Statistical Mechanics for B.Sc. 5th semester python

Statistical Distributions

Number of particles of energy ϵ

$$n(\epsilon) = g(\epsilon)f(\epsilon)$$

where, $g(\epsilon)$ = number of states of energy ϵ = statistical weight corresponding to energy ϵ

$f(\epsilon)$ = distribution function = average number of particles in each state of energy ϵ = probability of occupancy of each state of energy ϵ

There are three kinds of distribution

- Identical particles that are sufficiently far apart to be distinguishable, for instance, the molecules of a gas. In quantum terms, the wave functions of the particles overlap to a negligible extent. The Maxwell-Boltzman distribution function holds for such particles.
- Identical particles of 0 or integral spin that cannot be distinguished one from another because their wave functions overlap. Such particles, called bosons, do not obey the Pauli exclusion principle, and the Bose-Einstein distribution function

holds for them. Photons are in this category, and we shall use Bose-Einstein statistics to account for the spectrum of radiation from a blackbody.

- Identical particles with odd half-integral spin (1/2, 3/2, 5/2, . .) that also cannot be distinguished one from another. Such particles, called fermions, obey the exclusion principle, and the Fermi-Dirac distribution function holds for them. Electron are in this category, and we shall use Fermi-Dirac statistics to study the behavior of the free electrons in metal that are responsible for its ability to conduct electric current.

Maxwell-Boltzmann Statistics

Identical particles that are sufficiently far apart to be distinguishable, for instance, the molecules of a gas. In quantum terms, the wave functions of the particles overlap to a negligible extent. The Maxwell-Boltzman distribution function holds for such particles.

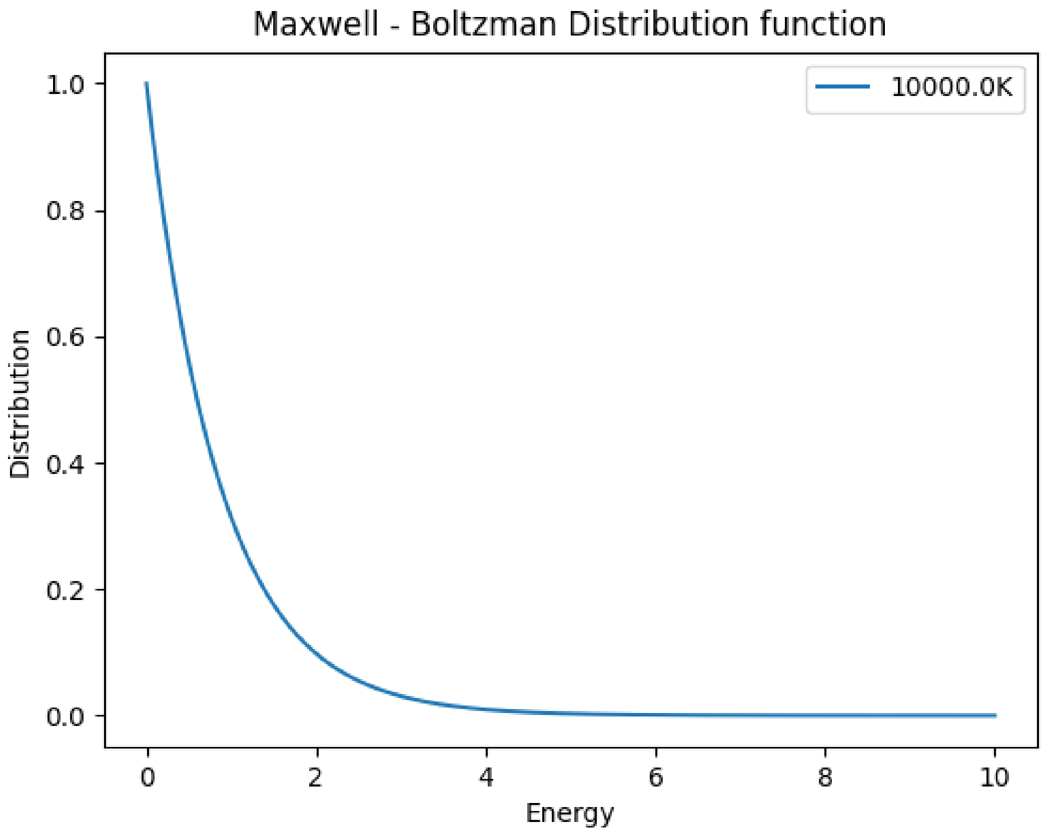
Maxwell-Boltzmann distribution function

$$f_{MB}(\epsilon) = Ae^{-\frac{\epsilon}{kT}}$$

```
from matplotlib import pyplot as plt
import numpy as np
from scipy import constants

x = np.linspace(0, 10, 10000)

# Maxwell Boltzman distribution
T = 10000.0
k = constants.k
kev = k/constants.e
y = np.exp(-x/(kev*T))
# Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.plot(x, y, label= str(T) + 'K')  # Plot some data on the axes.
# Plot more data on the axes...
ax.set_xlabel("Energy")  # Add an x-label to the axes.
ax.set_ylabel("Distribution")  # Add a y-label to the axes.
ax.set_title("Maxwell - Boltzman Distribution function")  # Add a title to the axes.
ax.legend()  # Add a legend.
plt.show()
```

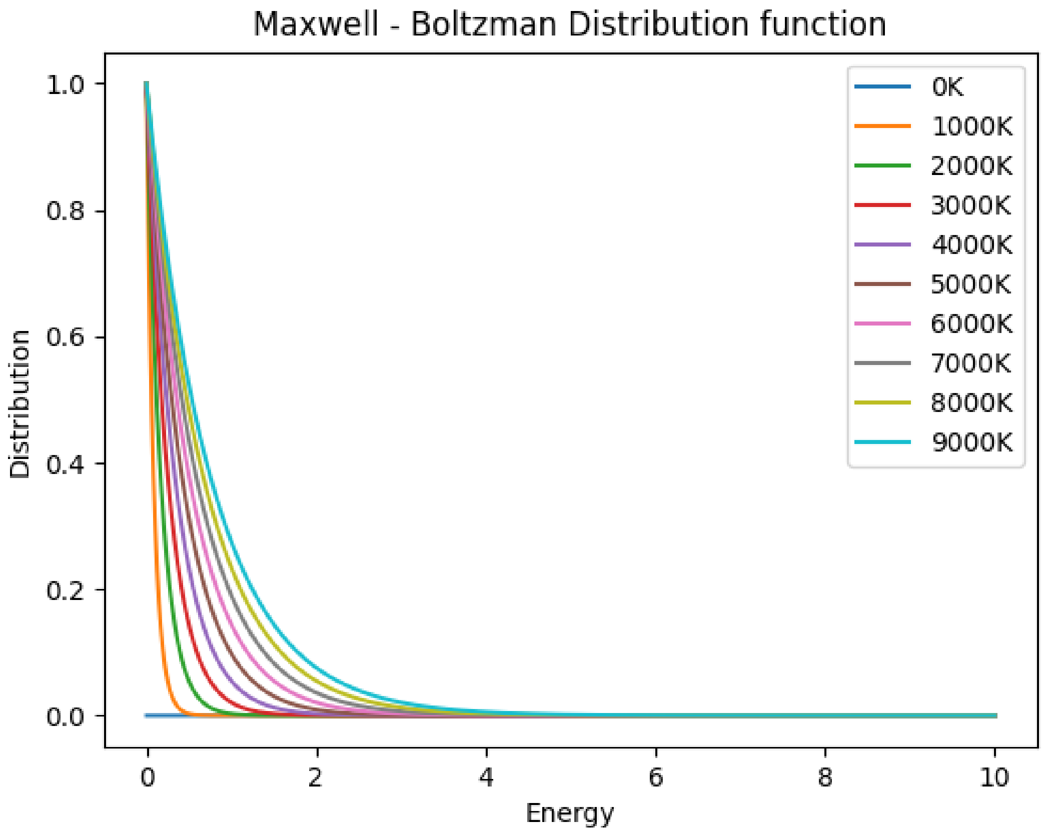


Variation of distribution function with temperature

```
from matplotlib import pyplot as plt
import numpy as np
from scipy import constants

x = np.linspace(0, 10, 10000)

# Maxwell Boltzman distribution
k = constants.k
kev = k/constants.e
fig, ax = plt.subplots()  # Create a figure and an axes.
for T in range(0,10000,1000):
    y = np.exp(-x/(kev*T))
    # Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
    ax.plot(x, y, label= str(T) + 'K')  # Plot some data on the axes.
# Plot more data on the axes...
ax.set_xlabel("Energy")  # Add an x-label to the axes.
ax.set_ylabel("Distribution")  # Add a y-label to the axes.
ax.set_title("Maxwell - Boltzman Distribution function")  # Add a title to the axes.
ax.legend()  # Add a legend.
plt.show()
```



Quantum Statistics

Bose-Einstein distribution function

Definition

Particles with 0 or integral spins, which are bosons. Bosons do not obey the exclusion principle, and the wave function of a system of bosons is not affected by the exchange of any pair of them. A wave function of this kind is called symmetric. Any number of bosons can exist in the same quantum state of the system.

Bose-Einstein distribution function

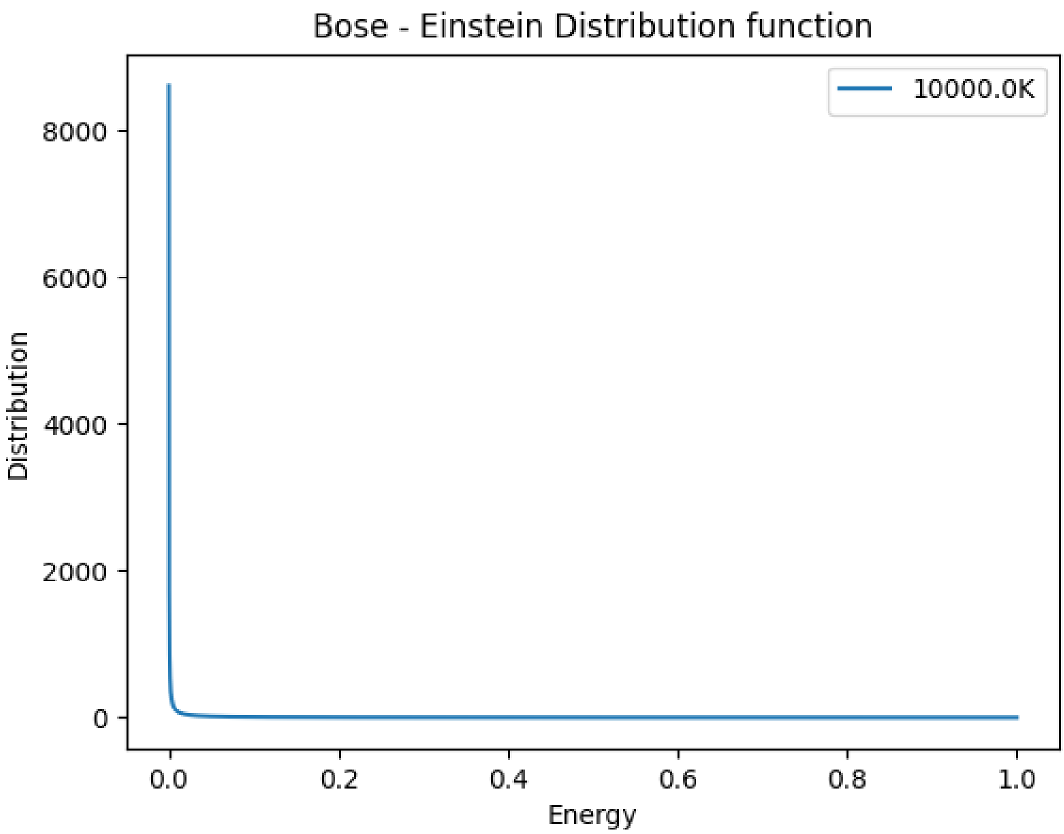
$$f_{BE}(\epsilon) = \frac{1}{e^{\alpha} e^{\epsilon/kT} - 1}$$

```
from matplotlib import pyplot as plt
import numpy as np
from scipy import constants

x = np.linspace(0, 1, 10000)

# Maxwell Boltzman distribution
```

```
T = 10000.0
k = constants.k
kev = k/constants.e
y = 1/(np.exp(x/(kev*T))-1)
# Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.plot(x, y, label= str(T) + 'K')  # Plot some data on the axes.
# Plot more data on the axes...
ax.set_xlabel("Energy")  # Add an x-label to the axes.
ax.set_ylabel("Distribution")  # Add a y-label to the axes.
ax.set_title("Bose - Einstein Distribution function")  # Add a title to the axes.
ax.legend()  # Add a legend.
plt.show()
```



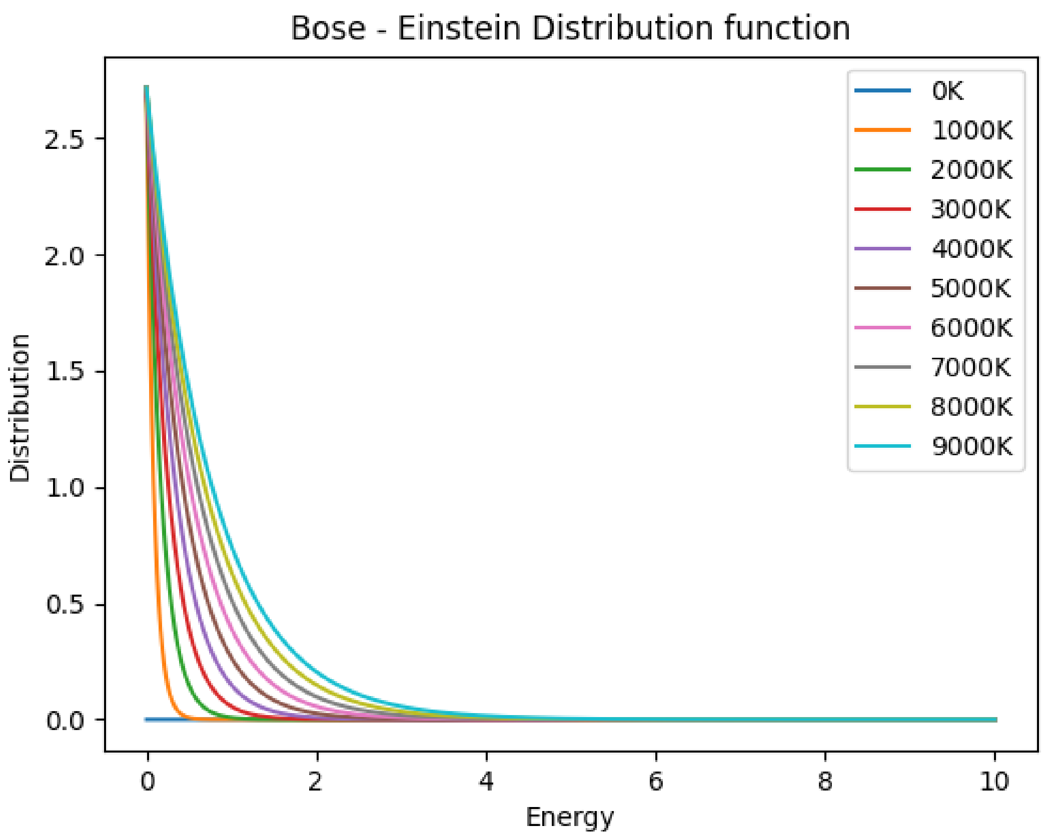
Variation of distribution function with temperature

```
from matplotlib import pyplot as plt
import numpy as np
from scipy import constants

x = np.linspace(0, 10, 10000)

# Maxwell Boltzman distribution
k = constants.k
```

```
kev = k/constants.e
fig, ax = plt.subplots()  # Create a figure and an axes.
for T in range(0,10000,1000):
    y = 1/(np.exp(x/(kev*T)-1))
    # Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
    ax.plot(x, y, label= str(T) + 'K')  # Plot some data on the axes.
# Plot more data on the axes...
ax.set_xlabel("Energy")  # Add an x-label to the axes.
ax.set_ylabel("Distribution")  # Add a y-label to the axes.
ax.set_title("Bose - Einstein Distribution function")  # Add a title to the axes.
ax.legend()  # Add a legend.
plt.show()
```



Fermi-Dirac distribution function

Definition

Identical particles with odd half-integral spin (1/2, 3/2, 5/2, . .) that also cannot be distinguished one from another. Such particles, called fermions, obey the exclusion principle, and the Fermi-Dirac distribution function holds for them. Electron are in this category, and we shall use Fermi-Dirac statistics to study the behavior of the free electrons in metal that are responsible for its ability to conduct electric current.

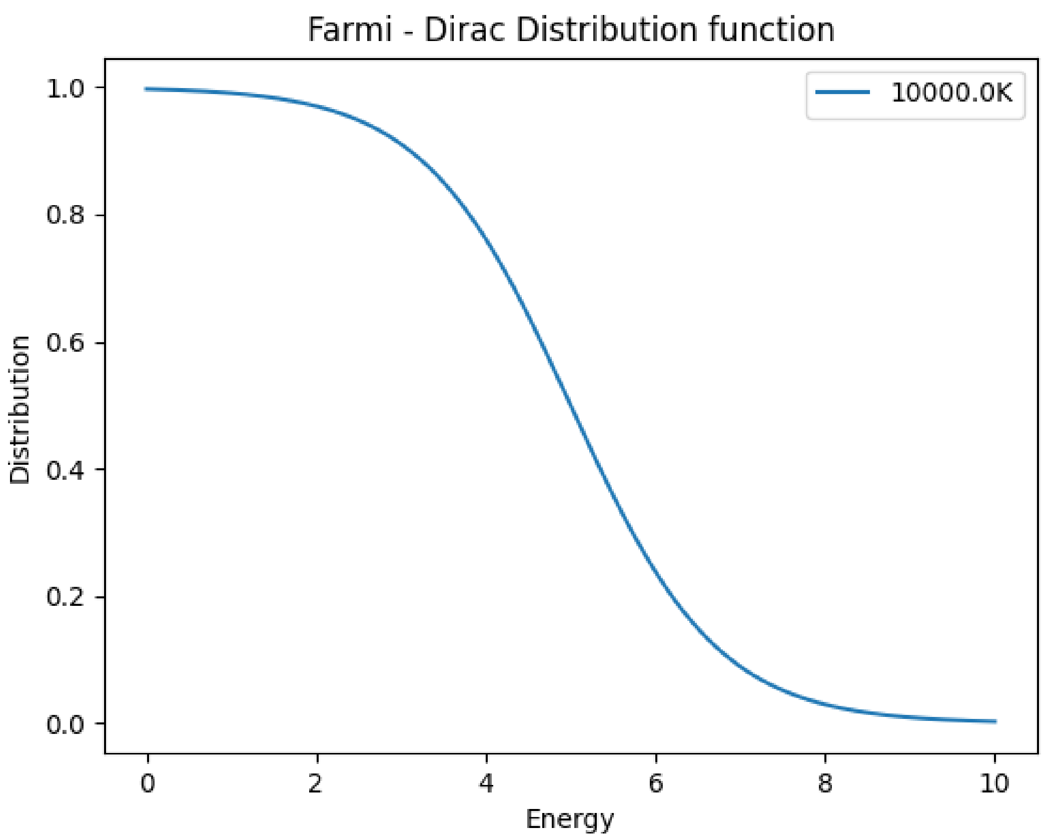
Fermi-Dirac distribution function

$$f_{FD}(\epsilon) = \frac{1}{e^{\alpha}e^{\epsilon/kT} + 1}$$

```
from matplotlib import pyplot as plt
import numpy as np
from scipy import constants

x = np.linspace(0, 10, 10000)

# Maxwell Boltzman distribution
T = 10000.0
k = constants.k
kev = k/constants.e
y = 1/(np.exp(x/(kev*T))+1)
# Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.plot(x, y, label= str(T) + 'K')  # Plot some data on the axes.
# Plot more data on the axes...
ax.set_xlabel("Energy")  # Add an x-label to the axes.
ax.set_ylabel("Distribution")  # Add a y-label to the axes.
ax.set_title("Farmi - Dirac Distribution function")  # Add a title to the axes.
ax.legend()  # Add a legend.
plt.show()
```

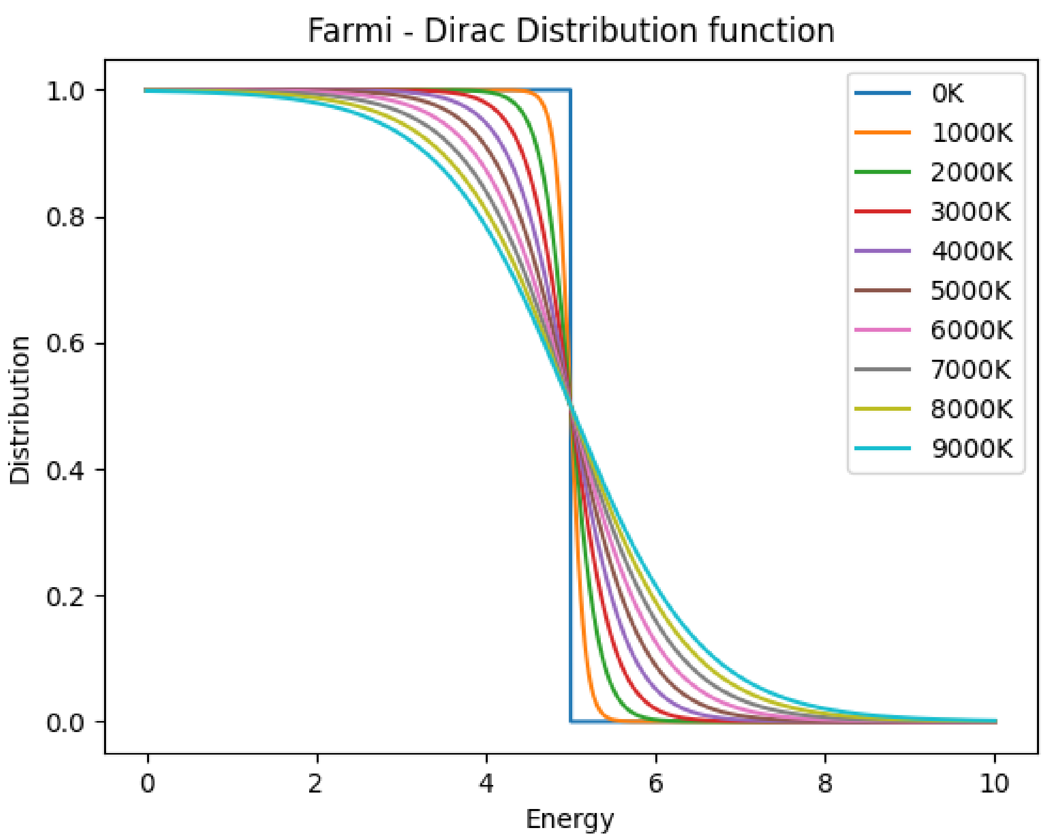


Fermi-Dirac distribution function

```
from matplotlib import pyplot as plt
import numpy as np
from scipy import constants

x = np.linspace(0, 10, 10000)

# Maxwell Boltzman distribution
k = constants.k
kev = k/constants.e
fig, ax = plt.subplots()  # Create a figure and an axes.
for T in range(0,10000,1000):
    y = 1.0/(np.exp(x/(kev*T)+1))
    # Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
    ax.plot(x, y, label= str(T) + 'K')  # Plot some data on the axes.
# Plot more data on the axes...
ax.set_xlabel("Energy")  # Add an x-label to the axes.
ax.set_ylabel("Distribution")  # Add a y-label to the axes.
ax.set_title("Farmi - Dirac Distribution function")  # Add a title to the axes.
ax.legend()  # Add a legend.
plt.show()
```

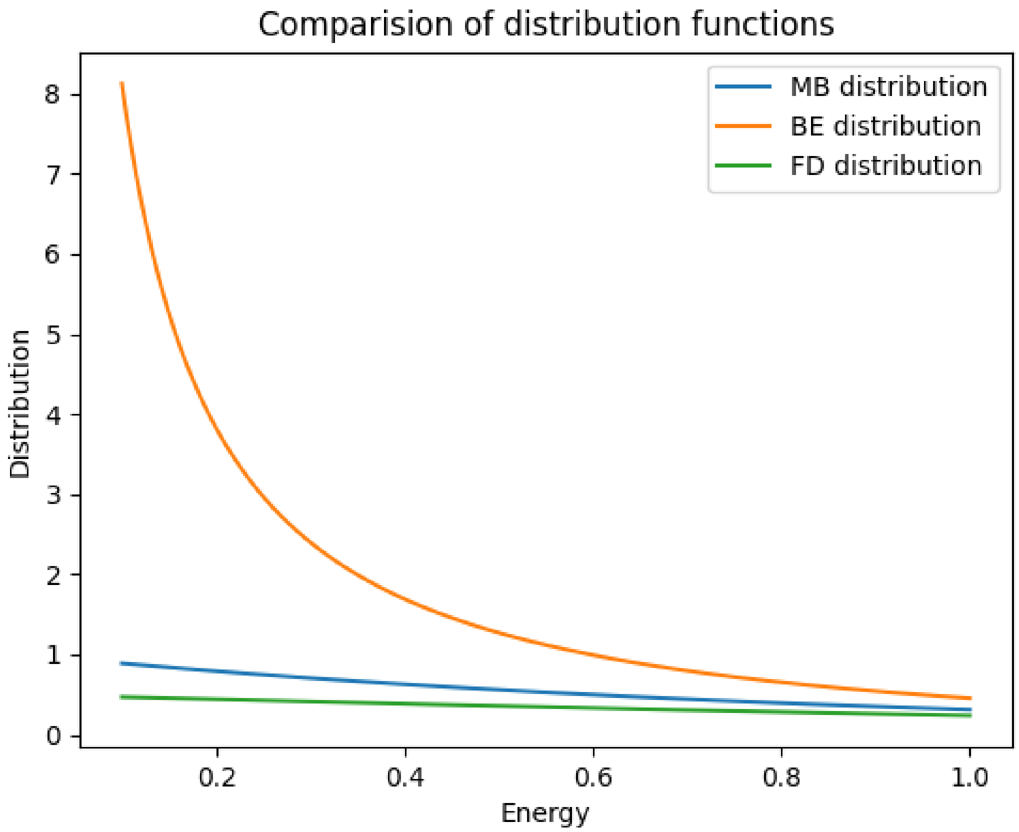


Comparison of three distributions

```
from matplotlib import pyplot as plt
import numpy as np
from scipy import constants

x = np.linspace(0.1, 1, 1000)

# Maxwell Boltzman distribution
T = 10000.0
k = constants.k
kev = k/constants.e
ymb = np.exp(-x/(kev*T))
ybe = 1.0/(np.exp(x/(kev*T))-1.0)
yfd = 1.0/(np.exp(x/(kev*T))+1.0)
# Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.plot(x, ymb, label= "MB distribution")  # Plot some data on the axes.
ax.plot(x, ybe, label= "BE distribution")
ax.plot(x, yfd, label= "FD distribution")
# Plot more data on the axes...
ax.set_xlabel("Energy")  # Add an x-label to the axes.
ax.set_ylabel("Distribution")  # Add a y-label to the axes.
ax.set_title("Comparision of distribution functions")  # Add a title to the axes.
ax.legend()  # Add a legend.
plt.show()
```


Comparision of distribution functions
MB distribution
BE distribution
FD distribution
Distribution
Energy
0
1
2
3
4
5
6
7
8
0.2
0.4
0.6
0.8
1.0