Contents

1 Note on this document

In order to run this code you obviously need Nim in your `PATH` as well as all packages, which are imported. In addition you need ntangle in your path. With those in place, run:

```
ntangle axionMass.org
nim c -r axionMass.nim
```

and the code is extracted and run, regenerating all plots.

2 Calculation of axion mass range of a helioscope

For BabyIAXO it would be nice to know:

1. the upper axion mass range for usage with vacuum
2. the resonant axion mass for a specific helium density

Starting point for this discussion is the IAXO gas phase study by Biljana and Kresimir.
The coherence condition for axions is

$$qL < \pi \tag{1}$$

$$\text{where } q = \frac{m_a^2}{2E_a} \tag{2}$$

with L the length of the magnetic field (20m for IAXO, 10m for BabyIAXO), m_a the axion mass and E_a the axion energy (taken from solar axion spectrum).

In the presence of a low pressure gas, the photon receives an effective mass m_γ, resulting in a new q:

$$q = \left| \frac{m_\gamma^2 - m_a^2}{2E_a} \right| \tag{3}$$

Thus, we first need some values for the effective photon mass in a low pressure gas, preferably helium.

From this we can see that coherence in the gas is restored if $m_\gamma = m_a$, $q \to 0$ for $m_a \to m_\gamma$. This means that in those cases the energy of the incoming axion is irrelevant for the sensitivity!

In order to calculate some values, we'll write some Nim code to calculate and plot the dependence of m_a on the gas pressure.

First we have to import some modules we'll need:

```
import sequtils, seqmath, ggplotnim, strformat, algorithm, nlopt, options, strutils
```

An approximation for the dependence of m_γ on the surrounding gas, was found in: `https://www.sciencedirect.com/science/article/pii/S0370269302018221` and can be written as:

$$m_\gamma = \sqrt{\frac{4\pi\alpha n_e}{m_e}} \tag{4}$$

where α is the fine structure constant, m_e the electronm mass and n_e the electron number density. However, this equation still has to be fixed regarding the units. The units as such are:

$$\text{eV} = \sqrt{\frac{1}{\text{eVm}^3}} \tag{5}$$

We can fix this, by replacing the `m^3` by their natural unit equivalents in eV: `1 eV^-1 == 1.97e-7 m` That is, replace the `1 / m^3` by `1.97e-7 eV^3`, for a final equation:

$$m_\gamma = \sqrt{\frac{4\pi\alpha n_e \cdot (1.97e-7)^3}{m_e}} \tag{6}$$

A proc to calculate an effective mass from an electron number density is thus:

```
proc effPhotonMass(ne: float): float =
  ## returns the effective photon mass for a given electron number density
  const alpha = 1.0 / 137.0
  const me = 511e3 # 511 keV
  # note the 1.97e-7 cubed to account for the length scale in `ne`
  result = sqrt( pow(1.97e-7, 3) * 4 * PI * alpha * ne / me )
```

This means we need to calculate the electron number density n_e for a given gas. For practical reasons it's probably easier to calculate it from a molar density in mol / m^3 for a gas (see here):

$$n_e = Z \cdot N_A \cdot c \tag{7}$$

where N_A is the Avogadro constant ($N_A = 6.022e23\,\text{mol}^{-1}$) and c the molar density of the gas in `mol / m^3` and Z appears since we have Z electrons per molecule in the gas.

```
proc numDensity(c: float): float =
  ## converts a molar concentration in mol / m^3 to a number density
  const Z = 2 # number of electron in helium atom
  result = Z * 6.022e23 * c
```

The standard atomic weight of helium is $A_r = 4.002602$, resulting in a molar mass of $M(\text{He}) = 4.002\,602\,\text{g}\,\text{mol}^{-1}$.

Given the ideal gas law:

$$PV = nRT \tag{8}$$

with the pressure P, the gas volume V, the amount of gas in mol n, the universal gas constant R: $R = 8.314\,\text{J}\,\text{K}^{-1}\,\text{mol}^{-1}$, and the temperature T.

So to calculate the molar gas amount from a pressure, volume and temperature:

```
proc molarAmount(p, vol, temp: float): float =
  ## calculates the molar amount of gas given a certain pressure,
  ## volume and temperature
  ## the pressure is assumed in mbar
  const gasConstant = 8.314 # joule K^-1 mol^-1
  let pressure = p * 1e2 # pressure in Pa
  result = pressure * vol / (gasConstant * temp)
  #echo "Molar amount for P = ", pressure, " Pa is ", result
```

We know the volume and temperature and want to calculate the dependency of m_a on different pressure values. Thus, we can calculate n for each pressure and insert it into the number density proc to calculate the effective photon mass.

As mentioned above, coherence in the gas is restored if $m_\gamma = m_a$. So all we have to do is to calculate m_γ for different pressures and then we know the effective axion mass we're sensitive to in BabyIAXO given a certain helium pressure.

So calculate the effective photon mass for a given pressure in BabyIAXO:

```
proc babyIaxoEffMass(p: float): float =
  ## calculates the effective photon (and thus axion mass) for BabyIAXO given
  ## a certain helium pressure in the BabyIAXO magnet
  const vol = 10.0 * (PI * pow(0.3, 2)) # 10m length, bore radius 30 cm
  # UPDATE: IAXO will be run at 4.2 K instead of 1.7 K
  # const temp = 1.7 # assume 1.7 K same as CAST
  const temp = 4.2
  once:
    echo "BabyIAXO magnet volume is ", vol, " m^3"
    echo "BabyIAXO magnet temperature is ", temp, " K"
  let amountMol = molarAmount(p, vol, temp) # amount of gas in mol
  let numPerMol = numDensity(amountMol / vol) # number of electrons per m^3
  #echo "Num electrons per m^3 ", numPerMol
  result = effPhotonMass(numPerMol)
```

All that is left to do then is to create a set of pressure values that we want to plot against and calculate the sensitive axion mass for those values for BabyIAXO. Since we probably want to cover a log space, create a helper proc `logspace` from `linspace`:

```
proc logspace(start, stop: float, num: int, base = 10.0): seq[float] =
  ## generates evenly spaced points between start and stop in log space
  let linear = linspace(start, stop, num)
  result = linear.mapIt(pow(base, it))
```

So generate values and create a plot. Let's write a proc to generate a plot given some mbar range and a plot filename:

```
proc makePlot(pstart, pstop: float, fname: string, log = false) =
  let pressures = logspace(pstart, pstop, 1000) # 1000 values from 1e-5 mbar to 500 mbar
  let masses = pressures.mapIt(babyIaxoEffMass(it)) # corresponding masses
  # convert both seqs to a dataframe
  let df = seqsToDf({"P / mbar" : pressures, "m_a / eV" : masses})
  let plt = ggplot(df, aes("P / mbar", "m_a / eV")) +
    geom_line() +
    ggtitle("Sensitive axion mass in eV depending on helium pressure in mbar")
  if not log:
    plt + ggsave(fname)
  else:
    plt + scale_x_log10() + scale_y_log10() + ggsave(fname)
```

First for a logspace from 1e-6 to 1e2 mbar:

```
makePlot(-6.0, 2.0, "axionMassesFullRange.pdf")
```

and now for a low pressure range:

```
makePlot(-6.0, -2.0, "axionMassesZoomed.pdf")
```

and finally as a log-log plot in the full range.

```
1 makePlot(-6.0, 2.0, "axionMassesLogLog.pdf", log = true)
```

These plots are shown in fig. 1 and 2 and the log-log plot shown in 3.

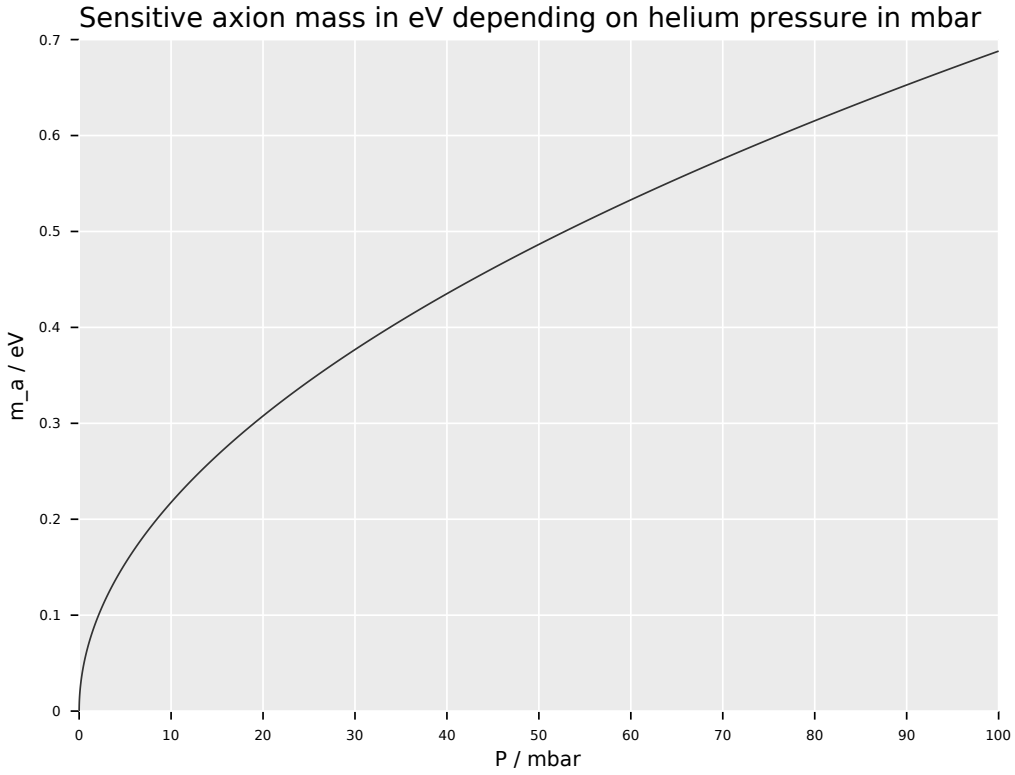

Figure 1: Axion masses in eV depending on the pressure in BabyIAXO in mbar in the full range from 10×10^{-6} mbar to 10×10^{2} mbar.

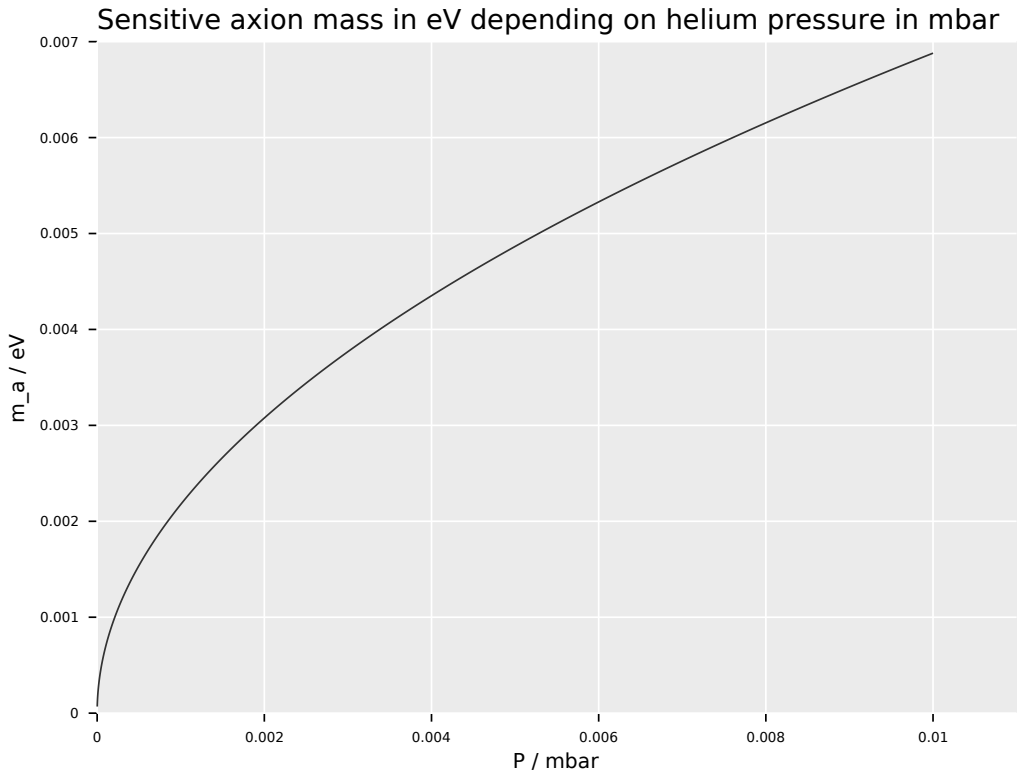

Figure 2: Axion masses in eV depending on the pressure in BabyIAXO in mbar in a zoomed range from 10×10^{-6} mbar to 10×10^{-2} mbar.

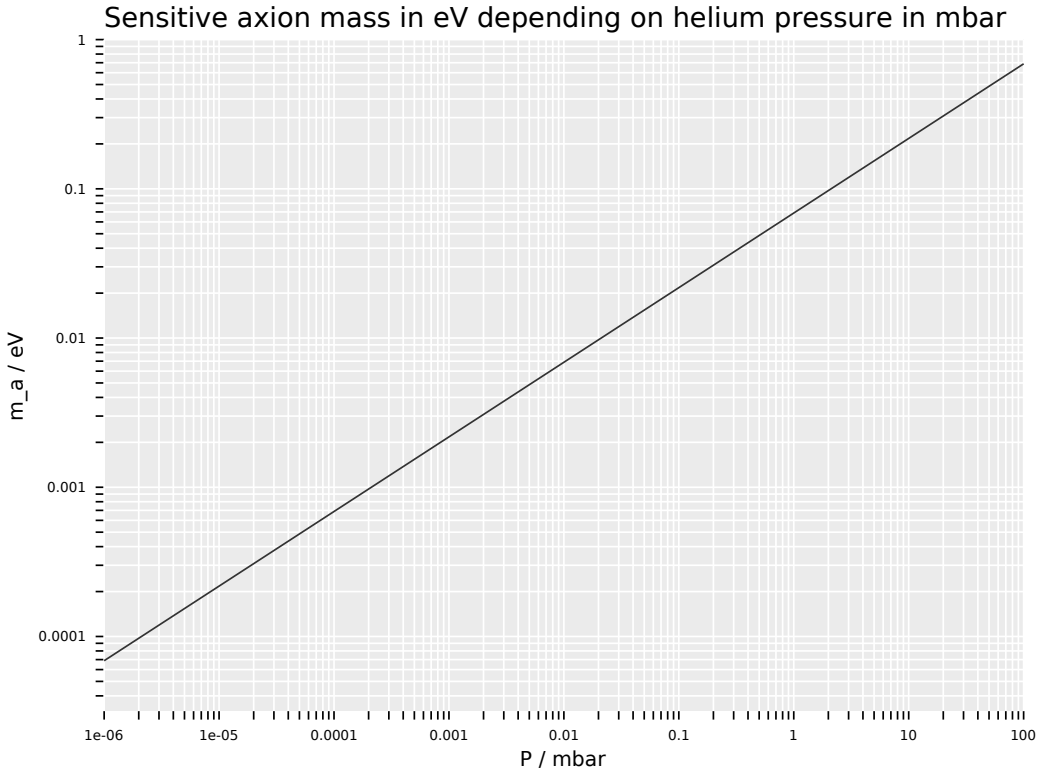

Figure 3: Axion masses in eV depending on the pressure in BabyIAXO in mbar in the full range from 10×10^{-6} mbar to 10×10^{2} mbar as a log-log plot.

2.1 Analytical formula for effective photon mass

UPDATE: updated to fix issues from 2.4.

We can easily derive an analytical expression for the above calculation, by simply inserting all equations into one another.

$$m_\gamma = \sqrt{\frac{(1.97 \times 10^{-7})^3 4\pi\alpha Z N_a 100 \cdot P(\text{mbar})}{m_e R T}} \tag{9}$$

if the pressure is given in mbar (factor 100 accounts for that).

which after inserting all numbers and multiplying them gives the following expression with:

- `Z = 2`
- `T = 4.2 K`
- `R = 8.314 J K^-1 mol^-1`
- `N_A = 6.022e23 mol^-1`
- `\alpha = 1 / 137`
- `m_e = 511,000 eV`

which results in:

$$m_\gamma = 1.940\,81 \times 10^{-2} \cdot \sqrt{4\pi P(\text{mbar})} \tag{10}$$

Let's define this as a function and check that the resulting values are actually the same as for the above calculation:

```
proc analyticalCalc(p: float): float =
  ## calculates the effective mass for a given pressure from a single equation
  ## derived from inserting all above into one
  result = 1.94081e-2 * sqrt(4 * PI * p )

block:
  let pressures = logspace(-6.0, 2.0, 1000)
  let massesNum = pressures.mapIt(babyIaxoEffMass(it))
  let massesAna = pressures.mapIt(analyticalCalc(it))
  #echo massesNum
  #echo massesAna
  func almostEqual(a, b: seq[float]): bool = zip(a, b).mapIt(abs(it[0] - it[1]) < 1e-5).allIt(it)
  doAssert massesNum.almostEqual(massesAna)
```

Which is indeed the case (as the assertion holds during runtime).

2.2 Upper mass range for vacuum

Finally also calculate the mass range one is sensitive to, if the magnet is not filled with helium.

In those cases, the limit is just given by the coherence condition mentioned at the top of the file $qL < \pi$. Using the boundary as the condition to find m_a we have:

$$qL = \pi \tag{11}$$

$$\frac{m_a^2}{2E_a} L = \pi \tag{12}$$

$$m_a = \sqrt{\frac{\pi 2 E_a}{L}} \tag{13}$$

where we again have to add a factor of `1.97e-7 m` for the length scale L:

$$m_a = \sqrt{\frac{\pi 2 E_a \cdot 1.97 \times 10^{-7}}{L}}$$

Note that in this case the effective mass one is sensitive to is directly dependent on the axion's energy, in contrast to the resonant case in the presence of an effective photon mass.

So we can calculate the mass limit in dependence of the axion's energy as:

```
proc vacuumMassLimit(energy: float, magnetLength = 10.0): float =
  ## given an axion energy in keV, calculate the limit of coherence
  ## for the vacuum case in BabyIAXO
  # note the length scale 1.97e-7 to take into account the meter scale in
  # babyIaxoLen
  let babyIaxoLen = magnetLength / 1.97e-7 # length in "eV"
  result = sqrt(PI * 2 * energy * 1e3 / babyIaxoLen) # convert energy to eV
```

So let's also create a plot covering the interesting energy range from 0 to 10 keV:

```
let energies = linspace(0.0, 10.0 , 1000) # from 0 to 10 keV
let massLimits = energies.mapIt(vacuumMassLimit(it))
let df = seqsToDf({"E / keV" : energies, "m_a / eV" : massLimits})
ggplot(df, aes("E / keV", "m_a / eV")) +
  geom_line() +
  ggtitle("Sensitive axion mass limit in eV for BabyIAXO in vacuum run") +
  ggsave("vacuumMassLimit.pdf")
```

Now cross check the value for the vacuum mass limit with the value calculated by Biljana and Kreso for IAXO, which was `m_a < 1.62e-2 eV` for IAXO (length 20m) at an energy of `4.2 keV`:

```
let iaxoLimit = vacuumMassLimit(4.2, 20.0)
func almostEqual(a, b: float, eps = 1e-3): bool = abs(a - b) < eps
doAssert iaxoLimit.almostEqual(1.61e-2) # value from IAXO gas phase study
echo "Full IAXO mass limit @ 4.2 keV = ", iaxoLimit, " eV"
```

And we see that this is indeed the case!

Compared with that the value of the vacuum limit at `4.2 keV` for BabyIAXO is:

```
const babyIaxoVacuumMassLimit = vacuumMassLimit(4.2)
echo "BabyIAXO mass limit @ 4.2 keV = ", babyIaxoVacuumMassLimit, " eV"
```

2.3 Calculating axion conversion probability

Now we will try to calculate the axion conversion probability in Helium at a given gas density and temperature.

The axion-photon conversion probability $P_{a\to\gamma}$ in general is given by:

$$P_{a\to\gamma} = \left(\frac{g_{a\gamma}B}{2}\right)^2 \frac{1}{q^2 + \Gamma^2/4}\left[1 + e^{-\Gamma L} - 2e^{-\frac{\Gamma L}{2}}\cos(qL)\right], \tag{14}$$

where Γ is the inverse absorption length for photons (or attenuation length).

Aside from Γ we have all values at hand. At the end we want to have $P_{a\to\gamma}$ as a function of $P_{a\to}\gamma(\Gamma, m_a)$, i.e. we pick a specific B and $g_{a\gamma}$, set L and wish to evaluate the conversion probabiltiy based on Γ (depends on the gas pressure) and the axion mass. That way we can calculate the conversion probability FWHM at a given pressure.

Following Theodoro's PhD thesis (see 3 below), the attenuation length can be calculated from:

$$\Gamma = \rho\left(\frac{\mu}{\rho}\right), \tag{15}$$

where μ/ρ is the so called mass attenuation coefficient. The values for the mass attenuation coefficient are tabulated, e.g. by NIST: `https://physics.nist.gov/PhysRefData/XrayMassCoef/tab3.html` Since

Energy MeV	μ/ρ $\mathrm{cm^2\,g^{-1}}$	μ_{en}/ρ $\mathrm{cm^2\,g^{-1}}$
1.00000E-03	6.084E+01	6.045E+01
1.50000E-03	1.676E+01	1.638E+01
2.00000E-03	6.863E+00	6.503E+00
3.00000E-03	2.007E+00	1.681E+00
4.00000E-03	9.329E-01	6.379E-01
5.00000E-03	5.766E-01	3.061E-01
6.00000E-03	4.195E-01	1.671E-01
8.00000E-03	2.933E-01	6.446E-02
1.00000E-02	2.476E-01	3.260E-02
1.50000E-02	2.092E-01	1.246E-02
2.00000E-02	1.960E-01	9.410E-03
3.00000E-02	1.838E-01	1.003E-02
4.00000E-02	1.763E-01	1.190E-02
5.00000E-02	1.703E-01	1.375E-02
6.00000E-02	1.651E-01	1.544E-02
8.00000E-02	1.562E-01	1.826E-02
1.00000E-01	1.486E-01	2.047E-02
1.50000E-01	1.336E-01	2.424E-02
2.00000E-01	1.224E-01	2.647E-02
3.00000E-01	1.064E-01	2.868E-02
4.00000E-01	9.535E-02	2.951E-02
5.00000E-01	8.707E-02	2.971E-02
6.00000E-01	8.054E-02	2.959E-02
8.00000E-01	7.076E-02	2.890E-02
1.00000E+00	6.362E-02	2.797E-02
1.25000E+00	5.688E-02	2.674E-02
1.50000E+00	5.173E-02	2.555E-02
2.00000E+00	4.422E-02	2.343E-02
3.00000E+00	3.503E-02	2.019E-02
4.00000E+00	2.949E-02	1.790E-02
5.00000E+00	2.577E-02	1.622E-02
6.00000E+00	2.307E-02	1.493E-02
8.00000E+00	1.940E-02	1.308E-02
1.00000E+01	1.703E-02	1.183E-02
1.50000E+01	1.363E-02	9.948E-03
2.00000E+01	1.183E-02	8.914E-03

Table 1: Table from NIST of the mass attenuation length of photons in helium. The right most column is the mass-energy absorption coefficient, which we do not care about here.

we mainly care about ^{4}He in the context of this document, we can take the helium entry straight from `https://physics.nist.gov/PhysRefData/XrayMassCoef/ElemTab/z02.html`, which conveniently lists these values in useful energy ranges, see tab. 1.

In order to evaluate the mass attenuation at any energy, we need an interpolation function. Conveniently Javier Galan (PhD thesis) fitted such a function for us already (taken from Theodoros' thesis),

$$\log\left(\frac{\mu}{\rho}\right)(E) = 1.5832 + 5.9195e^{-0.353808E} + 4.03598e^{-0.970557E} \tag{16}$$

We will now define this function in our Nim program and generate the datapoints for the interpolation in our energy range we consider above:

```
proc logMassAttenuation(e: float): float =
  ## calculates the logarithm of the mass attenuation coefficient for a given
  ## energy `e` in `keV` and the result in `cm^2/g`
  result = -1.5832 + 5.9195 * exp(-0.353808 * e) + 4.03598 * exp(-0.970557 * e)

let logMuOverRho = energies.mapIt(logMassAttenuation(it))
# now the non-log values
let muOverRho = logMuOverRho.mapIt(exp(it))
```

Before we can plot this together with the tabulated data, we have to parse the table above. The raw data is also stored in `mass_attenuation_nist_data.txt`. We will now parse it into a dataframe and then drop all values outside the range of the energies we consider:

```
const massAttenuationFile = "mass_attenuation_nist_data.txt"
# skip one line after header, second header line
var dfMuRhoTab = toDf(readCsv(massAttenuationFile, skipLines = 1, sep = ' ', header = "#")
  # convert MeV energy to keV
  .mutate(f{"Energy" ~ `Energy` * 1000.0})
  .filter(f{float: `Energy` >= energies.min and `Energy` <= energies.max})
```

which leaves us with the following table of values tab. 2:

Idx	Energy / keV	mu/rho	mu_{en}/rho
0	1	60.84	60.45
1	1.5	16.76	16.38
2	2	6.863	6.503
3	3	2.007	1.681
4	4	0.9329	0.6379
5	5	0.5766	0.3061
6	6	0.4195	0.1671
7	8	0.2933	0.06446
8	10	0.2476	0.0326

Table 2: All mass attenuation coefficients (μ/ρ) in the range of our energies considered 0 keV to 10 keV

Now we can finally compare the interpolation function with the real coefficients and see if they actually match:

```
# create df of interpolated values
let dfMuRhoInterp = seqsToDf({ "E / keV" : energies,
                               "mu/rho" : muOverRho,
                               "log(mu/rho)" : logMuOverRho})
# rename the columns of the tabulated values df and create a log column
dfMuRhoTab = dfMuRhoTab.rename(f{"E / keV" <- "Energy"})
  .mutate(f{float: "log(mu/rho)" ~ ln(c"mu/rho")})
# build combined DF
```

```
let dfMuRho = bind_rows([("Interpolation", dfMuRhoInterp),
                         ("NIST", dfMuRhoTab)],
                        id = "type")
echo dfMuRho
# create plot of log values
ggplot(dfMuRho, aes("E / keV", "log(mu/rho)", color = "type")) +
  geom_line(data = dfMuRho.filter(f{`type` == "Interpolation"})) +
  geom_point(data = dfMuRho.filter(f{`type` == "NIST"})) +
  ggtitle("Mass attenuation coefficient interpolation and data") +
  ggsave("log_mass_attenuation_function.pdf")

# and the plot of the raw mu/rho values
ggplot(dfMuRho, aes("E / keV", "mu/rho", color = "type")) +
  geom_line(data = dfMuRho.filter(f{`type` == "Interpolation"})) +
  geom_point(data = dfMuRho.filter(f{`type` == "NIST"})) +
  scale_y_log10() +
  ggtitle("Mass attenuation coefficient interpolation and data") +
  ggsave("mass_attenuation_function.pdf")
```

The figure of the mass attenuation function (non-log data) is shown in fig. 4.

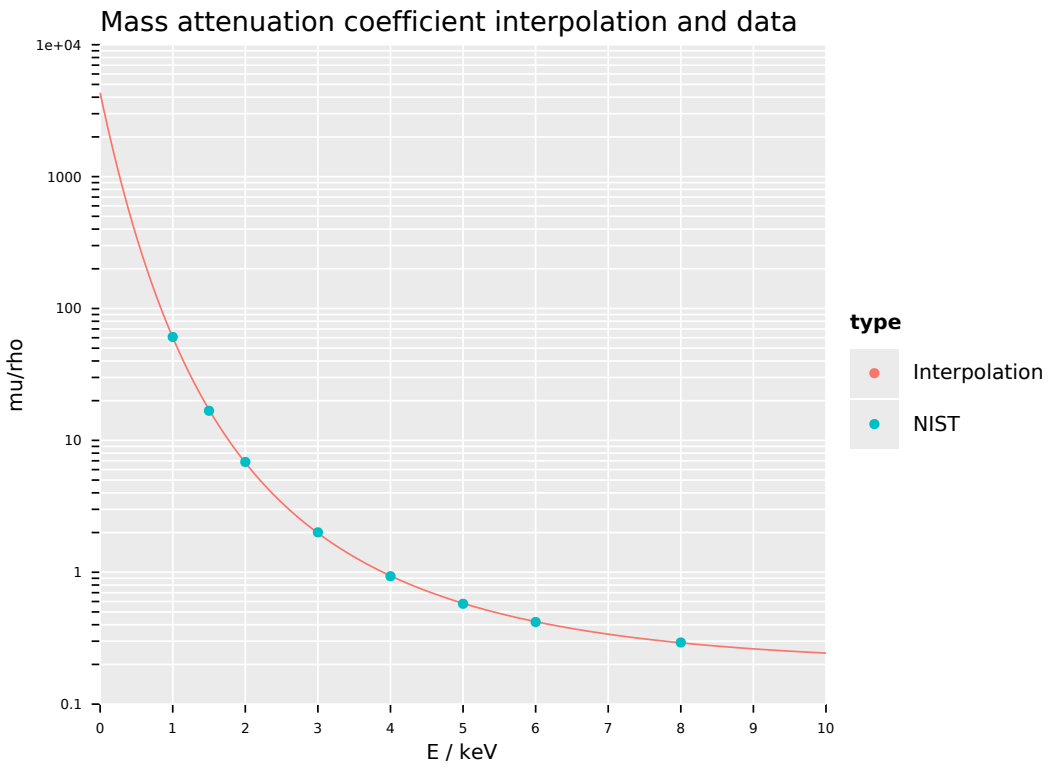

Figure 4: Mass attenuation coefficient comparison of interpolation function with the tabulated data from NIST for ^{4}He.

Now all that remains is to put the inerpolation function to use, as it apparently correctly describes the NIST data. The only sanity check left before we do that is to check whether the effective mass calculation of Theodoros' thesis matches with the function we obtained.

Our simplified formula ended up being:

$$m_\gamma = 1.372\,36 \times 10^{-2} \cdot \sqrt{4\pi P} \tag{17}$$

while according to equation 7.3:

$$m_\gamma = 28.77\sqrt{\frac{Z}{A}\rho} \tag{18}$$

where for ^{4}He $Z = 2$ and $A \sim 4\,\mathrm{g\,mol^{-1}}$.

The ideal gas law in molar form is:

$$P = \rho\frac{R}{M}T, \tag{19}$$

where M is the molar mass of our gas. Since we assume it's pure ^{4}He, this is $M = 4\,\mathrm{g\,mol^{-1}}$. However, if we simply replace the pressure in our equation by this equation, the result will be still dependent both on T, which is because equation 17 already includes our temperature of $T = 4.2\,\mathrm{K}$ (as well as R as a matter of fact). Let's fix that:

$$m_\gamma = 1.372\,36 \times 10^{-2} \cdot 4.2 \cdot 8.314\sqrt{\frac{4\pi P}{RT}} \tag{20}$$

$$m_\gamma = 0.479\,21\sqrt{\frac{4\pi P}{RT}}. \tag{21}$$

Now we can finally replace P:

$$m_\gamma = 0.479\,21\sqrt{\frac{4\pi\rho RT}{MRT}} \tag{22}$$

$$m_\gamma = 0.479\,21\sqrt{\frac{4\pi\rho}{M}} \tag{23}$$

and inserting the molar mass of helium $M(\mathrm{He}) = 4.002\,602\,\mathrm{g\,mol^{-1}}$ yields:

$$m_\gamma = 0.8491\sqrt{\rho} \tag{24}$$

and comparing that to eq. 18 by inserting Z and A:

$$m_\gamma = 28.77\sqrt{\frac{2}{4.002602}\rho} \tag{25}$$

$$m_\gamma = 20.336\,85\sqrt{\rho} \tag{26}$$

which, *ehm* does not match.

Why?

Will figure out the error later. For now continue on with the actual calculation of the conversion probabiltiy.

If we take one of our effective mass calculations for granted for the moment, we can calculate the probability as we defined it further above, eq. 14.

```
proc momentumTransfer(m_gamma, m_a: float, E_a = 4.2): float =
  ## calculates the momentum transfer for a given effective photon
  ## mass `m_gamma` and axion mass `m_a` at an axion energy of
  ## 4.2 keV `E_a` (by default).
  #const c = 299792458
  result = abs((m_gamma * m_gamma - m_a * m_a) / (2 * E_a * 1000.0))

proc axionConversionProb(m_a, m_gamma, gamma: float, length = 10.0): float =
  ## given an axion mass and an inverse absorption length
  ## calculates the conversion probabilty with BabyIAXOs magnet
  ## properties. Axion coupling constant taken to be `1` +1e-11+.
  ## By default uses BabyIAXO length of `10m`
  # both `g_agamma` and `B` only scale the absolute value `P`, does not matter
  const g_agamma = 1.0 #1e-11
  const B = 4.5 # T, actually don't know the real number right now
  # convert length in `m` to `eV`
  let L = length / 1.97e-7 # m
```

```
  let q = momentumTransfer(m_gamma, m_a)
  let term1 = pow(g_agamma * B / 2.0, 2)
  let term2 = 1.0 / (q * q + gamma * gamma / 4)
  let term3 = 1.0 + exp(-gamma * L) - 2 * exp(-gamma * L / 2) * cos(q * L)
  result = term1 * term2 * term3
```

Now all that is left to do is think up some reasonable numbers for the axion mass we want to investigate, calculate the effective photon masses for all pressures and the mass attenuation values and we're done. Start with effective photon masses,

```
let pressures = logspace(-6.0, 2.0, 4) # take only few values for a start
let m_gammas = pressures.mapIt(babyIaxoEffMass(it))
```

now for the mass attenuations, we need to derive a density from the pressure values we have:

```
proc density(p: float, temp = 4.2): float =
  ## returns the density of the gas for the given pressure.
  ## The pressure is assumed in `mbar` and the temperature (in `K`).
  ## The default temperature corresponds to BabyIAXO aim.
  ## Returns the density in `g / cm^3`
  const gasConstant = 8.314 # joule K^-1 mol^-1
  const M = 4.002602 # g / mol
  let pressure = p * 1e2 # pressure in Pa
  # factor 1000 for conversion of M in g / mol to kg / mol
  result = pressure * M / (gasConstant * temp * 1000.0)
  # convert the result to g / cm^3 for use with mass attenuations
  result = result / 1000.0
```

Let's add a convenience proc to directly calculate the attentuation length from a pressure in mbar:

```
proc attenuationLength(p: float): float =
  ## for a given pressure in `mbar` returns the attenuation length
  ## `Γ` in units of `eV`.
  # multiply by `100` to convert `Γ` from `1 / cm` to `1 / m`
  # multiply by `1.97e-7` to convert `1 / m` to `1 / eV`
  result = 1.97e-7 * 100.0 * density(p) * exp(logMassAttenuation(4.2))
```

So the densities are:

```
let densities = pressures.mapIt(density(it))
```

Let's first calculate the Γ values for a fixed energy:

```
let gammas = densities.mapIt(it * exp(logMassAttenuation(4.2)))
```

For each of the (Γ, q) pairs we will generate one plot, similar to fig. 5.

To generate it we still need a reasonable guess for the axion mass of interest. We can essentially use the effective photon mass as a baseline and generate N values around $O(1\,\%)$ of the nominal value. Let's try that:

```
proc genAxionConvPlot(gamma, m_gamma: float, nameSuffix = "",
                      start = 0.0, stop = 0.0, length = 10.0) =
  ## generates a single axion conversion probabilit plot
  # calculate reasonable `m_a` values
```

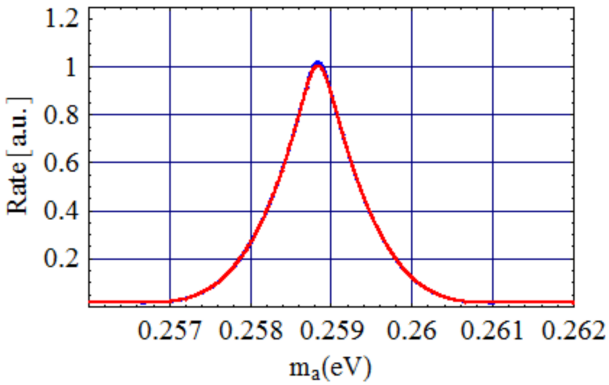

Figure 5: Example of an axion conversion probability taken from the IAXO gas phase study.

```
  echo "Gamma: ", gamma
  echo "m_gamma: ", m_gamma
  var m_a: seq[float]
  if start != stop:
    m_a = linspace(start, stop, 1000)
  else:
    m_a = linspace(m_gamma * 0.99, m_gamma * 1.01, 1000)

  let qs = m_a.mapIt(momentumTransfer(m_gamma, it))
  # plot momentum transfers by themselves
  let dfMom = seqsToDf(m_a, qs)
  ggplot(dfMom, aes("m_a", "qs")) +
    geom_line() +
    ggsave("momentumTransfers.pdf")

  let prob = m_a.mapIt(axionConversionProb(it, m_gamma, gamma, length = length))
  let df = seqsToDf({ "m_a / eV" : m_a,
                      "P_a->gamma" : prob })
  echo df
  #echo df.summarize(f{"P_a->gamma" ~ min("P_a->gamma")})
  #echo df.pretty(-1)
  ggplot(df, aes("m_a / eV", "P_a->gamma")) +
    geom_line() +
    ggtitle(&"Axion conversion probability for Γ = {gamma:.2e}, m_  = {m_gamma:.2e}") +
    ggsave(&"axion_conversion_prob_{nameSuffix}.pdf")
  if nameSuffix == "1":
    writeFile("dfdata.txt", df.pretty(-1))
```

and call it for all values:

```
doAssert gammas.len == m_gammas.len
for i in 0 ..< gammas.len:
  genAxionConvPlot(gammas[i], m_gammas[i], $i)
let dftest = seqsToDf({ "gammas" : gammas,
                        "m_gammas" : m_gammas })
ggplot(dftest, aes("gammas", "m_gammas")) +
  geom_line() +
  ggsave("test_gamma.pdf")
```

Which currently is plain wrong.
~~Let's debug.~~ **UPDATE:** see 2.4, the fixes from there were already applied to the code above.
To avoid any more mistakes, let's write some tests to check whether we can reproduce first the two

values we can read off from the IAXO gas phase study paper. Looking at both fig. 5 and the equivalent for a pressure equivalent of 3 bar at room temperature yields the following reference values, tab. 3.

We calculate the pressure corresponding to the room temperature pressures using the ideal gas law again:

```
proc pressureAtTemp(p: float, T = 4.2, T2 = 293.0): float =
  ## converts a given pressure `p` at `T` to an equivalent pressure at `T2`
  result = p * T2 / T

let p1bar = pressureAtTemp(1000.0, 293, 4.2)
let p3bar = pressureAtTemp(3000.0, 293, 4.2)
echo "Pressure for equivalent of 1 bar @ 293 K ", p1bar
echo "Pressure for equivalent of 3 bar @ 293 K ", p3bar
let m_gamma_1bar = babyIaxoEffMass(p1bar)
let m_gamma_3bar = babyIaxoEffMass(p3bar)
echo "m_gamma for pressure equivalent of 1 bar @ 293 K ", m_gamma_1_bar
echo "m_gamma for pressure equivalent of 3 bar @ 293 K ", m_gamma_3_bar
doAssert almostEqual(m_gamma_1_bar, 0.26)
doAssert almostEqual(m_gamma_3_bar, 0.4483, eps = 1e-2)
```

Which yields the values shown in the table.

P @ 293 K / bar	P @ 4.2 K / mbar	m_γ / eV
1.0	14.3345	0.26048
3.0	43.0034	0.45117

Table 3: Reference values extracted from the IAXO gas phase study paper.

Now we'll generate the exact plots for the conversion probability for these values.

```
let gamma_1bar = attenuationLength(p1bar)
let gamma_3bar = attenuationLength(p3bar)
genAxionConvPlot(gamma_1bar, m_gamma_1bar, "1bar_equiv")#, 0.2593, 0.2616
genAxionConvPlot(gamma_3bar, m_gamma_3bar, "3bar_equiv")#, 0.4485, 0.4535
# and now for reference the IAXO (`20m`) plots
genAxionConvPlot(gamma_1bar, m_gamma_1bar, "1bar_equiv_20m", length = 20.0)
genAxionConvPlot(gamma_3bar, m_gamma_3bar, "3bar_equiv_20m", length = 20.0)
```

However the plot, see the example of the 1 bar equivalent in fig. 20.

UPDATE: the reasons for the plot being broken are explained in 3.3 and in short have to do with the units of different products being wrong, namely of Γ, L and (in a way) q. They have been fixed in the code above. The broken plot has been moved to the mentioned section.

With the code now fixed, we can finally look at the correct axion conversion probability plots. First for the 1 bar equivalent in fig. 6 and for the 3 bar equivalent in fig. 7. In the appendix in fig. 21 and 22 are the same plots for the full IAXO length of 20 m.

Although we still have the more pronounced cos behavior in our conversion probability, at least the widths of the peaks seem to match, as far as one can extract by eye from fig. 5.

In order to be able to determine the FWHM of the peaks (if we calculate the above curves for many pressure values) we can't simply fit a simple function to it and extract some sigma. While we can try to fit a simple gaussian, I don't expect it to fit very well, given the function of the conversion probability eq. 14.

Let's try it anyways for lack of a better way for now. We'll add it to the `genAxionConvPlot` proc above.

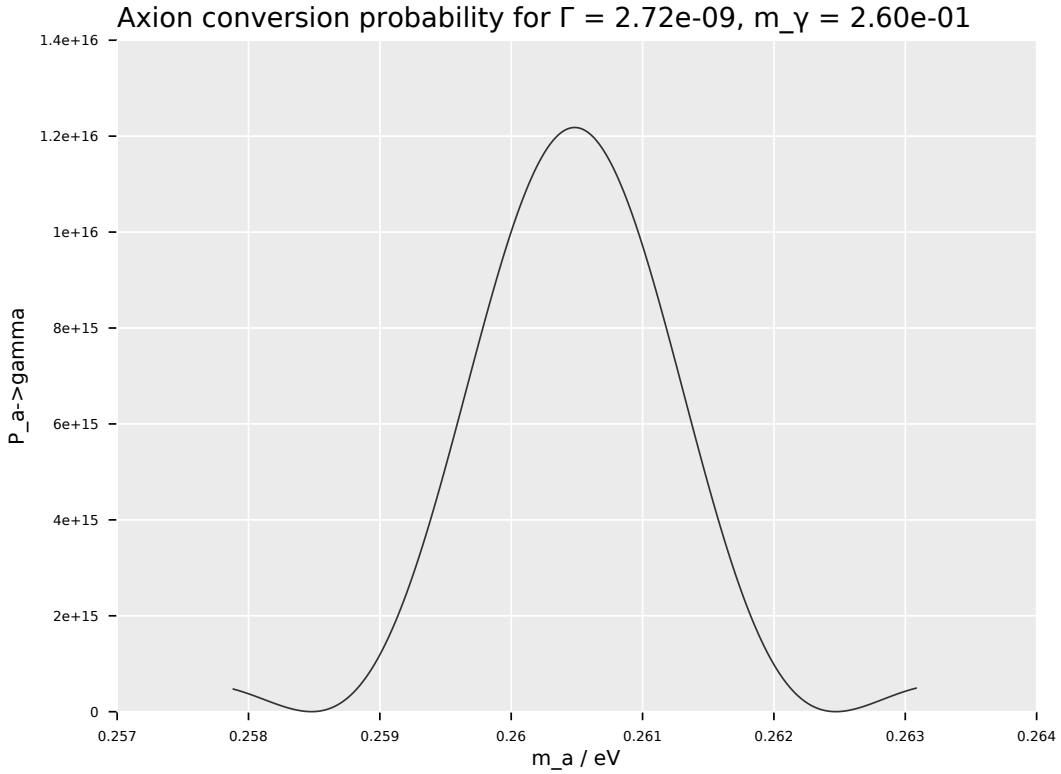

Figure 6: Example of the axion conversion probability at a pressure of $P = 14.3345\,\text{mbar}$ (corresponds to 1 bar at room temperature). This roughly reproduces the plot of fig. 5, although we see the influence of the cos term a lot more. The only difference is that this plot corresponds to the BabyIAXO length of $L = 10\,\text{m}$ instead fig. 5 $L = 20\,\text{m}$. For our IAXO equivalent plot see fig. 21 in the appendix. The absolute value of the probability is arbitrary, since $g_{a\gamma} = 1$ for this plot.

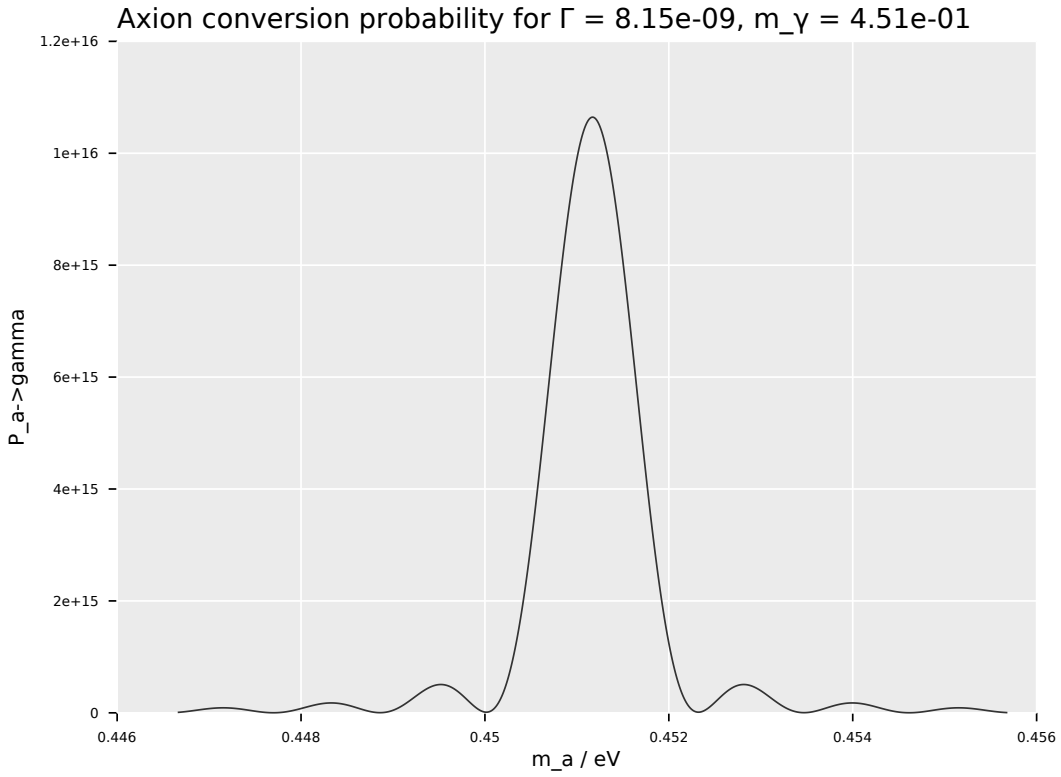

Figure 7: Example of the axion conversion probability at a pressure of $P = 43.0034\,\text{mbar}$ (corresponds to 3 bar at room temperature) and a magnet length of 10 m. For the IAXO equivalent plot see fig. 22 in the appendix.

```
import mpfit
proc gaussFit(p_ar: seq[float], x: float): float =
  result = p_ar[0] * gauss(x = x, mean = p_ar[1], sigma = p_ar[2])

proc calcConvProbCurve(gamma, m_gamma, pressure: float, length = 10.0,
                       massRange = none[tuple[low, high: float]]()):
    tuple[m_a, prob: seq[float]] =
  var m_a: seq[float]
  if massRange.isNone:
    if pressure < 0.01:
      echo "PRressure ", pressure
      m_a = linspace(m_gamma * 0.2, m_gamma * 1.8, 1000)
    else:
      m_a = linspace(m_gamma * 0.5, m_gamma * 1.5, 1000)
  else:
    m_a = linspace(massRange.get.low, massRange.get.high, 1000)
  let qs = m_a.mapIt(momentumTransfer(m_gamma, it))
  let prob = m_a.mapIt(axionConversionProb(it, m_gamma, gamma, length = length))
  result = (m_a: m_a, prob: prob)

proc fitToConvProb(gamma, m_gamma, pressure: float, nameSuffix = "",
                   length = 10.0, createPlot = true): (float, seq[float]) =
  ## generates a single axion conversion probabilit plot
  let (m_a, prob) = calcConvProbCurve(gamma, m_gamma, pressure, length)
  let p0 = @[prob.max, m_a[prob.argmax], 0.02]
  let (pRes, res) = fit(gaussFit, p0, m_a, prob, prob.mapIt(1.0))
  #echoResult(pRes, res = res)
  let probFit = m_a.mapIt(gaussFit(pRes, it))
  if createPlot:
    let df = seqsToDf({ "m_a / eV" : m_a,
                        "P_a->gamma" : prob,
                        "P_fit" : probFit })
    #echo df
    ggplot(df, aes("m_a / eV", "P_a->gamma")) +
      geom_line() +
      geom_line(aes(y = "P_fit", color = "Gaussian fit")) +
      ggtitle(&"Gaussian fit to P_a->gamma at p = {pressure:.4f} mbar") +
      ggsave(&"conv_prob_fit_{nameSuffix}.pdf")
  result = (pressure, pRes)

let (drop, pRes1bar) = fitToConvProb(gamma_1bar, m_gamma_1bar, p1bar, "1bar")
```

If we ignore the code duplication we have right now **cough**, we can see from fig. 8 that a gaussian sort of works, at least for the purpose of extracting a reasonable σ. But let's not talk about our χ^2/dof please (hint $\sim 6.21 \times 10^{28}$).

In order to extract a FWHM, we simply have to take

$$\text{FWMH} = 2\sqrt{2\ln(2)}\sigma \approx 2.355\sigma \tag{27}$$

of our σ.

Let's do that.

```
proc fwhm(sigma: float): float =
  ## returns the FWHM of a gaussian for a `sigma`
  result = 2 * sqrt(2 * ln(2.0)) * abs(sigma)
echo "FWHM @ 1bar room temperature = ", fwhm(pRes1bar[2])
```

So the FWHM of the fit in fig. 8 is:

$$\text{FWHM}_{@1\,\text{bar}} \approx 0.001\,691\,\text{eV} \tag{28}$$

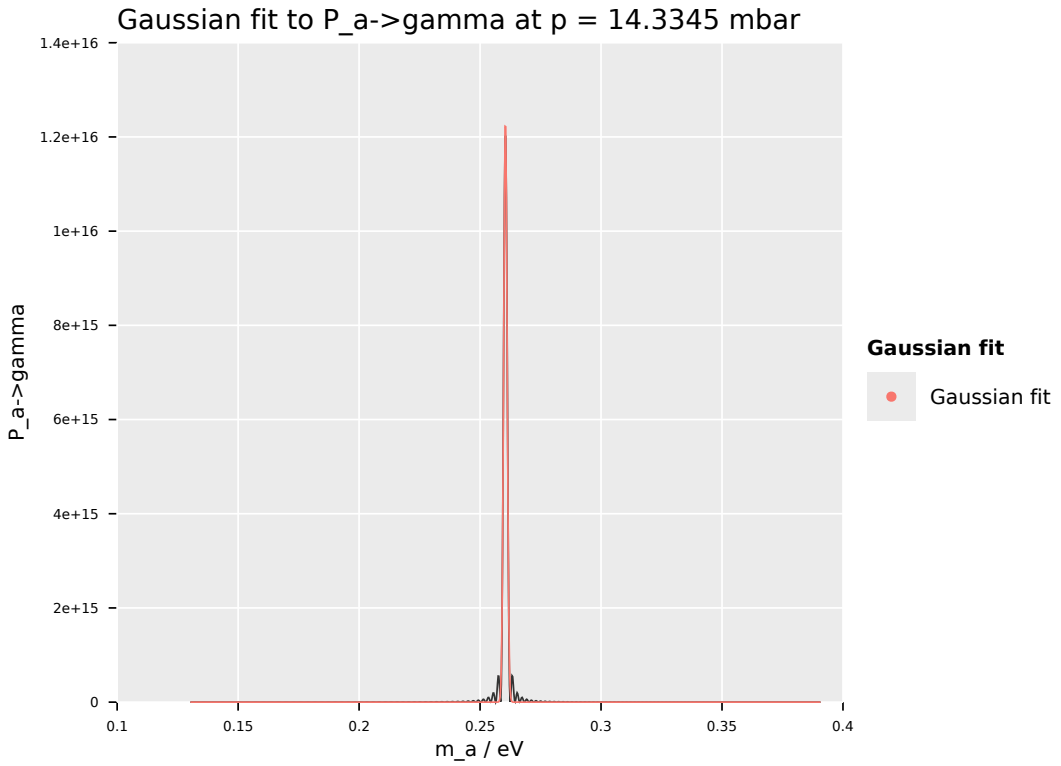

Figure 8: Gaussian fit applied to the axion conversion probability. While the χ^2 is beyond horrible, the fit is reasonable to extract the FWHM.

Thus, finally get back to our `logspace` derivation of some pressures, calculate $P_{a\to\gamma}$ for each, apply the fit and see where it leads us.

```
let pressuresFine = logspace(-6.0, 2.0, 1000)
let gammasFine = pressuresFine.mapIt(attenuationLength(it))
let mgammasFine = pressuresFine.mapIt(babyIaxoEffMass(it))
var fwhmFine: seq[float]
when true:
  # NOTE: set to `true` if you want to recreate the fits and plots!
  for i in 0 ..< pressuresFine.len:
    let (p, pResI) = fitToConvProb(gammasFine[i], mgammasFine[i],
                                   pressuresFine[i], &"{pressuresFine[i]:.6f}",
                                   createPlot = true)
    fwhmFine.add fwhm(pResI[2])
```

and now create the plot relating the pressure to the FWHM:

```
when true:
  # NOTE: set to `true` if you want to redo these calculations
  var dfFwhm = seqsToDf({ "Pressure / mbar" : pressuresFine,
                          "FWHM / eV" : fwhmFine })
  dfFwhm = dfFwhm.mutate(f{float: "FWHM / eV" ~ abs(`FWHM / eV`)})
```

```
  ggplot(dfFwhm, aes("Pressure / mbar", "FWHM / eV")) +
    geom_point() +
    ylab(margin = 1.5) +
    scale_y_log10() +
    scale_x_log10() +
    ggtitle("FWHM / eV of a->gamma probability depending on pressure / mbar") +
    ggsave("fwhm_vs_pressure_full_loglog.pdf")
  # and a subset filtered to above 0.01 mbar in linear
  dfFwhm = dfFwhm.filter(f{float: `Pressure / mbar` > 0.01})
  ggplot(dfFwhm, aes("Pressure / mbar", "FWHM / eV")) +
    geom_point() +
    scale_y_log10() +
    ggtitle("FWHM / eV of a->gamma probability depending on pressure / mbar") +
    ggsave("fwhm_vs_pressure_ylog.pdf")

  let elements = pressuresFine.len
  doAssert pressuresFine.len == elements
  doAssert gammasFine.len == elements, " was " & $gammas.len
  doAssert mgammasFine.len == elements, " was " & $mgammas.len
  doAssert fwhmFine.len == elements, " was " & $fwhmFine.len

# finally write the data to a file
proc writeCsv(pressures, gammas, mgammas, fwhm: seq[float])
when true:
  writeCsv(pressuresFine, gammasFine, mgammasFine, fwhmFine)
```

This gives us two plots. First in fig. 9 the full range from 1×10^{-6} mbar to 1×10^{2} mbar in log-log. We see that below $\sim 1 \times 10^{-4}$ mbar the behavior changes significantly and probably the fit starts to break down. But in those conditions we should be well in the coherent part of the phase space anyways. So take that with a grain of salt. Keep in mind that these values are the cryogenic pressures, so ~ 43 mbar already equate to rougly 3 bar equivalent pressure at room temperature!

Then in fig. 10 we see the same data zoomed to all values above 0.01 mbar. Here we see the curve changes even faster than an exponential. Given the conversion probability itself already decays as an $\exp^{-\Gamma L}$ and the additional suppression of q in the denominator, this is to be expected I suppose. Or maybe not quite? Since for one plot, i.e. one value in the FWHM plots the only changing values **is** q. So it's just a $\frac{1}{x^2}$ dependency? Well, if I had more time, we could fit, but not today…

For once I'm not going to append **all** of the gaussian fits to the conversion probabilities. Otherwise this will have $\mathcal{O}(1000)$ pages…

One example of a very low pressure scenario is shown in fig. 11.

And let's implement the `writeCsv` proc to store the results.

```
proc writeCsv(pressures, gammas, mgammas, fwhm: seq[float]) =
  var f = open("fwhm_results.csv", fmWrite)
  f.write("# Pressures / mbar\t Γ / eV\t m_  / eV & fwhm / eV\n")
  let elements = pressures.len
  var line = ""
  for i in 0 ..< elements:
    line = &"{pressures[i]},{gammas[i]},{mgammas[i]},{abs(fwhm[i])}\n"
    f.write(line)
  f.close()
```

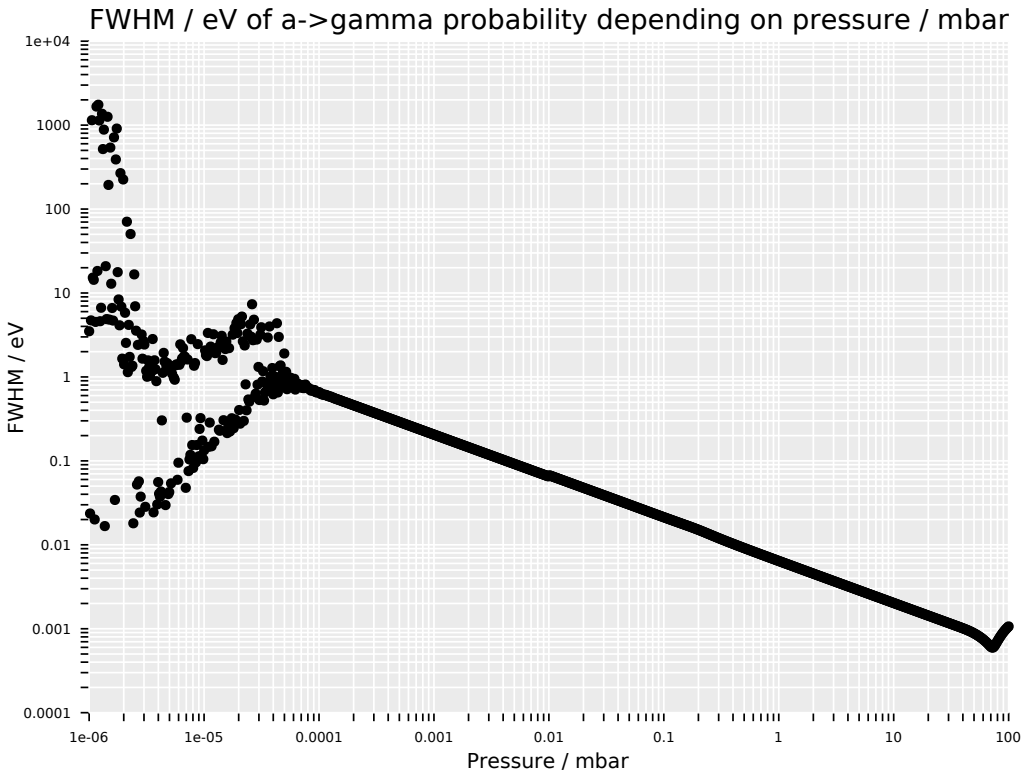

Figure 9: FWHM in eV of the axion conversion probability $P_{a\to\gamma}$ depending on the ^{4}He pressure inside BabyIAXO in mbar. The full range from 1×10^{-6} mbar to 1×10^{2} mbar at cryogenic temperatures is shown. Below $\sim 1 \times 10^{-4}$ mbar the fit probably breaks.

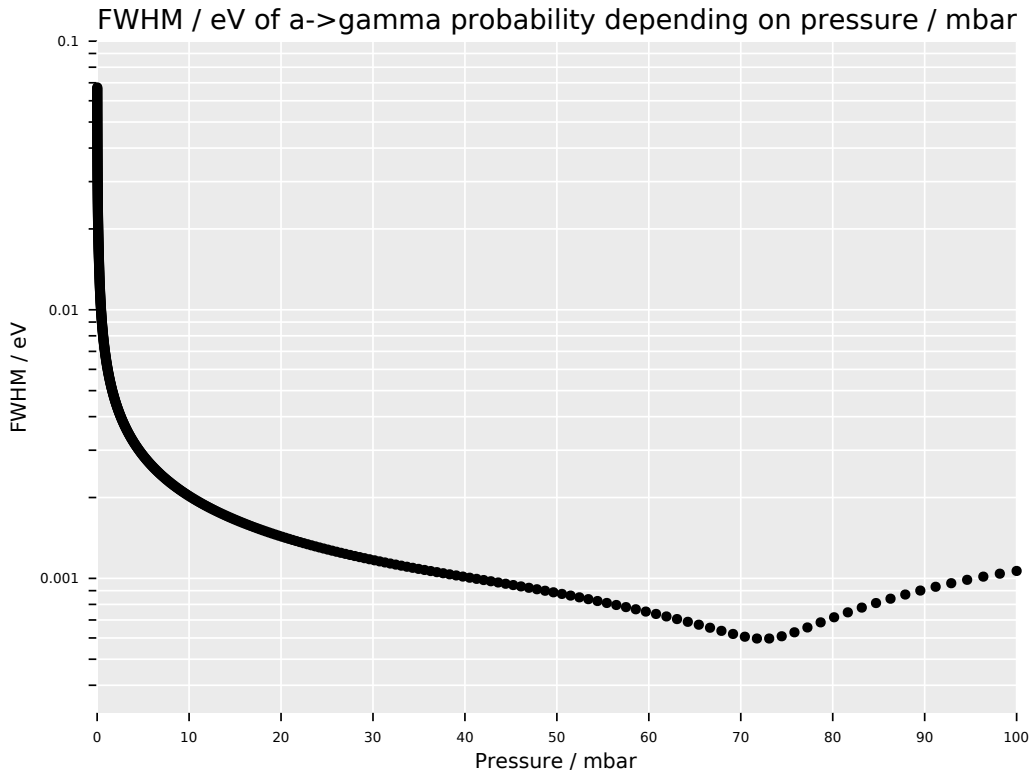

Figure 10: FWHM in eV of the axion conversion probability $P_{a\to\gamma}$ depending on the ^{4}He pressure inside BabyIAXO in mbar. The range is zoomed to 1×10^{-2} mbar to 1×10^{2} mbar at cryogenic temperatures. The curve seems to behave well in this range.

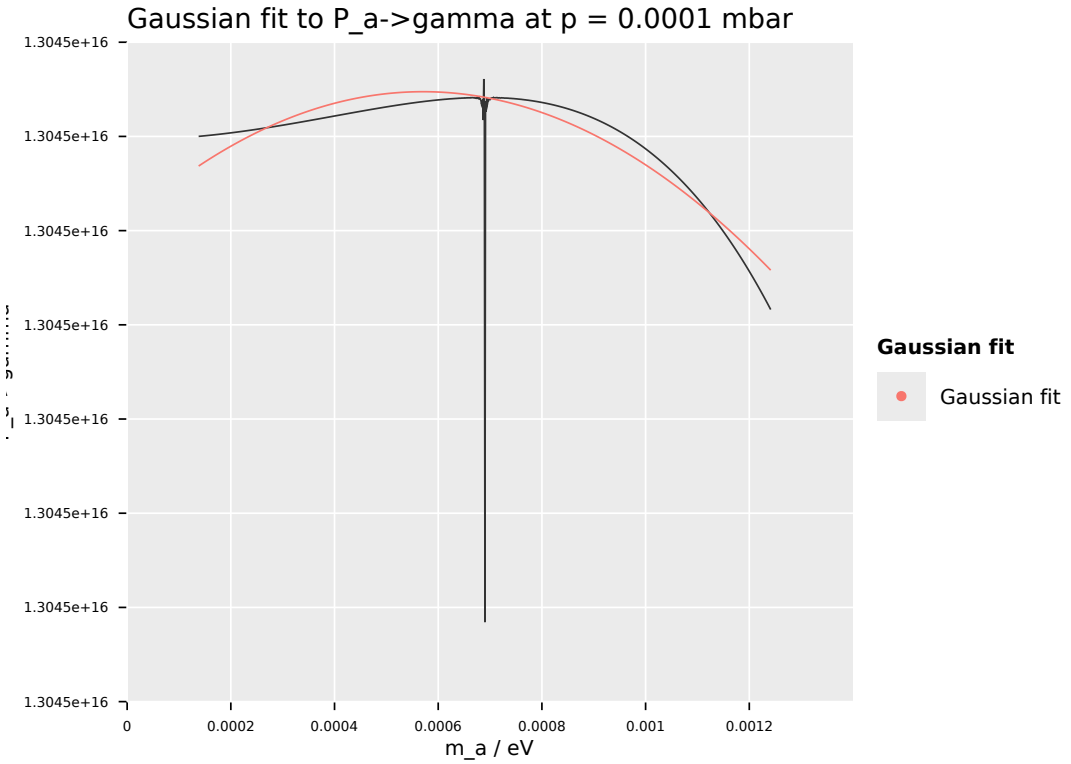

Figure 11: Example at which the gaussian fit fails at very low pressure. Even further below it's just a straight line with the dip.

2.4 Find our error in the calculation

It seems like calculating from the alternative m_γ calculation and putting in the pressure (derived by ideal gas law) equivalent of 1 bar at room temperature (yields 14.33 mbar at 4.2 K), matches with our expectation from the gas study, $m_\gamma \approx 0.261\,\text{eV}$.

```
echo "Room temp: ", pressures.mapIt(pressureAtTemp(it) / 1000.0)

proc m_gamma_alternative(p: float): float =
  # gallard alternative m_gamma. Says 0.02, more exact is:
  result = sqrt(0.01988 * p / 4.2)

let compPress = pressureAtTemp(1000, 293.0, 4.2)
echo "cryo pressure for 1 bar at room: ", compPress
# get m_a at compPress
echo "m_a @ 1 bar @ room temp: ", babyIaxoEffMass(compPress)
echo "m_a alt. @ 1 bar room temp ", m_gamma_alternative(compPress)
doAssert babyIaxoEffMass(compPress).almostEqual(m_gamma_alternative(compPress))
```

$$PV = nRT \tag{29}$$
$$n = \frac{m}{M} \tag{30}$$
$$PV = \frac{m}{M}RT \tag{31}$$
$$P = \frac{m}{V}\frac{R}{M}T \tag{32}$$
$$P = \rho\frac{R}{M}T \tag{33}$$
$$\rho = \frac{PM}{RT} \tag{34}$$

And we know that these two are equivalent:

$$n = N_A c \tag{35}$$
$$n = \frac{N_A}{M}\rho \tag{36}$$

So this is the number density of molecules **molecules** with molar mass M. That means the number of **electrons** is Z times larger! That's where the factor of Z in equation 60 comes from!:

$$n_e = Z\frac{N_A}{M}\rho \tag{37}$$

Then we can alternatively express this in terms of the pressure instead of density, via

$$n_e = Z\frac{N_A}{M}\frac{PM}{RT} \tag{38}$$
$$n_e = Z\frac{N_A P}{RT} \tag{39}$$
$$\tag{40}$$

which we can finally insert into the expression for m_γ eq. 4 to get:

$$m_\gamma = \sqrt{\frac{4\pi\alpha Z n_e P}{RT m_e}} \tag{41}$$

which after fixing the units and making P be in mbar:

$$m_\gamma = \sqrt{\frac{4\pi\alpha Z n_e (1.97\times10^{-7})^3 \cdot 100 \cdot P(\text{mbar})}{RT m_e}} \tag{42}$$

due to $100\,\text{Pa} = 1\,\text{mbar}$. Inserting all known constants:

$$m_\gamma = \sqrt{\frac{0.019\,88 \cdot P(\text{mbar})}{T(\text{K})}} \tag{43}$$

which is precisely equation 59.

Now let's make sure we can similarly derive the equation from Theodoros' thesis by inserting 60 instead:

$$m_\gamma = \sqrt{\frac{4\pi\alpha n_e}{m_e}} \tag{44}$$

$$m_\gamma = \sqrt{\frac{4\pi\alpha Z N_A \rho}{M m_e}} \tag{45}$$

Let's again consider the units of this:

$$\text{eV} = \sqrt{\frac{\text{kg}\cdot\text{mol}}{\text{m}^3\cdot\text{g}\cdot\text{eV}}}, \tag{46}$$

based on the fact that we typically give the molar mass in $\text{g}\,\text{mol}^{-1}$. Yet, for the NIST data about the mass attenuation coefficient, it is more convenient, if we give our density not int kg/m^3, but rather in g/cm^3, because the attenuation coefficients are also given in cm^2/g. Which means we end up with the following corrections:

$$m_\gamma = \sqrt{\frac{4\pi\alpha Z N_A \cdot 1000 \cdot 1000 \cdot (1.97\times10^{-7})^3 \rho}{M m_e}} \tag{47}$$

$$m_\gamma = \sqrt{\frac{826.571 Z\rho}{M}} \tag{48}$$

$$m_\gamma = 28.75\sqrt{\frac{Z}{M}\rho\,[\text{g/cm}^3]} \tag{49}$$

where one of the factors of 1000 is from the conversion of M to $\text{g}\,\text{mol}^{-1}$ and the other from the conversion of ρ to g/cm^3.

This means we have finally proven the seemingly arbitrary relation of m_γ.

TL;DR:

- we missed the factor Z in our initial calculation of the electron number density 7, because we forgot that there's Z electrons for each molecule apparently :P
- while trying to derive the equation for eq. 59 we were still missing a conversion from kg/m^3 to g/cm^3 for the density.

2.5 Absorption of X-rays in helium

In general the absorption of X-rays in a medium is also governed by the Beer-Lambert law, which for a uniform medium can be expressed via the attenuation length or the mass attenuation coefficient as:

$$I(d) = I_0 \exp\left(-\frac{\mu}{\rho}\rho d\right) = I_0 \exp\left(-\Gamma d\right) \tag{50}$$

where $\frac{\mu}{\rho}$ is again the mass attenuation coefficient and Γ the inverse absorption length.

The former expression is preferable, since we already have the values from tab. 1 and the interpolation eq. 16.

So for a given photon energy E we can calculate the X-ray flux after a traversed distance d:

```
proc intensitySuppression(energy, distance, pressure: float, temp = 293.15): float =
  ## calculates the suppression factor of the intensity of X-rays
  ## with `energy` (in keV) after `distance` (in `m`) at a helium `pressure`
  ## (in `mbar`) and temperature of the gas of `temp` (in `K`).
  ## By default assume room temperature for calc of beam line filled with
```

```
  ## gas.
  let massAtt = exp(logMassAttenuation(energy))
  # calculate density from pressure
  let rho = density(pressure, temp)
  # factor 100 is to convert `distance` to `cm` from `m`
  result = exp(-massAtt * rho * distance * 100)
```

2.5.1 Absorption in beam line at room temp @ 1 bar pressure

Assuming the BabyIAXO magnet is supposed to be filled with helium gas at room temperature (no cryo for the bore and even heated to 293.15 K), then we need to operate at pressures of $\mathcal{O}(1\,\text{bar})$. For a simple (and thin!) window setup it could be a nice solution to consider filling the beamline (incl. telescope) itself with helium at the same conditions. However, assuming a length of 5 m this results in a suppression of the intensity.

Let's calculate how large that suppression is and check how much intensity is lost after 5 m at 1000 mbar and 293.15 K for all energies 0 keV to 10 keV:

```
proc calcFilledBeamline(length: float, energies: seq[float],
                        toPlot = true): DataFrame =
  ## calculates the X-ray suppression of a beamline filled with gas at
  ## `1 bar`, `293.15 K` of `5 m` (default) length
  # take intensity suppression as is, assumes incoming intensity is `1`
  let p = 1000.0 # mbar
  let intensities = energies.mapIt(intensitySuppression(it, length, p))
  result = seqsToDf({ "E / keV" : energies,
                      "Transmission" : intensities })
  if toPlot:
    ggplot(result, aes("E / keV", "Transmission")) +
      geom_line() +
      ggtitle(&"Transmission after {length} m of He at 1 bar, 293.15 K") +
      ggsave(&"transmission_beamline_he_{length}_m.pdf")

let df5m = calcFilledBeamline(5.0, energies)
let df7_5m = calcFilledBeamline(7.5, energies)
echo "\n\n\n"
echo df5m.tail(10)
echo df7_5m.tail(10)
echo "done\n\n\n"
```

The same plot is also generated for 7.5 m. The plot for transmission after 5 m is shown in fig. 12.

Finally we wish to create a combined plot of the transmissions of the 5 m transmission, the 7.5 m case, a potential 10 µm polypropylene (C_3H_6) window and the axion electron flux.

The transmission of the polypropylene window is calculated from `http://henke.lbl.gov/optical_constants/filter2.html` and the data file is `polypropylene_window_10micron.txt`.

```
proc transmissionsPlusSpectrum =
  ## creates a plot of the expected transmissions for filled beamlines,
  ## the polypropylene window and the axion electron spectrum.
  let polypropDf = toDf(readCsv("polypropylene_window_10micron.txt", sep = ' ', skipLines = 1,
                                header = "#"))
    .mutate(f{float: "E / keV" ~ c"Energy/eV" / 1000.0})
    .rename(f{"10 µm PP" <- "Transmission"})
  # now using the `polyPropDf` energies, calculate the data frames for the He losses, so that
  # we have the exact same energies and we can easily multiply them afterwards
```

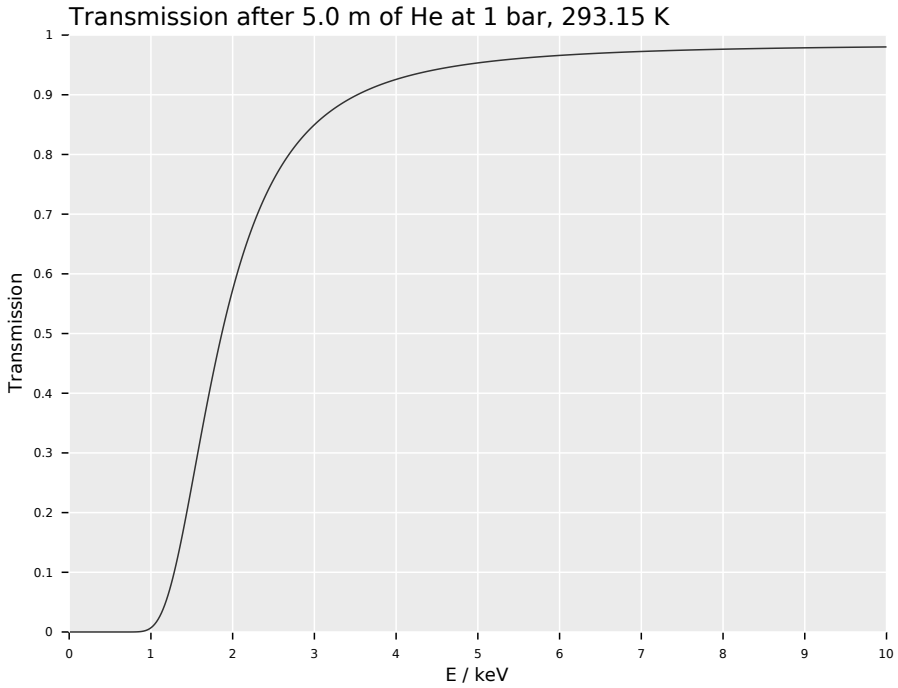

Figure 12: Transmission of X-rays after 5 m of ^{4}He at $P = 1000\,\text{mbar}$ and $T = 293.15\,\text{K}$.

```
let energiesPP = polyPropDf["E / keV", float].toRawSeq
let df5m = calcFilledBeamline(5.0, energiesPP).rename(f{"5 m He" <- "Transmission"})
let df7_5m = calcFilledBeamline(7.5, energiesPP).rename(f{"7.5 m He" <- "Transmission"})
# do an ugly multi join
var dfPPHe = inner_join(polyPropDf, df5m, by = "E / keV").innerJoin(df7_5m, by = "E / keV")
# finally add polyProp * He
dfPPHe = dfPPHe.mutate(f{"5mHe+PP" ~ c"10 µm PP" * c"5 m He"},
                       f{"7.5mHe+PP" ~ c"10 µm PP" * c"7.5 m He"})
  .gather(["10 µm PP", "5 m He", "5mHe+PP", "7.5 m He", "7.5mHe+PP"],
          key = "Setup", value = "Transmission")
let gaeDf = toDf(readCsv("axion_gae_flux.dat", sep = ' ', skipLines = 9, header = "#"))
  .rename(f{"E / keV" <- "Energy/keV"})
  .mutate(f{float: "Phi" ~ `Phi` / max(`Phi`) * 100.0})
  .filter(f{float: c"E / keV" <= 10.0})
var fullDf = bind_rows([("Axion-Electron flux", gaeDf),
                        ("Setups", dfPPHe)],
                       id = "Type")
  .select("Type", "E / keV", "Phi", "Transmission", "Setup")
  # `Transmission` contains many `VNull` values, math w/ `Value`
  .mutate(f{Value -> Value: "Transmission" ~ `Transmission` * (%~ 100.0)})
ggplot(fullDf.filter(f{string: `Type` != "Axion-Electron flux"}),
       aes(x = "E / keV", y = "Transmission")) +
  geom_line(aes(color = "Setup")) +
  geom_line(aes(x = "E / keV", y = "Phi"),
            data = fullDf.filter(f{string: `Type` == "Axion-Electron flux"})) +
  scale_y_continuous("Transmission / %", secAxis = secAxis(name = "Axion flux / a.u.")) +
  legend_position(x = 0.88, y = 0.1) +
  #xlim(0.0, 3.0, outsideRange = "drop") +
  ggtitle("Comparison of He filled beamline (1 bar, 293 K) and 10 µm window") +
  ggsave("window_he_transmissions_axion_flux.pdf", width = 853, height = 480)

```

```
41 transmissionsPlusSpectrum()
```

The resulting combined plot is shown in fig. 13.

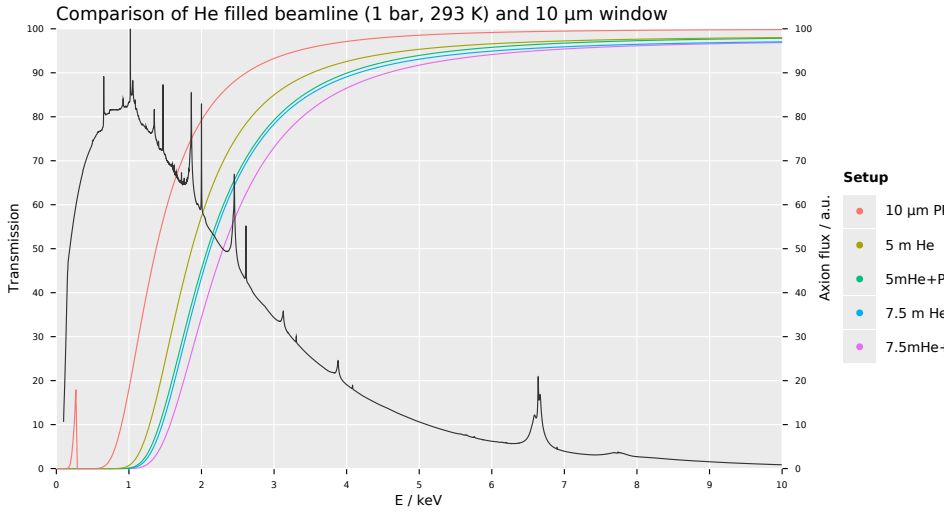

Figure 13: Comparison of three different possible setups. Either the beamline is filled with ^{4}He at \$P = 1 bar, $T = 293.15$ K and has a length of 5 m (red) or 7.5 m (green) or the beamline is under vacuum and a 10 µm C_3H_6 polypropylene window (blue) is used. In addition the expected solar axion-electron flux is shown in black in arbitrary units.

2.6 First pressure value for He filling

The next thing to consider is to calculate the first pressure (or rather density) setting, which would be taken on a ^{4}He run. That is, from the "2.2", that is the pressure (density) equivalent for $m_a = 0.0228$ eV, that pressure is sought, which results in a FWHM of the same value as the declining vacuum sensitivity at that mass.

So we have to calculate two things:

1. the full vacuum sensitivity curve
2. determine the density (pressure) that corresponds to the m_γ, which produces the resonance curve (cf. fig. 7) that lines up with the vacuum curve at its FWHM on the left edge.

Let's start with 1.

2.6.1 Full vacuum sensitivity curve

Analytically the vacuum conversion probability can be derived from the expression eq. 14 by simplifying q for $m_\gamma \to 0$ and $\Gamma = 0$:

$$P_{a\to\gamma,\text{vacuum}} = \left(\frac{g_{a\gamma}B}{2}\right)^2 \frac{1}{q^2}\left[1 + 1 - 2\cos(qL)\right] \tag{51}$$

$$P_{a\to\gamma,\text{vacuum}} = \left(\frac{g_{a\gamma}B}{2}\right)^2 \frac{2}{q^2}\left[1 - \cos(qL)\right] \tag{52}$$

$$P_{a\to\gamma,\text{vacuum}} = \left(\frac{g_{a\gamma}B}{2}\right)^2 \frac{2}{q^2}\left[2\sin^2\left(\frac{qL}{2}\right)\right] \tag{53}$$

$$P_{a\to\gamma,\text{vacuum}} = \left(g_{a\gamma}B\right)^2 \frac{1}{q^2}\sin^2\left(\frac{qL}{2}\right) \tag{54}$$

$$P_{a\to\gamma,\text{vacuum}} = \left(\frac{g_{a\gamma}BL}{2}\right)^2 \left(\frac{\sin\left(\frac{qL}{2}\right)}{\left(\frac{qL}{2}\right)}\right)^2 \tag{55}$$

$$P_{a\to\gamma,\text{vacuum}} = \left(\frac{g_{a\gamma}BL}{2}\right)^2 \left(\frac{\sin\left(\delta\right)}{\delta}\right)^2 \tag{56}$$

$$\tag{57}$$

Let's implement this and see if it reproduces the known plot.

```nim
proc vacuumConversionProb(m_a: float, length = 10.0): float =
  ## calculates the conversion probability in BabyIAXO for the given axion
  ## mass `m_a`
  # both `g_agamma` and `B` only scale the absolute value `P`, does not matter
  const g_agamma = 1.0 #1e-11
  const B = 4.5 # T, actually don't know the real number right now
  # convert length in `m` to `eV`
  let L = length / 1.97e-7 # m
  let E_a = 4.2 # keV mean value for Primakoff spectrum
  let q = momentumTransfer(m_gamma = 0.0, m_a = m_a)
  let term1 = pow(g_agamma * B * L / 2.0, 2)
  let term2 = pow(sin(q * L / 2.0) / (q * L / 2.0), 2.0)
  result = term1 * term2

proc vacuumConversionProbApprox(m_a: float, length = 10.0): float =
  ## approximates the vacuum conversion probability to second order of taylor
  ## expansion (4th power of `sin^2` argument)
  # both `g_agamma` and `B` only scale the absolute value `P`, does not matter
  const g_agamma = 1.0 #1e-11
  const B = 4.5 # T, actually don't know the real number right now
  # convert length in `m` to `eV`
  let L = length / 1.97e-7 # m
  let E_a = 4.2 # keV mean value for Primakoff spectrum
  let q = momentumTransfer(m_gamma = 0.0, m_a = m_a)
  let term1 = pow(g_agamma * B * L / 2.0, 2)
  let term2 = 1.0 - pow(q * L / 2.0, 2.0) / 3.0
  result = term1 * term2

proc vacuumConvProbPlot(useApprox = false) =
  ## generates the typical plot of the conversion probability in the range
  ## 1 μeV to 1 eV for g_a,gamma = 1.0 (for simplicity)
  # we use linspace so we have more fidelity where the curve is more interesting
  let vacMasses = linspace(1e-6, 1.0, 1000)
  var probs: seq[float]
  var outname = ""
  if not useApprox:
    probs = vacMasses.mapIt(vacuumConversionProb(it))
```

```
    outname = "vacuum_conversion_prob.pdf"
  else:
    probs = vacMasses.mapIt(vacuumConversionProbApprox(it))
    outname = "vacuum_conversion_prob_taylor.pdf"
  let dfVac = seqsToDf({ "m_a / eV" : vacMasses,
                         "P_a,gamma" : probs})

  ggplot(dfVac, aes("m_a / eV", "P_a,gamma")) +
    geom_line() +
    #geom_point() +
    scale_x_log10() +
    scale_y_log10() +
    ggsave(outname)
vacuumConvProbPlot()
# The below results in negative values for the probability, which breaks the
# log log plot. Why negative from approximation?
#vacuumConvProbPlot(useApprox = true)
```

The resulting vacuum conversion probability plot is shown in fig. 14.

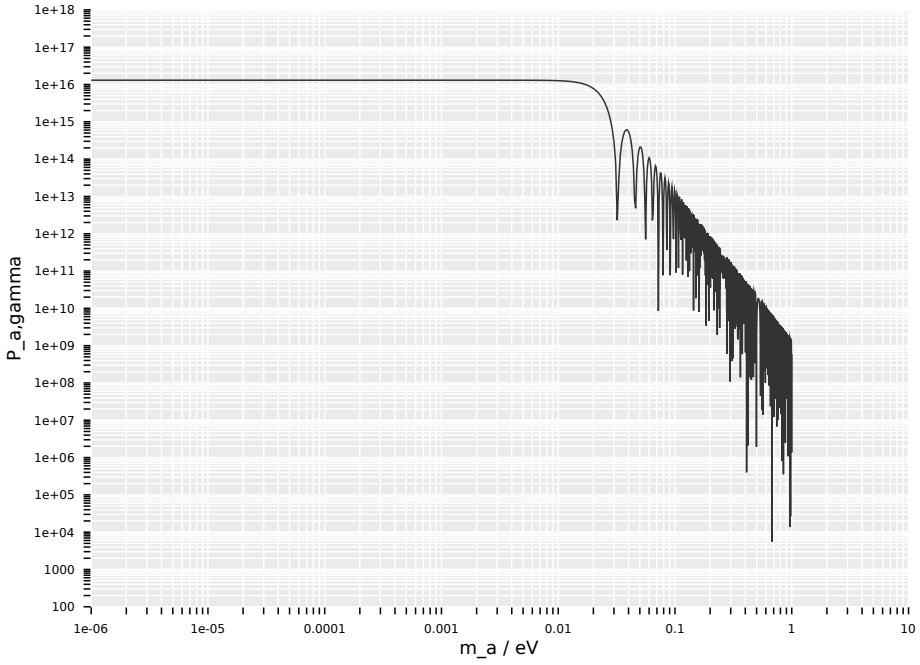

Figure 14: Normal axion photon conversion probability in the BabyIAXO magnet assuming $g_{a,\gamma} = 1$ in the typical mass range. The behavior for higher masses is due to being off-resonance ($qL > \pi$) and is usually not shown in the vacuum plots.

2.6.2 Determine the next pressure (density) step

Now starts the more complicated part of the calculation, namely 2.

It can be decomposed into a few parts:

- the vacuum curve
- a set of (m_γ, Γ, ρ), which defined the conversion probability with a buffer gas

- calculate the mass and conversion probability at the left edge of the FWHM of the buffer curve
- find m_a of the vacuum curve for the P of the (m_a, P) point of the buffer curve
- calculate the distance mass distance between $m_{a,\text{vacuum at P}}$ and $m_{a,\text{left FHWM}}$
- take absolute value of distance
- perform minimization of that distance. Minimum corresponds to desired density (pressure) point.

Let's build this up in code then.

```
proc pressureToGammaMGamma(pressure: float): tuple[gamma, m_gamma: float] =
  ## returns a tuple of the corresponding `gamma` and
  ## `m_gamma` values associated with the pressure for BabyIAXO filled with
  ## He.
  result = (gamma: attenuationLength(pressure),
            m_gamma: babyIaxoEffMass(pressure))

proc getDfVac(massRange: Option[tuple[low, high: float]]): DataFrame =
  let r = if massRange.isNone: (low: 1e-6, high: 1.0) else: massRange.get
  let vacMasses = linspace(r.low, r.high, 1000)
  let vacProbs = vacMasses.mapIt(vacuumConversionProb(it))
  result = seqsToDf({ "m_a / eV" : vacMasses,
                      "P_a->gamma" : vacProbs})
  # echo result.pretty(50)

type
  MassFit = object
    discard
proc vacuumMassGivenProb(prob: float): float =
  ## determines the mass corresponding to the given probability.
  ## Since `P` even for the vacuum case is not really invertible,
  ## we'll do it numerically.
  proc probDiff(p0: seq[float], dummy: MassFit): float =
    result = abs(vacuumConversionProb(p0[0]) - prob)
  var opt = newNloptOpt[MassFit](LN_COBYLA, 1)
  # hand the function to fit as well as the data object we need in it
  var varStr = newVarStruct(probDiff, MassFit())
  opt.setFunction(varStr)
  let (params, minVal) = opt.optimize(@[0.01])
  result = params[0]
  opt.destroy()
echo vacuumMassGivenProb(1.5e15)

proc pressureGivenEffPhotonMass(m_gamma: float, T = 4.2): float =
  ## calculates the inverse of `babyIaxoEffPhotonMass`, i.e. the pressure
  ## from a given effective photon mass in BabyIAXO
  result = m_gamma * m_gamma * T / 0.01988

template fwhmTail(name: untyped, op: untyped): untyped =
  proc `name`(m_a: float, outputInfo = false,
              massRange = none[tuple[low, high: float]]()):
    tuple[m_as, probs: seq[float], mAtFwhm, pHalf: float] =
    # calculate pressure corresponding to `m_a`
    let pressure = pressureGivenEffPhotonMass(m_a)
    # get gamma and m_gamma from pressure
    let (gamma, m_gamma) = pressureToGammaMGamma(pressure)
    doAssert abs(m_a - m_gamma) < 1e-4
    let (m_as, probs) = calcConvProbCurve(gamma, m_gamma,
```

```
                                          pressure, massRange = massRange)
    let (p, pResI) = fitToConvProb(gamma, m_gamma, pressure,
                                   nameSuffix = "teststest", createPlot = true)
    let fwhmVal = fwhm(pResI[2])
    let mAtFwhm = `op`(pResI[1], fwhmVal / 2.0)
    let pHalf = pResI[0] / 2.0
    result = (m_as: m_as, probs: probs, mAtFwhm: mAtFwhm, pHalf: pHalf)
    if outputInfo:
      echo "Axion mass m_a = ", m_a, " eV"
      echo "Pressure P = ", pressure, " mbar"
      echo "Attenuation length Gamma = ", gamma
      echo "Effective photon mass m_gamma = ", m_gamma, " eV"
      echo "FWHM ", fwhmVal

fwhmTail(lhsFwhmTail, `-`)
fwhmTail(rhsFwhmTail, `+`)

proc massDifference(m_a, prob: float, cmpMass = none[float]()): float =
  ## Given a conversion probability value at a certain mass, calculate the
  ## mass difference between this value and the vacuum curve
  var ma_cmp: float
  if cmpMass.isNone:
    ma_cmp = vacuumMassGivenProb(prob)
  else:
    let (m_as, probs, mAtFwhm, pHalf) = rhsFwhmTail(cmpMass.get)
    ma_cmp = mAtFwhm
    echo "using ", cmpMass
  echo "Corresponding ma in vacuum ", ma_cmp, " for ", prob
  result = abs(m_a - ma_cmp)

proc gasStepsPlot(massLhs, massRhs: tuple[name: string, m: float], title = "",
                  filterMass = none[tuple[low, high: float]]()) =
  ## generates a plot showing the resonance curves for the two masses
  ## (or one if one mass is 0.0). Compares it to the vacuum conversion
  ## probability.
  let (m_as, probs, mAtFwhm, pHalf) = lhsFwhmTail(massLhs.m, massRange = filterMass)
  let dfLhs = seqsToDf({ "m_a / eV" : m_as,
                         "P_a->gamma" : probs,
  })
  echo "LHS = ", massLhs
  echo "LHS P = ", pressureGivenEffPhotonMass(massLhs.m)
  var dfComb: DataFrame
  if massRhs.name.len == 0:
    dfComb = bind_rows([("vacuum", getDfVac(filterMass)),
                        (massLhs.name, dfLhs)],
                        #("xline", dfDummy),
                        #("yline", dfDummy2),
                       id = "case")
  else:
    echo "RHS = ", massRhs
    echo "RHS P = ", pressureGivenEffPhotonMass(massRhs.m)
    let (m_Rhs, probsRhs, _, _) = rhsFwhmTail(massRhs.m, massRange = filterMass)
    let dfRhs = seqsToDf({ "m_a / eV" : m_Rhs,
                           "P_a->gamma" : probsRhs,
    })
    dfComb = bind_rows([("vacuum", getDfVac(filterMass)),
                        (massRhs.name, dfRhs),
                        (massLhs.name, dfLhs)],
```

```
                        id = "case")
  if filterMass.isSome:
    let fm = filterMass.get
    dfComb = dfComb.filter(f{float: c"m_a / eV" >= fm.low and c"m_a / eV" <= fm.high})
  var plt = ggplot(dfComb, aes("m_a / eV", "P_a->gamma", color = "case")) +
    geom_line() +
    scale_y_log10() +
    ggtitle(title)
  if filterMass.isNone:
    plt = plt + scale_x_log10()
  let n1 = massLhs.name.replace(" ", "_")
  let n2 = massRhs.name.replace(" ", "_")
  plt + ggsave(&"vacuum_helium_cutoff_{mAtFWHM}_{n1}_{n2}.pdf")


proc massDifferenceAtMass(m_a: float, createPlot = true,
                          outputInfo = false, cmpMass = none[float]()): float =
  ## returns the pressure that corresponds to the mass that is required to
  ## achieve a FWHM overlap between the vacuum curve and the buffer gas setup
  # use `m_a` as start
  # calculate buffer curve with this value
  echo "Eff photon mass ", m_a
  let (m_as, probs, mAtFwhm, pHalf) = lhsFwhmTail(m_a, outputInfo)
  # find m_a of `pHalf` on vacuum curve
  result = massDifference(mAtFwhm, pHalf, cmpMass)
  if createPlot:
    if cmpMass.isNone:
      gasStepsPlot((name: "helium", m: m_a), massRhs = (name: "", m: 0.0))
    else:
      gasStepsPlot((name: "helium", m: m_a),
                   (name: "heliumRef", m: cmpMass.get))

proc findMassAtFwhm(m_a: float, cmpWithMassless = true): float =
  ## performs minimization of `massDifferenceAtMass` to find the mass `m_a` at which
  ## the FWHM of the buffer phase matches the vacuum line
  proc findMin(p0: seq[float], dummy: MassFit): float =
    if cmpWithMassless:
      result = massDifferenceAtMass(p0[0])
    else:
      result = massDifferenceAtMass(p0[0], cmpMass = some(m_a))

  var opt = newNloptOpt[MassFit](LN_COBYLA, 1, bounds = @[(1e-4, Inf)])
  # hand the function to fit as well as the data object we need in it
  var varStr = newVarStruct(findMin, MassFit())
  opt.setFunction(varStr)
  opt.xtol_rel = 1e-8
  opt.ftol_rel = 1e-8
  let (params, minVal) = opt.optimize(@[m_a])
  result = params[0]
  opt.destroy()

when true:
  let mstep1 = findMassAtFwhm(babyIaxoVacuumMassLimit)
  let mstep2 = findMassAtFwhm(mstep1, cmpWithMassless = false)
```

Finally generate the plots from the calculation:

```
when true:
  echo "\n\n------------------------------\n\n"
  proc toPressureDiff(m1, m2: float, T = 4.2): float
  let pdiffVacStep1 = toPressureDiff(mstep1, babyIaxoVacuumMassLimit)
  gasStepsPlot((name: "1st He step", m: mstep1),
               massRhs = (name: "", m: 0.0),
               title = &"First buffer gas step; match at FWHM, ΔP = {pdiffVacStep1:.4f} mbar (@ 4.2
  let pdiffFirst2Steps = toPressureDiff(mstep1, mstep2)
  gasStepsPlot((name: "2nd He step", m: mstep2),
               (name: "1st He step", m: mstep1),
               title = &"First two He steps, each match at FWHM, ΔP = {pdiffFirst2Steps:.4f} mbar (@
```

The first gas buffer step is shown in figure 15. We can see that we get the expected matching of the first coherent curve with the dropping vacuum curve.

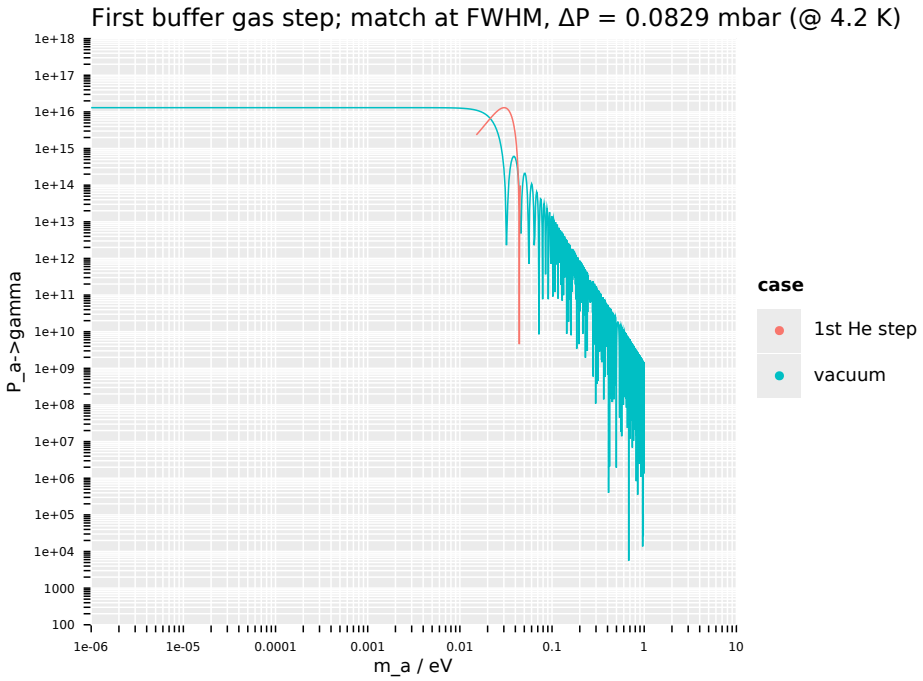

Figure 15: Plot of the vacuum conversion probability depending on the axion mass for the vacuum case and the first pressure step with ^{4}He buffer gas, such that the two produce a "best matching", i.e. the sensitivity does not drop below the value at the FWHM (= by a factor of 2 in probability). At 4.2 K the pressure is 0.1927 mbar.

The first two helium steps are then shown in fig. 16. Again we see that the matching of the two coherent curves is correct.

At 4.2 K the first pressure step is at 0.1927 mbar and the second at 0.3808 mbar, meaning a difference in pressure of 0.1881 mbar.

Now let's look at the same for the steps between two pressures at 1 bar equivalent and 3 bar equivalent. For this calculate the mass that corresponds to these and take values for `findMassAtFwhm`.

To better present the information, let's write a helper proc, which converts a difference in masses to a difference in pressures:

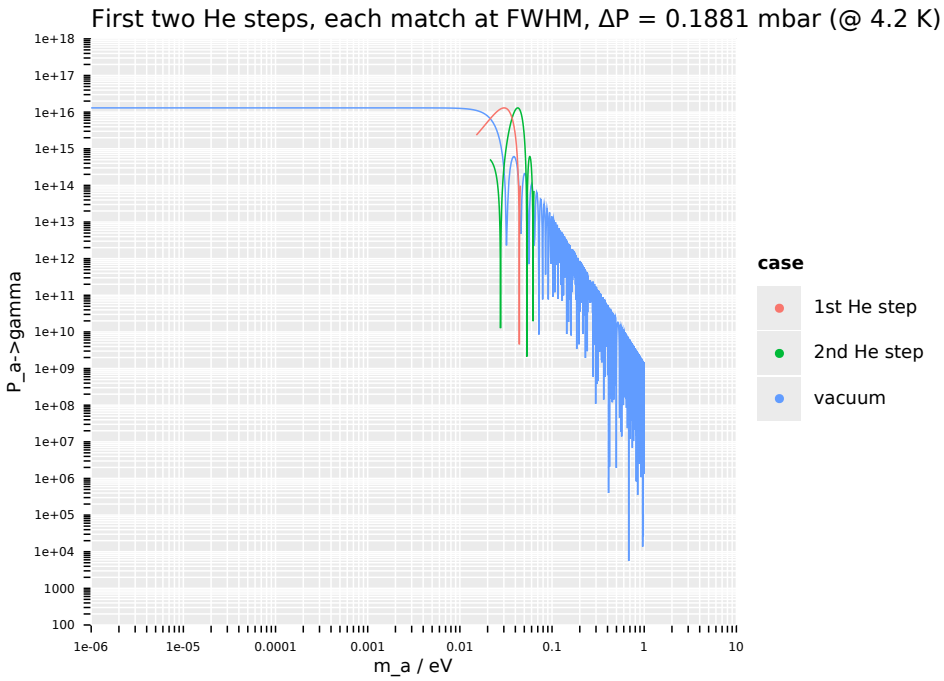

Figure 16: Plot of the vacuum conversion probability depending on the axion mass for the vacuum case and the first two pressure steps with ^{4}He buffer gas. We can see the mass difference achieved for the first two steps. At 4.2 K the first pressure step is at 0.1927 mbar and the second at 0.3808 mbar.

```nim
proc toPressureDiff(m1, m2: float, T = 4.2): float =
  ## assuming two masses `m1` and `m2` in `eV`, converts the
  ## mass difference to a corresponding pressure difference
  ## at temperature `T`
  result = abs(pressureGivenEffPhotonMass(m2, T = T) -
               pressureGivenEffPhotonMass(m1, T = T))
```

```nim
# 1 bar
when true:
  let mstep1Bar = findMassAtFwhm(m_gamma_1bar, cmpWithMassless = false)
  let pdiff1bar = toPressureDiff(m_gamma_1bar, mstep1Bar)
  gasStepsPlot((name: "Step 1", m: m_gamma_1_bar),
               (name: "Step 2", m: mstep1Bar),
               title = &"Two steps at 1 bar (@ 293 K) pressure, ΔP = {pdiff1bar:.4f} mbar (@ 4.2 K)"
               filterMass = some((low: 0.22, high: 0.2999)))
  #
  ### 3 bar
  let mstep3Bar = findMassAtFwhm(m_gamma_3bar, cmpWithMassless = false)
  let pdiff3bar = toPressureDiff(m_gamma_3bar, mstep3Bar)
  gasStepsPlot((name: "Step 1", m: m_gamma_3_bar),
               (name: "Step 2", m: mstep3Bar),
               title = &"Two steps at 3 bar (@ 293 K) pressure, ΔP = {pdiff3bar:.4f} mbar (@ 4.2 K)"
               filterMass = some((low: 0.42, high: 0.48)))
```

The step difference near 1 bar at $T = 293.15\,\mathrm{K}$ is shown in fig. 17. Here the pressure difference at cryo temperature 4.2 K is only

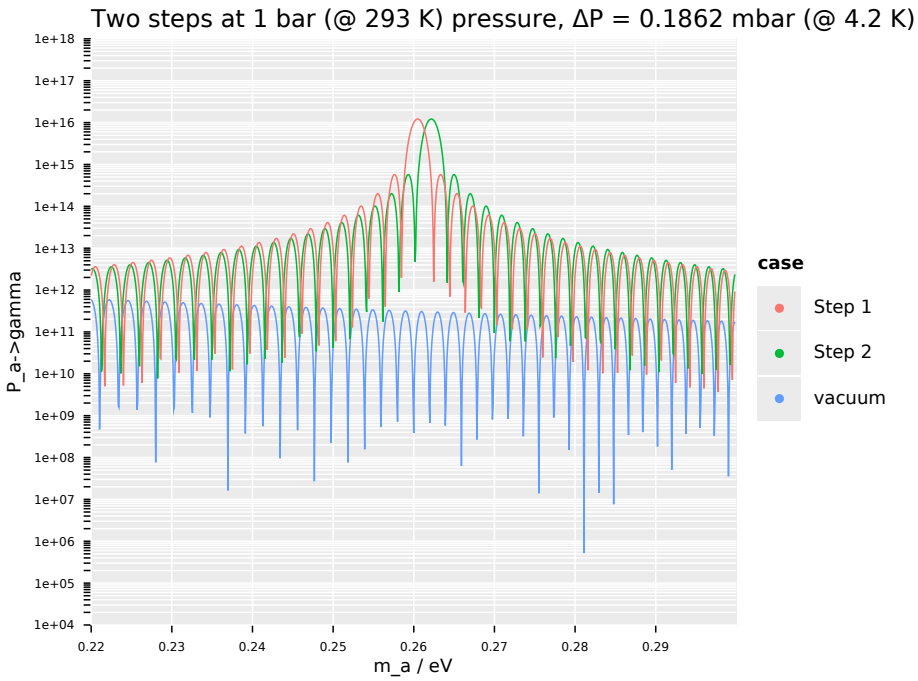

Figure 17: Plot of the vacuum conversion probability depending on the axion mass for the vacuum case and two pressure steps with ^{4}He buffer gas at 1 bar at $T = 4.2\,\mathrm{K}$.

And in fig. 18 we see the same plot near the 3 bar pressure equivalent.

Finally we still want to reproduce fig. 3 depending on the density instead of a pressure.

```
proc m_a_vs_density(pstart, pstop: float) =
  let pressures = logspace(pstart.log10, pstop.log10, 1000)
  let densities = pressures.mapIt(density(it, 4.2))
  let masses = pressures.mapIt(babyIaxoEffMass(it)) # corresponding masses
  # convert both seqs to a dataframe
  let ref1Bar = density(1000, 293.15)
  let df1bar = seqsToDf({" / g/cm^3" : @[ref1Bar, ref1Bar], "m_a / eV" : @[1e-2, 1.0]})
  let ref3Bar = density(3000, 293.15)
  let df3bar = seqsToDf({" / g/cm^3" : @[ref3Bar, ref3Bar], "m_a / eV" : @[1e-2, 1.0]})
  let refVacLimit = density(pressureGivenEffPhotonMass(babyIaxoVacuumMassLimit))
  let dfVacLimit = seqsToDf({" / g/cm^3" : @[refVacLimit, refVacLimit], "m_a / eV" : @[1e-2  1.0]})
  let df = seqsToDf({" / g/cm^3" : densities, "m_a / eV" : masses})
  let dfComb = bind_rows([("ma vs  ", df),
                          ("1 bar @ 293 K", df1bar),
                          ("3 bar @ 293 K", df3bar),
                          ("Vacuum limit", dfVacLimit)],
                         id = "Legend")
  ggplot(dfComb, aes(" / g/cm^3", "m_a / eV", color = "Legend")) +
    geom_line() +
    scale_x_log10() +
    scale_y_log10() +
    ggtitle("Sensitive axion mass in eV depending on helium density in g / cm^3") +
    ggsave("m_a_vs_density.pdf")

when true:
```

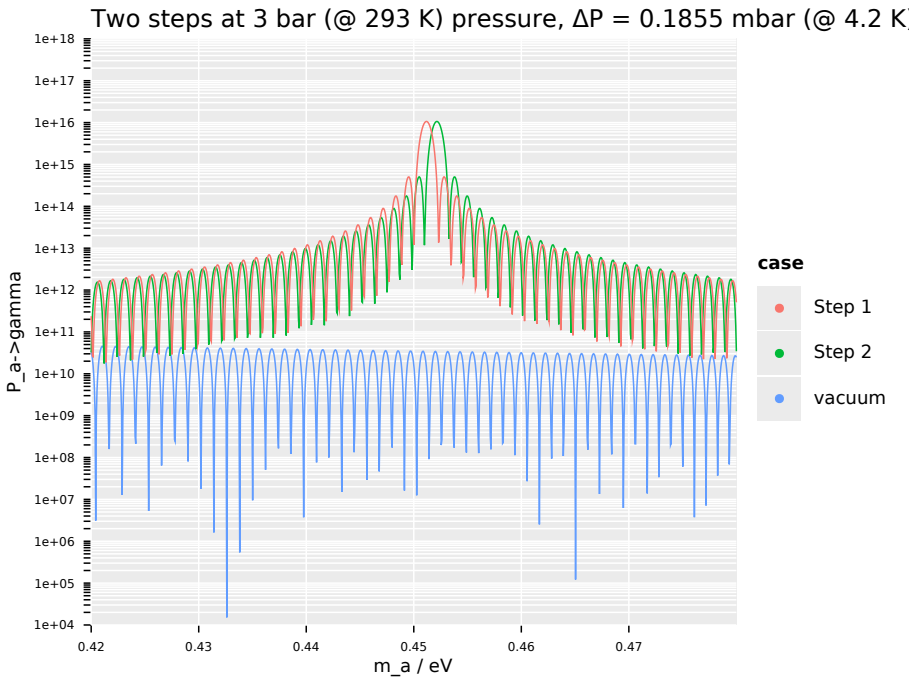

Figure 18: Plot of the vacuum conversion probability depending on the axion mass for the vacuum case and two pressure steps with ^{4}He buffer gas at 3 bar at $T = 4.2\,\mathrm{K}$.

```
m_a_vs_density(pressureGivenEffPhotonMass(babyIaxoVacuumMassLimit) * 0.9,
               pressureGivenEffPhotonMass(mstep3Bar) * 1.1)
```

The final plot then is the dependency of the axion mass m_a against the ^{4}He density in the magnet. This is shown in fig. 19.

3 Notes

3.1 TODO Evaluate if this m_γ results in the same value as our fn

According to Theodoros' PhD thesis: `https://cds.cern.ch/record/1607071/files/CERN-THESIS-2012-349.pdf` (page 186) the effective photon mass can also be written as:

$$m_\gamma = 28.77\sqrt{\frac{Z}{A}\rho} \tag{58}$$

where Z is the atomic number, ρ the density of the gas and A the atomic mass of the buffer gas.

"Distinguishing axion models at IAXO, Jaeckel & Thompson:" `https://iopscience.iop.org/article/10.1088/1475-7516/2019/03/039/meta` mentions another approximation for m_γ, namely:

$$m_\gamma \approx \sqrt{0.02\frac{p(\mathrm{mbar})}{T(\mathrm{K})}}\,\mathrm{eV} \tag{59}$$

which seems to be quite a bit closer to our assumption than Theodors'.

J. Gallan's PhD thesis seems to be the original source for the equation, `https://arxiv.org/pdf/1102.1406.pdf`.

He arrives at it from the relation that the electron number density can be written as:

$$n_e = Z\frac{N_A}{M}\rho \tag{60}$$

I should take a pen and paper and write it down. Continue on with the conversion probability.

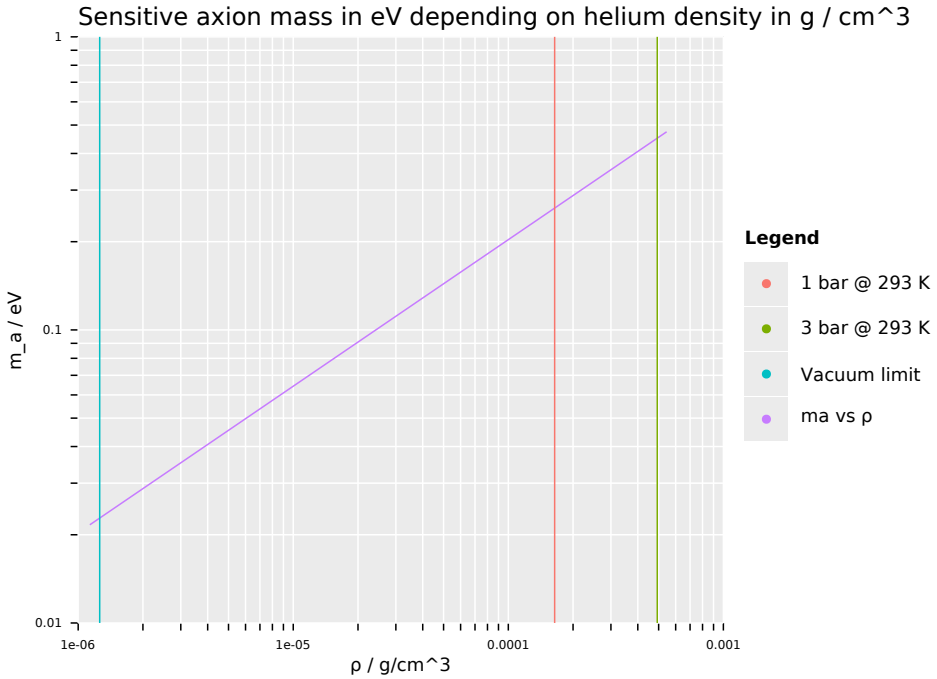

Figure 19: Sensitive axion mass m_a depending on the density of ^{4}He buffer gas in the magnet. Also added are the lines corresponding to the vacuum limit and the 1 bar and 3 bar lines (at 4.2 K).

3.2 Momentum transfer units

Momentum transfer has units of momentum.

Momentum has natural units of:

3.3 Attenuation length, momentum transfer and length

This section was to attempt to fix the plot in fig. 20.

So in principle: $\exp -\Gamma L$ has to be unit 1, i.e. Γ has to be unit of inverse length (which is by definition true of course, since it's defined as an inverse length). However, in the units we actually use for our numbers, that ain't quite the case. We use SI units for $L = 10\,\text{m}$ and have Γ from:

$$\Gamma = \rho \left(\frac{\mu}{\rho} \right) \tag{61}$$

Now, we use ρ in units of g/cm^3, while the NIST data supplies μ/ρ in cm^2/g. What does this mean? It means that

$$[\Gamma] = \left[\text{g/cm}^3\right] \left[\text{cm}^2/\text{g}\right] \tag{62}$$

$$[\Gamma] = \left[\text{cm}^{-1}\right] \tag{63}$$

ref eq 14.

This means two things for us. One: For all products of $\Gamma \cdot L$ we have to apply a correction factor of 100 to convert from cm to m, but maybe more importantly, we have to consider what this means for the denominator of the second term of the conversion probability, namely:

$$q^2 + \Gamma^2/4 \tag{64}$$

Do we have to fix one of these?

The same about the products holds also for the argument to the cosine:

$$\cos(q \cdot L) \tag{65}$$

Since we have by definition q in eV and

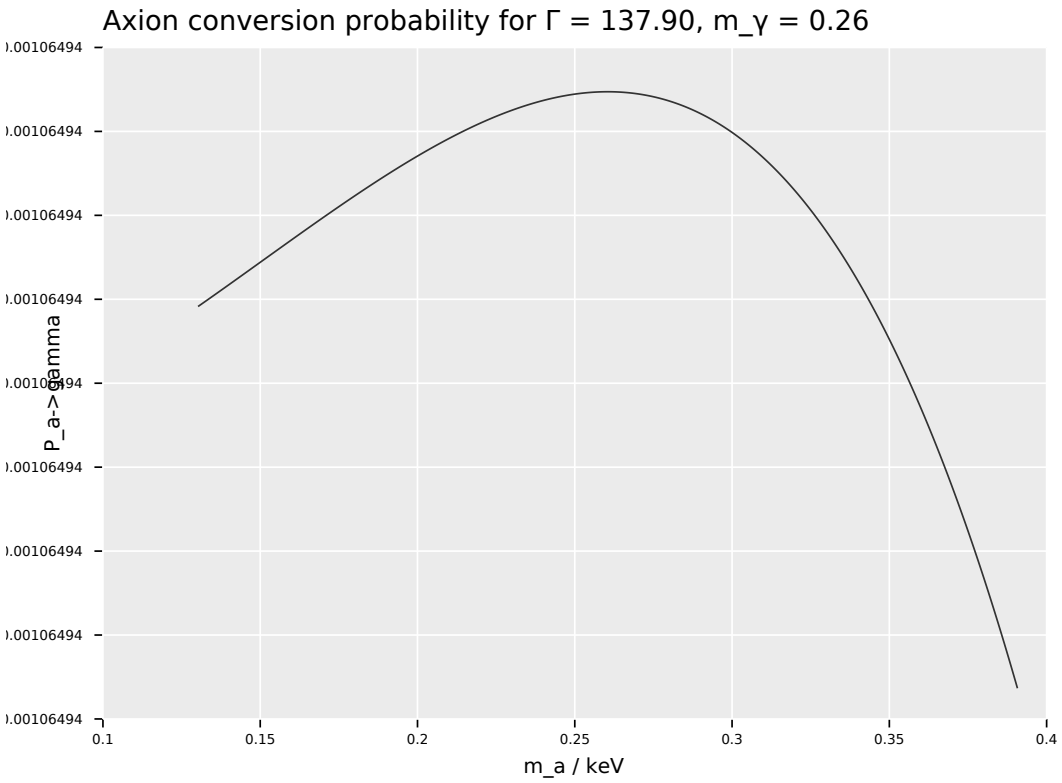

Figure 20: Example of the axion conversion probability at a pressure of $P = 14.3345\,\text{mbar}$ (corresponds to 1 bar at room temperature). This should reproduce fig. 5. That apparently does not work yet.

4 Appendix appendix

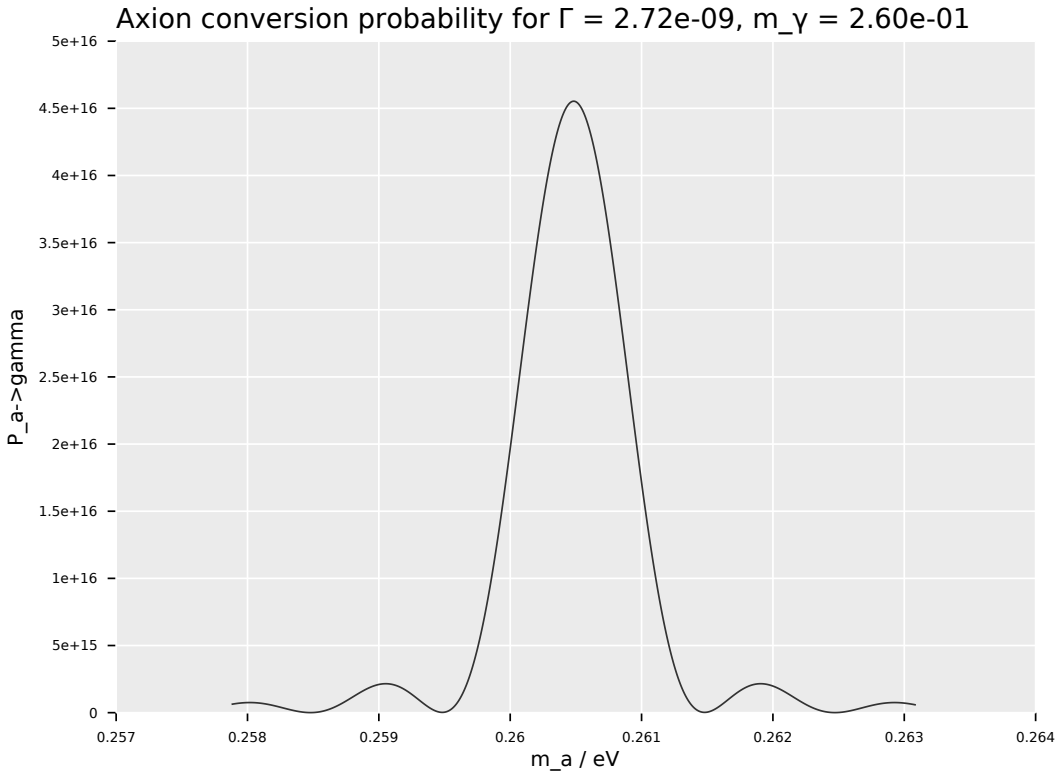

Figure 21: Example of the axion conversion probability at a pressure of $P = 14.3345\,\text{mbar}$ (corresponds to 1 bar at room temperature) for the full IAXO length 20 m. This is equivalent to the plot of fig. 5, although we see the influence of the cos term a lot more.

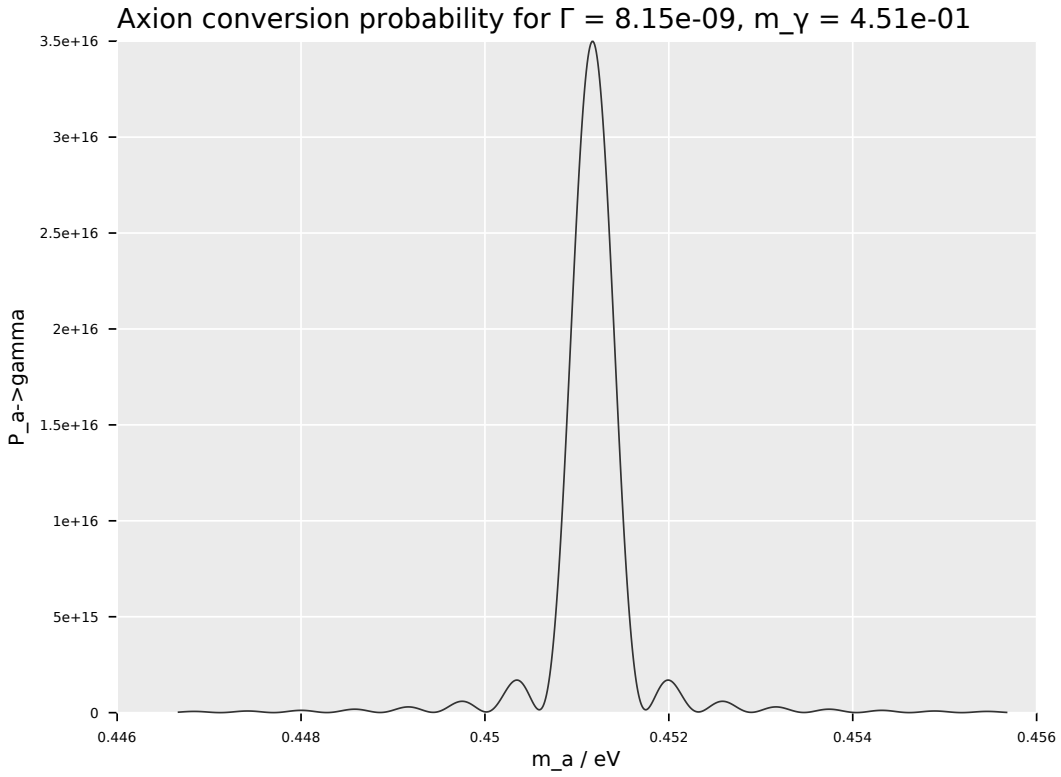

Figure 22: Example of the axion conversion probability at a pressure of $P = 43.0034\,\text{mbar}$ (corresponds to 3 bar at room temperature) and a magnet length of 20 m. This plot should be exactly the equivalent to the 3 bar plot of the IAXO gas phase study paper.