# Fitting an ellipsoid to an arbitrary 3D shape

November 8, 2023

The idea of this method is mostly based on this paper: https://doi.org/10.1016/S0191-8141(03)00093-2, so I would recommend reading it first (especially Appendix B). Here I extend that idea into fitting a 3D ellipsoid to an arbitrary 3D shape.

## 1 General equation of an ellipsoid in three dimensions

From basic geometry, the equation of the locus of a point $\mathbf{p}' = (x', y', z')$ on the boundary of an ellipsoid, with centre $\mu' = (\mu'_x, \mu'_y, \mu'_z)$, and $l_1$, $l_2$, $l_3$ being the length of the semi-axes, respectively, and major axis parallel to the $x'$-axis, is given by:

$$\frac{(x' - \mu'_x)^2}{l_1^2} + \frac{(y' - \mu'_y)^2}{l_2^2} + \frac{(z' - \mu'_z)^2}{l_3^2} = 1 \tag{1}$$

In general, the major axis orientation can be represented by 3D elemental rotation, that is rotation about x-, y-, or z-axis using the right-hand rule. Specifically:

- Rotation by an angle $\theta$ about the x-axis using the right-hand rule:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

- Rotation by an angle $\theta$ about the y-axis using the right-hand rule:

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

- Rotation by an angle $\theta$ about the z-axis using the right-hand rule:

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In our contexts, we are interested in *Euler angles*, and using matrix multiplication we can represent an extrinsic rotation whose (improper) Euler angles are $\alpha$, $\beta$, and $\gamma$:

$$R = R_z(\gamma)R_y(\beta)R_x(\alpha) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

$$= \begin{bmatrix} \cos\beta\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \cos\beta\sin\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{bmatrix}$$

Therefore, from the reference coordinate $\mathbf{p}' = (x', y', z')$, the rotation produces new variables $\mathbf{p} = (x, y, z)$ as:

$$\mathbf{p} = R\mathbf{p}' \tag{2}$$

Now, notice that the Eq. (1) can be written in this format as follows:

$$(\mathbf{p}' - \mu')^T L^{-1} (\mathbf{p}' - \mu') = \begin{bmatrix} x' - \mu'_x & y' - \mu'_y & z' - \mu'_z \end{bmatrix} \begin{bmatrix} l_1^2 & 0 & 0 \\ 0 & l_2^2 & 0 \\ 0 & 0 & l_3^2 \end{bmatrix}^{-1} \begin{bmatrix} x' - \mu'_x \\ y' - \mu'_y \\ z' - \mu'_z \end{bmatrix} = 1$$

The choice of axis length matrix $L$ will be shown shortly after. Now, the change of variables can be introduced using the matrix form of the rotation transformation in Eq. (2) $\mathbf{p}' - \mu' = R^{-1}(\mathbf{p} - \mu)$, yielding:

$$(\mathbf{p} - \mu)^T (R^{-1})^T L^{-1} R^{-1} (\mathbf{p} - \mu) = 1$$

Here, $\mu = (\mu_x, \mu_y, \mu_z) = R\mu'$ are the rotated coordinates of the ellipsoid centre. Let $\Sigma = R^T L R$, we get the general equation of an ellipsoid in three dimensions (in matrix notation) as:

$$(\mathbf{p} - \mu)^T \Sigma^{-1} (\mathbf{p} - \mu) = 1 \tag{3}$$

## 2 Fitting an ellipsoid to a 3D region identified by points

In this section a method for fitting an ellipsoid $E$ to a given region $R$ containing $n$ points $\mathbf{p}_i = (x_i, y_i, z_i)$, $i = 1, 2, ..., n$ is derived. We define our error function to be a variant of the squared Euclidean distance:

$$\mathbf{d}(\mathbf{p}, E) = \sum_{i=1}^{n} (\mathbf{p}_i - \mu)^T \Sigma^{-1} (\mathbf{p}_i - \mu) + \frac{n}{2} \log |\Sigma| \tag{4}$$

We can see that letting $\Sigma = \mathbf{I}$ would give us the well-known least-square error function. Geometrically, $\mathbf{d}(\mathbf{p}, E)$ can be viewed as the Euclidean distance computed on variables transformed in such a way that the ellipsoid would appear as a sphere in the transformed space. We can also see that the solution to minimising $\mathbf{d}(\mathbf{p}, E)$ is the same as the estimation of the mean vector $\mu$ and dispersion matrix $\Sigma$ of a trivariate Gaussian distribution, given a sample of $n$ data points. This is shown in *Anderson, T.W., 1984. Multivariate Statistical Analysis, 2nd ed, John Wiley*, Theorem 3.2.1, and the solution is given as:

- Mean:

$$\bar{\mu} = \begin{bmatrix} \mu_x & \mu_y & \mu_z \end{bmatrix}^T = \begin{bmatrix} \frac{1}{n}\sum_{i=1}^{n} x_i & \frac{1}{n}\sum_{i=1}^{n} y_i & \frac{1}{n}\sum_{i=1}^{n} z_i \end{bmatrix}^T \tag{5}$$

- Dispersion matrix:

$$\bar{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{p_i} - \bar{\mu})(\mathbf{p_i} - \bar{\mu})^T \tag{6}$$

$$= \begin{bmatrix} \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu_x)^2 & \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y) & \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu_x)(z_i-\mu_z) \\ \frac{1}{n}\sum_{i=1}^{n}(y_i-\mu_y)(x_i-\mu_x) & \frac{1}{n}\sum_{i=1}^{n}(y_i-\mu_y)^2 & \frac{1}{n}\sum_{i=1}^{n}(y_i-\mu_y)(z_i-\mu_z) \\ \frac{1}{n}\sum_{i=1}^{n}(z_i-\mu_z)(x_i-\mu_x) & \frac{1}{n}\sum_{i=1}^{n}(z_i-\mu_z)(y_i-\mu_y) & \frac{1}{n}\sum_{i=1}^{n}(z_i-\mu_z)^2 \end{bmatrix} \tag{7}$$

This result implies that the best fitting ellipsoid (in the sense of minimising distance $\mathbf{d}(\mathbf{p}, E)$) has a center vector $\mu = \bar{\mu}$ and dispersion matrix $\Sigma = \bar{\Sigma}$. At this stage, it remains to identify the axis length parameters $l_1$, $l_2$, $l_3$, and Euler angles $\alpha$, $\beta$, $\gamma$ that correspond to this closest fit. This can be done by making the substitution $\Sigma = R^T L R$ and get:

$$R^T L R = \bar{\Sigma} \tag{8}$$

Now, we can see that the rotational matrix $R$ is orthonormal, therefore $R^T = R^{-1}$. This leads to Eq. (8) be the diagonalization of dispersion matrix $\bar{\Sigma}$, therefore the major-axis length of our ellipsoid will be $\sqrt{\lambda_{max}(\bar{\Sigma})}$, and the minor-axis length will be $\sqrt{\lambda_{min}(\bar{\Sigma})}$. However, this is the ellipsoid that minimizes the error function given above with correct aspect ratios, and to get an ellipsoid that best "covers" our grain, we want to scale it with equivalent volume: $vol(E) = n$. Notice that:

$$vol(E) = \frac{4}{3}\pi\prod_{i=1}^{3} l_i = \frac{4}{3}\pi\prod_{i=1}^{3}\sqrt{\lambda_i(\bar{\Sigma})} = \frac{4}{3}\pi\prod_{i=1}^{3}\sqrt{L_{ii}} = \frac{4}{3}\pi\sqrt{\det(L)}$$

The last equation is from the fact that $L$ is a diagonal matrix. Therefore, the correct scaling for our 1D axes is $\left[\frac{n}{vol(E)}\right]^{\frac{1}{3}}$, and our semi-major and semi-minor axes are:

$$\max(l_i) = \left[\frac{3n}{4\pi\sqrt{\det(L)}}\right]^{\frac{1}{3}}\sqrt{\max(L_{ii})} \tag{9}$$

$$\min(l_i) = \left[\frac{3n}{4\pi\sqrt{\det(L)}}\right]^{\frac{1}{3}}\sqrt{\min(L_{ii})} \tag{10}$$

Finally, the Euler angles $\alpha$, $\beta$, $\gamma$ can be extracted from $R$ using these scheme: http://eecs.qmul.ac.uk/~gslabaugh/publications/euler.pdf, https://nghiaho.com/?page_id=846 to get:

$$\alpha = atan2(R_{32}, R_{33}) \tag{11}$$

$$\beta = atan2(-R_{31}, \sqrt{R_{32}^2 + R_{33}^2}) \tag{12}$$

$$\gamma = atan2(R_{21}, R_{11}) \tag{13}$$

where $atan2$ is the 2-argument arctangent.

## 3 Implementation on Python

First, we will import some necessary libraries for our code.

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import lognorm, beta
```

The next step is to extract all points (or $xyz$-coordinates) inside a grain by the grain's ID. The function is provided by Ben Nguyen.

[2]:
```python
# Get a grain by grain ID
RVE = {}
file_open = open("text1.txt", 'r')
for line in file_open:
    line_split = line.split(" ")
    line_split.pop()
    grainID = int(line_split[-1])
    if grainID in RVE:
        RVE[grainID].append([int(line_split[3]), int(line_split[4]),␣
 ↪int(line_split[5])])
    else:
        RVE[grainID] = [[int(line_split[3]), int(line_split[4]),␣
 ↪int(line_split[5])]]
```

Here, each voxel is only given as coordinates, so we will create another data for storing the 8 vertices of each voxel for fitting. The reason for this is that if the method consider each voxel as a point, then in the case that the grain is a single $1 \times 1 \times 1$ voxel would lead to the ellipsoid collapses into a single point. Considering a voxel as a cube with 8 vertices and fit the ellipsoid to the vertices solves the problem.

[3]:
```python
vertices = {}
for i in RVE:
    indices = np.array(RVE[i])
    x = indices
    x = np.append(x, indices - [1, 0, 0], axis=0)
    x = np.append(x, indices - [0, 1, 0], axis=0)
    x = np.append(x, indices - [0, 0, 1], axis=0)
    x = np.append(x, indices - [1, 1, 0], axis=0)
    x = np.append(x, indices - [1, 0, 1], axis=0)
    x = np.append(x, indices - [0, 1, 1], axis=0)
    x = np.append(x, indices - [1, 1, 1], axis=0)
    vertices[i] = np.unique(x, axis=0)
```

Now, we pick a random grain ID as an example for our function.

[4]:
```python
A = 579
```

The main part of our implementation is the function `fitEllipsoid` below, which is based on the theory discussed above. This function will take an array of $xyz$-coordinates as input, and return outputs as all necessary statistics for us to define our ellipsoid.

[5]:
```
# Set RVE dimension and resolution
dim = 100
res = 5

# Fit an ellipsoid to the grain
def fitEllipsoid(i):
    """Returns the best-fitted ellipsoid to a 3D region defined by a set of␣
 ↪points on xyz-coordinate.
       Input:
        i    : our grain ID
       Output
        mu_x, mu_y, mu_z: the center of our ellipsoid
        Sigma           : a matrix such that inv(Sigma) define our quadric␣
 ↪surface
        L               : an array containing the length of the semi-axes of␣
 ↪our ellipsoid
        R               : the rotational matrix
    """

    # Get the grain's vertices and volume
    A = np.transpose(vertices[i])
    n = np.shape(RVE[i])[0]
    # Calculate the center of our ellipsoid
    mu_x = np.average(A[0])
    mu_y = np.average(A[1])
    mu_z = np.average(A[2])
    # Define our quadric surface, which is equal to the dispersion matrix of␣
 ↪our data
    Sigma = [[np.cov(A[0],A[0])[0][1], np.cov(A[0],A[1])[0][1], np.
 ↪cov(A[0],A[2])[0][1]],
             [np.cov(A[1],A[0])[0][1], np.cov(A[1],A[1])[0][1], np.
 ↪cov(A[1],A[2])[0][1]],
             [np.cov(A[2],A[0])[0][1], np.cov(A[2],A[1])[0][1], np.
 ↪cov(A[2],A[2])[0][1]]]
    # Extract the axes and the rotation matrix
    L, R = np.linalg.eig(Sigma)
    # Scale the axes by volume
    scaling = np.cbrt(3*n/(4*np.pi*np.sqrt(np.prod(L))))
    L = np.sqrt(L) * scaling
    # Correct the rotation matrix
    R = np.transpose(R)

    # Return the results
    return mu_x, mu_y, mu_z, Sigma, L, R
```

Now that our function `fitEllipsoid` is implemented, we can fit an ellipsoid to our chosen grain.

```
[6]: mu_x, mu_y, mu_z, Sigma, L, R = fitEllipsoid(A)
```

From the calculated rotation matrix $R$, we can extract Euler angles $\alpha$, $\beta$, $\gamma$ using the discussed scheme.

```
[7]: # Euler angles
     a = np.arctan2(R[2][1], R[2][2])
     b = np.arctan2(-R[2][0], np.sqrt(R[2][1]**2 + R[2][2]**2))
     g = np.arctan2(R[1][0], R[0][0])
     a, b, g
```

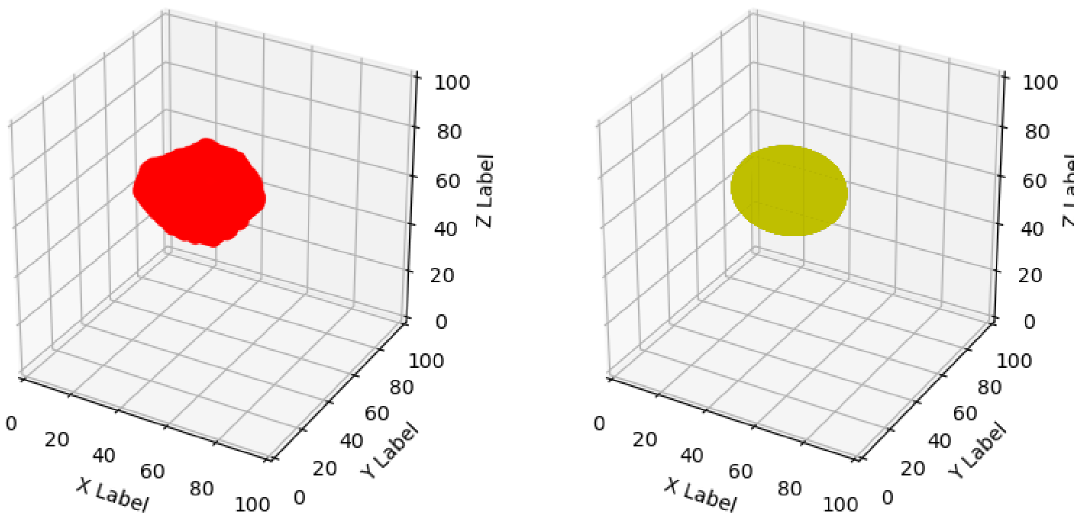
```
[7]: (-1.2890483182380252, -0.03812276033118074, 0.1212130202022888)
```

To verify that our ellipsoid is indeed "fitted" to the grain, we will visualize both our grain and the ellipsoid inside the dimension of our RVE. Visualizing the ellipsoid can be done by generating a data of points lying on the ellipsoid surface, and this can be easily done by applying sphere-to-ellipsoid transformation on $\mathbf{p}' = (x', y', z')$ and coordinate transformation from $\mathbf{p}'$ to $\mathbf{p}(x, y, z) = R\mathbf{p}'(x', y', z')$.

```
[8]: # Pre-transformed center
     mu_ = np.transpose(R) @ [mu_x, mu_y, mu_z]

     ellipsoid = []
     for i in range(0,100000):
         # Simulate points on a sphere
         x = np.random.uniform(-1,1)
         y = np.random.uniform(-1,1)
         z = np.random.uniform(-1,1)
         [x, y, z] = [x, y, z]/np.sqrt(x**2 + y**2 + z**2)
         # Transform into an ellipsoid
         x = x * L[0] + mu_[0]
         y = y * L[1] + mu_[1]
         z = z * L[2] + mu_[2]
         # Rotate the ellipsoid
         p = R @ [x, y, z]
         ellipsoid += [p]
```

Now we can visualize our result.

```
[9]: fig = plt.figure(figsize=plt.figaspect(0.5))

     # Visualize our grain
     ax = fig.add_subplot(1, 2, 1, projection='3d')
     ax.scatter(np.transpose(vertices[A])[0],np.transpose(vertices[A])[1],np.
      ↪transpose(vertices[A])[2],color='r')
     ax.set_xlim([0,100])
     ax.set_ylim([0,100])
ax.set_zlim([0,100])
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
ax.set_box_aspect([1.0, 1.0, 1.0])

# Visualize the fitted ellipsoid
ax = fig.add_subplot(1, 2, 2, projection='3d')
ax.scatter(np.transpose(ellipsoid)[0],np.transpose(ellipsoid)[1],np.
 ↪transpose(ellipsoid)[2],color='y',alpha=0.1)
ax.set_xlim([0,100])
ax.set_ylim([0,100])
ax.set_zlim([0,100])
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
ax.set_box_aspect([1.0, 1.0, 1.0])
```



We are also interested in the grain size and shape distribution of our generated RVE. For this purpose, we will calculate for each grain a fitted ellipsoid, and from the ellipsoid the equivalent diameter and grain aspect ratio can be calculated as:

$$d_{RVE} = 2\left(\prod_{i=1}^{3} l_i\right)^{\frac{1}{3}} \tag{14}$$

$$a_{RVE} = \frac{\min(l_i)}{\max(l_i)} \tag{15}$$

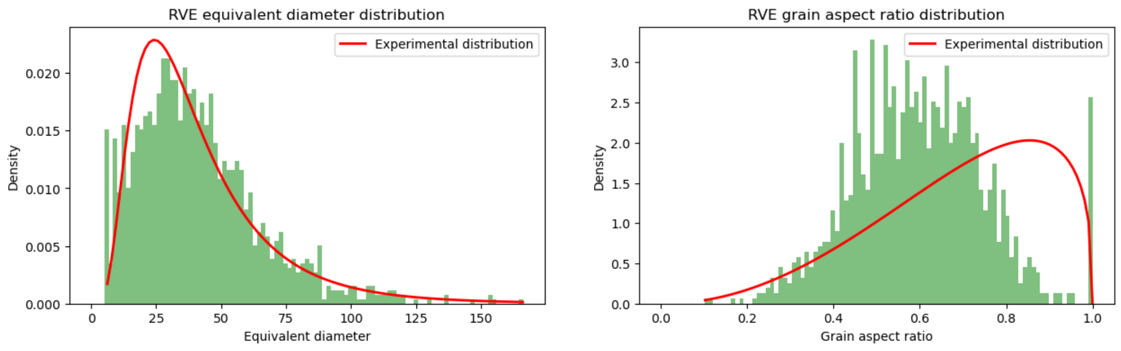
The code below implement these calculations.

[10]:
```
d = []
a = []
for i in RVE:
    mu_x, mu_y, mu_z, Sigma, L, R = fitEllipsoid(i)
    # Calculate equivalent diameter, scaled with the resolution
    d.append(np.cbrt(np.prod(L)) * 2 * res)
    # Calculate grain aspect ratio
    a.append(min(L)/max(L))
```

Finally, we can plot the characteristic distribution of the grains inside our RVE. The experimental distribution of our RVE is given as reference.

[11]:
```
fig, axes = plt.subplots(1, 2, figsize=(15,4))

# Plot the RVE equivalent diameter distribution
axes[0].hist(d, color='g', bins=np.linspace(0, max(d), 100), alpha=0.5,␣
 ↪density=True)
axes[0].plot(np.linspace(min(d), max(d), 100), lognorm.pdf(np.linspace(min(d),␣
 ↪max(d), 100), 0.5996, scale=np.exp(3.5503)),
             'r-', lw=2, label='Experimental distribution')
axes[0].set_xlabel("Equivalent diameter")
axes[0].set_ylabel("Density")
axes[0].set_title("RVE equivalent diameter distribution")
axes[0].legend()

# Plot the RVE grain aspect ratio distribution
axes[1].hist(a, color='g', bins=np.linspace(0, 1, 100), alpha=0.5, density=True)
axes[1].plot(np.linspace(min(a), max(a), 100), beta.pdf(np.linspace(min(a),␣
 ↪max(a), 100), 3.1206, 1.3618),
             'r-', lw=2, label='Experimental distribution')
axes[1].set_xlabel("Grain aspect ratio")
axes[1].set_ylabel("Density")
axes[1].set_title("RVE grain aspect ratio distribution")
axes[1].legend()
```

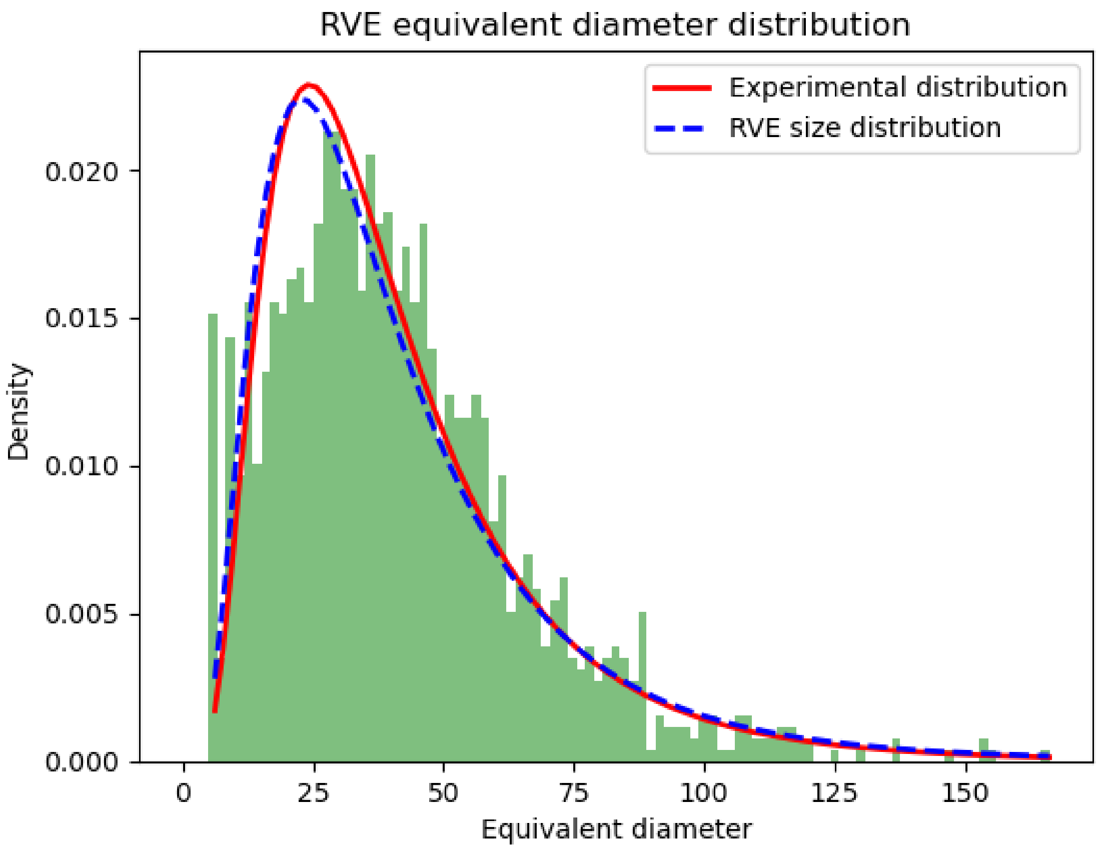
[11]:
```
<matplotlib.legend.Legend at 0x7fb54ad6e770>
```



To evaluate our results, i.e. how accurately our RVE characterises the material, we will fit our experimental data into the corresponding distribution using the *Maximum Likelihood Estimator* (MLE) and get the parameter. The plot below shows the log-normal distribution of our RVE grain size compared to the experimentally fitted data.

[12]:
```python
# Find the PDF of our RVE size using MLE
shape, loc, scale = lognorm.fit(d, floc=0)
mu_fit, sigma_fit = np.log(scale), shape

# Plot the RVE equivalent diameter distribution
plt.hist(d, color='g', bins=np.linspace(0, max(d), 100), alpha=0.5,␣
 ↪density=True)
plt.plot(np.linspace(min(d), max(d), 100), lognorm.pdf(np.linspace(min(d),␣
 ↪max(d), 100), 0.5996, scale=np.exp(3.5503)),
             'r-', lw=2, label='Experimental distribution')
plt.plot(np.linspace(min(d), max(d), 100), lognorm.pdf(np.linspace(min(d),␣
 ↪max(d), 100), sigma_fit, scale=np.exp(mu_fit)),
             'b--', lw=2, label='RVE size distribution')
plt.xlabel("Equivalent diameter")
plt.ylabel("Density")
plt.title("RVE equivalent diameter distribution")
plt.legend()
```

[12]:
```
<matplotlib.legend.Legend at 0x7fb549223100>
```



Now, the difference between two distributions can be quantitated using the **Hellinger distance**, which quantify the similarity between two probability distributions $P$ and $Q$. The value ranges from 0 (being the same distribution) to 1 ($P$ assigns probability zero to every set to which $Q$ assigns a positive probability, and vice versa). In this specific case of log-normal distribution with $P(\mu_1, \sigma_1)$ and $Q(\mu_2, \sigma_2)$, the Hellinger distance is defined as:

$$H(P,Q) = \left[1 - \sqrt{\frac{2\sigma_1\sigma_2}{\sigma_1^2+\sigma_2^2}}\, e^{-\frac{1}{4}\frac{(\mu_1-\mu_2)^2}{\sigma_1^2+\sigma_2^2}}\right]^{\frac{1}{2}} \tag{16}$$

[13]:
```
# Calculate the Hellinger distance
hellinger_size = np.sqrt(1 - np.sqrt(2*sigma_fit*0.5996/(sigma_fit**2 + 0.
  ↪5996**2))
                                *np.exp(-1/4 * (mu_fit - 3.5503)**2/(sigma_fit**2 + 0.
  ↪5996**2)))
# Print the results
print(f"The Hellinger distance of the RVE and experimental size distribution is␣
  ↪{hellinger_size}.")
```

```
The Hellinger distance of the RVE and experimental size distribution is
```

```
0.032235357561045744.
```

For now, the problem is that I can't seem to fit the beta distribution into the data properly (in other words, the parameters look insane).

```
[14]: # Find the PDF of our RVE size using MLE
ap, be, loc, scale = beta.fit(a, floc=0)
print(f"The parameters of the RVE beta shape distribution is ({ap}, {be}).")
```

```
The parameters of the RVE beta shape distribution is (14.487545335525471,
3015393.3186525526).
```

```
/opt/software/lib/python3.10/site-packages/scipy/optimize/_minpack_py.py:178:
RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
```