Chapter 1

Naked planet model

1.1 The Moon, a rock in space

The temperature of any planet or planet-like body is determined by the rate at which it receives energy and the rate at which it loses energy. We will build a climate model from the ground up, module by module, based on this simple fact.

Due to its relative simplicity and proximity to the Earth, there may be no better place in the Universe for laying the foundations of an Earth-centric planetary climate model than the Moon. The Moon is on average the same distance from the Sun as Earth, making it a decent analogue to consider for building a simple energy balance model for a point on the surface of an airless, naked planet. Recent robotic observations of lunar surface temperatures by NASA's Diviner radiometer enable comparison of lunar and terrestrial temperatures. 19th century physics (particularly the Stefan-Boltzmann law) allow simple modeling of various points on the surface of the Moon.

In this chapter, we'll use the Python interpreter and then short Python scripts to calculate radiative equilibrium temperatures and generate plots of simulated diurnal temperature variation under various assumptions. Then in Chapter 3, we will develop a dynamic soil heat exchange model in order to provide a reasonable simulation of the Moon's diurnal temperature curve.

1.2 Lunar temperature observations

Observations from Earth's airless sibling, the Moon

About 75 kilometers (45 miles) from the Moon's north pole, beneath the 2500 m (8200 ft) tall south rim of the Hermite crater [1], there is a spot that probably hasn't seen the light of day for millions of years. Unlike Earth, where the 23.5 degree tilt of the planet's rotational axis causes its polar regions to see half a year of sunshine and half a year of nighttime darkness, the Moon has an inclination of only 1.5 ° relative to the ecliptic [6], meaning that the topography of certain craters is sufficient to cast permanent shadows over parts of the Moon's airless surface.

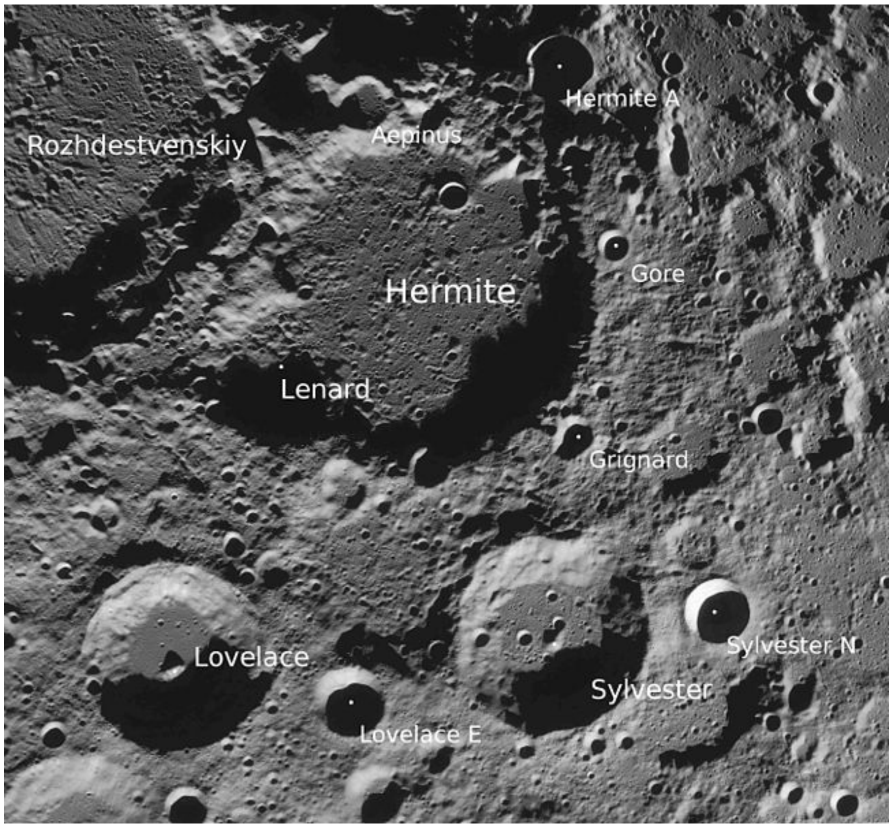

Fig. 1.1 Visual imagery of the Moon's Hermite crater, at 85 degrees north latitude (Image source: NASA / LRO Team)

In addition to the complete lack of solar energy at certain points beneath the rims of polar craters, the Moon is a rock in space with no atmosphere to trap what little heat there is or bring it in from warmer latitudes. Therefore we have long expected the Hermite crater to be very, very cold. But we didn't know precisely how cold until just the last few years.

In 2009 NASA launched a robot with a variety of instruments to map and observe the surface of the Moon. The Lunar Reconnaissance Orbiter (LRO) and its Diviner lunar radiometer have been in an eccentric polar orbit around the Moon ever since. Scientists were excited to find that Hermite crater harbors the coldest known place in the entire solar system, with surface temperatures measured as low as 26 degrees above absolute zero (26 Kelvins, −247 °C, −413 °F), which is even colder than distant Pluto [3].

The Moon is an essentially airless body, with less gas pressure on its surface than the atm ultra-high vacuum (UHV) systems used for scientific research on Earth. Since there is no atmosphere to absorb or scatter incoming sunlight, at its equator the Moon's surface has been observed to reach temperatures as high as 399 K (+124 °C, +255 °F) [5]. Likewise, without an atmosphere to trap nighttime thermal radiation, temperatures can dip below 100 K (−173 °C, −180 °F) at the same equatorial locations.

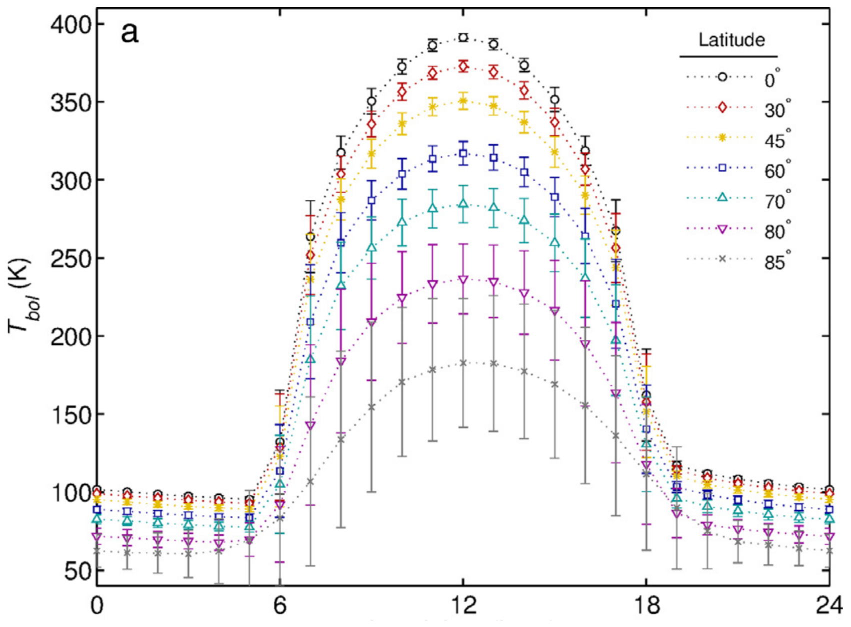

Fig. 1.2 Zonal average temperature observations for each lunar hour on the Moon at various latitudes as measured by the Diviner radiometer, aboard NASA's Lunar Reconnaissance Orbiter. Reproduced from [6]

The nights are very long on the Moon, with a synodic day of 29.53 days. In other words, if you were on the surface of the Moon, it would take 29 1/2 Earth days for the Sun to return to the same position in the sky. Nighttime is nearly two Earth weeks long. Diviner radiometry observations have shown that the Moon's surface cools rapidly in the days before sunset and then cools more slowly throughout the night until sunrise starts the diurnal temperature cycle all over again. Except in those shady places where the sun never shines, which simply stay cold all the time.

Orbiting the sun at the same average distance as Earth, the Moon is something like what our planet would be without an atmosphere, with extreme temperature swings that would be intolerable to life, at least not without extensive protection. Compare the Moon's extreme temperature range (26-399 K) to Earth's coldest in situ measured temperature of 184 K ($-89.2\,^{\circ}$C; $-128.6\,^{\circ}$F) at Vostok in Antarctica, and its hottest in situ temperature of 330 K ($+56.7\,^{\circ}$C; $+134.1\,^{\circ}$F) at Death Valley in California (WMO, 2018). To be fair, the lunar temperatures cited here weren't measured with bulb thermometers. Instead they are so-called bolometric temperatures calculated from satellite radiometer data. The calculation is based on the known relationship between the temperature of objects and the wavelengths of light that such bodies emit (Williams et al., 2017).

Similar temperature measurements have also been made on Earth. Mildrexler et al. (2011) searched through NASA's MODIS satellite radiometer dataset, part of the Earth Observing System (EOS), and found the hottest yet known land surface temperature yet: 344 K ($+70.7\,^{\circ}$C; $+159.3\,^{\circ}$F) in the Lut Desert of Iran during the summer of 2005. Scambos et al. (2018) report finding the coldest known surface temperature on Earth by the same method, 175 K ($-98.6\,^{\circ}$C; $-145\,^{\circ}$F) on the high Dome Fuji-Dome Argus region of the East Antarctic Plateau, at 82° south latitude and just over 3,800 m (12,500 ft) elevation.

On average, the Moon receives exactly the same amount of solar radiation as Earth, with a tiny amount of variation as it moves into its orbital extremes slightly closer to the Sun and slightly farther away. However, presumably because of its near-surface composition (an atmosphere, biosphere, and hydrosphere), Earth doesn't get quite as hot or nearly as cold as the Moon. These similarities and differences make the Moon a powerful, relatively simple test case for developing our physical understanding of the climate of Earth or any other planet-like body through modeling.

1.3 Simple 0-D surface energy balance

Consider a point on the surface of the Moon at the subsolar point, i.e. where the Sun is directly overhead. How hot will it get, and what controls its nighttime cooling? Unlike on Earth, where the high temperature is typically sometime in the late afternoon due to the thermal inertia of the surface and atmosphere, on an idealized airless Moon surface, the high temperature should happen at exactly lunar noon. We can define a lunar hour as 1/24 of a lunar day (lunation) whose length corresponds to Moon's synodic period of 29.53 Earth days. This means that lunar noon is six lunar hours after sunrise, the equivalent of about a week on Earth. Due to the Moon's slight axial tilt, the subsolar point isn't exactly on the equator at all times, but it isn't far.

Assuming no heat exchange with the subsurface, and that thermal equilibrium is achieved instantaneously, the radiant flux in watts per square meter received by our rock surface q_{in} and the power radiated back out to space q_{out} are equal:

$$q_{in} = q_{out} \tag{1.1}$$

Guemard (2018) and others have provided a value of the solar "constant" for Earth (the flux of solar energy at the top of the atmosphere): $S_E = 1361.1 \pm 0.5\, Wm^{-2}$. This value is the same on average for the Moon, therefore if we assume that there is no scattering of incoming light, and that sunlight is the only energy source to the surface:

$$q_{in} = S_E \tag{1.2}$$

19th century physics (the Stefan-Boltzmann law) tells us that a body with emissivity ε will radiate energy proportional to the fourth power of its temperature T. For now we will assume that the Moon's surface is a perfect radiator, that is, $\varepsilon = 1.00$. We can write the following expression for the power radiating outward (in W/m2) from the Moon's surface at the subsolar point and at an equilibrium surface temperature T_s, where σ is the Stefan-Boltzmann constant $\sigma = 5.67 \times 10^{-8} Wm^{-2}K^{-4}$:

$$q_{out} = \varepsilon \sigma T_s^4 \tag{1.3}$$

Setting the input and output terms equal to each other (assuming thermal equilibrium):

$$S_E = \varepsilon \sigma T_s^4 \tag{1.4}$$

Solving for the surface temperature:

$$T_s = \left(\frac{S_E}{\varepsilon \sigma} \right)^{1/4} \tag{1.5}$$

We should verify that the units on both side of our equation correspond with each other:

$$[K] = \left(\frac{[W]m^{-2}}{[Wm^{-2}K^{-4}]}\right)^{1/4}$$

$$[K] = [\left(K^4\right)^{1/4}]$$

$$[K] = [K]$$

Inserting our data and assuming emissivity of 1.00,

$$T_s = \left(\frac{1361.1\ Wm^{-2}}{(1.00)(5.67 \times 10^{-8}\ Wm^{-2}K^{-4})}\right)^{1/4}$$

It's quite easy to perform this calculation in Python. First we fire up the Python interpreter at the command line:

```
  $ python
```

A new prompt will appear, where we can type Python commands one line at a time. Amazingly, we can simply type the expression that we just wrote above for the surface equilibrium temperature:

```
>>> T_s = (1361.1/1.00/5.67e-8)**(1/4)
```

When you hit enter, Python calculates the value of T_s right away. Now we just need to display its value using the `print()` function, with `T_s` as its only argument:

```
>>> print(T_s)
```

Hitting return will give us

```
393.61962546775345
```

which expressed with three significant digits is

$$T_s = 394\,K$$

This is remarkably close to the maximum temperature of 399 K that the Diviner radiometer has measured on the Moon's surface!

What happens if we include real ballpark values for the measured reflectivity (albedo) and emissivity of the Moon's surface? Defining the normal albedo A as the fraction of insolation that is not absorbed (Bond albedo), Vasavada et al. [5] used Diviner data to calculate mean albedo values of 0.07 and 0.16 for mare and highland areas of the moon, respectively. We can use a ballpark mean of $A = 0.115$ here. Vasavada et al. also reported T_7 (IR/thermal channel 7) spectral emissivity of $\varepsilon_7 = 0.98$. With albedo A removing a fraction of insolation,

$$(1-A)S_E = \varepsilon\sigma T_s^4 \tag{1.6}$$

Solving for T_s,

$$T_s = \left(\frac{(1-A)S_E}{\varepsilon\sigma}\right)^{1/4} \tag{1.7}$$

To perform this calculation, this time let's write a Python script in a text editor such as Sublime or vim so that we can reuse and modify our code. We can save it in a `<filename>.py` file such as `nakedplanet_simple.py` that we can run at the command line using `python nakedplanet_simple.py`. The `scipy` module includes a catalog of commonly used scientific constants, so we can include it in our code using the line `from scipy import constants`. For now we will hard-code our variable values right in this script, but later on we will store them in a separate data file for better program organization (modularity).

```python
# nakedplanet_simple.py

from scipy import constants

sigma = constants.sigma
alpha = 0.115   # Mean bond albedo value (ranges from 0.07 to
    0.16) from Vasavada et al. (2012)
epsilon = 0.98  # T7 emissivity (Vasavada et al., 2012)
S_E = 1361.1    # Solar constant at Earth TOA, +/- 0.5 [W m^-2] (
    Gueymard, 2018)


T_s = ( ( ( 1 - alpha ) * S_E ) / epsilon / sigma )**(1/4)
print(T_s)
```

```
  $ python nakedplanet_simple.py
  383.7063753575507
```

$$T_s = 384\,K$$

What other sources of heat might there be to the lunar surface? These fluxes become most significant at night, so for this purpose there is no better place to think about than the eternal shade below the rim of the Hermite crater, which is at a perpetual low temperature of 26 K.

Shaded from the Sun, q_{in} for such an environment should have only two terms (which we have not yet considered for simplicity): q_g, the surface heat flux from the Moon's interior (primarily from radioactive decay), and q_c, the radiative flux from the cosmic microwave background. Siegler and Smrekar [4] used radiogenic thermal conduction models and measurements taken by the Apollo, GRAIL, and Selene missions to estimate global lunar geothermal surface heat flux values of q_g between $9-13\,mW\,m^{-2}$. Fixsen [2] calculated a cosmic background flux of $q_c = 3.13 \times 10^6\,Wm^{-2}$.

With a mean value of $q_g = 11 \times 10^{-3}\,Wm^{-2}$, $q_c \approx 0$ since $q_c \ll q_g$. Therefore $q_{in} \approx 11 \times 10^{-3}\,Wm^{-2}$ and the energy balance for regions of permanent shadow on the Moon becomes:

$$q_{in} \approx q_g = \varepsilon\sigma T_s^4 \tag{1.8}$$

Solved for T_s:

$$T_s = \left(\frac{q_g}{\varepsilon\sigma}\right)^{1/4} \tag{1.9}$$

We can perform this calculation by adding a few lines of code to our `nakedplanet_simple.py` script.

```
q_g = 11e-3     # Lunar geothermal surface heat flux [Wm-2], mean
     of 9-13e-3 Wm-2 (Siegler and Smrekar, 2014)
```

Then we modify our temperature calculations accordingly:

```
# Subsolar max temp calculation
T_s = ( ( ( 1 - alpha ) * S_E ) / epsilon / sigma )**(1/4)
print("Subsolar␣max␣temp:", T_s)

# Permanent shadow temp calculation (q_g only heat source)
T_s = ( q_g / epsilon / sigma )**(1/4)
print("Permanent␣shadow␣temp:", T_s)

$ python nakedplanet_simple.py
```

```
Subsolar max temp: 383.7063753575507
Permanent shadow temp: 21.09302465387775
```

Our ”permanent shadow temp” of 21 K is quite close to the observed coldest value of 26 K, and keeping in mind various possible values for emissivity (some materials are better radiators than others), we might interpret this calculated value as the theoretical coldest surface temperature possible anywhere on the Moon.

At this point we might as well format our `print` output to round to the correct number of significant digits. When formatting strings for the `print` function, the string format modifier `f` allows you to specify the number of digits to the right of the decimal point, for example `:.2f` would always print two digits to the right of the decimal point. However we would like the decimal point to move depending on the number of significant digits, so we can use the modifiers `g` and `G` (where ”g” stands for ”general” format). The two are the same except that `G` will use scientific notation for larger numbers, which is desirable in our case. Strictly speaking, since we are using estimates for radiogenic heat flux that have only two significant digits, we should be expressing our calculations using `:.2G`, or two significant digits. However since our temperature values are three digit numbers, this is fairly cumbersome to read. Therefore for readability we will use `:.3G` or three significant digits.

String objects in Python have a format method that can be used to modify the string according to certain parameters. To express our calculations with three digits, and indicate the units for our numbers, we can rewrite our print statements as:

```python
print("Subsolar␣max␣temp:", "{:.3G}".format(T_s), "K")

print("Permanent␣shadow␣temp:", "{:.3G}".format(T_s), "K")
```

```
Subsolar max temp: 384 K
Permanent shadow temp: 22 K
```

With this we have a reasonable and simple model for the range of temperatures found on the Moon, including the coldest known place in the solar system, right in Earth’s backyard.

1.4 Periodic solar forcing

A first step to simulating the diurnal temperature cycle on the surface of the Moon is to simply assume instantaneous thermal equilibrium and recalculate temperatures at each new time step.

For this we want to derive an expression for input flux to the surface as a function of time $q_{in}(t)$. On a horizontal surface at a point on the solar equator,

$$q_{in}(t) = S_E(t) + q_g + q_c \tag{1.10}$$

where q_g and q_c are the geothermal and cosmic heat fluxes, respectively. Since $S_E \gg q_g, q_c$, to a first approximation between sunrise and sunset

$$q_{in}(t) \approx S_E(t) \tag{1.11}$$

The geothermal and cosmic heating terms become more important at night, however, so for our time-dependent model we should leave them.

To simulate a time series of temperatures through a period of 24 lunar hours, we need to model how solar irradiance varies as the Moon rotates around its axis. Solar irradiance on a rotating body is a periodic function of the time.

Let Θ be the local zenith angle of the Sun (angle from the vertical) and $S_E(t)$ be the actual solar irradiance as a function of time t, measured in sidereal seconds.

$$cos\,\Theta(t) = \frac{S_E(t)}{S_E} \tag{1.12}$$

$$S_E(t) = (1-A)\,S_E\,cos\,\Theta(t) \tag{1.13}$$

We consider here a point on the moon at the solar equator. When the solar zenith angle $\Theta = 0$, the sun is directly overhead. If we start the clock in our model ($t_0 = 0$) at one of these noontime solar maxima, the next time that the sun will be overhead is P_{moon} seconds later, once the Moon has undergone a rotation of 2π. The period of the moon's rotation P_{moon} (in seconds) is the length of the lunar day in sidereal days times the number of seconds in a day: $P_{moon} = 29.53059 \times 24 \times 3600\,s = 2.55 \times 10^6\,s$.

Likewise from our perspective, the zenith angle of the Sun proceeds to $\Theta = \frac{\pi}{2}$ at sunset ($t = t_0 + 0.25\,P_{moon}$), $\Theta = \pi$ at local midnight ($t = t_0 + 0.5\,P_{moon}$), $\frac{3\pi}{2}$ at sunrise, and back to $\Theta = 2\pi$ (coterminal with $\Theta = 0$) at $t = t_0 + P_{moon}$.

If we define the hour angle $h(t)$ such that $cos\,h(t) = cos\,\Theta$, we can express the hour angle in terms of the rotational period P_{moon}:

$$h(t) = 2\pi(\frac{t}{P_{moon}}) \tag{1.14}$$

The hour angle increases linearly with time to infinity, but has the important property that at the end of each lunation, when time t is an integer multiple of P_{moon}, the value of $h(t)$ is an integer multiple of 2π.

Since $cos\,h(t) = cos\,\Theta$,

$$S_E(t) = S_E\,cos\,h(t) \tag{1.15}$$

between sunrise and sunset.

At nighttime, however, this function would yield a negative insolation value, so we need to use a function that drops to exactly zero at all angles between $\frac{\pi}{2}$ and $\frac{3\pi}{2}$. The clipping function

$$\psi(x) = \frac{1}{2}\,(cos\,x + \mid cos\,x \mid) \tag{1.16}$$

has this behavior. Therefore we can define the solar irradiance as a function of time at the solar equator as:

$$S_E(t) = (1-A)\,S_E\,cos\,\psi(h(t)) \tag{1.17}$$

Just to make sure the sun is shining realistically on our model point on the Moon, let's use Python to plot this solar irradiance function over a couple of lunar days. We'll write our code in a Python file that we will call `solar.py`.

Very soon we will modularize our Python code into different files, objects, and functions, but still for now we will store all of our information in a single file. Our file will be organized as follows:

```
# filename: solar.py

# Import modules

# Set planetary parameters

# Set model parameters

# Create and initialize arrays
# Run time loop

# Plot calculated array values using matplotlib
```

The code below will produce two plots using matplotlib. One will show the calculated solar irradiance as a function of time in sidereal seconds, and the second will show the solar irradiance as a function of time in lunar hours. Comments are provided for explanation.

```
# solar.py

import matplotlib.pyplot as plt    # for plotting
import numpy as np    # numpy arrays for calculations and
    matplotlib plots

pi = np.pi    # so we don't have to type np.pi all the time

# Moon climate parameters

S_E = 1361.1 # Solar constant at Earth TOA, +/- 0.5 [W m-2] (
    Gueymard, 2018)
albedo = 0.115 # Mean bond albedo value (ranges from 0.07 to
    0.16) from Vasavada et al. (2012)
P_moon = 29.53059*24.*3600. #Mean length of Moon SYNODIC day [s]

Sabs = S_E * (1.0 - albedo)

# Model parameters
modelruntime = 4*P_moon
dt = 1.0 * 3600. # timestep in seconds (1.0*3600*24 = 1 earth day
    )
Nt = int(round(modelruntime/dt))   # Calculate number of
    timesteps (rounded and converted to integer)

# Create numpy arrays for calculations and plotting
t = np.zeros(Nt) # start the clock at zero
h = np.zeros(Nt) # hour angle
psi = np.zeros(Nt) # clipping function
solar = np.zeros(Nt) # array of insolation values to be
    calculated

# Create numpy array for lunar hour

lunarhour = np.zeros(Nt)

# Time loop, goes until modelruntime
for n in range(0, Nt):
        t[n] = n*dt    # Calculate the model time based on the
            current timestep number
        h[n] = 2*pi*(t[n]/P_moon) # Calculate hour angle
        psi[n] = 0.5 * ( np.cos(h[n]) + np.abs(np.cos(h[n]))) #
            Calculate clipping function
        solar[n] = Sabs * psi[n]  # Calculate solar flux

        lunarhour[n] = t[n]/P_moon*24  # Calculate lunar hour (
            for plotting only)

# Plot solar and t arrays with matplotlib
plt.scatter(t, solar)
plt.plot(t, solar)
plt.xlabel('Time␣(s)')
plt.ylabel('Solar␣irradiance␣(W/m2)')
plt.show()

plt.scatter(lunarhour, solar)
plt.plot(lunarhour, solar)
plt.xlabel('Time␣(lunar␣hours)')
plt.ylabel('Solar␣irradiance␣(W/m2)')
plt.show()
```

The output is a beautiful simulation of solar irradiance at a single point on the solar equator of the Moon over the course of four lunar days (about four Earth months). Note that the daily maxima occur at hours 24, 48, 72, and 96, and that every 12 hours starting at lunar 6 P.M. there is zero insolation for the entire nighttime. Only the second plot from solar.py (in lunar hours) is shown here.

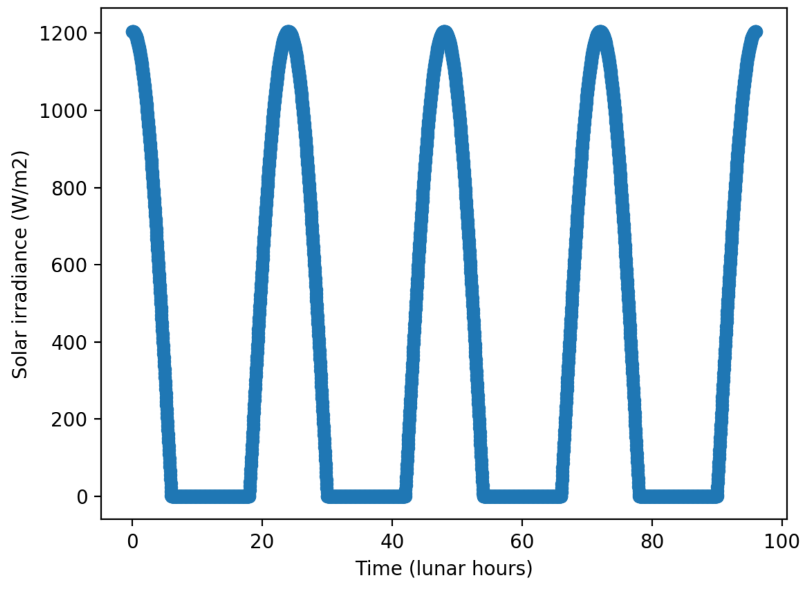

Fig. 1.3 Calculated solar irradiance at a point on the solar equator of the Moon over four lunations

If we'd like to see what just one lunar day of sunshine looks like, we can start the lunarhour array at lunar midnight:

```
lunarhour[n] = t[n]/P_moon*24 - 12
```

and tell matplotlib to show only the lunar hours from 0 to 24, with ticks at 0, 6, 12, 18, and 24:

```
plt.xlim(0, 24)
plt.xtick(0, 6, 12, 18, 24)
```

We can modify modelruntime in solar.py to show only one lunar rotational period:

```
modelruntime = 1*P_moon
```

and for comparison with the Diviner diurnal temperature observations, modify the time loop to start at lunar midnight, halfway through the first lunation (because of how we set modelruntime, it will end 24 lunar hours later). Since for loops only deal with integers, we need to typecast the starting point of our for loop as an int:

```
for n in range(0 + int(P_moon/2), Nt):
```

We don't really need the first matplotlib plot (expressed in sidereal seconds), so we can simply delete it from the script.

It is convenient to save this new modified code as solar_oneday.py. Running it at the command line:

```
python solar_oneday.py
```

provides us with the beautiful result seen in Figure 1.4.

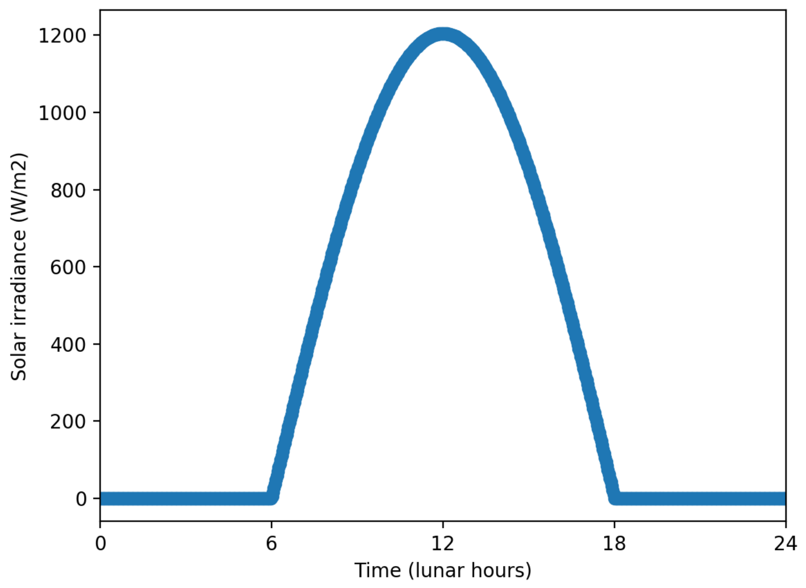

Fig. 1.4 Calculated solar irradiance at a point on the solar equator of the Moon over a single lunar day

The curve we have generated here isn't so different from the insolation curve on a clear day at any given location in the tropics on Earth: The Sun rises around six a.m., sets around six p.m., and insolation reaches a local maximum of $(1-A)S_E \approx 1200\,Wm^{-2}$ right around twelve noon. The difference here is of course that a lunar day lasts nearly an Earth month, and each lunar hour is a bit more than an Earth day long. Hopefully the analogy is clear to the reader.

Once we calculate the diurnal equilibrium temperature curve for our point at the solar equator on the Moon (neglecting soil heat fluxes), it should follow the diurnal insolation curve fairly closely.

1.5 Linking solar forcing to the 0-D surface energy balance

It won't take much to generate a time-dependent surface temperature curve from our existing solar.py module. Actually, just couple lines of Python code will do.

You may have noticed that the file solar.py is already pushing past 50 lines. It works just fine the way it is, and that's not something that we want to take for granted. In the next chapter, we will look at how Git and GitHub can be used for version control, that is, for saving our working versions with comments as we go along. We'll also clean up our model's code by organizing it into modules that can be added, used, and re-used like tools in a toolbox.

But for now let's keep all our code in one file and save our working version of `solar.py` the old-fashioned, pre-Git (pre-2005) way by simply copying its contents into another file, let's call it `solar_working_notemp.py`:

```
cp solar.py solar_working_notemp.py
```

For now we'll just add our temperature code to solar.py, a file we are going to pick apart and separate into modules in the next chapter.

We can add a numpy array for surface temperature to the end of the initialization section:

```
# Create numpy arrays for calculations and plotting
t = np.zeros(Nt) # start the clock at zero
h = np.zeros(Nt) # hour angle
psi = np.zeros(Nt) # clipping function
solar = np.zeros(Nt) # array of insolation values to be
    calculated
T_s = np.zeros(Nt) # time array of surface temperatures
```

Then we can add an equilibrium temperature calculation to the time loop section. Recalling that if we neglect geothermal, soil, and cosmic heating so that insolation is the only energy input to the system and thermal radiation is the only output, the equilibrium surface temperature is:

$$T_s = \left(\frac{(1-A)S_E}{\varepsilon\sigma} \right)^{1/4} \tag{1.18}$$

We can express the time-dependence of this function with a simple change of notation:

$$T_s(t) = \left(\frac{(1-A)S_E(t)}{\varepsilon\sigma} \right)^{1/4} \tag{1.19}$$

In Python form, this can be written in solar.py as:

```
    # Calculate surface temperature given solar flux
    T_s[n] = ((1-albedo)*solar[n]/epsilon/sigma)**(1/4)
```

Be careful, however, because we haven't yet defined the variables epsilon and sigma in solar.py, so we should make sure to do this by adding the following lines in the appropriate sections above the time loop:

```
    from scipy import constants
    sigma = constants.sigma

    epsilon = 0.98       # T7 emissivity (Vasavada et al., 2012)
```

And since if we make a calculation but we don't present it graphically, it might as well never have happened at all, we should make sure to add in some matplotlib code to the appropriate section to generate a nice temperature plot.

```
plt.scatter(lunarhour, T_s)
plt.plot(lunarhour, T_s)
plt.xlabel('Time␣(lunar␣hours)')
plt.ylabel('Surface␣temperature␣(K)')
plt.ylim(-20, 400)
plt.xticks(np.arange(0, 25, step=6))
plt.show()
```

Before we run the code, what do we expect to see in our temperature plot? The total energy input to the system should be $q_{in}(t) = S_E(t) + q_g + q_c + q_{soil}(t)$. However we have neglected all heating terms except insolation so that $q_{in}(t) \approx S_E(t)$. This means that at night, since we've designed such a beautiful clipping function, there is no insolation and therefore no heating of the surface from any source. So far there is no thermal inertia in our system so we should expect the temperature to be exactly absolute zero between sunset and sunrise.

When we run the code to see our result:

```
python solar.py
```

since we have no print statements there is no output to the terminal, but a matplotlib plot does appear. The first is our plot of insolation through the day as we saw before. If we close that window, our desired temperature plot will appear.

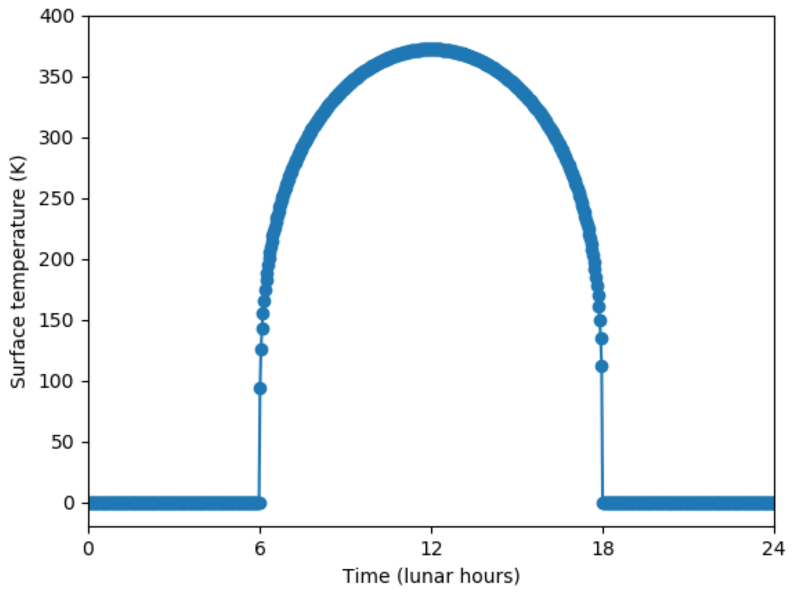

Fig. 1.5 Calculated surface temperature at a point on the solar equator of the Moon over a single lunar day

1.5.1 Improving the visual representation of data in matplotlib

We'd like to see what happens when we add in the various other heat flux terms to the energy balance. We can make our plots a little bit better by generating a text box that automatically displays the high and low temperatures (max and min). Since there's already a fair amount of visual information on the left and right margins of the plot, let's put this annotation in the upper right corner.

We'll need some background information on matplotlib to proceed with any confidence. First of all, matplotlib has two interfaces: a state-based one and an object-oriented one. For the couple of plots we've generated so far, we've taken the simplest possible route available to us: matplotlib.pyplot, which is the state-based, procedural interface for simple plotting. With pyplot, everything has to be executed in a one-off sequence, and nothing can be re-used. The advantage of pyplot's state-based interface is that it is easier to use for those unfamiliar with matplotlib.

However for any more sophisticated, publication-quality customization of plots, we have to make use of the object-oriented interface to matplotlib. This approach is a bit more nuanced but more flexible. Its building blocks are the ”Figure” object and the ”Axes” object.

These terms are important for understanding how to modify matplotlib plots. A ”Figure” in matplotlib-speak is the final rendered image that you see for example in Figure 1.5. It is composed of one or more ”Axes,” which are individual plots. As one official matplotlib tutorial states, ”The Figure is like a canvas, and the Axes is a part of that canvas on which we will make a particular visualization, [and] Figures can have multiple Axes on them” (https://matplotlib.org/tutorials/introductory/lifecycle.html). Another official explanation is: ”The Figure is the final image that may contain 1 or more Axes. The Axes represent an individual plot (don't confuse this with the word ”axis”, which refers to the x/y axis of a plot).” (https://matplotlib.org/tutorials/introductory/lifecycle.html).

The reason this is important to know is that most modifications to a plot require modifying an ”Axes” object, almost always called ”ax” by convention. There isn't much to be done with ”Figure” objects, called ”fig” by convention, but in order to access an ax object we call the plt.subplots() method, which returns a fig, ax tuple. So in order to make anything but the most basic customizations, we need to use the following essentially boilerplate code to access an Axes object:

```
fig, ax = plt.subplots()
```

You'll live a lot longer if you just accept that we'll hardly ever use the fig object, but to make pretty plots we'll often want to play with the ax object, and figs go along with axes because of the plt.subplots() method. And be careful! plt.subplots() for this purpose has a plural ”s” on the end. plt.subplot() without an ”s” is an entirely different function.

The methods for matplotlib Axes objects are written slightly differently from the functions for pyplot/plt, so we'll have to rewrite our plotting code accordingly.

It is a good idea to try to write this code on your own by using the matplotlib API documentation for the Axes object. The code below is the object-oriented Axes equivalent of the pyplot code we used earlier, and generates a surface temperature plot identical to that seen in Figure 1.5. We can now replace the pyplot-based code with this Axes-based code in solar.py.

```
fig, ax = plt.subplots()
ax.scatter(lunarhour, T_s)
ax.plot(lunarhour, T_s)
ax.set_xlim(0,24)
ax.set_ylim(-20,400)
ax.set_xticks(np.arange(0, 25, step=6))
ax.set_xlabel('Time␣(lunar␣hours)')
ax.set_ylabel('Surface␣temperature␣(K)')
plt.show()
```

In a sense we're right back where we started, but this time by more sophisticated means. Now we can finally annotate our plot with the max and min temperatures as we set out to do earlier. For this we use the ax.annotate() method. Two arguments are enough to use the ax.annotate() method in this case: a string to display and the location to display the string.

```
ax.annotate('Max␣temp:', xy=(17,350))
ax.annotate('Min␣temp:', xy=(17,325))
```

This won't display any values, but will put some text in a good spot on the plot. The string is the first argument by default and is enclosed in either single or double quotes. The location can be given as a keyword argument xy=(xlocation, ylocation), where x and y are measured in terms of the data you are plotting. In the code snippet below, I place the annotations at lunar hour 17, at temperature values of 350 and 325, respectively. Other keyword argument options are available for placement of the annotation, such as by pixel coordinates, but since the numbers in this plot are nice and round, it made sense to use point coordinates.

Next we want to display a variable value. To do this we can use Python's format() method. A good user-friendly reference with examples for use of this method can be found at: https://www.geeksforgeeks.org/python-format-function/

To use format(), we insert a placeholder with curly braces inside the string literal (the text inside the quotation marks). Then we invoke the format method on the string object using the dot operator:

```
ax.annotate('Max␣temp:␣{}␣K'.format(400), xy=(17,350))
ax.annotate('Min␣temp:', xy=(17,325))
```

If we put this in our code and run it, we'll have an annotation that says "Max temp: 400" and "Min temp:" without a value for the min temp. Let's test whether we can insert a variable where we wrote "400" in the previous code snippet:

```
ax.annotate('Max␣temp:␣{}␣K'.format(Sabs), xy=(17,350))
ax.annotate('Min␣temp:', xy=(17,325))
```

This will display the value of Sabs (the solar constant reduced by the albedo) as the maximum temperature, which doesn't make much sense but at least it shows that we can use a variable as an argument to format().

It would be nice to format our value. In our current context, we don't really need any decimal places for our temperature results, so inside the curly brace placeholder we can let Python know that we would like to display a float with zero decimal places. We should try to understand this notation, since it will always come in handy for expressing our modeling results. If we wanted to express multiple variables with the same string, we could number our curly brace placeholders: 'Max temperatures:

1:.0f K, 2:.0f K'.format(temp1, temp2). In this little example, "1" indicates that it is the first variable to be displayed, and "0" is the number of decimal places to be shown for the float value. Likewise "2" in the second placeholder indicates that it is the second variable to be displayed. These index numbers aren't required, since format() will just display the variables in the order they are written by default if no indices are specified. So for display of a single float variable within a string with zero decimal places, we can leave out the index number to the left of the colon, and just write ':.0f'.format(variable):

```
ax.annotate('Max␣temp:␣{:.0f}␣K'.format(Sabs), xy=(17,350))
ax.annotate('Min␣temp:', xy=(17,325))
```

Alright, great, we've formatted Sabs, but that's just a dummy variable, and we want to express the max and min temps. For this we can use numpy's amax() and amin() methods, which will find the maximum and minimum values for given arrays. We can apply amax() to our surface temperature array `T_s`:

```
ax.annotate('Max␣temp:␣{:.0f}␣K'.format(np.amax(T_s)), xy
    =(17,350))
```

and do the same with amin() for the minimum temperature:

```
ax.annotate('Min␣temp:␣{:.0f}␣K'.format(np.amin(T_s)), xy
    =(17,325))
```

Lastly, as we start generating a plethora of plots, we'll be glad when our image files have at least somewhat explanatory titles, so we should give our plot a title:

```
ax.set_title('Solar␣heating␣only')
```

The final result of this incremental development is a decent plot of surface temperature at the solar equator on the Moon under solar forcing for one lunation, from lunar midnight to lunar midnight:

```
ax.annotate('Max␣temp:␣{:.0f}␣K'.format(np.amax(T_s)), xy
    =(17,350))
ax.annotate('Min␣temp:␣{:.0f}␣K'.format(np.amin(T_s)), xy
    =(17,325))
ax.set_title('Model␣lunar␣equatorial␣sfc␣temperature,␣solar␣
    heating␣only')
```

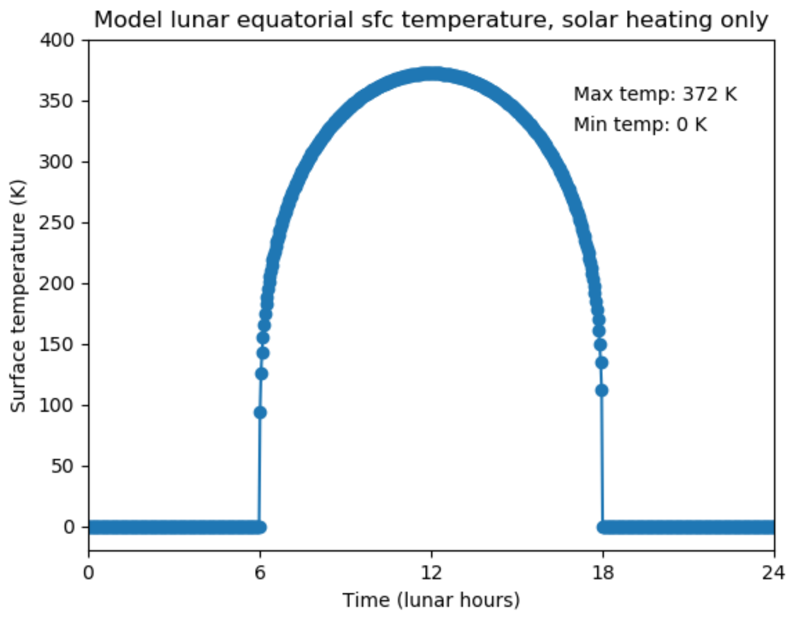

Fig. 1.6 Calculated surface temperature at a point on the solar equator of the Moon over a single lunar day, solar heating only, annotated with maximum and minimum temperatures

That annotation took a little bit of upfront work, but it will pay off now that are a bit better at effectively displaying our model results in a single visual representation. Later we should modularize our code so that we can reuse it, but for the next numerical experiments we will simply copy and paste the code we wish to modify and comment out the older versions.

1.6 Numerical experiment: Including cosmic background radiation in the surface energy balance

We can see from Figure 1.6 that our energy balance model is obviously incomplete. The nighttime temperatures are too cold (absolute zero). First of all we know that the coldest known place in the solar system (Hermite crater) doesn't even get that cold, and second we are trying to bring our model into harmony with the Diviner bolometric temperature observations, which indicate equatorial minimum temperatures around 100 K, with slow nighttime cooling. Let's see what happens when we add terms to the input side of our energy balance equation.

We know that the cosmic background radiation flux contribution to the surface energy budget is small, but does it affect the nighttime minimum temperature?

Starting with our good old equilibrium energy conservation equation, with each term expressed as a function of time:

$$q_{in}(t) = q_{out}(t) \tag{1.20}$$

in this iteration of our experiment we can add the cosmic background radiation flux by letting $q_{in}(t) = (1-A)S_E(t) + q_c$, where q_c is assumed to be constant. We can recall here Fixsen's [2] calculated cosmic background flux of $q_c = 3.13 \times 10^6 \, Wm^{-2}$.

Rewriting the equilibrium condition:

$$(1-A)S_E(t) + q_c = \varepsilon\sigma T_s^4 \tag{1.21}$$

and solving for the surface temperature:

$$T_s = \left(\frac{(1-A)S_E + q_c}{\varepsilon\sigma}\right)^{1/4} \tag{1.22}$$

we can see that adding terms to the input side of the energy balance can be accomplished in the temperature equation by simply adding our desired terms to the numerator. This can serve as a shortcut to full repeated derivations as we proceed.

Let's define `q_c` in solar.py:

```
# Moon climate parameters

q_c = 3.13e-6 # Cosmic background radiation flux [W m^-2] (Fixsen
    , 2009)
```

then comment out our old temperature equation in solar.py so that we can modify a copy to include q_C:

```
#T_s[n] = ((1-albedo)*solar[n]/epsilon/sigma)**(1/4) # Calculate
    surface temperature given solar flux
T_s[n] = (((1-albedo)*solar[n] + q_c)/epsilon/sigma)**(1/4) #
    Calculate surface temperature given solar flux and cosmic
    flux
```

We should also copy/comment out our plotting code and update the title in the new version (soon we will deal with this sort of code reuse more efficiently through modularization):

```
"""
fig, ax = plt.subplots()
ax.scatter(lunarhour, T_s)
ax.plot(lunarhour, T_s)
ax.set_xlim(0,24)
ax.set_ylim(-20,400)
ax.set_xticks(np.arange(0, 25, step=6))
ax.set_xlabel('Time (lunar hours)')
ax.set_ylabel('Surface temperature (K)')
ax.annotate('Max temp: {:.0f} K'.format(np.amax(T_s)), xy
    =(17,350))
ax.annotate('Min temp: {:.0f} K'.format(np.amin(T_s)), xy
    =(17,325))
ax.set_title('Model lunar equatorial sfc temperature, solar
    heating only')
plt.show()
"""

fig, ax = plt.subplots()
ax.scatter(lunarhour, T_s)
ax.plot(lunarhour, T_s)
ax.set_xlim(0,24)
ax.set_ylim(-20,400)
ax.set_xticks(np.arange(0, 25, step=6))
ax.set_xlabel('Time (lunar hours)')
ax.set_ylabel('Surface temperature (K)')
ax.annotate('Max temp: {:.0f} K'.format(np.amax(T_s)), xy
    =(17,350))
ax.annotate('Min temp: {:.0f} K'.format(np.amin(T_s)), xy
    =(17,325))
ax.set_title('Model lunar equatorial sfc temperature, solar +
    cosmic heating')
plt.show()
```

Now we can generate new model output:

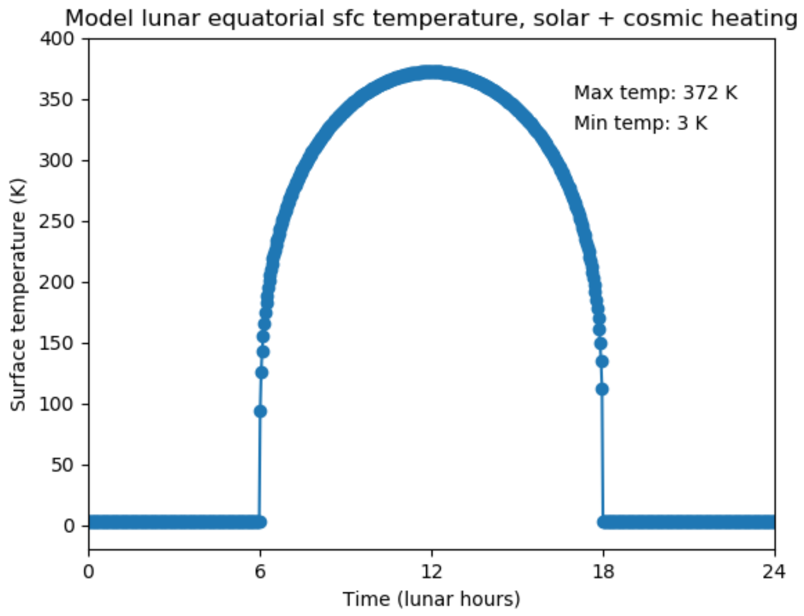

Fig. 1.7 Calculated surface temperature at a point on the solar equator of the Moon over a single lunar day, solar + cosmic heating, annotated with maximum and minimum temperatures

We recall that space is supposed to have an average temperature of 2.7 K. Is that what our model has reproduced? We can change the ax.annotate() line on our plot to show a single decimal point for the minimum temperature to see if this is the case:

```
ax.annotate('Min temp: {:.1f} K'.format(np.amin(T_s)), xy
    =(17,325))
```

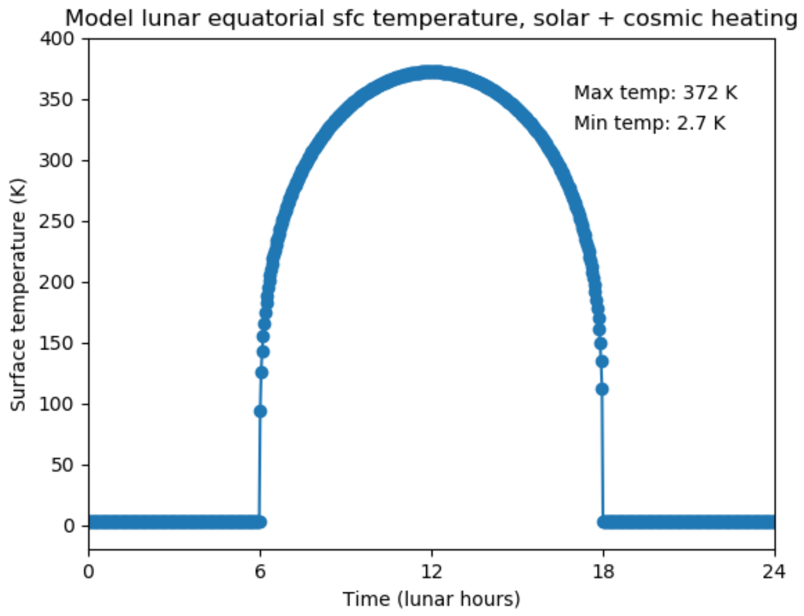

Fig. 1.8 Calculated surface temperature at a point on the solar equator of the Moon over a single lunar day, solar + cosmic heating, annotated with maximum and minimum temperatures, minimum expressed with one decimal point

Well that's certainly interesting, we've managed to simulate something closer to reality, but it's still too cold at night.

1.7 Including geothermal heating

We've already seen that all we have to do to add a term to the energy input in our temperature equation is add it to the numerator, and we can recall Siegler and Smrekar's [4] estimate of global lunar geothermal surface heat flux values q_g between $9-13\,mW\,m^{-2}$. Using a ballpark medium of $q_g = 11 \times 10^{-3}\,W\,m^{-2}$, we can declare this as a variable in solar.py:

```
# Moon parameters
q_g = 11e-3 # Mean lunar geothermal surface flux [W m^-2] (
    Siegler and Smrekar (2014)
```

Now we can modify the temperature calculation:

```
#T_s[n] = ((1-albedo)*solar[n]/epsilon/sigma)**(1/4) # Calculate
    surface temperature given solar flux
#T_s[n] = (((1-albedo)*solar[n] + q_c)/epsilon/sigma)**(1/4) #
    Calculate surface temperature given solar flux and cosmic
    flux
T_s[n] = (((1-albedo)*solar[n] + q_c + q_g)/epsilon/sigma)**(1/4)
     # Calculate surface temperature given solar, cosmic, and
    geothermal flux
```

and modify our plot code to make sure we have a new title and express the minimum temperature with only one decimal point:

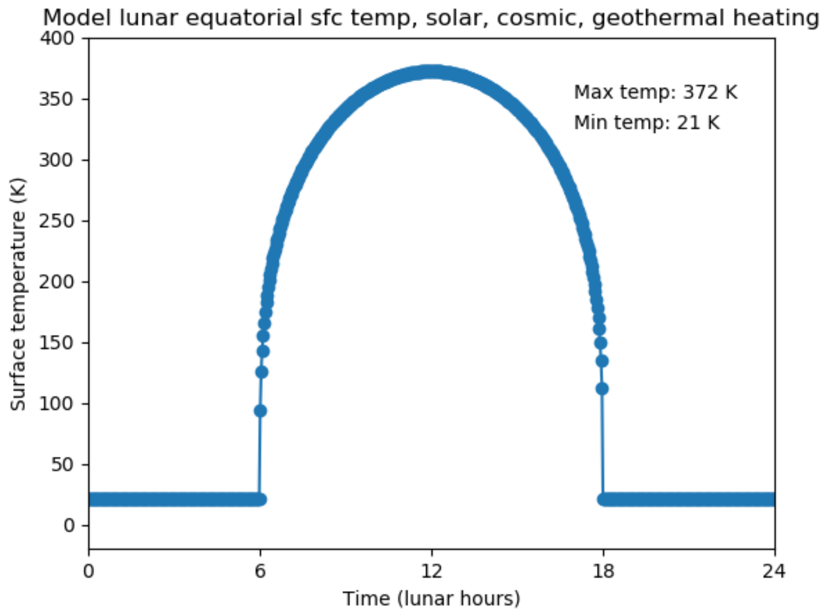

Fig. 1.9 Calculated surface temperature at a point on the solar equator of the Moon over a single lunar day, solar, cosmic, and geothermal heating, annotated with maximum and minimum temperatures

Now our nighttime temperature represents something close to the temperature we have observed in the permanent shade regions of Hermite crater, which can be understood to be controlled primarily by cosmic and geothermal heating. Notice that the max temp does not change under these varying scenarios, since insolation dwarfs other heating contributions. Still, to get our equatorial nighttime minimum to match observations, we have to account for yet another factor, the thermal inertia of the soil.

1.8 Approximating the influence of soil thermal inertia

A true dynamic calculation of the soil heat flux term requires a numerical solution of the heat conduction equation, which we will dive into in Chapter 3. However we can approximate the soil heating term q_{soil} by solving the energy balance equation given a ballpark nighttime temperature $T_s = 100\,K$.

$$q_{in} = q_{out} \tag{1.23}$$

$$q_c + q_g + q_{soil} = \varepsilon \sigma T_s^4 \tag{1.24}$$

$$q_{soil} = \varepsilon \sigma T_s^4 - q_c - q_g \tag{1.25}$$

Given that we have already laid down the values of each of these variables in solar.py, we can quickly perform this calculation by adding a line to the script:

```
# Moon climate parameters
# <other variables>

# estimate mean equatorial soil heat flux from ballpark nighttime
     temp of 100 K
q_soil = epsilon*sigma*(100**4) - q_c - q_g
print('q_soil: {:.2f}  W m-2'.format(q_soil))
```

```
q_soil: 5.55 W m-2
```

or in pretty math notation:

$$q_{soil} \approx 5.55\,W m^{-2} \tag{1.26}$$

at the solar equator.

It's a bit of a punt to just stick this number into our energy balance model, but it would be nice to see what our temperature curve looks like with a ballpark mean soil flux thrown in. We'll also update the title in our plotting code.

```
T_s[n] = (((1-albedo)*solar[n] + q_c + q_g + q_soil)/
    epsilon/sigma)**(1/4) # Calculate surface temperature
     given solar, cosmic, and geothermal flux

ax.set_title('Model lunar equatorial sfc temp with: S,
    q_c, q_g, q_soil')
```

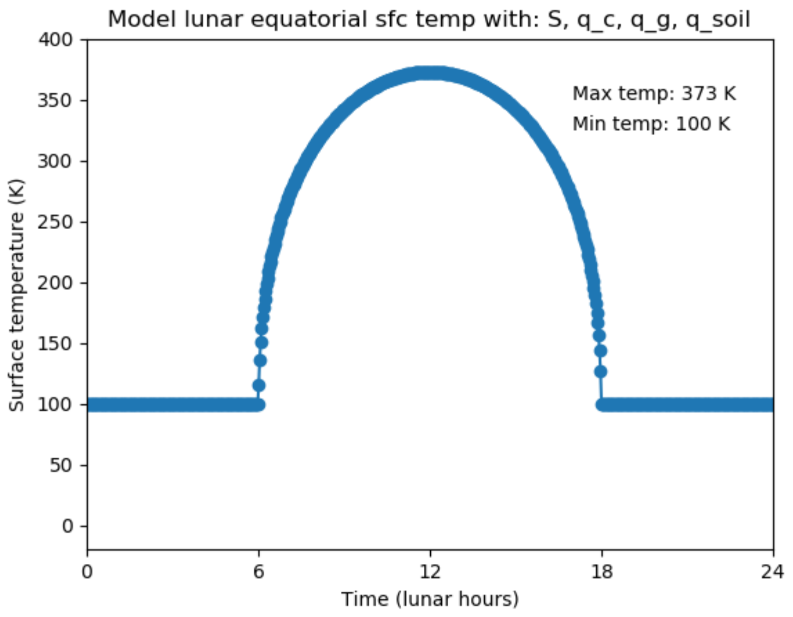

Fig. 1.10 Calculated surface temperature at a point on the solar equator of the Moon over a single lunar day, solar, cosmic, geothermal, and estimated soil heating, annotated with maximum and minimum temperatures

Considering the fact that we are using real data and real physics (except for the soil heat flux estimate), this is a pretty satisfactory approximation of the Diviner observed diurnal temperature curve at the solar equator. The only way we will be able to improve upon this model is to 1) clean up our code through modularization and version control (introduced in Chapter 2) and 2) model the conduction of solar energy into and out of the soil, providing our model with dynamic thermal inertia (which we will do in Chapter 3).

References

[1] Arya, A. S., Rajasekhar, R. P., Thangjam, G., Gopala Krishna, B. and Kiran Kumar, A. S. [2011]. Surface age and morphology of Hermite crater of lunar North Pole using high resolution datasets, *EPSC-DPS Joint Meeting 2011*, p. 1850.

[2] Fixsen, D. J. [2009]. The temperature of the cosmic microwave background, *The Astrophysical Journal* **707**(2): 916.

[3] Freeberg, A. and Jones, N. [2009]. Goddard Release No. 09-87. retrieved 11 Dec 2018.
URL: $https://www.nasa.gov/mission_pages/LRO/news/agu - results - 2009.html$

[4] Siegler, M. A. and Smrekar, S. E. [2014]. Lunar heat flow: Regional prospective of the Apollo landing sites, *Journal of Geophysical Research: Planets* **119**(1): 47–63.

[5] Vasavada, A. R., Bandfield, J. L., Greenhagen, B. T., Hayne, P. O., Siegler, M. A., Williams, J.-P. and Paige, D. A. [2012]. Lunar equatorial surface temperatures and regolith properties from the Diviner Lunar Radiometer Experiment, *Journal of Geophysical Research: Planets* **117**(E12).

[6] Williams, J.-P., Paige, D. A., Greenhagen, B. T. and Sefton-Nash, E. [2017]. The global surface temperatures of the Moon as measured by the Diviner Lunar Radiometer Experiment, *Icarus* **283**: 300–325.