1-gVirtualXRay_vs_XCOM-attenuation_coefficients

March 2, 2022

[1]:
```
from IPython.display import display
from IPython.display import Image
```

Main contributor: F. P. Vidal

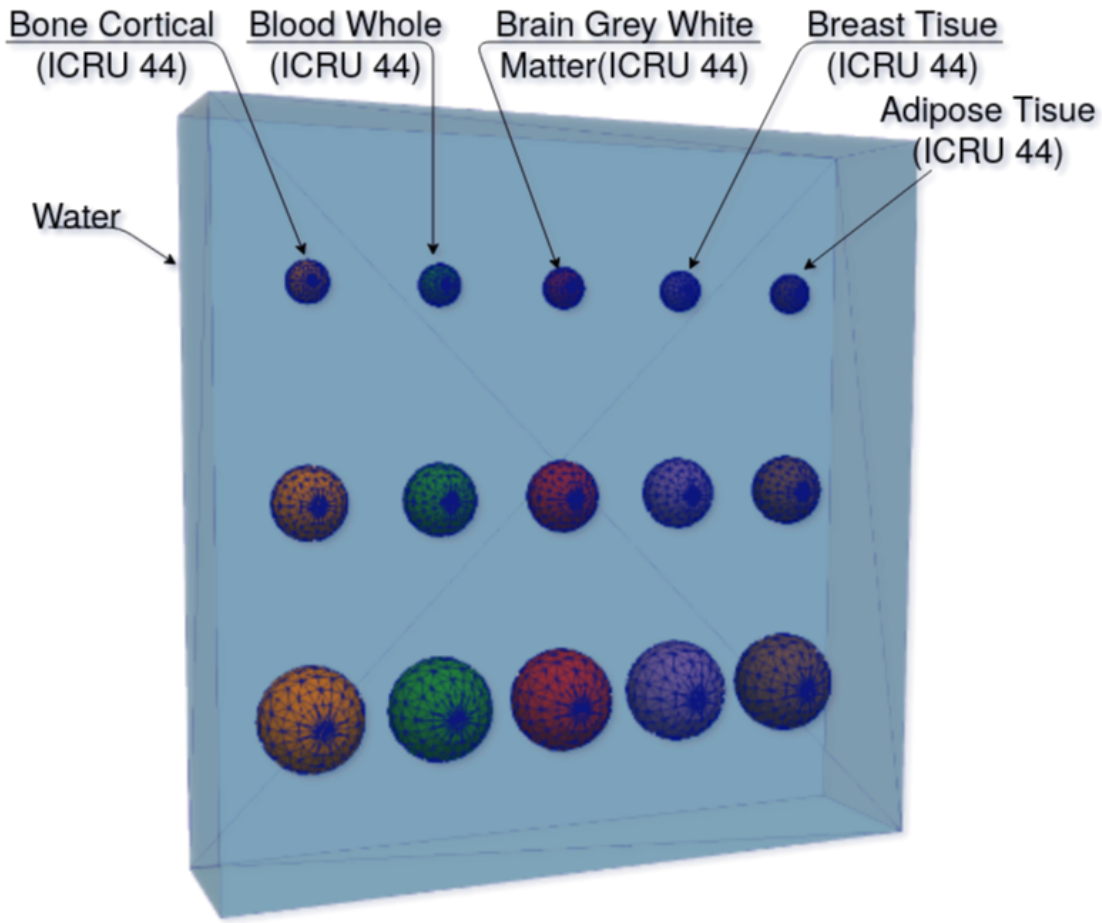
Purpose: In this notebook, we aim to demonstrate that gVirtualXRay is using mass attenuation coefficients comparable with those from NIST's XCOM Photon Cross Sections Database.

Material and Methods: To generate an X-ray image with an analytic simulation, we must solve the Beer-Lambert law, which relies on attenuation coefficients. The first step in validating gVirtualXRay is to ascertain that the attenuation coefficients used in gVirtualXRay are in perfect agreement with those found in the literature. Ground truth data is provided in the NIST Standard Reference Database 126.

The sample is made of a 70x70x15mm box of water, in which 5 columns of 3 spheres of different radii (2, 3.5, and 5mm) have been inserted. A given material is associated to the spheres of each column (bone (cortical), blood (whole), brain (grey/white matter), breast tissue, and adipose tissue). The columns are ordered in decreasing density. We use the definitions of tissue substitutes provided in the ICRU Report 44 by the International Commission on Radiation Units and Measurements. The material composition is available at `https://physics.nist.gov/PhysRefData/XrayMassCoef/tab2.html`.

[2]:
```
Image(filename="doc/sample.png", width=400)
```

[2]:



Results: The correlation coefficient between the attenuation coefficients computed with gVirtualXRay and those provided in NIST's *XCOM: Photon Cross Sections Database* is 100% for bone (cortical), blood (whole), brain (grey/white matter), breast tissue, and adipose tissue. It demonstrates that the photon cross section calculations performed in gVirtualXRay are accurate.

1 Import packages

[3]:
```
%matplotlib inline

import os # Locate files

import numpy as np # Who does not use Numpy?
import pandas as pd # Load/Write CSV files

import matplotlib
import matplotlib.pyplot as plt # Plotting
```

```
import matplotlib.image as mpimg # Read file

font = {'family' : 'serif',
        #'weight' : 'bold',
         'size'   : 22
       }
matplotlib.rc('font', **font)
matplotlib.rc('text', usetex=True)

from scipy.stats import pearsonr # Compute the correlatio coefficient
from sklearn.metrics import mean_absolute_percentage_error as mape

import viewscad # Use OpenSCAD to create STL files
import gvxrPython3 as gvxr # Simulate X-ray images

import json2gvxr # Set gVirtualXRay and the simulation up
```

```
SimpleGVXR 1.0.1 (2022-02-22T14:00:25) [Compiler: GNU g++] on Linux
gVirtualXRay core library (gvxr) 1.1.5 (2022-02-22T14:00:25) [Compiler: GNU g++]
on Linux
```

2 Setting up gVirtualXRay

Before simulating an X-ray image using gVirtualXRay, we must create an OpenGL context.

[4]:
```
json2gvxr.initGVXR("notebook-1.json", "EGL")
```

```
Create an OpenGL context: 800x450
Wed Mar  2 12:12:45 2022 ---- Create window gvxrStatus: Create window
Wed Mar  2 12:12:45 2022 ---- EGL version: Wed Mar  2 12:12:45 2022 ---- OpenGL
version supported by this platform OpenGL renderer:   GeForce RTX 2080
Ti/PCIe/SSE2
OpenGL version:    4.5.0 NVIDIA 455.45.01
OpenGL vender:     NVIDIA Corporation
Wed Mar  2 12:12:45 2022 ---- Use OpenGL 4.5.0 0 500 500

0

0 0 800 450


1.5
4.5.0 NVIDIA 455.45.01
```

3 Creating the test object

We now create CAD models using OpenSCAD and extract the corresponding STL files.

[5]:
```
openscad_make_spheres_str = """

module make_column_of(sphere_radius, height, count)
{
    step = height / (count - 1);
    for (a = [0 : count - 1]) {
        offset = -height / 2 + step * a ;
        translate([0, offset, 0])
            sphere(sphere_radius[a], $fn=25);
    }
}

module make_row_of(radius, count, id)
{
    step = radius / (count - 1);
    for (a = [0 : count - 1]) {
        if (id == -1 || id == a) {
            offset = -radius / 2 + step * a ;
            translate([offset, 0, 0])
                children();
        }
    }
}

module make_spheres(sphere_radius, ring_radius, ring_count, column_height,␣
 ↪column_count, id = -1)
{
    make_row_of(radius = ring_radius, count = ring_count, id = id)
        make_column_of(sphere_radius, height = column_height, count =␣
 ↪column_count);
}
"""
```

The matrix

[6]:
```
openscad_matrix_str = """

color("red")
    difference() {
        scale([70, 70, 15])
            cube(1, center = true);
        make_spheres([2, 3.5, 5], 50, 5, 40, 3, -1);
}
```

```
"""
```

[7]:
```
fname = 'CAD_models/matrix.stl'
if not os.path.isfile(fname):

    r = viewscad.Renderer()
    r.render(openscad_matrix_str + openscad_make_spheres_str,␣
 ↪outfile='CAD_models/matrix.stl')
```

[8]:
```
openscad_cube_str = """

color("red")
    scale([70, 70, 15])
        cube(1, center = true);

"""
```

[9]:
```
fname = 'CAD_models/cube.stl'
if not os.path.isfile(fname):

    r = viewscad.Renderer()
    r.render(openscad_cube_str, outfile=fname)
```

The spheres

[10]:
```
openscad_col_str_set = []

for i in range(5):
    openscad_col_str_set.append("""
    color("blue")
        make_spheres([2, 3.5, 5], 50, 5, 40, 3, """ + str(i) + ");")

    fname = 'CAD_models/col_' + str(i) + '.stl'
    if not os.path.isfile(fname):

        r = viewscad.Renderer()
        r.render(openscad_col_str_set[-1] + openscad_make_spheres_str,␣
 ↪outfile=fname)
```

Load the samples. `verbose=2` is used to print the material database for Gate. To disable it, use `verbose=0` or `verbose=1`.

[11]:
```
json2gvxr.initSamples(verbose=2)
```

```
Load the 3D data

Bone_Cortical_ICRU_44: d=1.92 g/cm3 ; n=9 ; state=solid
```

```
        +el: name=Hydrogen ; f=0.034
        +el: name=Carbon ; f=0.155
        +el: name=Nitrogen ; f=0.042
        +el: name=Oxygen ; f=0.435
        +el: name=Sodium ; f=0.001
        +el: name=Magnesium ; f=0.002
        +el: name=Phosphor ; f=0.103
        +el: name=Sulfur ; f=0.003
        +el: name=Calcium ; f=0.225

Blood_Whole_ICRU_44: d=1.06 g/cm3 ; n=10 ; state=solid
        +el: name=Hydrogen ; f=0.102
        +el: name=Carbon ; f=0.11
        +el: name=Nitrogen ; f=0.033
        +el: name=Oxygen ; f=0.745
        +el: name=Sodium ; f=0.001
        +el: name=Phosphor ; f=0.001
        +el: name=Sulfur ; f=0.002
        +el: name=Chlorine ; f=0.003
        +el: name=Potassium ; f=0.002
        +el: name=Iron ; f=0.001

Brain_Grey_White_Matter_ICRU_44: d=1.04 g/cm3 ; n=9 ; state=solid
        +el: name=Hydrogen ; f=0.107
        +el: name=Carbon ; f=0.145
        +el: name=Nitrogen ; f=0.022
        +el: name=Oxygen ; f=0.712
        +el: name=Sodium ; f=0.002
        +el: name=Phosphor ; f=0.004
        +el: name=Sulfur ; f=0.002
        +el: name=Chlorine ; f=0.003
        +el: name=Potassium ; f=0.003

Breast_Tissue_ICRU_44: d=1.02 g/cm3 ; n=8 ; state=solid
        +el: name=Hydrogen ; f=0.106
        +el: name=Carbon ; f=0.332
        +el: name=Nitrogen ; f=0.03
        +el: name=Oxygen ; f=0.527
        +el: name=Sodium ; f=0.001
        +el: name=Phosphor ; f=0.001
        +el: name=Sulfur ; f=0.002
        +el: name=Chlorine ; f=0.001

Adipose_Tissue_ICRU_44: d=0.95 g/cm3 ; n=7 ; state=solid
        +el: name=Hydrogen ; f=0.114
        +el: name=Carbon ; f=0.598
        +el: name=Nitrogen ; f=0.007
        +el: name=Oxygen ; f=0.278
```

```
        +el: name=Sodium ; f=0.001
        +el: name=Sulfur ; f=0.001
        +el: name=Chlorine ; f=0.001
```

```
CAD_models/col_0.stl    nb_faces:       1938    nb_vertices:    5814
bounding_box (in cm):   (-2.99606, -2.19961, -0.496354) (-2, 2.49901, 0.496354)
CAD_models/col_1.stl    nb_faces:       1938    nb_vertices:    5814
bounding_box (in cm):   (-1.74606, -2.19961, -0.496354) (-0.75, 2.49901,
0.496354)
CAD_models/col_2.stl    nb_faces:       1938    nb_vertices:    5814
bounding_box (in cm):   (-0.496057, -2.19961, -0.496354)        (0.5, 2.49901,
0.496354)
CAD_models/col_3.stl    nb_faces:       1938    nb_vertices:    5814
bounding_box (in cm):   (0.753943, -2.19961, -0.496354) (1.75, 2.49901,
0.496354)
CAD_models/col_4.stl    nb_faces:       1938    nb_vertices:    5814
bounding_box (in cm):   (2.00394, -2.19961, -0.496354)  (3, 2.49901, 0.496354)
CAD_models/cube.stl     nb_faces:       12      nb_vertices:    36
bounding_box (in cm):   (-3.5, -3.5, -0.75)     (3.5, 3.5, 0.75)
```

4 Mass attenuation coefficients

Before computing an X-ray image, we can check that the mass attenuation coefficients are accurate. We downloaded tabulated data from the XCOM database and compare the values with the ones used in gVirtualXRay.

4.1 Chemical elements

We do it for relevant chemical elements from https://physics.nist.gov/PhysRefData/XrayMassCoef/tab3.html:

- Carbon
- Chlorine
- Hydrogen
- Iron
- Magnesium
- Nitrogen
- Oxygen
- Phosphorus
- Potassium
- Sodium
- Sulfur

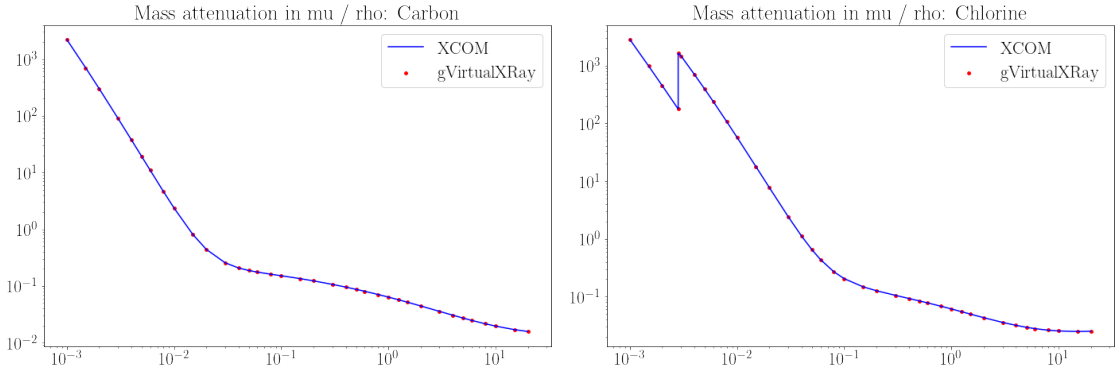
[12]:
```
elements = ["Carbon", "Chlorine", "Hydrogen", "Iron", "Magnesium", "Nitrogen",␣
↪"Oxygen", "Potassium", "Sodium", "Sulfur"] #  "Phosphorus"]#, ]
```

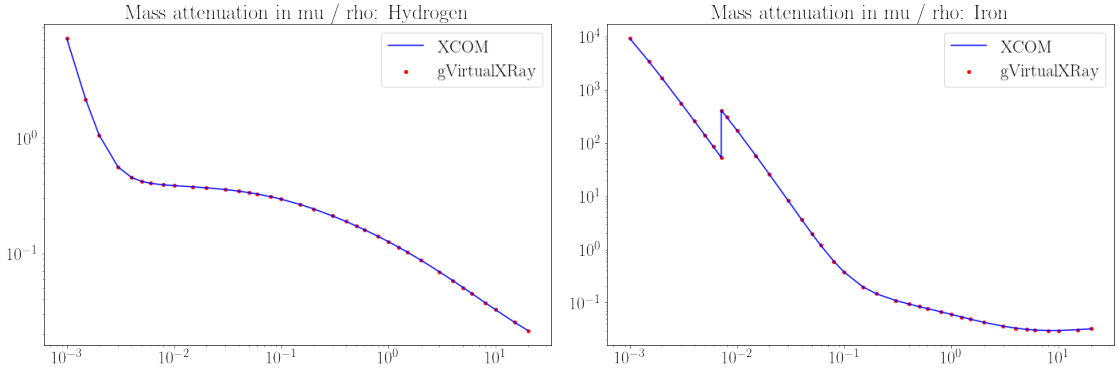
```
i = 1

plt.figure(figsize= (20,40))

cols = ["Element", "Pearson correlation (in %)", "MAPE (in %)"]
rows = []

print("Element    &    Pearson correlation    &    MAPE \\\\")
print("\\hline")

for element in elements:
    if os.path.isfile("XCOM_data/" + element + ".csv"):
        df = pd.read_csv("XCOM_data/" + element + ".csv")

        gvxr_mu_rho = []
        gvxr_energy = []
        for energy_id in range(len(df["Photon in MeV"])):
            if energy_id == len(df["Photon in MeV"]) - 1 or energy_id == 0:
                energy = df["Photon in MeV"][energy_id]
            elif df["Photon in MeV"][energy_id] == df["Photon in␣
↪MeV"][energy_id + 1]:
                energy = df["Photon in MeV"][energy_id] - df["Photon in␣
↪MeV"][energy_id] * 1e-3
            elif df["Photon in MeV"][energy_id] == df["Photon in␣
↪MeV"][energy_id - 1]:
                energy = df["Photon in MeV"][energy_id] + df["Photon in␣
↪MeV"][energy_id] * 1e-3
            else:
                energy = df["Photon in MeV"][energy_id]

            gvxr_mu_rho.append(gvxr.getMassAttenuationFromElement(element,␣
↪energy, "MeV"))
            gvxr_energy.append(energy)

        df["Mass attenuation coefficient (gVirtualXRay)"] = gvxr_mu_rho
        df["|XCOM - gVirtualXRay| / XCOM (in %)"] = np.abs(np.array(100 *␣
↪(df["Mass attenuation coefficient"] - gvxr_mu_rho) / df["Mass attenuation␣
↪coefficient"])).astype(int)

        if df["|XCOM - gVirtualXRay| / XCOM (in %)"].mean() > 1.0:
            print("\tWARNING:")
            print("\t\tAverage:", df["|XCOM - gVirtualXRay| / XCOM (in %)"].
↪mean())
            print("\t\tMax:", df["|XCOM - gVirtualXRay| / XCOM (in %)"].abs().
↪max())
```

```
        df.to_csv("gVirtualXRay_output_data/" + element + "-validation.csv",␣
 ↪index=False )

        plt.subplot(6, 2, i)
        plt.title("Mass attenuation in mu / rho:  " + element)
        plt.plot(df["Photon in MeV"], df["Mass attenuation coefficient"],␣
 ↪label="XCOM", color="b")
        plt.scatter(gvxr_energy, gvxr_mu_rho, label="gVirtualXRay", s=15,␣
 ↪color="r")
        plt.legend()
        plt.xscale('log')
        plt.yscale('log')

        corr, _ = pearsonr(df["Mass attenuation coefficient"], gvxr_mu_rho)
        MAPE = mape(df["Mass attenuation coefficient"], gvxr_mu_rho)

        row = [element, 100.0 * corr, 100.0 * MAPE]
        rows.append(row)

        percent = "%"
        print(element + "    &    %.2f" % (100 * corr) + "\\" +  percent + "    ␣
 ↪&    %.2f" % (100 * MAPE) + "\\" +  percent + " \\\\")

        i = i + 1

df = pd.DataFrame(columns=cols, data=rows)
print("Overall" + "    &    %.2f" % (df["Pearson correlation (in %)"].mean()) +␣
 ↪"\\" + percent + "$\\pm$%.2f" % (df["Pearson correlation (in %)"].std()) +
      "    &    %.2f" % (df["MAPE (in %)"].mean()) + "\\" + percent + "$\\pm$%.
 ↪2f" % (df["MAPE (in %)"].std()) + " \\\\")

plt.tight_layout()

plt.savefig('plots/mass_attenuation_coefficients_from_elements.pdf')
plt.savefig('plots/mass_attenuation_coefficients_from_elements.png')
```

```
Element    &    Pearson correlation    &    MAPE \\
\hline
Carbon    &    100.00\%    &    0.06\% \\
Chlorine    &    100.00\%    &    0.07\% \\
Hydrogen    &    100.00\%    &    0.06\% \\
Iron    &    100.00\%    &    0.07\% \\
Magnesium    &    100.00\%    &    0.12\% \\
Nitrogen    &    100.00\%    &    0.06\% \\
Oxygen    &    100.00\%    &    0.06\% \\
Potassium    &    100.00\%    &    0.06\% \\
Sodium    &    100.00\%    &    0.07\% \\
```

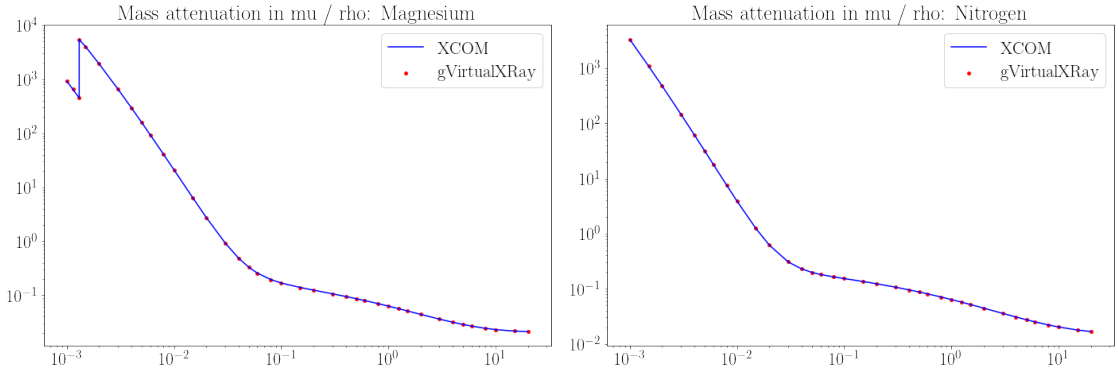
```
Sulfur    &    100.00\%    &    0.08\% \\
Overall    &    100.00\%$\pm$0.00    &    0.07\%$\pm$0.02 \\
```


Mass attenuation in mu / rho: Carbon
XCOM
gVirtualXRay

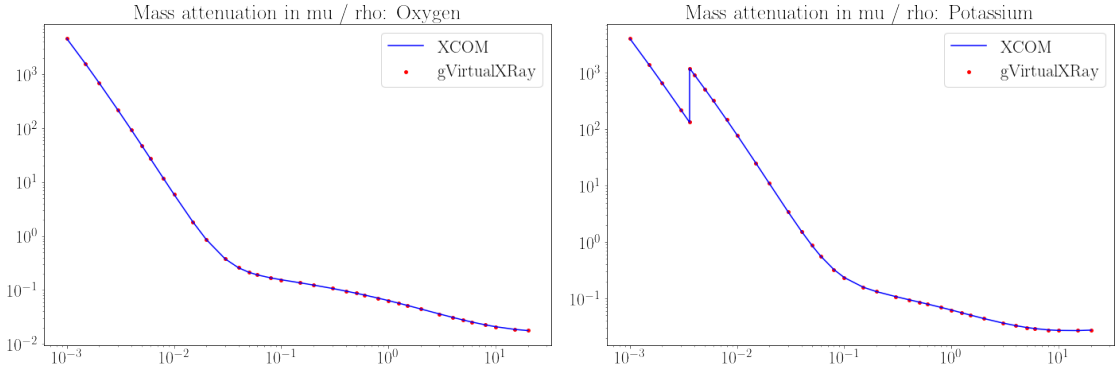

Mass attenuation in mu / rho: Chlorine
XCOM
gVirtualXRay

Mass attenuation in mu / rho: Hydrogen
XCOM
gVirtualXRay

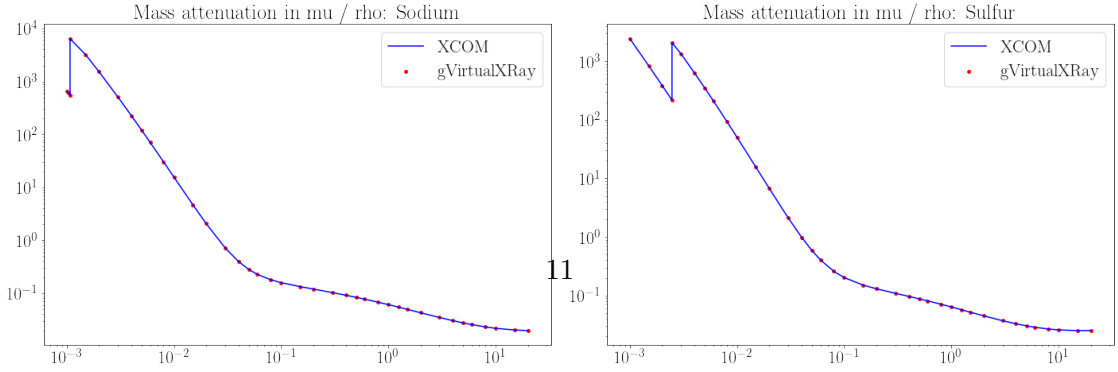
Mass attenuation in mu / rho: Iron
XCOM
gVirtualXRay


Mass attenuation in mu / rho: Magnesium
XCOM
gVirtualXRay

Mass attenuation in mu / rho: Nitrogen
XCOM
gVirtualXRay


Mass attenuation in mu / rho: Oxygen
XCOM
gVirtualXRay

Mass attenuation in mu / rho: Potassium
XCOM
gVirtualXRay


Mass attenuation in mu / rho: Sodium
XCOM
gVirtualXRay

Mass attenuation in mu / rho: Sulfur
XCOM
gVirtualXRay

[13]:
```
print(df)
```

```
     Element  Pearson correlation (in %)  MAPE (in %)
0     Carbon                   99.999999     0.060139
1   Chlorine                   99.999971     0.067245
2   Hydrogen                   99.999999     0.061734
3       Iron                   99.999999     0.073480
4  Magnesium                   99.999698     0.121077
5   Nitrogen                   99.999998     0.056702
6     Oxygen                  100.000000     0.058795
7  Potassium                   99.999978     0.064172
8     Sodium                   99.999958     0.073294
9     Sulfur                   99.999940     0.084094
```

[14]:
```
print("The smallest Pearsons correlation is %.2f" % df["Pearson correlation (in␣
↪%)"].min() + "%")
print("The largest MAPE is %.2f" % df["MAPE (in %)"].max() + "%")
```

```
The smallest Pearsons correlation is 100.00%
The largest MAPE is 0.12%
```

The smallest Pearsons correlation is almost 100%, it is as good as it can get. The largest MAPE is about 0.1%, which is an extremely small error. When we plot the data generated with gVirtualXRay against NIST's, we cannot tell the difference.

4.2 ICRU-44 materials

We can now try something more complex, some clinically relevant materials from https://physics.nist.gov/PhysRefData/XrayMassCoef/tab4.html:

- Adipose Tissue (ICRU-44)
- Blood, Whole (ICRU-44)
- Bone, Cortical (ICRU-44)
- Brain, Grey/White Matter (ICRU-44)
- Breast Tissue (ICRU-44)
- Water, Liquid

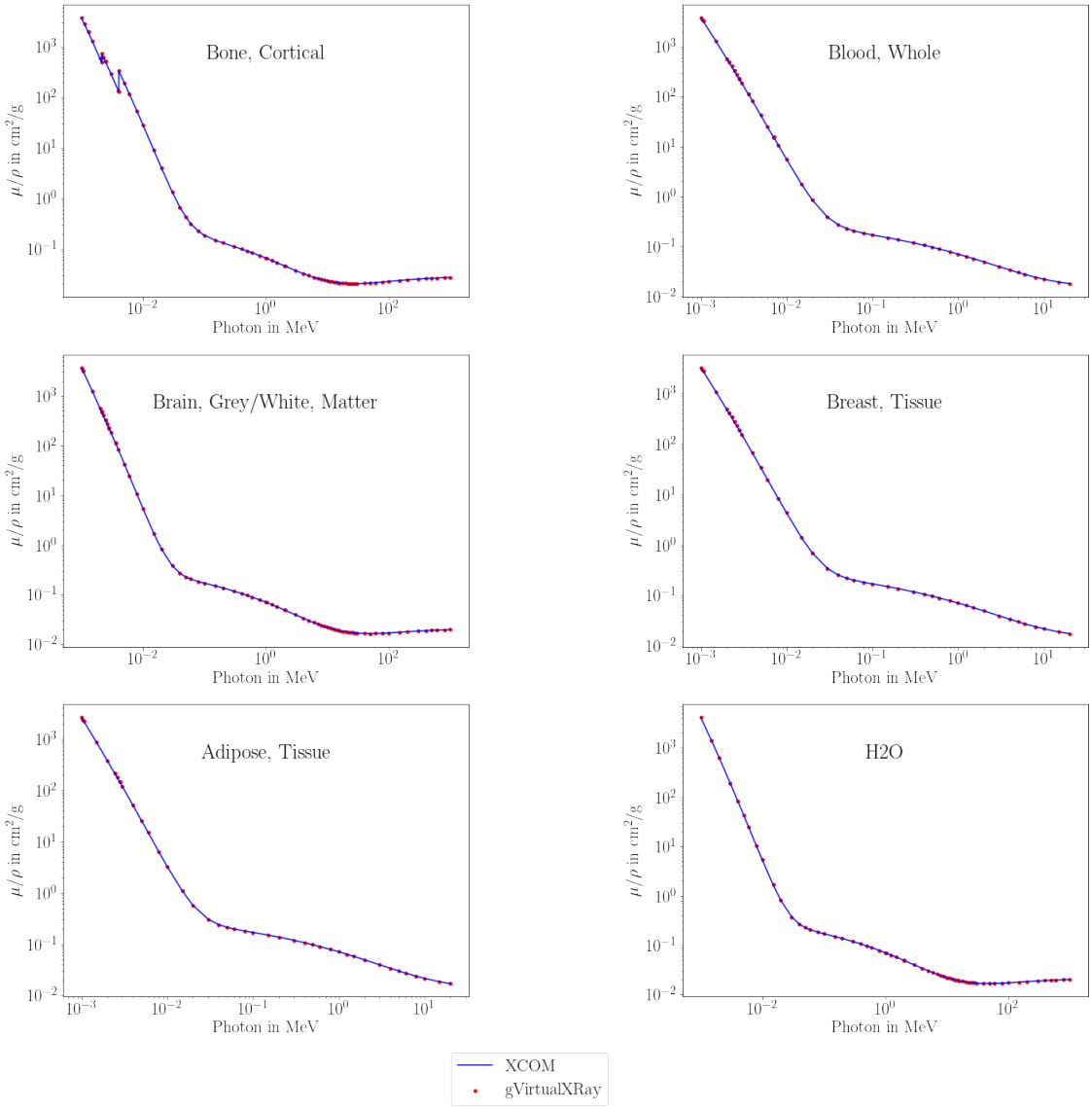
[15]:
```
i = 1

plt.figure(figsize= (20,20))

min_pearsons_correlation = 100.0 # initialise with the largest possible value,␣
↪+100%
max_MAPE = -100.0 # initialise with the smallest possible value, -100%
```

```
cols = ["Material", "Pearson correlation (in %)", "MAPE (in %)"]
rows = []

print("Material    &    Pearson correlation    &    MAPE \\\\")
print("\\hline")

for sample in json2gvxr.params["Samples"]:
    label = sample["Label"]
    if os.path.isfile("XCOM_data/" + label + ".csv"):
        df = pd.read_csv("XCOM_data/" + label + ".csv")

        gvxr_mu_rho = []
        gvxr_energy = []

        for energy_id in range(len(df["Photon in MeV"])):
            if energy_id == len(df["Photon in MeV"]) - 1 or energy_id == 0:
                energy = df["Photon in MeV"][energy_id]
            elif df["Photon in MeV"][energy_id] == df["Photon in␣
↪MeV"][energy_id + 1]:
                energy = df["Photon in MeV"][energy_id] - df["Photon in␣
↪MeV"][energy_id] * 1e-3
            elif df["Photon in MeV"][energy_id] == df["Photon in␣
↪MeV"][energy_id - 1]:
                energy = df["Photon in MeV"][energy_id] + df["Photon in␣
↪MeV"][energy_id] * 1e-3
            else:
                energy = df["Photon in MeV"][energy_id]

            gvxr_mu_rho.append(gvxr.getMassAttenuationCoefficient(label,␣
↪energy, "MeV"))
            gvxr_energy.append(energy)


        df["Mass attenuation coefficient (gVirtualXRay)"] = gvxr_mu_rho
        df["|XCOM - gVirtualXRay| / XCOM (in %)"] = np.abs(np.array(100 *␣
↪(df["Mass attenuation coefficient"] - gvxr_mu_rho) / df["Mass attenuation␣
↪coefficient"])).astype(int)

        if df["|XCOM - gVirtualXRay| / XCOM (in %)"].mean() > 1.0:
            print("\tWARNING:")
            print("\t\tAverage:", df["|XCOM - gVirtualXRay| / XCOM (in %)"].
↪mean())
            print("\t\tMax:", df["|XCOM - gVirtualXRay| / XCOM (in %)"].abs().
↪max())
```

```
        df.to_csv("gVirtualXRay_output_data/" + label + "-validation.csv",␣
 ↪index=False)

        plt.subplot(3, 2, i)
        plt.title(label.replace('_ICRU_44','').replace('Grey_White','Grey/
 ↪White').replace('_',', '), y=0.80)
        plt.plot(df["Photon in MeV"], df["Mass attenuation coefficient"],␣
 ↪label="XCOM", color="b")
        plt.scatter(gvxr_energy, gvxr_mu_rho, label="gVirtualXRay", s=15,␣
 ↪color="r")
        plt.xlabel('Photon in MeV')
        plt.ylabel('${\\mu}/{\\rho}$ in cm$^2$/g')
        plt.xscale('log')
        plt.yscale('log')

        corr, _ = pearsonr(df["Mass attenuation coefficient"], gvxr_mu_rho)
        MAPE = mape(df["Mass attenuation coefficient"], gvxr_mu_rho)

        label = label.replace("_ICRU_44", "")
        label = label.replace("_", ", ")

        row = [label, 100.0 * corr, 100.0 * MAPE]
        rows.append(row)

        percent = "%"
        print(label + "    &    %.2f" % (100 * corr) + "\\" + percent + "    & ␣
 ↪  %.2f" % (100 * MAPE) + "\\" + percent + " \\\\")

        if i == 5:
            plt.legend(loc='lower center', bbox_to_anchor=(0.9, -0.4, 0.5, 0.5))

        i = i + 1


df = pd.DataFrame(columns=cols, data=rows)
print("Overall" + "    &    %.2f" % (df["Pearson correlation (in %)"].mean()) +␣
 ↪"\\" + percent + "$\\pm$%.2f" % (df["Pearson correlation (in %)"].std()) +
      "    &    %.2f" % (df["MAPE (in %)"].mean()) + "\\" + percent + "$\\pm$%.
 ↪2f" % (df["MAPE (in %)"].std()) + " \\\\")

plt.tight_layout()

plt.savefig('plots/mass_attenuation_coefficients_from_ICRU44.pdf')
plt.savefig('plots/mass_attenuation_coefficients_from_ICRU44.png')
```

```
Material    &    Pearson correlation    &    MAPE \\
\hline
```

```
Bone, Cortical    &    99.97\%    &    0.50\% \\
Blood, Whole    &    99.99\%    &    1.06\% \\
Brain, Grey, White, Matter    &    99.99\%    &    0.64\% \\
Breast, Tissue    &    99.99\%    &    0.91\% \\
Adipose, Tissue    &    99.99\%    &    0.73\% \\
H2O    &    100.00\%    &    0.06\% \\
Overall    &    99.99\%$\pm$0.01    &    0.65\%$\pm$0.35 \\
```



[16]:
```
print(df)
```

```
                  Material  Pearson correlation (in %)  MAPE (in %)
0          Bone, Cortical                   99.968093     0.499700
1            Blood, Whole                   99.991698     1.055023
```

```
2  Brain, Grey, White, Matter                  99.991079      0.642719
3                Breast, Tissue                99.991442      0.908802
4              Adipose, Tissue                 99.991395      0.732040
5                          H2O                100.000000      0.059322
```

4.3 Worse case scenario

[17]:
```
print("The smallest Pearsons correlation is %.2f" % df["Pearson correlation (in␣
↪%)"].min() + "%")
print("The largest MAPE is %.2f" % df["MAPE (in %)"].max() + "%")
```

```
The smallest Pearsons correlation is 99.97%
The largest MAPE is 1.06%
```

The smallest Pearsons correlation is almost 100%, it is as good as it can get. The largest MAPE is about 1%, which is an extremely small error.

5 Shuting down

[18]:
```
gvxr.destroyAllWindows()
```

```
0(0x563c6d613d20)
```