Energy

April 19, 2022

1 Calculations for Power vs. Half-life

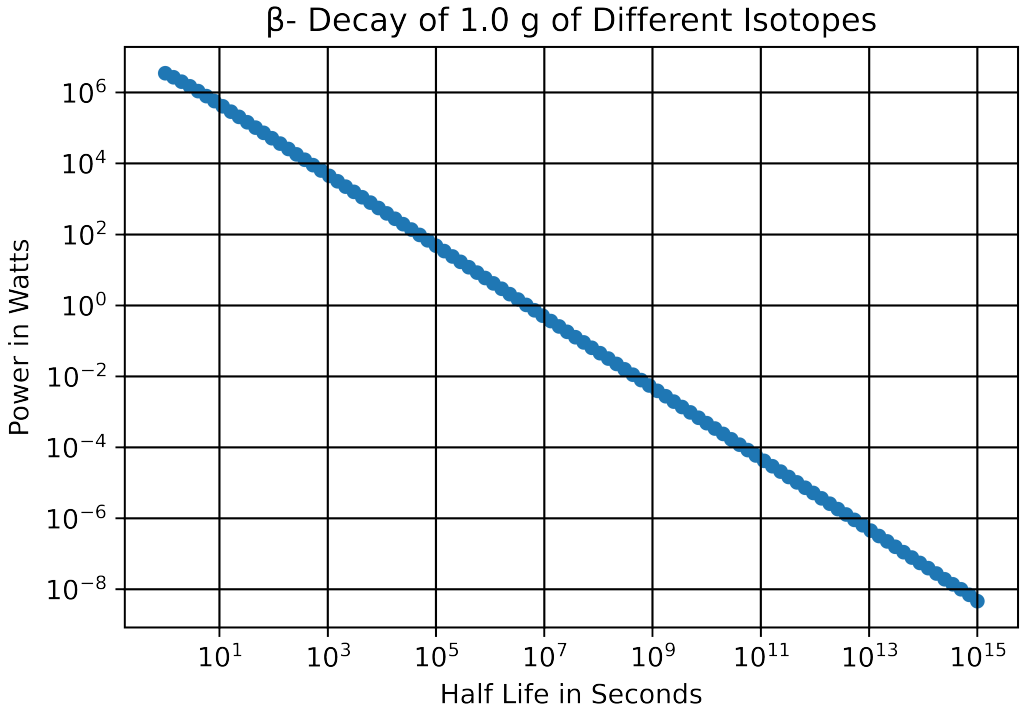
[ ]:
```
import astropy.units as units
import astropy.constants as constants
import matplotlib.pyplot as plt
import sympy as sym
import numpy as np
import pandas as pd
import plotly.express as px
a, b, c, d, e, f, g, h, i, j, k, l, m = sym.symbols('a b c d e f g h i j k l m')
n, o, p, q, r, s, t, u, v, w, x, y, z = sym.symbols('n o p q r s t u v w x y z')
symbol_list = (a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v,
w, x, y, z)
A, B, C, D, E, F, G, H, I, J, K, L, M = sym.symbols('A B C D E F G H I J K L M')
N, O, P, Q, R, S, T, U, V, W, X, Y, Z = sym.symbols('N O P Q R S T U V W X Y Z')
symbol_list = (A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
               V, W, X, Y, Z)
#%matplotlib notebook #incompatible with mpmath
def half_life_to_energy(half_life, time, initial_mass, decay_type, molar_mass):
    #we assume half life and time have consistent units (e.g. both in seconds)
    #all masses are in grams
    initial_counts = initial_mass * 6.0221408 * (10**23) / molar_mass
    decay_count = initial_counts * (1-(.5 ** (time / half_life)))
    energy = decay_energy(decay_count, decay_type) / 2 #conservative estimate
    return (energy) #joules, counts

def decay_energy(decay_counts, decay_type):
  if(decay_type == "beta_minus"):
    return decay_counts * 2.7237003 * (10 ** -15) #Joules

half_life_U_238 = 1.41*(10 ** 17) #seconds
half_life_CA_48 = 6.4 * (10 ** 26.5) #seconds
#around 10^9.5 years :)
half_life = 10 ** 11
mass = 10 ** -3 #grams
time = 1 #second
power = half_life_to_energy(half_life, time, mass, "beta_minus",
```

```
            63)
mass, power, time = mass * units.g, power * units.W, time * units.s
print(mass, "generates", power, "and loses", mass,
      "of mass over the first\nsecond :).\n Half of the energy" +
      " will be lost in", (time).to(units.year))
```

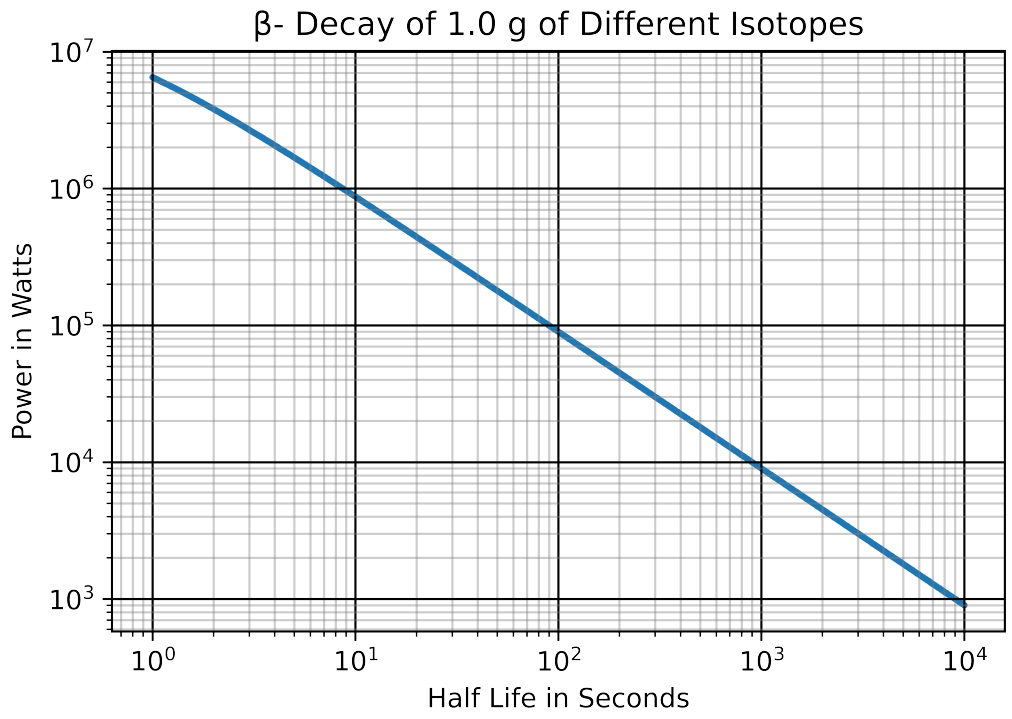
```
0.001 g generates 9.023273319236257e-08 W and loses 0.001 g of mass over the
first
second :).
 Half of the energy will be lost in 3.168808781402895e-08 yr
```

[ ]:
```
'''
Here we will plot half-life versus power generated in first second of existence.
Based on the plot this makes, since there are about 10^9.5 seconds in a century,
the ideal half life is around 10^10 seconds. This will mean after a century,
it will only half of the remaining mass (and thus presumably only produce half
of the energy indicated in this plot).

'''
def calc_half_power(min_half_life, max_half_life, steps, mass, molar_mass):
  precise_half_lives, power_array = [], []
  time = 1
  for exponent in np.linspace(min_half_life, max_half_life, steps):
        half_life = 10 ** exponent
        precise_half_lives.append(half_life)
        power = half_life_to_energy(half_life, time, mass, "beta_minus",
                                    molar_mass)
        power_array.append(power)
  return (precise_half_lives, power_array)

def plot_power_vs_half_life(min_half_life, max_half_life, steps, mass,
                            molar_mass, point_size, provide_fit, dpi):
  '''
  Note that the half-lifes give are assumed to be a power of 10
  '''
  precise_half_lives, power_array = calc_half_power(min_half_life,
                                      max_half_life, steps, mass, molar_mass)
  plt.figure(dpi = dpi)
  plt.scatter(x = precise_half_lives, y= power_array, s = point_size)
  plt.xscale("log"), plt.yscale("log")
  plt.title("- Decay of " + str(mass * units.g) + " of Different Isotopes")
  plt.ylabel("Power in Watts")
  plt.xlabel("Half Life in Seconds")
  plt.grid(which='major', color='black')
  plt.grid(which='minor', color='grey', alpha=0.4)
  plt.minorticks_on()
  logX, logY = np.log10(precise_half_lives), np.log10(power_array)
```

```
  plt.show()
  if provide_fit:
    return np.polyfit(logX, logY, 1)
plot_power_vs_half_life(0, 15, 100, 1, 118, point_size = 20,
                        provide_fit = False, dpi = 1000)
```



[ ]:
```
'''
Interestingly, zooming in we have a half-life of 10^9 seconds (30 years)
corresponds to about 1 W/g.
Conveniently, on this log-log graph, the slope is also about -1.
'''
plot_power_vs_half_life(0, 4, 500, 1, 63, 2, provide_fit = True, dpi = 10**3)
```



[ ]:
```
array([-0.98133518,  6.90205566])
```

[ ]:
```
half_life_Si_31 = (157 * units.minute).to(units.s) / units.s
power_Si_31_per_gram = half_life_to_energy(half_life_Si_31, 1, 1, "beta_minus",
                                   31) * units.W
power_Si_31_per_gram
```

[ ]: 1946.6018 W

[ ]:
```
half_life_Ni_63 = (100.1 * units.year).to(units.s) / units.s
power_Ni_63_per_gram = half_life_to_energy(half_life_Ni_63, 1, 1, "beta_minus",
                                   63) * units.W
power_Ni_63_per_gram #200kg could power a car for 50 years :)
```

[ ]: 0.0028564536 W

2 Background physics

For the nuclear reaction x + X $\rightarrow$ y + Y we define the Q as the energy from mass loss:

$$Q = [x_{mass} + X_{mass} - y_{mass} - Y_{mass}]c^2 \tag{1}$$

For β^- decay we have

$$
\begin{aligned}
n &\rightarrow p + e^{-} + \bar{v} \\
Q &= (n_{mass} - p_{mass} - e^{-}_{mass} - \bar{v}_{mass})c^2 \\
Q &= .782MeV - \bar{v}_{mass}c^2
\end{aligned}
\tag{2}
$$

Assuming a massless neutrino this simplifies to .782 MeV

[ ]:
```
#half_life_to_energy(.987 * 3.1536 * (10 ** 9), 1, 10**-6)[0]
avg_energy_per_decay = 17 * units.keV
mass_predecay = constants.m_n
mass_postdecay = constants.m_p + constants.m_e
Ni63_half_life = 1.02 * 3.1536 * (10 ** 9) * units.s
time = 1 * units.s
initial_mass = (10 ** -6) * units.kg
recalc_energy_per_decay = (mass_predecay - mass_postdecay) * (constants.c ** 2)
recalc_energy_per_decay.to(units.keV) / avg_energy_per_decay
```

[ ]:
46.019612

3 Potential Supply Chain

https://drive.google.com/file/d/1Mhe_WbmmahkeAE_JPnbYRvphh6IAWPwn/view?usp=sharing

This is very similar to how neutrons are currently produced in particle accelerators :)

https://drive.google.com/file/d/1wPwC2eu6CqND1JRK_Dgwao7nJKzyh7Xj/view?usp=sharing

https://drive.google.com/file/d/1t76jd7mjup9C_4SqJ5opnoNNZjVcNmSs/view?usp=sharing

Thermal neutrons (0.025 eV) are sufficient (neutron temperature, Bryskin et al. 2004. Figure 4.)

[ ]:
```
class ni63_setup:
  '''
  This will call helper classes depending on the different steps using in the
  production
  '''

  def __init__(self, **kwargs):
    self.assign_attributes(**kwargs)
    self.solar_panel_dict = {}

  def assign_attributes(self, **kwargs):
      for key in kwargs:
          setattr(self, key, kwargs[key])
```

[ ]:
```
class solar_panel:
  '''
```

```
  Use to create a solar_panel object with a given efficiency, area, and solar␣
↪flux
  '''
  def __init__(self, **kwargs):
    self.assign_attributes(**kwargs)

  def assign_attributes(self, **kwargs):
      for key in kwargs:
          setattr(self, key, kwargs[key])

  def calc_voltage(self):
    '''
    By definition
    power = solar_flux * efficiency * area
    power = voltage * current
    Taking the equivalent resistance of the entire circuit, we have
    voltage = current * resistance
    current = voltage / resistance
    power = voltage^2 / resistance
    voltage = sqrt(power * resistance)
    voltage = sqrt(solar_flux * efficiency * area * resistance)
    '''
    voltage =  np.sqrt(self.solar_flux * self.solar_panel_efficiency *
                       self.solar_panel_area * self.resistance)
    self.voltage = voltage.to(V)

  def calc_charge_plane(self):
  #Modify appropriately for the shape of the capacitor
    self.charge = self.capacitance * self.voltage

  def calc_gamma_ray_flux(self):
    self.gamma_ray_flux = ((self.charge**2) * (self.charge_e_distance**-4) *
                           self.brehm_coeff * self.cathode_ray_flux)

  def calc_ni63_production_speed(self):
    self.ni63_production_speed = (self.gamma_ray_flux * self.donor_cross_section
                                  * self.target_cross_section)
```

[ ]:
```
def calc_brehm_coeff():
  echarge = 1.6021766 * (10 ** -19) * C
  return ((echarge**4) / (96 * ((math.pi * c * 8.8541878128 * (10**-12) * F /␣
↪m)**3) *
                    (m_e ** 2)))

def calc_cathode_ray_flux():
  return
```

[ ]:
```
'''
https://www.thermofisher.com/order/catalog/product/1517021A

50 W is sufficient for 10^8 n/s
'''
```

[ ]:
```
'\nhttps://www.thermofisher.com/order/catalog/product/1517021A \n\n50 W is sufficient for 10^8 n/s\n'
```

4 Theoretical Maximum Production per m^2 solar panel

With a power input of P and a E_γ joules for each gamma ray, assuming each gamma ray has a ρ probability of producing ^{63}Ni we have

$$N = \frac{P\rho}{E_\gamma} \tag{3}$$

[ ]:
```
def g_per_year(solar_flux, area, efficiency, energy_per_neutron, molar_mass):
  power = solar_flux * area * efficiency
  nGamma = power / energy_per_neutron
  isotopes_per_second = nGamma.to(units.s ** -1)
  molPerS = (isotopes_per_second / (constants.N_A)).to(units.mol * units.s ** -1)
  return (molPerS * (molar_mass)).to(units.g * units.year ** -1)

solar_flux = 1000 * units.W / (units.m ** 2)
area = 1 * units.m ** 2
efficiency = 1 #any real world factors that affect the production rate
energy_per_neutron = .075 * units.MeV
molar_mass = 63 * units.g / units.mol
rate = g_per_year(solar_flux, area, efficiency, energy_per_neutron, molar_mass)
rate
```

[ ]:
274.74004 $\frac{\text{g}}{\text{yr}}$

4.1 It will take 5,000 years/m^2 to make enough for a car to work for 50 years.

We have a theoretical maximum of 10^{-6} mol nickel-63 per second for every square meter of sunlight collected per second = 36 moles per year > 2kg. Every car will need 5 years. We need 400 million m^2

[ ]:
```
earth_rad = 6400 * units.km
earth_surface_area = np.pi * 4 * (earth_rad ** 2)
panels_area = 400 * (10**6) * (units.m ** 2)
panels_area.to(units.km ** 2) / earth_surface_area
```

[ ]:
7.7712375×10^{-7}

5 We need to cover 1-millionth of the Earth in solar panels. More realistically, 400 mi^2

5.1 Caltech People who have done photoneutron work

- S.R. Golwala
- T. Aralis

5.2 What is the energy per gamma ray and probability of Ni63 production needed to make this technology competiting in the long term compared to current forms of energy storage?

The emission of radiation by accelerating charges is derived in Chapter 14 and 15 of the 2nd edition of Jackson's E&M.

[ ]:
```
'''
Source:
https://www.iea.org/reports/key-world-energy-statistics-2021/final-consumption
'''
annual_global_energy_consumed = (450 * units.EJ).to(units.W * units.year)
global_power_consumed = annual_global_energy_consumed / (1 * units.year)
efficiency = .2
power_per_area = 500 * units.W / (units.m ** 2) * efficiency
area_needed = (global_power_consumed / power_per_area).to(units.km ** 2)
area_needed
```

[ ]: 142596.4 km^2

[ ]:
```
area_California_state = 423970.694 * (units.km**2)
area_California_state / area_needed
```

[ ]: 2.9732217

6 If we cover 1/3 of California in solar panels, we could power the world.

7 We need to cover half a million square miles with solar panels and rapidly replant native flora.

[ ]:
```
#https://doi.org/10.1103/PhysRevX.7.041003
input = 40 * units.PW
Egamma= 2 * units.MeV
max_photon_flux = (input/Egamma).to(units.s ** -1)
max_photon_flux
```

[ ]: $1.2483018 \times 10^{29} \frac{1}{s}$

7.1 Rewrite clas to be based on generating the right energy neutrons for Ni63 production.

[ ]:
```
frequency = ((2 * units.MeV).to(units.J) / constants.h).to(units.Hz)
frequency
```

[ ]: 4.8359785×10^{20} Hz

8 A system that can convert between different isotopes depending on the power demand would be ideal.

- Requires a fast, compact, and energy efficient way to convert between emitted electrons and neutrons.

Possibilities include using the Brehmsstrahlung effect to create gamma rays which can then be used to generate photoneutrons. This seems like the crux of the system.

-

[ ]:
```
def total_brehmsstrahlung_power(velocity, charge, acceleration):
  '''
  Source:
  https://en.wikipedia.org/wiki/Bremsstrahlung#Total_radiated_power
  '''
  beta = velocity / constants.c
  gamma = (1 - (beta ** 2)) ** -.5
  beta_dot = (acceleration / constants.c)
  beta_term = (beta_dot ** 2) + ((beta * beta_dot) ** 2)/(1 - (beta ** 2))
  power = (charge ** 2) * (gamma ** 4) * beta_term / (6 *
                  float(sym.N(sym.pi)) * constants.eps0 * constants.c)
  return power.to(units.W)
```

[ ]:
```
total_brehmsstrahlung_power(.0000001 * constants.c, 1 * constants.e.si, .0000001␣
↪*
                            constants.c / units.s)
```

[ ]: $5.1303882 \times 10^{-51}$ W

[ ]:
```
total_energy_stored = half_life_Ni_63 * units.s * power_Ni_63_per_gram
total_energy_stored.to(units.TW * units.hour) #underestimate, and also per gram
```

[ ]: 2.5064712×10^{-9} TW h

9 Using the technology of the Andasol Solar Power Station, an area half the size of Pennsylvania could power the entire world.

[ ]:
```
#https://en.wikipedia.org/wiki/Andasol_Solar_Power_Station
andasol = 2000 * units.kW * units.hour / ((units.m ** 2 ) * units.year)
andasol_efficiency = andasol.to(units.W / (units.m ** 2))
andasol_efficiency
```

[ ]: 228.15423 $\frac{\mathrm{W}}{\mathrm{m}^2}$

[ ]:
```
needed_area = (global_power_consumed / andasol_efficiency).to(units.km ** 2)
needed_area
```

[ ]: 62500 km^2

[ ]:
```
pennsylvania_area = 119281.9 * (units.km ** 2)
needed_area / pennsylvania_area
```

[ ]: 0.52396885

[ ]:
```
neutrino_mass = (1 * units.keV / (constants.c ** 2)).to(units.g)
neutrino_mass
```

[ ]: $1.7826619 \times 10^{-30}$ g

10 Simualted Spectrum Using the Fermi Distribution

Approximated Energy Spectrum

$$N(T_e) = \frac{C}{c^5}(Q - T_e)^2(T + m_e c^2)\sqrt{T_e^2 + 2T_e m_e c^2} \tag{4}$$

Integrating this from 0 to Q and then normalizing the distribution

$$\int_0^{T_e\ max} \frac{C}{c^5}(Q - T_e)^2(T_e + m_e c^2)\sqrt{T_e^2 + 2T_e m_e c^2}\, dT_e = 1 \tag{5}$$

$$C = \frac{c^5}{\int_0^Q (Q - x)^2(x + a)\sqrt{x^2 + 2a}\, dx}$$

For some reason, WolframAlpha won't evaluate the integral. So using sympy, where Q = .782 MeV, a = $m_e c^2$, and $x = T_e$ All units are in kg-m-s SI.

[ ]:
```
unnormalized_fermi_fun = ((Q - x) ** 2) * (x + a) * sym.sqrt(
    (x ** 2) + (2 * a))
denominator = sym.integrate(unnormalized_fermi_fun, (x, 0, Q))
```

[ ]:
```
Q_fermi_distr = float((.782 * units.MeV).to(units.J) / units.J)
Q_fermi_distr
```

[ ]:
```
1.252902127788e-13
```

[ ]:
```
a_fermi_distr = float((constants.m_e * (constants.c ** 2)).to(units.J)
/ units.J)
a_fermi_distr
```

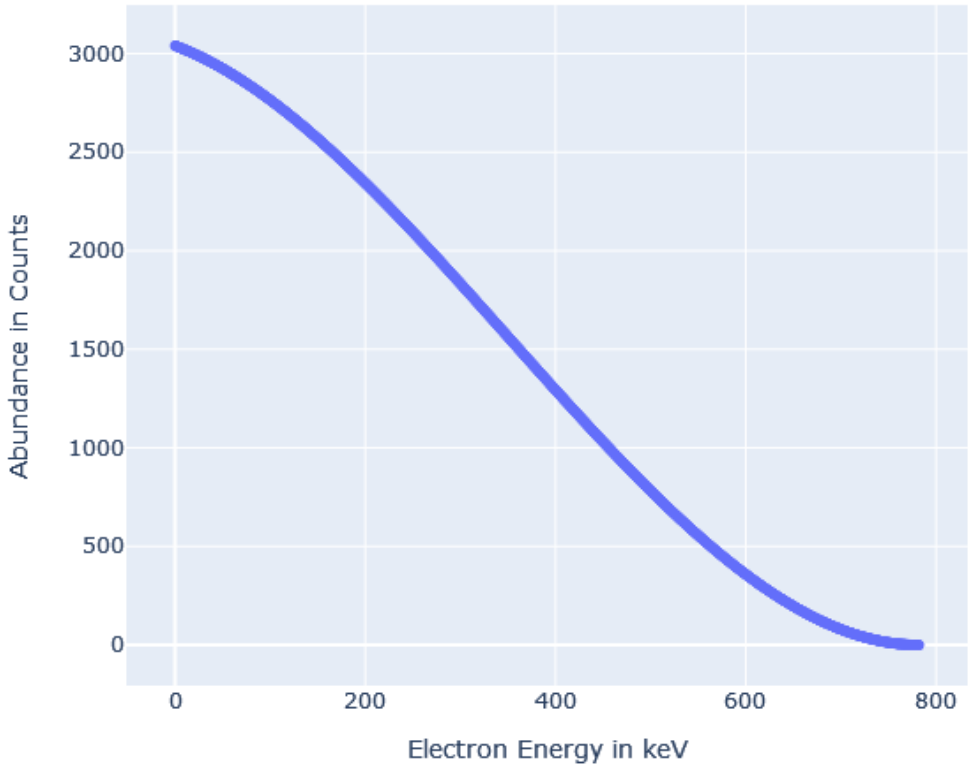
[ ]:
```
8.187105776823886e-14
```

[ ]:
```
c_to_the_fifth = float((constants.c * units.s / units.m) ** 5)
c_to_the_fifth
```

[ ]:
```
2.4216061708512208e+42
```

[ ]:
```
integrated_dist = sym.lambdify((Q, a), denominator)
fermi_distr_const = c_to_the_fifth / integrated_dist(Q_fermi_distr,
                                                     a_fermi_distr)
```

[79]:
```
fermi_distr_fun = C * unnormalized_fermi_fun / c_to_the_fifth
fermi_distr = sym.lambdify((Q, a, C, x), fermi_distr_fun)

electron_energies = []
abundance = []
log_res = 3
for electron_energy in np.linspace(0, Q_fermi_distr, 10 ** log_res):
  abundance.append(fermi_distr(Q_fermi_distr, a_fermi_distr,
                               fermi_distr_const, electron_energy))
  electron_energy_keV = float((electron_energy * units.J).to(units.keV)
        / units.keV)
  electron_energies.append(electron_energy_keV)
xlabel, ylabel = 'Electron Energy in keV', 'Abundance in Counts'
data = {xlabel: electron_energies, ylabel: abundance}
Ni63_spectrum = pd.DataFrame.from_dict(data)
fig = px.scatter(Ni63_spectrum, x = xlabel, y = ylabel)
fig.show()
```



[94]:
```
fermi_distr_fun
```

[94]:
$$4.12949063327043 \cdot 10^{-43} C \left(Q - x\right)^{2} \left(a + x\right) \sqrt{2a + x^{2}}$$

[92]:
```
average_fun = (sym.integrate(fermi_distr_fun, (x, 0, fermi_distr_fun)) /
           fermi_distr_fun)
average = sym.lambdify((Q, a, C, x), average_fun)
average(Q_fermi_distr, a_fermi_distr, fermi_distr_const, Q_fermi_distr)
```

```
---------------------------------------------------------------------------

SyntaxError                               Traceback (most recent call last)

/usr/local/lib/python3.7/dist-packages/sympy/core/sympify.py in sympify(a,␣
↪locals, convert_xor, strict, rational, evaluate)
    479         a = a.replace('\n', '')
--> 480         expr = parse_expr(a, local_dict=locals,␣
↪transformations=transformations, evaluate=evaluate)
```

```
      481     except (TokenError, SyntaxError) as exc:


      /usr/local/lib/python3.7/dist-packages/sympy/parsing/sympy_parser.py in␣
↪parse_expr(s, local_dict, transformations, global_dict, evaluate)
     1007
  -> 1008     return eval_expr(code, local_dict, global_dict)
     1009


      /usr/local/lib/python3.7/dist-packages/sympy/parsing/sympy_parser.py in␣
↪eval_expr(code, local_dict, global_dict)
      902     expr = eval(
  --> 903         code, global_dict, local_dict)  # take local objects in␣
↪preference
      904


      SyntaxError: invalid syntax (<string>, line 1)


  During handling of the above exception, another exception occurred:


      SympifyError                              Traceback (most recent call last)

      <ipython-input-92-15fd32dc4f1e> in <module>()
  ----> 1 average_fun = (sym.integrate(str(fermi_distr), (x, 0,␣
↪fermi_distr_fun)) /
        2            fermi_distr_fun)
        3 average = sym.lambdify((Q, a, C, x), average_fun)
        4 #average(Q_fermi_distr, a_fermi_distr, fermi_distr_const,␣
↪Q_fermi_distr)
        5 average


      /usr/local/lib/python3.7/dist-packages/sympy/integrals/integrals.py in␣
↪integrate(meijerg, conds, risch, heurisch, manual, *args, **kwargs)
     1539         'manual': manual
     1540         }
  -> 1541     integral = Integral(*args, **kwargs)
     1542
     1543     if isinstance(integral, Integral):


      /usr/local/lib/python3.7/dist-packages/sympy/integrals/integrals.py in␣
↪__new__(cls, function, *symbols, **assumptions)
```

```
        85                 useinstead="the as_expr or integrate methods of Poly").
↪warn()
        86
   ---> 87         obj = AddWithLimits.__new__(cls, function, *symbols,␣
↪**assumptions)
        88         return obj
        89


      /usr/local/lib/python3.7/dist-packages/sympy/concrete/expr_with_limits.py␣
↪in __new__(cls, function, *symbols, **assumptions)
       492
       493     def __new__(cls, function, *symbols, **assumptions):
   --> 494         pre = _common_new(cls, function, *symbols, **assumptions)
       495         if type(pre) is tuple:
       496             function, limits, orientation = pre


      /usr/local/lib/python3.7/dist-packages/sympy/concrete/expr_with_limits.py␣
↪in _common_new(cls, function, *symbols, **assumptions)
        23     (function, limits, orientation). This code is common to
        24     both ExprWithLimits and AddWithLimits."""
   ---> 25     function = sympify(function)
        26
        27     if isinstance(function, Equality):


      /usr/local/lib/python3.7/dist-packages/sympy/core/sympify.py in sympify(a,␣
↪locals, convert_xor, strict, rational, evaluate)
       480         expr = parse_expr(a, local_dict=locals,␣
↪transformations=transformations, evaluate=evaluate)
       481     except (TokenError, SyntaxError) as exc:
   --> 482         raise SympifyError('could not parse %r' % a, exc)
       483
       484     return expr


      SympifyError: Sympify of expression 'could not parse '<function␣
↪_lambdifygenerated at 0x7ff9f6ee84d0>'' failed, because of exception being␣
↪raised:
   SyntaxError: invalid syntax (<string>, line 1)
```

[ ]:
```
def normalized_counts(T_e, Q):
  count = (np.sqrt(((T_e **  2) + 2 * (T_e * m_e * (c ** 2))))
           * ((Q - T_e) ** 2) * (T_e + m_e * (c ** 2)) / (c ** 5))
  return count
```

[88]:

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-88-206dee9302f3> in <module>()
----> 1 average.variables


AttributeError: 'function' object has no attribute 'variables'
```

[ ]:

```
'''
We need to account for the fact that not all of the electrons will have the
maximum energy (Q value)!
stuck for right now. Use
https://github.com/MarcosP7635/Computing-and-Formatting/blob/main/
 ↪error_propagation.py
as a sympy reference
They average out to 17 keV. Rewrite to draw from a database of beta emission
spectra.
'''
```

[95]:

```
energy_Ni_63_per_gram = (power_Ni_63_per_gram * 50 * units.year).to(units.J)
'''
The best lithium-ion batteries store less than 1 kJ/g
Source: https://doi.org/10.1039/D0EE02681F
'''
energy_Ni_63_per_gram
```

[95]:

4507141 J

[ ]:

```
'''
if google drive won't mount to the colab session, then
you need to download this current python notebook and upload
it to the colab session, then right click on it to copy the
path for the command below.
'''
!jupyter nbconvert --to LaTeX /Energy.ipynb
#The above line makes a .tex file to format this Jupyter Notebook
```

[ ]: