Module `fermi_eos`

Functions

```
def beta_equilibrium_condition(k_p, k_n)
```

Condition of beta equilibrium in order to compute k_p as a function of k_n, in the case of a generic case of degenerate ideal Fermi gas of neutrons, protons and electrons. If it returns zero, then the Fermi momentum of the proton is the expected one, since it means that chemical potentials of neutrons, protons and electrons match the equilibrum condition for the (beta) weak interactions. Therefore, the k_p given in input is the desidered value of k_p.

Parameters

`k_p` : `float`
: Fermi momentum of protons in g cm / s.

`k_n` : `float`
: Fermi momentum of neutrons in g cm / s.

Returns

`float`
: Chemical potential in erg.

```
def compute_energy_density_emp(p)
```

Compute energy density from pressure, in the case of empirical interactions.

Parameters

`p` : `float`
: Pressure in dyne/cm^2.

Raises

`ValueError`
: If energy density is negative.

Returns

`float`
: Energy density in erg/cm^3.

```
def compute_energy_density_from_u_emp(u)
```

Compute energy density from u = n / n_0, in the case of empirical interactions.

Parameters

u : `float`
: Dimensionless u = n / n_0.

Returns

`float`
: Energy density in erg/cm^3.

```
def compute_energy_density_from_u_sky(u)
```

Compute energy density from u = n / n_0, in the case of Skyrme Hatree-Fock interactions.

Parameters

u : `float`
: Dimensionless u = n / n_0.

Returns

`float`
: Energy density in erg/cm^3.

```
def compute_energy_density_from_x_gen(x)
```

Compute energy density from x = k / (m c), in the case of a generic case of degenerate ideal Fermi gas of neutrons.

Parameters

x : `float`
: Dimensionless x = k / (m c).

Raises

`ValueError`
: If energy density is negative.

Returns

`float`
: Energy density in erg/cm^3.

```
def compute_energy_density_from_x_npe(x, e_0)
```

Compute energy density from x = k / (m c) and prefactor e_0, in the case of a generic case of degenerate ideal Fermi gas of neutrons, protons and electrons.

Parameters

x : `float`
: Dimensionless x = k / (m c).

e_0 : `float`
: Prefactor in erg/cm^3.

Returns

`float`
: Energy density in erg/cm^3.

```
def compute_energy_density_gen(p)
```

Compute energy density from pressure, in the case of a generic case of degenerate ideal Fermi gas of neutrons.

Parameters: - p : float Pressure in dyne/cm^2.

Returns: - float Energy density in erg/cm^3.

```
def compute_energy_density_npe(k_n)
```

In presence of protons, electrons and neutrons, compute energy density from Fermi momentum of neutrons, in the case of a generic case of degenerate ideal Fermi gas of neutrons, protons and electrons.

Parameters

k_n : `float`
: Fermi momentum of neutrons in g cm / s.

Raises

`ValueError`
: If energy density is negative.

`Returns:`

`float`
: Energy density in erg/cm^3.

```
def compute_energy_density_pe(k_p)
```

In presence of protons and electrons (no neutrons), compute energy density from Fermi momentum of protons, in the case of a generic case of degenerate ideal Fermi gas of neutrons, protons and electrons.

Parameters

k_n : `float`

Fermi momentum of protons in g cm / s.

Raises

`ValueError`
: If energy density is negative.

Returns

`float`
: Energy density in erg/cm^3.

```
def compute_energy_density_sky(p)
```

Compute energy density from pressure, in the case of Skyrme Hatree-Fock interactions.

Parameters

p : `float`
: Pressure in dyne/cm^2.

Raises:

ValueError If energy density is negative.

Returns

`float`
: Number density in erg/cm^3.

```
def compute_fermi_momentum_nonrel(p)
```

Compute Fermi momentum from pressure in the case of a non-relativistic degenerate ideal Fermi gas of neutrons.

Parameters

p : `float`
: Pressure in dyne/cm^2.

Returns

`float`
: Fermi momentum in g cm/s.

```
def compute_k_n(n)
```

Compute Fermi momentum of neutrons from number density, in the case of a generic case of degenerate ideal Fermi gas of neutrons, protons and electrons.

Parameters

n : `float`
Number density in cm^-3.

Returns

`float`
Fermi momentum of neutrons in g cm / s.

```
def compute_k_p(k_n)
```

Compute Fermi momentum of protons from Fermi momentum of neutrons, in the case of a generic case of degenerate ideal Fermi gas of neutrons, protons and electrons. The toms748 algorithm is used to find the right value of k_p that matches the beta equilibrium condition for chemical potential.

Parameters

k_n : `float`
Fermi momentum of neutrons in g cm / s.

Raises

`ValueError`
If Fermi momentum of protons is negative.

Returns

`float`
Fermi momentum of protons in g cm / s.

```
def compute_number_density_emp(p)
```

Compute number density from pressure, in the case of empirical interactions.

Parameters

p : `float`
Pressure in dyne/cm^2.

Raises

`ValueError`
If number density is negative.

Returns

`float`

Number density in cm^-3.

```
def compute_number_density_gen(p)
```

Compute number density from pressure, in the case of a generic case of degenerate ideal Fermi gas of neutrons.

Parameters: - p : float Pressure in dyne/cm^2.

Returns: - float Number density in erg/cm^3.

```
def compute_number_density_nonrel(p)
```

Compute number density from pressure in the case of a non-relativistic degenerate ideal Fermi gas of neutrons.

Parameters

p : `float`

Pressure in dyne/cm^2.

Returns

`float`

Number density in cm^-3.

```
def compute_number_density_sky(p)
```

Compute number density from pressure, in the case of Skyrme Hatree-Fock interactions.

Parameters

p : `float`

Pressure in dyne/cm^2.

Raises

`ValueError`

If number density is negative.

Returns:

float Number density in cm^-3.

```
def compute_pressure_from_u_emp(u, p)
```

Compute calculated pressure minus expected pressure from the latter and u = n / n_0, in the case of empirical interactions. If it returns zero, then the pressure is exactly the desidered one. Therefore, the u given in input is the desidered value of u.

Parameters

u : `float`

Dimensionless u = n / n_0.

p : `float`

Pressure in dyne/cm^2.

Returns

`float`

Calculated pressure minus expected.

```
def compute_pressure_from_u_sky(u, p)
```

Compute calculated pressure minus expected pressure from expected pressure and u = n / n_0, in the case of Skyrme Hatree-Fock interactions. If it returns zero, then the pressure is exactly the desidered one. Therefore, the u given in input is the desidered value of u.

Parameters

u : `float`

Dimensionless u = n / n_0.

p : `float`

Pressure in dyne/cm^2.

Returns

`float`

Calculated pressure minus expected.

```
def compute_pressure_from_x_npe(x, e_0)
```

Compute pressure from x = k / (mc) and prefactor e_0, in the case of a generic case of degenerate ideal Fermi gas of neutrons, protons and electrons.

Parameters

x : `float`

Dimensionless x = k / (m c).

e_0 : `float`

Prefactor in erg/cm^3.

Returns

`float`

Pressure in dyne/cm^2.

```
def compute_pressure_gen(x, p)
```

Compute calculated pressure minus expected pressure given the latter and x = k / (m c), in the case of a generic case of degenerate ideal Fermi gas of neutrons. If it returns zero, then the pressure is exactly the desidered one. Therefore, the x given in input is the desidered value of x.

Parameters

x : `float`
: Dimensionless x = k / (m c).

p : `float`
: Pressure in dyne/cm^2.

Returns

`float`
: Calculated pressure minus expected.

```
def compute_pressure_npe(k_n)
```

In presence of protons, electrons and neutrons, compute pressure from Fermi momentum of neutrons, in the case of a generic case of degenerate ideal Fermi gas of neutrons, protons and electrons.

Parameters

k_n : `float`
: Fermi momentum of neutrons in g cm / s.

Raises

`ValueError`
: If pressure is negative.

Returns

`float`
: Pressure in dyne/cm^2.

```
def compute_pressure_pe(k_p)
```

In presence of protons and electrons (no neutrons), compute pressure from Fermi momentum of protons, in the case of a generic case of degenerate ideal Fermi gas of neutrons, protons and electrons.

Parameters

k_p : `float`
: Fermi momentum of protons in g cm / s.

Raises

`ValueError`
: If pressure is negative.

Returns

`float`
: Pressure in dyne/cm^2.

```
def compute_u_emp(p)
```

Compute u = n / n_0 from pressure, in the case of empirical interactions. The toms748 algorithm is used to find the right value of u that matches the expected pressure with the calculated one.

Parameters

`p` : `float`
: Pressure in dyne/cm^2.

Raises

`ValueError`
: If u is negative.

Returns

`float`
: Dimensionless u = n / n_0.

```
def compute_u_sky(p)
```

Compute u = n / n_0 from pressure, in the case of Skyrme Hatree-Fock interactions. The toms748 algorithm is used to find the right value of u that matches the expected pressure with the calculated one.

Parameters

`u` : `float`
: Dimensionless u = n / n_0.

Raises

`ValueError`
: If u is negative.

Returns

`float`
: Energy density in erg/cm^3.

```
def compute_x_gen(p)
```

Compute x = k / (m c) from pressure, in the case of a generic case of degenerate ideal Fermi gas of neutrons. The toms748 algorithm is used to find the right value of x that matches the expected pressure with the calculated one.

Parameters

`p` : `float`
: Pressure in dyne/cm^3.

Raises

`ValueError`
: If x is negative.

Returns

`float`
: Dimensionless x = k / (m c).

```
def equation_of_state_emp(pressures)
```

Compute and save to file the equation of state for empirical interactions. The equation of state is in the form energy density as function of pressure. Also number densities are computed and saved to file.

Parameters

`pressures` : `NumPy array`
: Numpy array of pressures in dyne/cm^2.

Raises: - ValueError If pressure is negative.

```
def equation_of_state_gen(pressures)
```

Compute and save to file the equation of state for a generic degenerate ideal Fermi gas of neutrons. The equation of state is in the form energy density as function of pressure. Also number densities are computed and saved to file.

Parameters

`pressures` : `NumPy array`
: Numpy array of pressures in dyne/cm^2.

Raises: - ValueError If pressure is negative.

```
def equation_of_state_nonrel(pressures)
```

Compute and save to file the equation of state for a non-relativistic degenerate ideal Fermi gas of neutrons. The equation of state is in the form of energy density as function of pressure. Also number densities are computed and saved to file.

Parameters

pressures : `NumPy array`
: Numpy array of pressures in dyne/cm^2.

Raises: - ValueError If pressure is negative.

```
def equation_of_state_npe(number_densities)
```

Compute and save to file the equation of state for a generic degenerate ideal Fermi gas of neutrons, protons and electrons in beta equilibrium. The equation of state is in the form energy density as function of pressure.

Parameters

number_densities : `NumPy array`
: Numpy array of number densities in cm^-3.

Raises: - ValueError If number density is negative.

```
def equation_of_state_sky(pressures)
```

Compute and save to file the equation of state for Skyrme Hartee-Fock interactions. The equation of state is in the form energy density as function of pressure. Also number densities are computed and saved to file.

Parameters

pressures : `NumPy array`
: Numpy array of pressures in dyne/cm^2.

Raises: - ValueError If pressure is negative.

```
def polytropic_equation_of_state(p, k,
gamma)
```

Compute energy density from pressure using polytropic equation of state p(e) = k e^gamma or e(p) = (p / k)^(1 / gamma). k is the prefactor and gamma is the exponent of energy density.

Parameters

p : `float`
: Pressure in dyne/cm^2.

k : `float`
: K in (dyne/cm^2)^(1 - gamma).

gamma : `float`
: Dimensionless gamma.

Returns

`float`
: Energy density in erg/cm^3.

Functions