Experiment 212 - Viscosity of fluids

Felix Fleischle

18.10.2021

Introduction

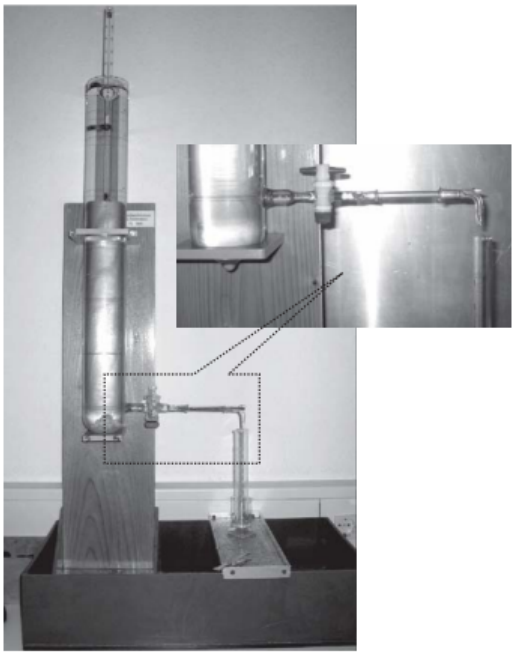


Figure 1: Image of the experimental setup

In this experiment, the objetives are to determine the viscosity η of polyethylene glycol with a falling sphere viscometer after Stoke's law, as well as with a capillary viscometer according to Hagen-Poiseuille. Additionally, we will check the validity limit of Stoke's law by determining the transition from laminar to turbulent flow. In the end we will compare the two viscosities we determined.

If a body moves in a medium, there are several forces acting on the body. First we have the force that is moving the body, in our case the gravitional force

$$F_g = mg = \rho V g \tag{1}$$

Additionally, there is a friction between the body and the medium. Due to the friction, the medium is moved alongside the body to a certain extent. Thus we can divide the medium in to liquid layers between the body and the resting wall. This results in a velocity gradient:

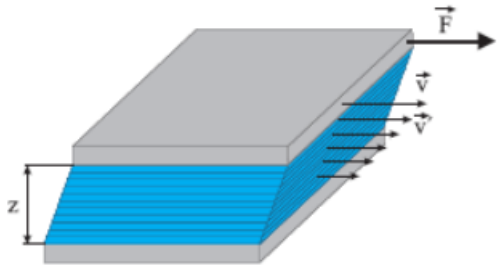


Figure 2: Velocity gradient due to friction

The friction force is

$$F_r = \eta A \frac{v}{z} = -6\pi\eta r v \tag{2}$$

where η is the viscosity, A the surface area, v the velocity of the body, and r the radius of the balls we will use for the measurement. This is called the law of Stokes.

This equation is only accurate for laminar flow, where the liquid layers are not getting mixed. When the flow is turbulent, and thus the layers are getting mixed, which occurs at higher speeds or specific body shapes, the friction force is much higher. We can estimate whether laminar of turbulent flow will occur through the Reynolds number

$$Re = \frac{2E_{kin}}{W_{friction}} \tag{3}$$

If the Reynolds number is above a certain threshold, which depends on the experimental setup, the flow will be turbulent. Below the threshhold the flow

will be laminar. We can estimate the Reynolds number with the equation

$$Re = \frac{\rho v L}{\eta} \tag{4}$$

where L is a "typical length", which equals the diameter of the ball if a ball moves through liquid, where $Re_{cr} = 2300$, or the diameter of the pipe for a pipe flow, where $Re_{cr} = 1$.

As mentioned before, we will have two different parts of the experiment. In the first part we will determine the viscosity through (2), using the force equilibrium

$$F_g + F_a + F_r = 0 \tag{5}$$

where $F_a = -\rho_f V_k g$ is the buoyancy force. Our experimental setup is a falling sphere viscometer, where we drop balls of different diameters in the viscometer and measure the sinking velocity. We can get an equation for the viscosity:

$$\eta = \frac{2}{9} g \frac{(\rho_K - \rho_f)}{v} r^2 \tag{6}$$

We will measure the sinking velocity for the diameters of 2, 3, 4, 5, 6, 7.144, 8 and 9 mm, 5 times for each diameter. We will then plot $v/(\rho_k - \rho_f)$ against r^2. Because Stokes law is only applicable for an infinite fluid, we will use the correction term

$$F_r = -6\pi\eta\lambda r v \tag{7}$$

$$\lambda = (1 + 2,1\frac{r}{R}) \tag{8}$$

where R is the radius of the pipe. We will then plot the result in the same diagram. We can then determine the viscosity through the slope of the linear part of the graph.

Then we will calculate the value of v_{lam} through (6) for each radius, as well as the Reynolds number with (4) for each diameter. We can then plot $\frac{v}{v_{lam}}$ against $\log Re$ and determine the point where a crease appears. This is exactly the point at which Stokes law loses its validity and turbulent flow occurs. We will also estimate the numerical value for Re_{cr}.

In the second part of the experiment, we will determine the viscosity after Hagen-Poiseuille, which describes laminar flow through a pipe

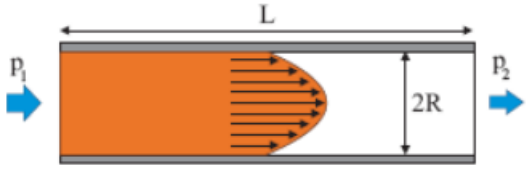


Figure 3: Laminar flow through a pipe

There is a force that acts on the liquid due to the pressure difference:

$$F_p = \pi r^2 (p_1 - p_2) \tag{9}$$

The newtonian friction is also present

$$F_r = -2\pi r L \eta \frac{dv}{dr} \tag{10}$$

If we use the force equilibrium and integrate over r, we get

$$v(r) = \frac{p_1 - p_2}{4\eta L}(R^2 - r^2 \tag{11}$$

where R is the radius of the whole pipe. If we integrate $2\pi r v(r)$ over r from 0 to R, we get the flow rate:

$$\frac{dV}{dt} = \frac{\pi (p_1 - p_2) R^4}{8\eta L} \tag{12}$$

So we can use the capillary viscometer to create a laminar flow, and measure the height difference in the pipe after $25cm^3$ of fluid have left the viscometer.

212 Viscosity of fluids - measurement protocol

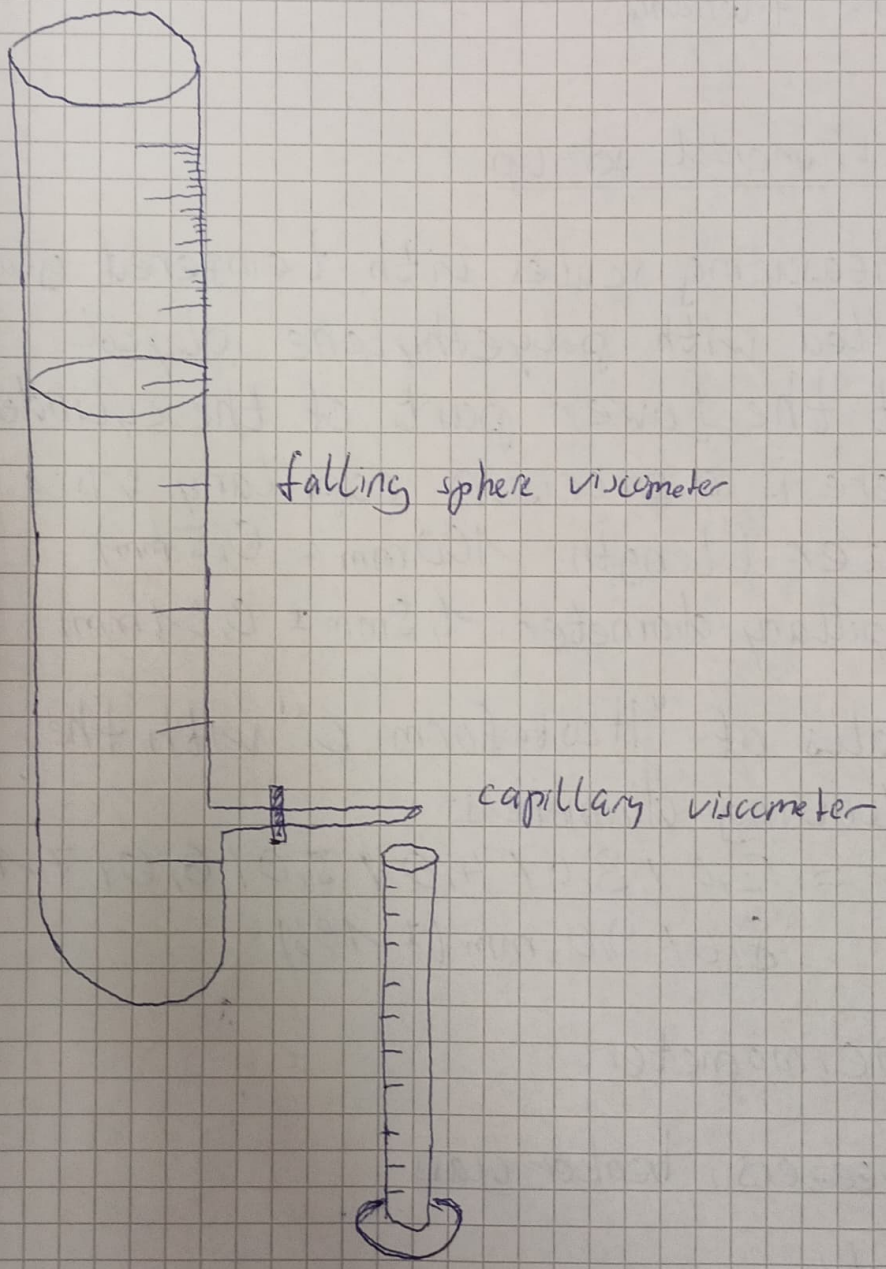
18.10.21

Lara Ziegler
Felix Fleischle

Instrumental set-up

- Measuring scales with tempered glass, filled with polyethylene glycol. At the lower part of the cylinder there is a precision capillary viscometer (length: 100mm ± 0,5mm) (capillary diameter 1,5mm ± 0,01mm)
- Balls of "Hostaform C" with the following diameters:
  $2r$ = 2,0 / 3,0 / 4,0 / 5,0 / 6,0 / 7,144 / 8,0 / 9,0 mm (± 1%)
- Thermometer
- Tweezers, beaker glass
- Scale
- Stopwatches

Sketch of the experiment



Experimental procedure

Determination of the viscosity after Stokes with a falling sphere viscometer

We will measure the falling time Δt five times for each diameter. We will also note the temperature and the distance of the markers Δs.

Table 1: Measuring Δt in the falling ~~ype~~ sphere viscometer

Measurement Nr.	diameter [mm]	Δs [mm]	Δt [s]	T [°C]
1		~~40~~ 20±2	~~9,30~~ 8,81	22±1
2		~~40~~ 20±2	8,~~64~~	22±1
3	2,0	~~40~~ 20±2	8,72	22±1
4		20±2	8,67	22±1
5		20±2	8,61	22±1
6		20±2	4,37	22±1
7		20±2	4,04	22±1
8	3,0	20±2	4,51	22±1
9		20±2	4,36	22±1
10		20±2	4,37	22±1

11		50±2	6,42	22±1
12		50±2	6,58	22±1
13	4,0	50±2	6,31	22±1
14		50±2	6,82	22±1
15		50±2	6,34	22±1
16		50±2	4,14	22±1
17		50±2	4,18	22±1
18	5,0	50±2	4,45	22±1
19		50±2	4,17	22±1
20		50±2	4,34	22±1
21		50±2	3,10	22±1
22		50±2	3,18	22±1
23	6,0	50±2	3,15	22±1
24		50±2	3,26	22±1
25		50±2	3,50	22±1
26		100±2	4,59	22±1
27		100±2	4,42	22±1
28	7,144	100±2	4,48	22±1
29		100±2	4,43	22±1
30		100±2	4,62	22±1
31		~~1800~~ 200±2	8,12	22±1
32		200±2	8,20	22±1
33	8,0	200±2	8,12	22±1
34		200±2	7,97	22±1
35		200±2	8,15	22±1

36	" "	200±2	6,37	22±1
37		200±2	6,29	22±1
38	90	200±2	6,20	22±1
39		200±2	6,45	22±1
40		200±2	6,14	22±1

Inner diameter of the ball drop vessel: d = 75mm

Stopwatch accuracy: 0,01s

~~Height accuracy:~~

Room Temperature T_0 = 21,7°C

Determination of the viscosity after Hagen-Poiseuille with a capillary viscometer

We will measure the beginning height h_A of the liquid column, as well as the end height h_E after 25cm³ are inside the measuring cylinder.
Additionally, we will measure the time until 5, 10, .., 25 cm³ are inside the cylinder.

h_A = 550mm ± 1mm

h_E = 544 ± 1 mm

Table 2: Outflow time measurement

V [cm^3]	t [s]
5	1.52 ± 5
10	412 ± 5
15	625 ± 5
20	855 ± 5
25	1114 ± 5

Temperature $T = 22 \pm 1\ °C$

Room Temperature $T_0 = 21,8\ °C$

Air pressure: $p_0 = 1012,6 \pm 0,2$ hPa

Valle

Viscosity of fluids - Analysis

Felix Fleischle - 18.10.2021

This is my first analysis done with Python, so there are probably many ways to do this way more efficient, like putting all the values into arrays, which i did not realize until later, so apologies for the redundancies.

Determination of the viscosity with the falling sphere viscometer

For the first part of the experiment, we will put our measured data for each diameter into an array:

In [1]:
```
# First we import the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
%matplotlib qt5
```

In [2]:
```
# Now we enter our data into numpy arrays and set the distances over which the times were measured.
# Additionally, we will enter the errors of the measured numbers.
# The time error is equal to 0.3s due to reaction time.

times_2mm = np.array([8.81 , 8.64 , 8.72 , 8.67 , 8.61])
distance_2mm = 0.02   # Unit is meters

times_3mm = np.array([4.37 , 4.04 , 4.51 , 4.36 , 4.37])
distance_3mm = 0.02

times_4mm = np.array([6.42 , 6.58 , 6.31 , 6.82 , 6.34])
distance_4mm = 0.05

times_5mm = np.array([4.14 , 4.18 , 4.45 , 4.17 , 4.34])
distance_5mm = 0.05

times_6mm = np.array([3.10 , 3.18 , 3.15 , 3.26 , 3.50])
distance_6mm = 0.05

times_7144mm = np.array([4.59 , 4.42 , 4.48 , 4.43 , 4.62])
distance_7144mm = 0.1
```

```
times_8mm = np.array([8.12 , 8.20 , 8.12 , 7.92 , 8.15])
distance_8mm = 0.2

times_9mm = np.array([6.37 , 6.29 , 6.20 , 6.45 , 6.14])
distance_9mm = 0.2

# We create an array for the distances
distancearr = np.array([distance_2mm , distance_3mm , distance_4mm , distance_5mm , distance_6mm , distance_7144mm , distance_8mm

# Errors
err_times = 0.3
err_distance = 0.002

# Now we will set the temperature, which remained constant throughout the experiment.
T = 22
err_T = 1

# Room Temperature
T_0 = 21.7
err_T_0 = 0.1

# Inner diameter of the pipe
d_inner = 0.075   # again in meters

# We print two examples, to see if our arrays were correctly created
print("Times: ",times_2mm, "[s]")
print("Times error: ",err_times, "s")
```

```
Times:  [8.81 8.64 8.72 8.67 8.61] [s]
Times error:  0.3 s
```

Now we can calculate the mean sinking velocity for each diameter: The velocity is equal to $v = \frac{\Delta s}{\Delta t}$

In [3]:
```
# We define a function that calculates the velocities
def velocityarray(distance, times):
    return distance / times
# We define another function that calculates the standard deviation of the mean
def stdmeanvelocity(velocities, a):
    return np.std(velocities)/np.sqrt(a)

velocities_2mm = velocityarray(distance_2mm, times_2mm) # Units are meters per second
meanvelocity_2mm = np.mean(velocities_2mm) # Mean
stdmeanvelocity_2mm = stdmeanvelocity(velocities_2mm, 5) # Standard deviation of the mean
```

```
velocities_3mm = velocityarray(distance_3mm, times_3mm)
meanvelocity_3mm = np.mean(velocities_3mm)
stdmeanvelocity_3mm = stdmeanvelocity(velocities_3mm, 5)

velocities_4mm = velocityarray(distance_4mm, times_4mm)
meanvelocity_4mm = np.mean(velocities_4mm)
stdmeanvelocity_4mm = stdmeanvelocity(velocities_4mm, 5)

velocities_5mm = velocityarray(distance_5mm, times_5mm)
meanvelocity_5mm = np.mean(velocities_5mm)
stdmeanvelocity_5mm = stdmeanvelocity(velocities_5mm, 5)

velocities_6mm = velocityarray(distance_6mm, times_6mm)
meanvelocity_6mm = np.mean(velocities_6mm)
stdmeanvelocity_6mm = stdmeanvelocity(velocities_6mm, 5)

velocities_7144mm = velocityarray(distance_7144mm, times_7144mm)
meanvelocity_7144mm = np.mean(velocities_7144mm)
stdmeanvelocity_7144mm = stdmeanvelocity(velocities_7144mm, 5)

velocities_8mm = velocityarray(distance_8mm, times_8mm)
meanvelocity_8mm = np.mean(velocities_8mm)
stdmeanvelocity_8mm = stdmeanvelocity(velocities_8mm, 5)

velocities_9mm = velocityarray(distance_9mm, times_9mm)
meanvelocity_9mm = np.mean(velocities_9mm)
stdmeanvelocity_9mm = np.std(velocities_9mm)/np.sqrt(5)

# Now we create an array for the mean velocities
meanvelocityarr = np.array([meanvelocity_2mm , meanvelocity_3mm , meanvelocity_4mm , meanvelocity_5mm , meanvelocity_6mm , meanvel
print("Mean velocities: ",meanvelocityarr, "[m/s]")
```

```
Mean velocities:  [0.00230165 0.00462511 0.00770574 0.01175722 0.01546969 0.02219015
 0.02468875 0.03180656] [m/s]
```

Now we will calculate the errors of the velocities. The error is composed of the standard deviation of the mean, as well as the reaction time error and distance error through propagation of uncertainty.

In [4]:

```
# We define a function that calculates the error of the mean velocity
def errMeanVelo(distance, times, err_distance, err_times, stdmeanvelocity):
    return np.sqrt( (err_distance / np.mean(times))**2 + ((distance*err_times) / (np.mean(times))**2)**2 + (stdmeanvelocity)**2 )

# Now we calculate the error for each diameter
err_meanvelocity_2mm = errMeanVelo(distance_2mm, times_2mm, err_distance, err_times, stdmeanvelocity_2mm)
```

```
err_meanvelocity_3mm = errMeanVelo(distance_3mm, times_3mm, err_distance, err_times, stdmeanvelocity_3mm)
err_meanvelocity_4mm = errMeanVelo(distance_4mm, times_4mm, err_distance, err_times, stdmeanvelocity_4mm)
err_meanvelocity_5mm = errMeanVelo(distance_5mm, times_5mm, err_distance, err_times, stdmeanvelocity_5mm)
err_meanvelocity_6mm = errMeanVelo(distance_6mm, times_6mm, err_distance, err_times, stdmeanvelocity_6mm)
err_meanvelocity_7144mm = errMeanVelo(distance_7144mm, times_7144mm, err_distance, err_times, stdmeanvelocity_7144mm)
err_meanvelocity_8mm = errMeanVelo(distance_8mm, times_8mm, err_distance, err_times, stdmeanvelocity_8mm)
err_meanvelocity_9mm = errMeanVelo(distance_9mm, times_9mm, err_distance, err_times, stdmeanvelocity_9mm)

err_meanvelocityarr = np.array([err_meanvelocity_2mm , err_meanvelocity_3mm , err_meanvelocity_4mm , err_meanvelocity_5mm , err_me
print("Mean velocities error: ",err_meanvelocityarr, "[m/s]")
```

```
Mean velocities error:  [0.00024362 0.00056718 0.00048053 0.00096314 0.00158479 0.00155196
 0.00095595 0.00156999] [m/s]
```

Our results for the sinking velocities are:

2mm: $v = 2.30 \pm 0.24 \frac{mm}{s}$

3mm: $v = 4.6 \pm 0.6 \frac{mm}{s}$

4mm: $v = 7.7 \pm 0.5 \frac{mm}{s}$

5mm: $v = 11.8 \pm 1.0 \frac{mm}{s}$

6mm: $v = 15.5 \pm 1.6 \frac{mm}{s}$

7.144mm: $v = 22.2 \pm 1.6 \frac{mm}{s}$

8mm: $v = 24.7 \pm 1.0 \frac{mm}{s}$

9mm: $v = 31.8 \pm 1.6 \frac{mm}{s}$

For plotting the velocity against the radius of the balls, we have to put our calculated data into arrays, as well as set the values for the densities of the balls and the polyethylene glycol.

In [5]:
```
# We create an array for the radius of the balls as well as their 1% error
r = np.array([0.001 , 0.0015 , 0.002 , 0.0025 , 0.003 , 0.003572 , 0.004 , 0.0045]) # Units: meters
err_r = r * 0.005

# We enter the values of the densities and their errors
rho_k = np.array([1377.5 , 1377.5 , 1377.5 , 1377.5 , 1377.5 , 1377.5 , 1357.5 , 1362.5]) # Units: kilograms per cubic meter
err_rho_k = np.array([2.5 , 2.5 , 2.5 , 2.5 , 2.5 , 2.5 , 2.5 , 2.5])

# We determine the density of polyethylene glycol through the diagram in the instruction as (T= 22 +- 1 °C)
rho_f = 1147.6 # Unit kilograms per cubic meter
err_rho_f = 1.2

# We have to plot v/(rho_k-rho_f) against r^2, thus we create arrays that include these values
y_values = meanvelocityarr / (rho_k - rho_f)
```

```
x_values = r**2

# The errors of the y and x values are calculated here:
err_y = np.sqrt( ((err_meanvelocityarr) / (rho_k - rho_f))**2 + ((meanvelocityarr*err_rho_k)/(rho_k-rho_f)**2)**2 + ((meanvelocity
err_x = 2*r*err_r

print("y-values: {}".format(y_values))
print("x-values: {}".format(x_values))

print("Error of the y-values: {}".format(err_y))
print("Error of the x-values: {}".format(err_x))
```

```
y-values: [1.00115057e-05 2.01179286e-05 3.35177817e-05 5.11405979e-05
 6.72887831e-05 9.65208703e-05 1.17621468e-04 1.48006313e-04]
x-values: [1.0000000e-06 2.2500000e-06 4.0000000e-06 6.2500000e-06 9.0000000e-06
 1.2759184e-05 1.6000000e-05 2.0250000e-05]
Error of the y-values: [1.06653040e-06 2.47896719e-06 2.12890480e-06 4.23457669e-06
 6.94099269e-06 6.85026402e-06 4.81213728e-06 7.55117715e-06]
Error of the x-values: [1.0000000e-08 2.2500000e-08 4.0000000e-08 6.2500000e-08 9.0000000e-08
 1.2759184e-07 1.6000000e-07 2.0250000e-07]
```

Now we can plot our calculated data

In [6]:
```
plt.errorbar(x_values, y_values, linestyle="None", marker = ".", yerr = err_y, xerr = err_x, ecolor = "red", capsize = 2)
plt.xlabel("$r^2$ [$m^2$] ")
plt.ylabel("$\\frac{v}{(\\rho_k - \\rho_f)}$ [$\\frac{m^4}{s\\cdot kg}$]")
```

Out[6]:
```
Text(0, 0.5, '$\\frac{v}{(\\rho_k - \\rho_f)}$ [$\\frac{m^4}{s\\cdot kg}$]')
```

Our next goal is to introduce the correction factor $\lambda = (1 + 2.1\frac{r}{R})$, which compensates that the liquid is not infinite. It depends on the radius r, so we will create an array for λ.

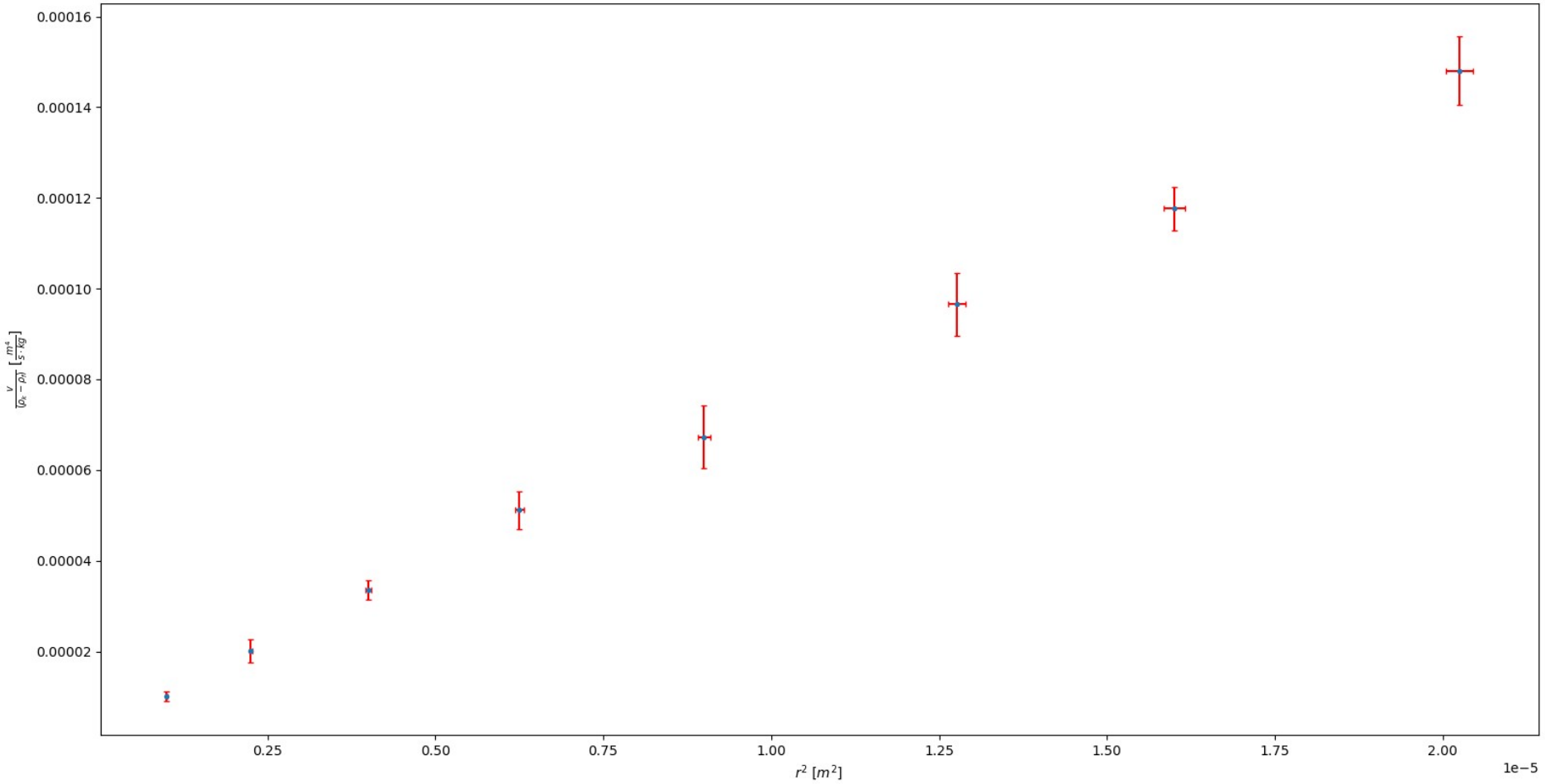
In [7]:
```
lmbda = 1 + 2.1 * (r / (0.5*d_inner))
err_lmbda = 2.1 * (err_r / (0.5*d_inner))

print("Lambda: ",lmbda)
print("Error of Lambda: ",err_lmbda)

y_values_corr = lmbda * y_values # corrected y-values
err_y_corr = np.sqrt( (lmbda * err_y)**2 + (y_values * err_lmbda)**2 ) # errors for the corrected y-values

print("Corrected y-values: ",y_values_corr)
print("Error corrected y-values: ", err_y_corr)
```

```
Lambda:  [1.056    1.084    1.112    1.14     1.168    1.200032 1.224    1.252   ]
```


0.00016
0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e−5
$r^2\ [m^2]$
$\frac{v}{(\rho_k - \rho_l)}\ [\frac{m^4}{s \cdot kg}]$

```
Error of Lambda:  [0.00028    0.00042    0.00056    0.0007     0.00084    0.00100016
 0.00112    0.00126   ]
Corrected y-values:  [1.05721500e-05 2.18078346e-05 3.72717733e-05 5.83002816e-05
 7.85932987e-05 1.15828133e-04 1.43968676e-04 1.85303903e-04]
Error corrected y-values:  [1.12625959e-06 2.68721372e-06 2.36741655e-06 4.82755016e-06
 8.10727650e-06 8.22110284e-06 5.89152904e-06 9.45591291e-06]
```

Now we can plot the corrected values in the same diagram

In [8]:
```
plt.errorbar(x_values, y_values, linestyle="None", marker = ".", yerr = err_y, xerr = err_x, label="original values", capsize = 2)
plt.errorbar(x_values, y_values_corr, linestyle="None", marker = ".", yerr = err_y_corr, xerr = err_x, color="green", label="corre
plt.xlabel("$r^2$ [$m^2$] ")
plt.ylabel("$\\frac{v}{(\\rho_k - \\rho_f)}$ [$\\frac{m^4}{s\\cdot kg}$]")
plt.legend()
```

Out[8]:
```
<matplotlib.legend.Legend at 0x24607bf1670>
```

Now we can do a fit and determine the viscosity η by defining the function v(r)

In [9]:
```
def fit(x, eta):
    return (2/9)*(9.81*x)/(eta)

# Now we can perform the fit
popt, pcov = curve_fit(fit, x_values, y_values_corr, sigma = err_y_corr)

eta = popt
err_eta = np.sqrt(pcov[0])

print("Viscosity: ",eta, "[Pas]")
print("Error of the viscosity: ",err_eta, "[Pas]")
```

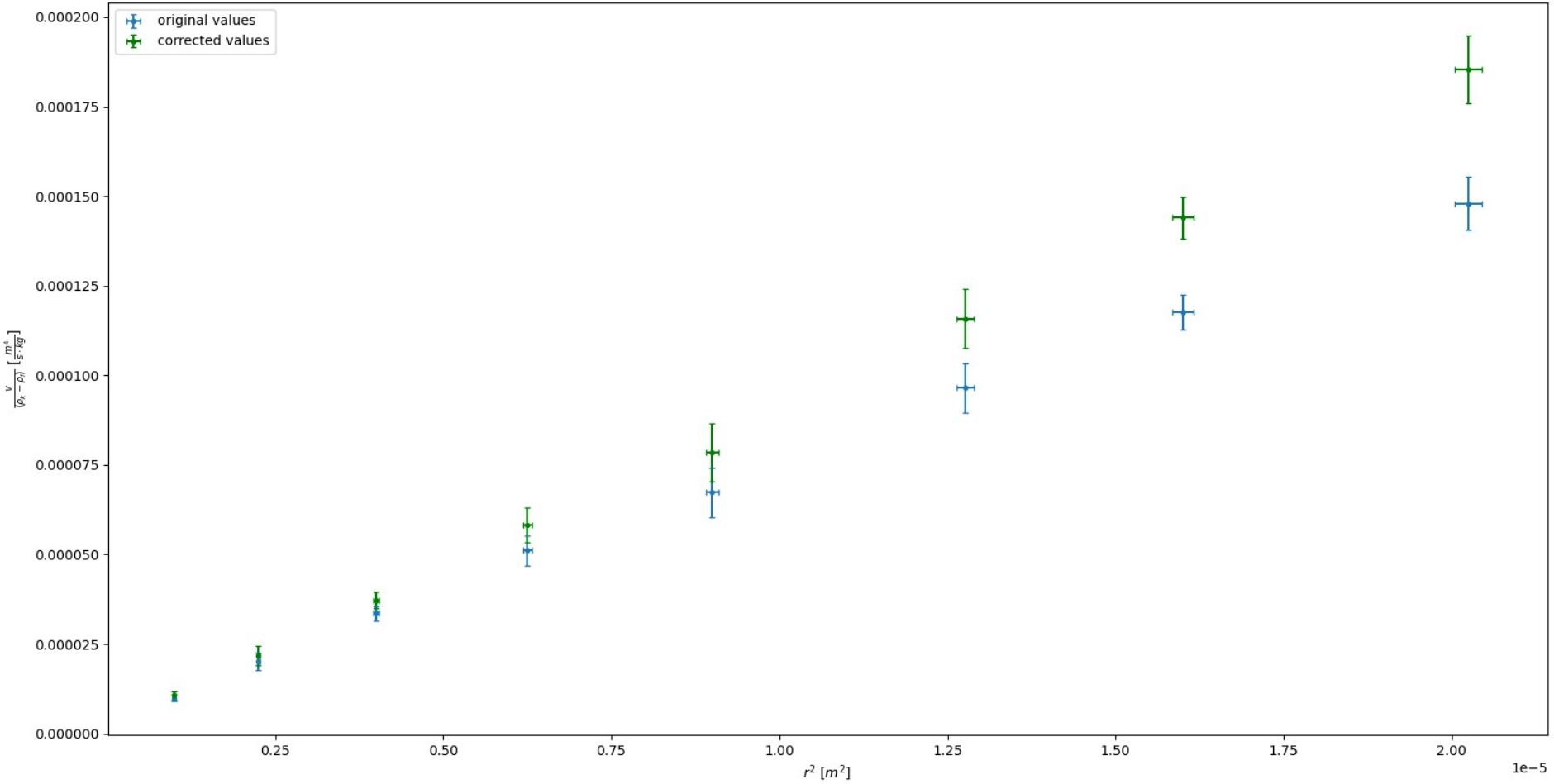
```
Viscosity:  [0.23768138] [Pas]
Error of the viscosity:  [0.00320927] [Pas]
```

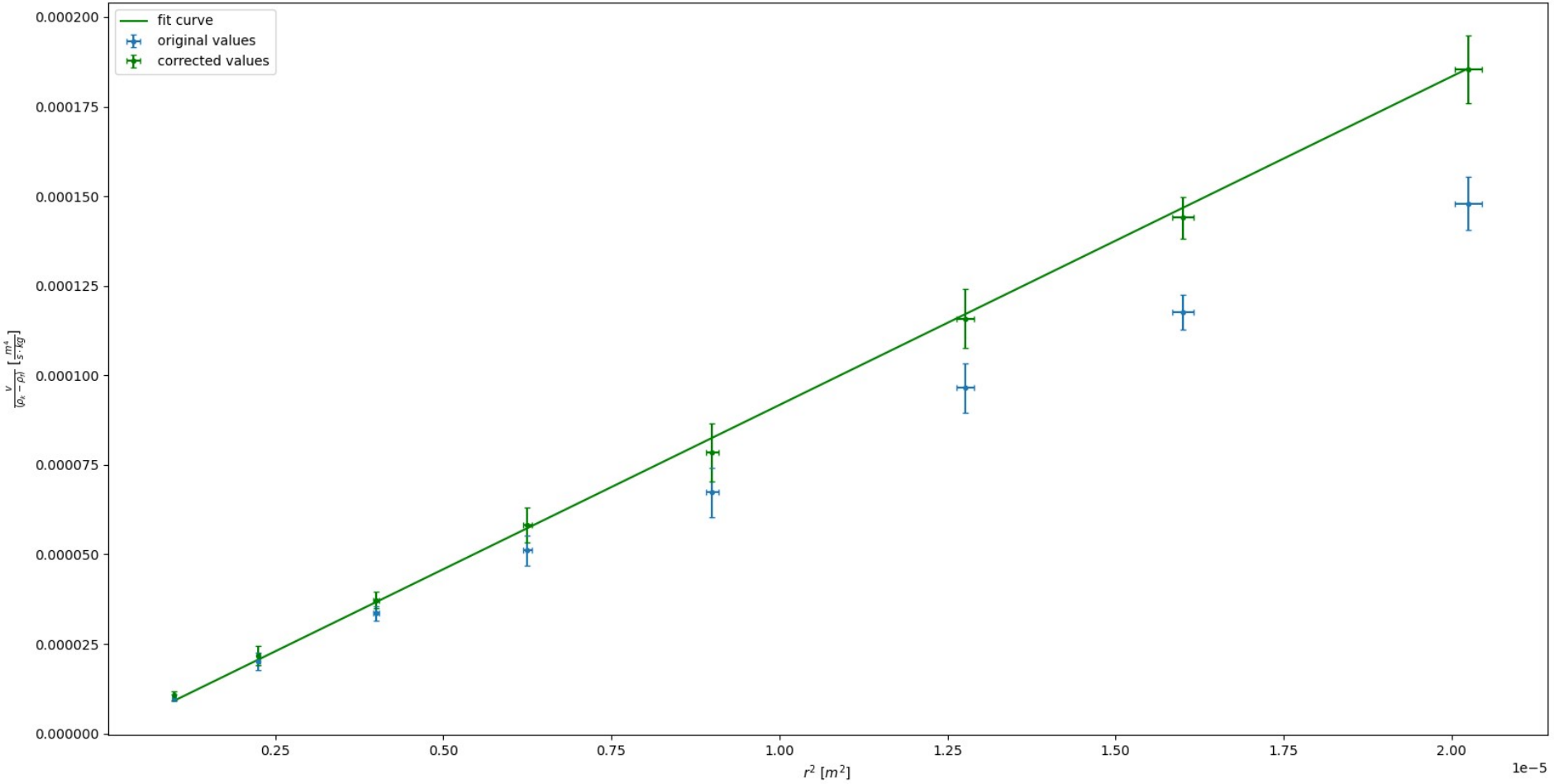
Our result for the viscosity of polyethlyene glycol with the falling sphere viscometer is

$\eta = 0.238 \pm 0.003 Pa \cdot s$

We will enter the fitted curve into our diagram.

In [10]:
```
plt.errorbar(x_values, y_values, linestyle="None", marker = ".", yerr = err_y, xerr = err_x, label="original values", capsize = 2)
plt.errorbar(x_values, y_values_corr, linestyle="None", marker = ".", yerr = err_y_corr, xerr = err_x, color="green", label="corre
plt.plot(x_values, fit(x_values, eta), color="green", label="fit curve")
plt.xlabel("$r^2$ [$m^2$] ")
plt.ylabel("$\\frac{v}{(\\rho_k - \\rho_f)}$ [$\\frac{m^4}{s\\cdot kg}$]")
plt.legend()
```


original values
corrected values
0.000200
0.000175
0.000150
0.000125
0.000100
0.000075
0.000050
0.000025
0.000000
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e−5
$r^2\ [m^2]$
$\frac{v}{(\rho_k - \rho_l)}\ [\frac{m^4}{s \cdot kg}]$


fit curve
original values
corrected values
0.000200
0.000175
0.000150
0.000125
0.000100
0.000075
0.000050
0.000025
0.000000
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e−5
$r^2\ [m^2]$
$\frac{v}{(\rho_k - \rho_l)}\ [\frac{m^4}{s \cdot kg}]$

Out[10]: <matplotlib.legend.Legend at 0x24607e963a0>

Calculation of the theroetical values v_{lam} with our determined viscosity and determining the critical Reynolds number

We can define a function that calculates v_{lam} for each radius (without the factor lambda):

In [11]:
```
def vlam(r, rho_k):
    return (2/9)*9.81*(rho_k - rho_f)/(eta)*(r)**2
def err_vlam(vlam, r, rho_k):
    return ( np.sqrt( (err_eta / eta)**2 + (2.5 / (rho_k - rho_f))**2 + (err_rho_f / (rho_k - rho_f))**2 + (2*0.01*r / r)**2 ) ) *

# Now we can calculate the theoretical velocities:
vlamarr = vlam(r,rho_k)
err_vlamarr = err_vlam(vlamarr, r, rho_k)

print("Theoretical laminar velocities: ", vlamarr, "[m/s]")
print("Error of the theoretical laminar velocities: ",err_vlamarr, "[m/s]")
```

```
Theoretical laminar velocities:  [0.00210863 0.00474442 0.00843452 0.01317893 0.01897767 0.02690439
 0.03080305 0.03991377] [m/s]
Error of the theoretical laminar velocities:  [5.68865272e-05 1.27994686e-04 2.27546109e-04 3.55540795e-04
 5.11978745e-04 7.25825667e-04 8.47424307e-04 1.09223096e-03] [m/s]
```

Next we have to calculate $\frac{v}{v_{lam}}$ and its error:

In [12]:
```
# First we define functions that calculate the values we want
def vfrac(v,vlam):
    return v / vlam
def err_vfrac(vfrac, v, vlam, err_v, err_vlam):
    return np.sqrt( (err_v / v)**2 + (err_vlam / vlam)**2 ) * vfrac

# Now we can calculate the fraction for each diameter:
vfracarr = vfrac(meanvelocityarr, vlamarr)
err_vfracarr = err_vfrac(vfracarr, meanvelocityarr, vlamarr, err_meanvelocityarr, err_vlamarr)


print("Velocity fractions: ",vfracarr)
print("Error of the velocity fractions: ",err_vfracarr)
```

```
Velocity fractions:  [1.09153603 0.97485364 0.9135955  0.89212243 0.81515245 0.82477788
 0.80150325 0.79688176]
```

```
Error of the velocity fractions:  [0.1192278  0.122405   0.06207448 0.07694313 0.08635505 0.06182709
 0.03807024 0.04497466]
```

Since our goal is to plot these fractions against the Reynolds number, we have to calculate the Reynolds Number for each diameter with equation (4).

In [13]:
```
def reynolds(v,d):
    return (rho_f * v * d)/(eta)
def err_reynolds(reynolds, v, d, err_v):
    return np.sqrt( (err_v / v)**2 + (0.01*d / d)**2 + (err_rho_f / rho_f)**2 + (err_eta / eta)**2 ) * reynolds

reynoldsarr = reynolds(meanvelocityarr,  2*r)
err_reynoldsarr = err_reynolds(reynoldsarr, meanvelocityarr, 2*r, err_meanvelocityarr)

print("Reynolds numbers: ",reynoldsarr)
print("Error of the reynolds numbers: ",err_reynoldsarr)
```

```
Reynolds numbers:  [0.02222612 0.06699446 0.14882284 0.28383774 0.44815502 0.76541508
 0.95363985 1.38214795]
Error of the reynolds numbers:  [0.0023821  0.00829258 0.0096128  0.02373771 0.04652685 0.05506158
 0.0402642  0.07208219]
```

Now we can plot the fractions against the reynolds number on a logarithmic scale

In [14]:
```
plt.errorbar(reynoldsarr , vfracarr, marker = ".", linestyle="None", yerr=err_vfracarr, xerr=err_reynoldsarr, ecolor="Red", capsiz
plt.xlabel("$Re$")
plt.ylabel("$\\frac{v}{v_{lam}}$")
#plt.xscale=("log")
```

Out[14]:
```
Text(-1.777777777777784, 0.5, '$\\frac{v}{v_{lam}}$')
```

Our result for the critical Reynolds Number is

$Re_{cr} = 0.45 \pm 0.07$

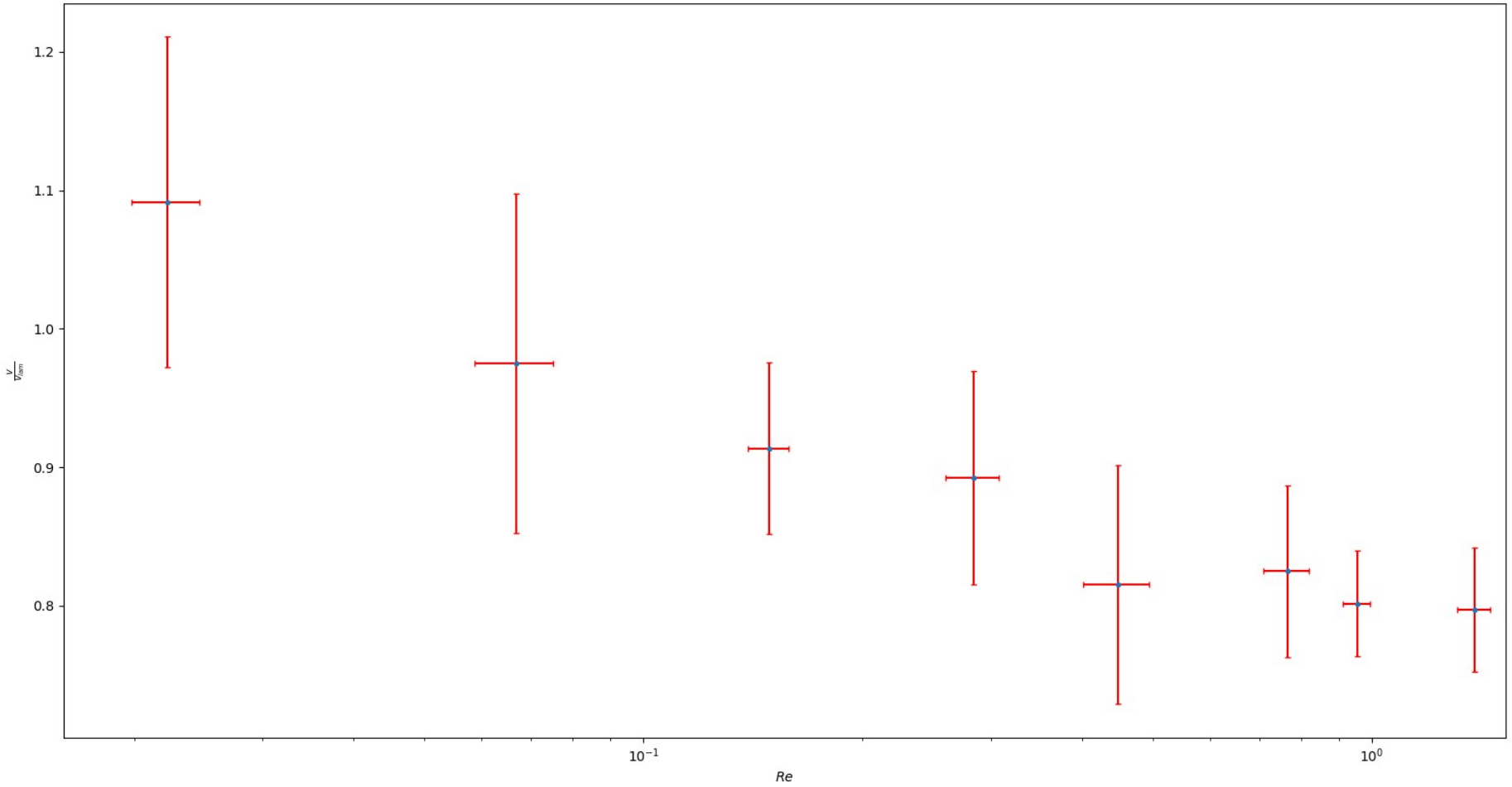
The result was estimated from the diagram.

We are also supposed to estimate the critical Reynolds number. In the instructions we are given, that when a ball is moved through the viscometer, the critical Reynolds number is approximately equal to 1.

Determining the viscosity after Hagen-Poiseuille

In the second part of the experiment, we have to determine the viscosity through equation (14):

In [15]:
```
# First, we enter our measured values for the height and time
h_a = 0.550 # Unit is meters
```


1.2
1.1
1.0
0.9
0.8
$\frac{v}{v_{lam}}$
10^{-1}
10^{0}
Re

```
h_e = 0.544
err_h = 0.001

V = np.array([0.000005 , 0.00001, 0.000015, 0.00002, 0.000025]) # Volume in m^3
t = np.array([152, 412, 625, 855, 1114]) # Times in seconds

# We define a linear function to determine dV / dt later
def linear(t, dVdt):
    return dVdt * t

p_0 = 101260 #Pa - air pressure
err_p_0 = 0.2

# Diameter of the capillary viscometer
d_cap = 0.0015 # meters
err_d_cap = 0.00001

# Length of the capillary viscometer
L = 0.1 # meters
err_L = 0.0005
```

Now we calculate $\frac{\Delta V}{\Delta t}$

In [16]:
```
#dVdt = (np.pi * (0.5*d_inner)**2 * (h_a - h_e)) / delta_t
#err_dVdt = np.sqrt( (np.pi * (0.5*d_inner)**2 * err_h / delta_t)**2 + (np.pi * (0.5*d_inner)**2 * err_h / delta_t)**2 + (np.pi *

# We perform the fit to determine dV / dt
popt, pcov = curve_fit(linear, t, V)

dVdt = popt
err_dVdt = np.sqrt(pcov[0])
print("dV / dt: ", dVdt, "[m^3/s]")
print("Error dV / dt: ",err_dVdt, "[m^3/s]")
```

```
dV / dt:  [2.3167732e-08] [m^3/s]
Error dV / dt:  [5.72851323e-10] [m^3/s]
```

Next we have to calculate the pressure inside the capillary viscometer with the mean of the two heights

In [17]:
```
mean_h = (h_a + h_e)/2
err_mean_h = np.sqrt( (err_h / 2)**2 + (err_h / 2)**2 )
print("Mean height: ",mean_h, "m")
print("Error mean height: ",err_mean_h, "m")
```

```
# Hydrostatic pressure minus p_0
p_h = rho_f * 9.81 * mean_h
err_p_h = np.sqrt( (rho_f * 9.81 * err_mean_h)**2 + (9.81 * mean_h * err_rho_f)**2)
print("Pressure difference: ",p_h, "Pa")
print("Error of the pressure difference: ",err_p_h, "Pa")
```

```
Mean height:  0.547 m
Error mean height:  0.0007071067811865475 m
Pressure difference:  6158.1019320000005 Pa
Error of the pressure difference:  10.238904486400095 Pa
```

Now we can use equation (14):

In [18]:
```
def hagenPoiseuille(p_h, d_cap, L, dVdt):
    return np.pi * p_h * (0.5*d_cap)**4 / (8*L*dVdt)
def err_hagenPoiseuille(eta_HP, p_h, d_cap, L, dVdt, err_p_h, err_d_cap, err_L, err_dVdt):
    return np.sqrt( (err_p_h / p_h)**2 + (4*0.5*err_d_cap / (0.5*d_cap))**2 + (err_L / L)**2 + (err_dVdt / dVdt)**2 ) * eta_HP

eta_HP = hagenPoiseuille(p_h, d_cap, L, dVdt)
err_eta_HP = err_hagenPoiseuille(eta_HP, p_h, d_cap, L, dVdt, err_p_h, err_d_cap, err_L, err_dVdt)

print("Viscosity after Hagen-Poiseuille: ",eta_HP, "[Pas]")
print("Error of the viscosity: ",err_eta_HP, "[Pas]")
```

```
Viscosity after Hagen-Poiseuille:  [0.33026937] [Pas]
Error of the viscosity:  [0.01213606] [Pas]
```

Our result for the viscosity with Hagen-Poiseuille is:

$\eta_{HP} = 0.330 \pm 0.012 Pa \cdot s$

Out next goal is to calculate the reynolds number of the capillary:

In [19]:
```
def reynoldsHPcalc(v_cap,d):
    return (rho_f * v_cap * d)/(eta_HP)
def err_reynoldsHP(reynoldsHP, v_cap, d_cap, err_v_cap, err_d_cap):
    return np.sqrt( (err_v_cap / v_cap)**2 + (err_d_cap / d_cap)**2 + (err_rho_f / rho_f)**2 + (err_eta_HP / eta_HP)**2 ) * reynol

# We calculate the Area perpendicular to the capillary viscometer
A = np.pi * (0.5 * d_cap)**2
err_A = np.pi * 2 * 0.5 * d_cap *  0.5 * err_d_cap

print("Area perpendicular to the capillary viscometer: ", A, "[m^2]")
print("Error of Area: ", err_A, "[m^2]")
```

```
# We calculate v
v_cap = dVdt / (np.pi * (0.5 * d_cap)**2)
err_v_cap = np.sqrt( (err_dVdt / dVdt)**2 + (err_A / A)**2 ) * v_cap

print("Capillary velocity: ",v_cap, "[m/s]")
print("Error Capillary velocity: ", err_v_cap, "[m/s]")

reynoldsHP = reynoldsHPcalc(v_cap, d_cap)
err_reynoldsHP = err_reynoldsHP(reynoldsHP, v_cap, d_cap, err_v_cap, err_d_cap)
print("Reynolds number: ",reynoldsHP)
print("Error Reynolds number: ", err_reynoldsHP)
```

```
Area perpendicular to the capillary viscometer:  1.7671458676442586e-06 [m^2]
Error of Area:  2.356194490192345e-08 [m^2]
Capillary velocity:  [0.01311025] [m/s]
Error Capillary velocity:  [0.00036829] [m/s]
Reynolds number:  [0.06833208]
Error Reynolds number:  [0.00319409]
```

$Re_{HP} = 0.068 \pm 0.003$

We can see that it was in fact laminar flow, because the Reynolds number was lower than the approximate 2300 at which turbulent flow occurs in the capillary viscometer.

Comparing our two determined viscosities

We will calculate the deviation of our two values:

In [20]:
```
sigma = (eta_HP - eta)/(np.sqrt(err_eta**2 + err_eta_HP**2))

print("Deviation: ", sigma)
```

```
Deviation:  [7.37563738]
```

In [ ]:

Summary and discussion

Our goals in this experiment were to determine the viscosity of polyethylene glycol with a falling sphere viscometer after Stoke's law, as well as with a capillary viscometer according to Hagen-Poiseuille. Additionally, we will check the validity limit of Stoke's law by determining the transition from laminar to turbulent flow.
First, we determined the sinking velocities for each ball diameter. Our results were

$$v_{2mm} = 2,30 \pm 0,24 \frac{mm}{s}$$

$$v_{3mm} = 4,6 \pm 0,6 \frac{mm}{s}$$

$$v_{4mm} = 7,7 \pm 0,5 \frac{mm}{s}$$

$$v_{5mm} = 11,8 \pm 1,0 \frac{mm}{s}$$

$$v_{6mm} = 15,5 \pm 1,6 \frac{mm}{s}$$

$$v_{7,144mm} = 22,2 \pm 1,6 \frac{mm}{s}$$

$$v_{8mm} = 24,7 \pm 1,0 \frac{mm}{s}$$

$$v_{9mm} = 31,8 \pm 1,6 \frac{mm}{s}$$

After adding a correction factor to compensate for the fact that our liquid is not infinite, we have determined the viscosity after Stokes as

$$\eta_S = 0,238 \pm 0,003 Pa \cdot s$$

We also determined the critical Reynolds number, where turbulent flow occurs, as approximately

$$Re_{cr} = 0,45 \pm 0,07$$

For the second part of the experiment, we determined the viscosity after Hagen-Poiseuille as

$$\eta_{HP} = 0,330 \pm 0.012$$

Our result for the Reynolds number in this part of the experiment was

$$Re_{HP} = 0,068 \pm 0,003$$

which showed that in the experiment laminar flow occured. The deviation of the two viscosity values was

$$\sigma = 7,38$$

There were several results that did not quite match our expectations. First, the deviation between the viscosity values was way too high. One explanation might be that we used all diameters to calculate the viscosity after Stokes, because the graph seemed very linear for all values, although for the higher diameters, Stokes law should now have been applicable, which we could see through the critical Reynolds number. If we ignored the last few values, the result may have been different, but we could not determine a clearly linear part in the graph, which is why we did not do that.
In the second part of the experiment, our time differences from 5 to 25 cm^3 were not very evenly spaced, so it might be that our calculated $\frac{dV}{dt}$ was imprecise, which would make the second viscosity inaccurate.
Additionally, the critical Reynolds for the falling sphere viscometer number did not match the given critical Reynolds number in the instruction, which was approximately 1. Our results for the Reynolds Numbers did not quite make up a good linear function (on the logarithmic scale), so it was hard approximating a critical Reynolds number.