EAS B9018 - Homework 1 Code

Problem 2: Plot the spectral emittance of 5 bodies in our solar system listed here:

- Sun (6000 K)
- Venus (600 K)
- Earth (300 K)
- Mars (200 K)
- Titan (120 K)

At which wavelength is the emittance a maximum for each body?

In [ ]:

```
# Suppress warnings
import logging, warnings
warnings.filterwarnings("ignore", category=FutureWarning)
logging.captureWarnings(True)

# Import analytical packages
import matplotlib.pyplot as plt, numpy as np

def S(lambda_, T):
    ''' Function to compute irradiance between two frequencies. '''
    # Planck constant, J-s
    h = 6.626e-34
    # Boltzmann constant, J K^-1
    k = 1.38e-23
    # Speed of light, m s^-1
    c = 3e8
    # Calculate spectral radiance
    s = (2*np.pi*h*c**2 / (lambda_**5))*(1/(np.exp(h*c/(lambda_*k*T))-1))

    # Return spectral radiance for the given spectrum in W sr^-1 m^-3
    return s

# Define body temperature (K)
bodies = {'Sun': 6000,
          'Venus': 600,
          'Earth': 300,
          'Mars': 200,
          'Titan': 120}
# Define wavelength spectrum to iterate over
wavelengths = np.arange(1e-9, 30e-6, 1e-9)

''' Part a. Plotting '''
# Initialize list to hold emittance results
emittances = []
# Iterate through all bodies and get emittances
for key, temperature in bodies.items():
    # Adjust so units are in W m^-2 um^-1
    emittance = [S(s, temperature) / (1e6) for s in wavelengths]
    # Get wavelength of maximum emittance using Wien's
    lambda_peak = 2.898e-3/temperature
    print('Peak emission wavelength of {0} is: {1:.2f} um'.format(key, lambda_peak/1e-6))
    emittances.append(emittance)

fig, ax = plt.subplots(dpi=300)
for i, emittance in enumerate(emittances):
    im = ax.plot(wavelengths * 1e6, emittance, label=list(bodies.keys())[i])
    ax.legend()
ax.set_xlim([0, 30])
ax.set_xlabel('Wavelength [$\mu m$]')
```

```
ax.set_ylabel('Emittance [$W m^{-2} \mu m^{-1}$]')
ax.set_yscale('log')
ax.set_ylim([1e-9, 1e9])
fig.tight_layout()
plt.show();
```

Problem 3: Assume that the sun emittance spectrum follows exactly Planck's formula, with T = 6000 K. Calculate the percent of solar energy in the following specctral regions:

1. Channel 1: 400 - 515 nm
2. Channel 2: 525 - 605 nm
3. Channel 3: 630 - 690 nm
4. Channel 4: 750 - 900 nm
5. Channel 5: 1550 - 1750 nm
6. Channel 6: 10400 - 12500 nm
7. Channel 7: 2090 - 2350 nm
8. Panchromatic: 520 - 900 nm

In [37]:

```
import matplotlib.pyplot as plt, numpy as np

def S(lambda_min, lambda_max, T):
    ''' Function to compute irradiance between two frequencies. '''
    # Planck constant, J-s
    h = 6.626e-34
    # Boltzmann constant, J K^-1
    k = 1.38e-23
    # Speed of light, m s^-1
    c = 3e8
    # Calculate spectral radiance
    s_max = (2 * np.pi * h * (c**2) / ((lambda_max**5)*(np.exp(h*c/(lambda_max * k * T)) -
    s_min = (2 * np.pi * h * (c**2) / ((lambda_min**5)*(np.exp(h*c/(lambda_min * k * T)) -

    # Return spectral radiance for the given spectrum in W sr^-1 m^-3
    return (lambda_max-lambda_min)*abs(s_max)

def integration(start=1e-9, d_lambda=1e-6, temperature=6000, criteria=0.15):
    ''' Basic numerical integration scheme. '''
    # Define list to hold all values
    irradiances = [0]
    # Define initial wavelength
    i = start
    # Convergence boolean - false if not converged, true if so
    convergence = False
    # While the solution hasn't converged (integral not fully computed), sum
    while not convergence:
        # Sum from a wavelength to an infinitesimally larger one (lambda + d_lambda)
        s = S(i, i + d_lambda, temperature)
        # Check for convergence
        if (s / irradiances[-1]) < criteria:
            convergence = True
        else:
            irradiances.append(s)
            i += d_lambda

    return np.nansum(irradiances)
```

In [ ]:

```
import matplotlib.pyplot as plt, numpy as np

def S(lambda_min, lambda_max, T):
    ''' Function to compute irradiance between two frequencies. '''
```

```python
    # Planck constant, J-s
    h = 6.626e-34
    # Boltzmann constant, J K^-1
    k = 1.38e-23
    # Speed of light, m s^-1
    c = 3e8
    # Calculate spectral radiance
    s_max = (2 * np.pi * h * (c**2) / ((lambda_max**5)*(np.exp(h*c/(lambda_max * k * T)) -
    s_min = (2 * np.pi * h * (c**2) / ((lambda_min**5)*(np.exp(h*c/(lambda_min * k * T)) -

    # Return spectral radiance for the given spectrum in W sr^-1 m^-3
    return (lambda_max-lambda_min)*abs(s_max)

def integration(start=1e-9, d_lambda=1e-6, temperature=6000, criteria=0.15):
    ''' Basic numerical integration scheme. '''
    # Define list to hold all values
    irradiances = [1e-9]
    # Define initial wavelength
    i = start
    # Convergence boolean - false if not converged, true if so
    convergence = False
    # While the solution hasn't converged (integral not fully computed), sum
    while not convergence:
        # Sum from a wavelength to an infinitesimally larger one (lambda + d_lambda)
        s = S(i, i + d_lambda, temperature)
        ratio = abs((s - irradiances[-1]) / s)
        # Optional print statement for troubleshooting
        #print('Wavelength: {0:.4e} | Current: {1:4e} | Previous: {2:.4e} | Ratio: {3:.5e
        # Conditional: if the previous-to-current ratio goes below the convergence ratio c
        # Alternate condition: if 100 um reached, break. Most of the solar spectrum should
        if ratio < criteria:
            break
        elif i > 100e-6:
            break
        else:
            irradiances.append(s)
            i += d_lambda

    return np.nansum(irradiances)

# Define temperature (K)
temperature = 6e3
# Define channels
channels = {'Channel 1': (400e-9, 515e-9),
            'Channel 2': (525e-9, 605e-9),
            'Channel 3': (630e-9, 690e-9),
            'Channel 4': (750e-9, 900e-9),
            'Channel 5': (1550e-9, 1750e-9),
            'Channel 6': (10400e-9, 12500e-9),
            'Channel 7': (2090e-9, 2350e-9),
            'Panchromatic': (520e-9, 900e-9)}
# Initialize dictionary to hold solar energy fractions
fractions = {}
# Get total solar energy
solar = integration(temperature=temperature, d_lambda=1e-10, criteria=1e-8)
# Definte temperature
# For each channel, get the fraction of solar energy in the spectral region
for key, value in channels.items():
    print(key)
    fractions[key] = (100 * S(value[0], value[1], temperature) / solar)
# Print
for key, value in fractions.items():
    print('{0}: {1:.2f} %'.format(key, value))
```