In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

from pprint import pprint
```

In [2]:
```
N_Avo = 6e23 # particles/mole, Avogadro's number
c = 3e10 # cm/s, speed of light
hbar = 1e-27 # erg.s, Planck constant (=h/(2pi))
h_Planck = hbar * 2*np.pi # erg/Hz, Planck constant (=hbar*2pi)
k_B = 1.4e-16 # erg/K, Boltzmann constant
r_sun = 8e10 # cm, radius of the sun
pc2cm = 3e18 # no. of cm per pc; 1pc = 3e18 cm
```

1: Practice with j_ν, α_ν, S_ν, and I_ν

1.1

In [3]:
```
n_gas = 10 # molecules/cm^3, number density of H2
r_grain = 1e-5 # cm, radius of grain
rho_grain = 3 # g/cm^3, material density of grain
ratio_dust_gas = 0.01 # ratio of dust to gas
```

In [4]:
```
# Grain mass (g)
m_grain = 4/3 * np.pi * r_grain**3 * rho_grain

# Gas + dust macro density (g/cm^3)
rho_gas = n_gas * 2/N_Avo
rho_dust = rho_gas * ratio_dust_gas

# Dust number density (cm^-3)
n_dust = rho_dust / m_grain

print(f"Dust Mass:\t\t {m_grain}\tg/grain")
print(f"Gas Macro Density:\t {rho_gas}\tg/cm^3")
print(f"Dust Macro Density:\t {rho_dust}\tg/cm^3")
print(f"Dust Number Density:\t {n_dust}\tgrains/cm^3")
```

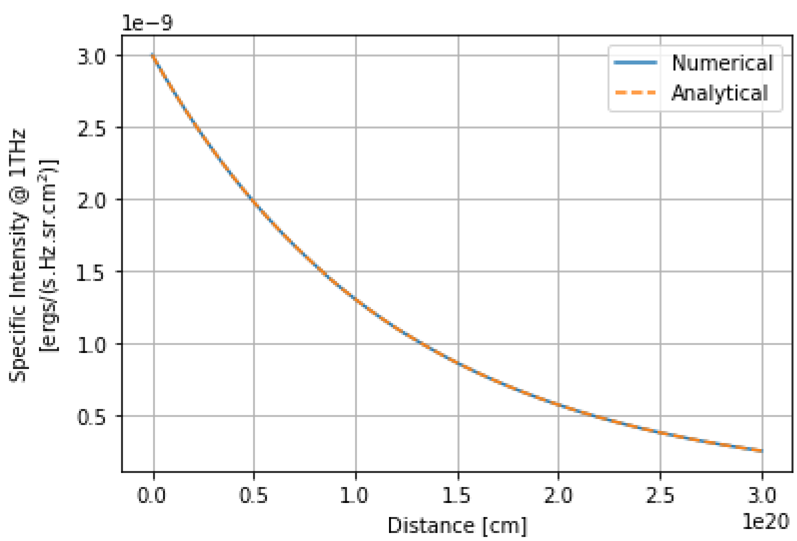
```
Dust Mass:               1.2566370614359175e-14 g/grain
Gas Macro Density:       3.3333333333333333e-23 g/cm^3
Dust Macro Density:      3.3333333333333335e-25 g/cm^3
Dust Number Density:     2.652582384864922e-11  grains/cm^3
```

1.2: Dust Extinction

In [5]:
```
I_nu0 = 3e-9 # erg/(s.Hz.sr.cm^2), backlight specific intensity
nu = 1e12 # Hz, backlight frequency
s_max = 100 * pc2cm # cm, thickness of the medium
```

In [6]:
```
# Calculating the extinction coefficient
sigma_grain = np.pi * r_grain**2 # cm^2, x-section of a dust particle
alpha_nu = n_dust * sigma_grain # 1/cm

N_steps = 10000 # Sufficiently small steps
s_vec = np.linspace(0, s_max, N_steps)

# Numerically solving
I_nu_extinction = [I_nu0] + [0]*(N_steps-1)
for i in range(1, N_steps):
    ds = s_vec[i] - s_vec[i-1]
    dI_nu = -alpha_nu * I_nu_extinction[i-1] * ds
    I_nu_extinction[i] = I_nu_extinction[i-1] + dI_nu

# Analytically solving
I_nu_extinction_ideal = I_nu0*np.exp(-alpha_nu*s_vec)

# Plot
plt.plot(s_vec, I_nu_extinction, label='Numerical')
plt.plot(s_vec, I_nu_extinction_ideal, '--', label='Analytical')

plt.legend()
plt.grid(True)
plt.xlabel('Distance [cm]')
plt.ylabel('Specific Intensity @ 1THz\n[ergs/(s.Hz.sr.cm$^2$)]')
```

Out[6]:
```
Text(0,0.5,'Specific Intensity @ 1THz\n[ergs/(s.Hz.sr.cm$^2$)]')
```

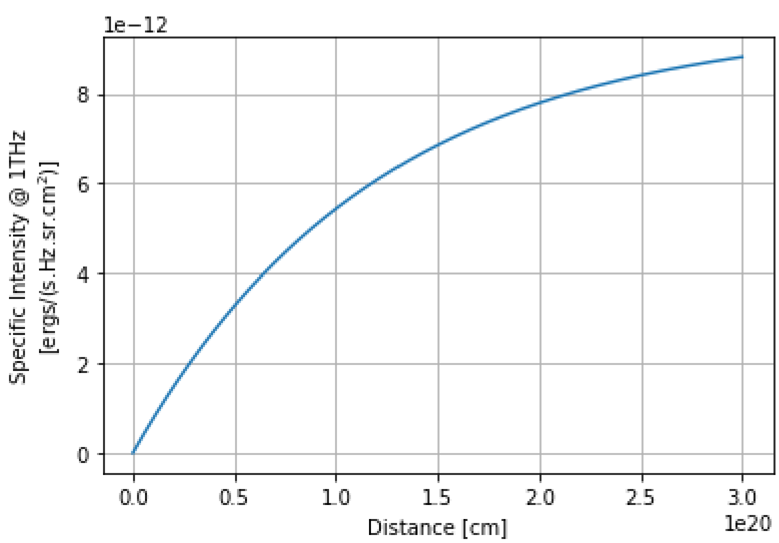


1.3: Dust Emission

In [7]:
```
T = 50
nu = 1e12

def fun_planck(nu, T):
    '''
    Inputs:
        nu: Float. Frequency of interest in Hz.
        T: Float. Temperature in Kelvin.
    Returns:
        The spectral radiance of a body given nu and T, in (erg/s)/(sr.Hz.cm^2)
    '''
    return (2 * h_Planck * nu**3 / c**2) * (1/(np.exp(h_Planck*nu/(k_B*T)) - 1))
```

In [8]:
```
# Spectral radiance (erg/s)/(sr.Hz.cm^2) at 1THz, 50K
B_nu = fun_planck(nu, T)

# Emissivity
j_nu = B_nu * sigma_grain * n_dust

# Accounting for self-absorption for observation
S_nu = j_nu / alpha_nu
```

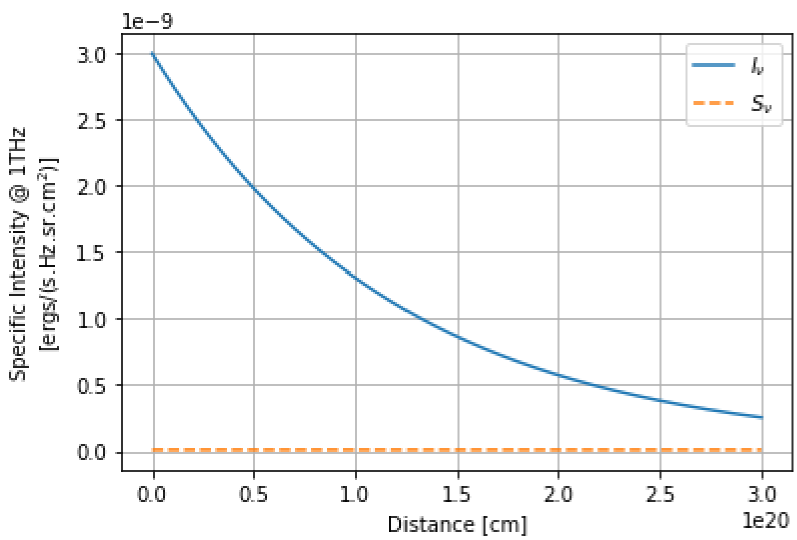
In [9]:
```
I_nu_sans_backlight = S_nu * (1 - np.exp(-alpha_nu * s_vec))

plt.plot(s_vec, I_nu_sans_backlight)
plt.xlabel('Distance [cm]')
plt.ylabel('Specific Intensity @ 1THz\n[ergs/(s.Hz.sr.cm$^2$)]')
plt.grid(True)
```

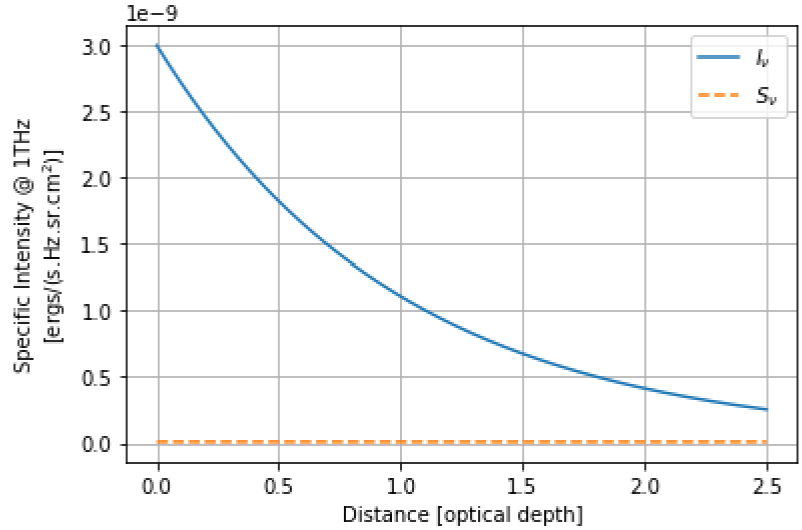


1.4: Extinction and Emission

In [10]:
```
I_nu = I_nu_extinction + I_nu_sans_backlight

# Specific intensity vs. distance
plt.plot(s_vec, I_nu, label=r'$I_\nu$')
plt.plot(s_vec, [S_nu]*len(s_vec), '--', label=r'$S_\nu$')
plt.xlabel('Distance [cm]')
plt.ylabel('Specific Intensity @ 1THz\n[ergs/(s.Hz.sr.cm$^2$)]')
plt.grid(True)
plt.legend()

# Specific intensity vs. optical depth
plt.figure()
plt.plot(s_vec*alpha_nu, I_nu, label=r'$I_\nu$')
plt.plot(s_vec*alpha_nu, [S_nu]*len(s_vec), '--', label=r'$S_\nu$')
plt.xlabel('Distance [optical depth]')
plt.ylabel('Specific Intensity @ 1THz\n[ergs/(s.Hz.sr.cm$^2$)]')
plt.grid(True)
plt.legend()
```

Out[10]: <matplotlib.legend.Legend at 0x189191c9f60>





2: Brightness, Magnitudes, and Photons

In [11]:
```
def calc_m_i(F_i, F_0i, m_0i):
    '''
    Inputs:
        F_i: Float. The observed flux integrated over a given filter.
        F_0i: Float. The calibration flux factor.
        m_0i: Float. The calibration magnitude factor.
    Returns:
        m_i: The magnitude of an object described by F_i, calibrated against F_0i
    '''
    return -2.5*np.log10(F_i/F_0i) + m_0i
```

2.1

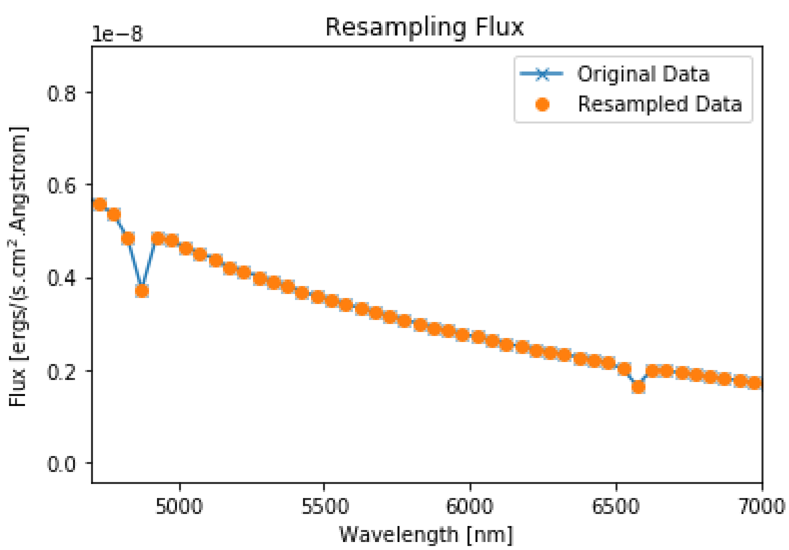
In [12]:
```
fname_filter = '../../bessel_V.dat'
fname_vega = '../../vega_spectrum.dat'

data_filter = np.loadtxt(fname_filter)
data_vega = np.loadtxt(fname_vega)

d_Keck = 10 # m, telescope diameter
A_Keck = np.pi*(d_Keck/2)**2 # m^2, telescope area
```

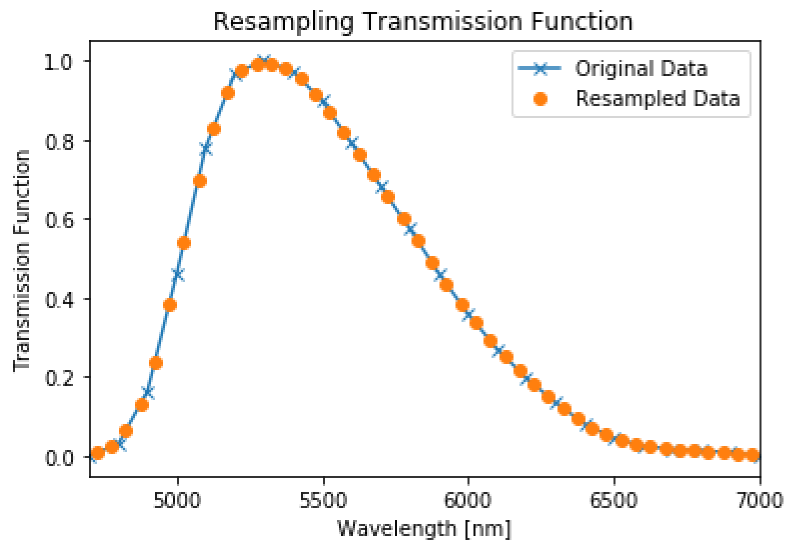
In [13]:
```
# lamb_vec = data_filter[:,0] # Angstroms
lamb_vec = data_vega[:,0] # Angstroms
E_vec = [h_Planck * c /(lamb*1e-8) for lamb in lamb_vec] # ergs, energy for a giv

# Interpolating the Vega spectrum vs. wavelength
F_lamb_spline = interp1d(data_vega[:,0],
                         data_vega[:,1],
                         kind='linear',
                         fill_value=0)

# Filter transmission interpolation
phi_spline = interp1d(data_filter[:,0],
                      data_filter[:,1],
                      kind='linear',
                      bounds_error=False,
                      fill_value=0)

# Integrate
F_vec = [0] * len(lamb_vec)
for i, lamb in enumerate(lamb_vec[:-1]):
    dlamb = lamb_vec[i+1] - lamb
    F_lamb = F_lamb_spline(lamb)
    phi = phi_spline(lamb)
    F_vec[i] = F_lamb*phi * dlamb

# F_i = sum(F_vec)
count_vec = {lamb_vec[i]:(F_vec[i]/E_vec[i]) for i,_ in enumerate(lamb_vec)} # wa
count_Keck = sum(count_vec.values())*1e4 * A_Keck
print(f'Photons/Second: {int(round(count_Keck))}')
```

```
Photons/Second: 729648964153
```

In [14]:
```
lamb_min = min(data_filter[:,0])
lamb_max = max(data_filter[:,0])

# Graphical demonstration of resampling the filter data
plt.plot(data_vega[:,0], data_vega[:,1], '-x', label='Original Data')
plt.plot(lamb_vec, F_lamb_spline(lamb_vec), 'o', label='Resampled Data')

plt.xlim([lamb_min, lamb_max])
plt.legend()
plt.ylabel('Flux [ergs/(s.cm$^2$.Angstrom]')
plt.xlabel('Wavelength [nm]')
plt.title('Resampling Flux')

# Graphical demonstration of resampling the transmission function
plt.figure()

plt.plot(data_filter[:,0], data_filter[:,1], '-x', label='Original Data')
plt.plot(lamb_vec, phi_spline(lamb_vec), 'o', label='Resampled Data')

plt.xlim([lamb_min, lamb_max])
plt.legend()
plt.ylabel('Transmission Function')
plt.xlabel('Wavelength [nm]')
plt.title('Resampling Transmission Function')
```

Out[14]:
```
Text(0.5,1,'Resampling Transmission Function')
```





2.2

In [15]:
```
# Keck receive beamwidth (given wavelength) in radians; mind units
calc_beamwidth = lambda lamb, D : 1.22*lamb/D
beamwidth_vec = calc_beamwidth(lamb_vec*1e-10, d_Keck)

x_vega = 8 * pc2cm # cm, distance to Vega
d_vega = 2.5 * r_sun # cm, Vega diameter

theta_vega = np.arctan(d_vega / x_vega) # radians, angle of Vega in the sky
Omega_vega = theta_vega**2 # ish; solid angle of Vega in the sky

# Checking that Vega doesn't appear larger than the telescope's receiving beam...
idx_trouble_lst = [i for i, theta in enumerate(beamwidth_vec) if theta < theta_ve
if idx_trouble_lst:
    print(f'Vega larger than beamwidth at wavelengths (A):\n{idx_trouble_lst}')
```

In [16]:
```
# Passlength of interest (cm)
lamb_bw = (max(lamb_vec) - min(lamb_vec))*1e-8

F_i = sum(F_vec) # erg/(s.cm^2), observed flux integrated over a given filter
I_lamb = F_i/(lamb_bw * Omega_vega)
print(f'Specific Intensity: {"{:.2e}".format(I_lamb)} erg/(s.cm^2.sr.cm)')

# i.e. Vega's a fair bit brighter than the sun
```

```
Specific Intensity: 8.12e+13 erg/(s.cm^2.sr.cm)
```

2.3

Halving the distance to Vega would increase the number of photons by 4x.

The specific intensity wouldn't change; this is because the solid angle that Vega occupies in the sky (Ω_{Vega} in 2.2) would scale up by roughly 4x as well. (The angle that Vega occupies in the sky is still smaller than the field of view of the telescope.)

3: Dust Bowl

In [17]:
```
r_grain = 100e-6/2 # m, dust particle radius
tau = 3 # approximate optical depth
x_vis = 1.5 # m, drop-off point for visibility
```

3.1

In [18]:
```
sigma_grain = np.pi*r_grain**2 # m^2, x-section of a single dust particle

alpha = tau/x_vis
n_air = alpha/sigma_grain
print(f'Dust Number Density:\t{"{:.2e}".format(n_air)} particles/m^3')
```

```
Dust Number Density:    2.55e+08 particles/m^3
```

3.2

In [19]:
```
n_ground = 1/(r_grain*2)**3 # no. particles/m^3, number density of particles on t
z_air = 8e4 # cm, height in the air
z_ground = n_air*z_air / n_ground

print(f'Lost Topsoil:\t{z_ground} cm')
```

```
Lost Topsoil:   20.371832715762604 cm
```

In [ ]:
```
```