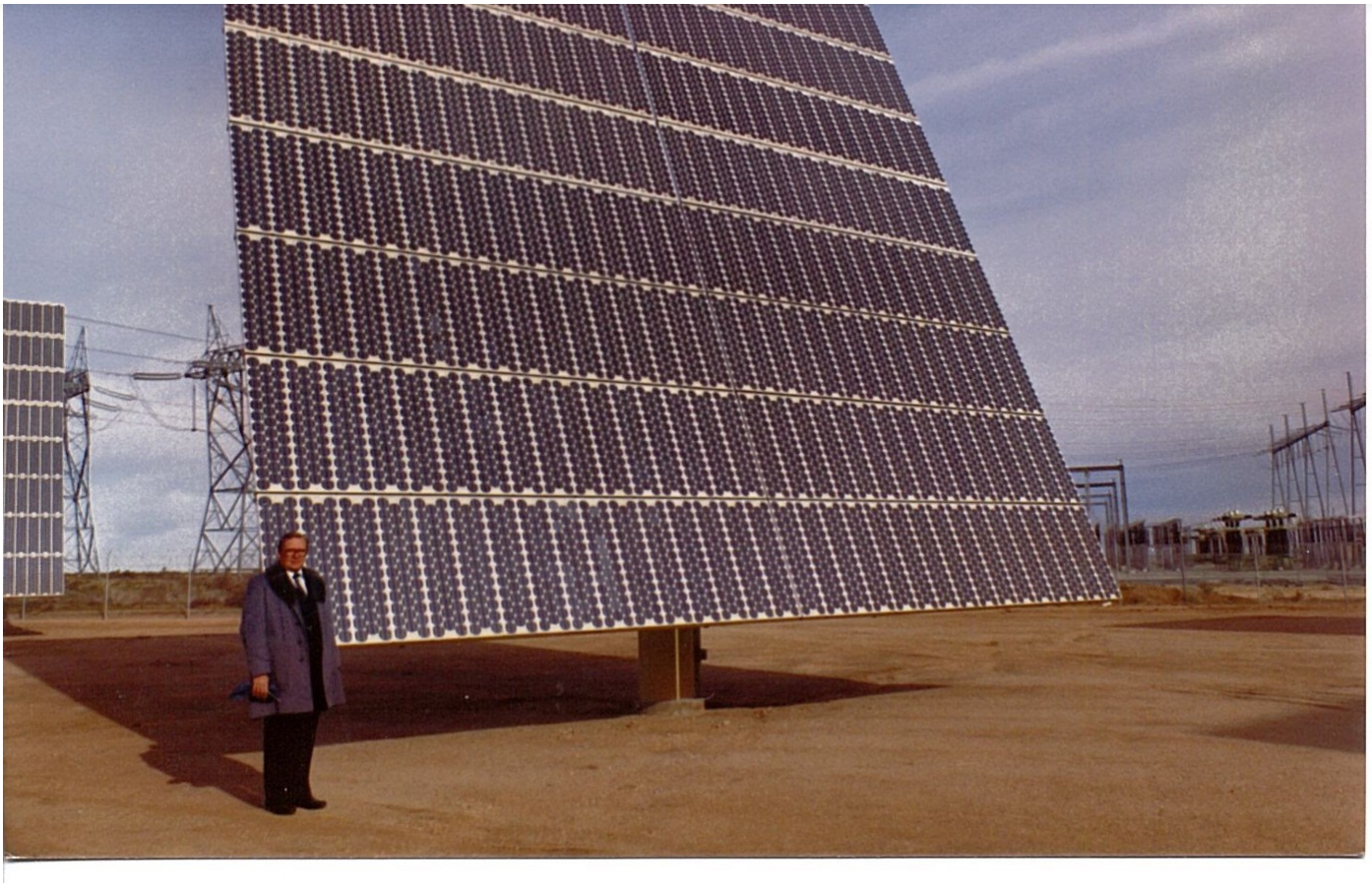
I've developed dozens of useful solar energy programs all based on this core sun position algorithm, many of which are used by my wife for her analysis as she sells optimally efficient and optimally money-saving solar panel systems for homeowners around the country.



A sample of the many solar utility programs by the author

It's a Family Affair

I married the boss's daughter. My boss was a top notch engineer who worked on the Viking Mars lander, among many other projects of note. His small solar engineering group created, built, and programmed (my responsibility) several of the world's largest solar power production facilities, at the time. It was the best job I've ever had, and it included one awesome side benefit; meeting the boss's daughter, EJ!



The author's boss, Floyd Blake, at one of our solar test facilities

Today, EJ is a "President's Club" sales person with hundreds of happy customers, because she helps them save a lot of money, while helping the planet, by customizing the installation of those solar panels you see popping up on your neighbor's roofs. I'm honored to be helping her with a variety of software tasks, many based on the sun position code presented here.

Full disclosure: If you want to know more about how EJ might be able to save you a lot of money by replacing your electric utility bill with a customized installation of solar panels, just click here. (EJ will be the person in her company to be notified)

What's Next?

The core sun position program presented here has found its way into many solar energy applications I've developed over the years. My plan is to write more Medium.com articles that present some of these useful Python utilities that anyone, from homeowners to solar professionals, can use as we all work to help make the world a better place. Stay tuned!

• • •