Auswertung Versuch 212 PAP 2.1.

In [2]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from scipy.optimize import curve_fit
from scipy.stats import chi2
from scipy.odr import Data, RealData, Model, ODR
import pandas as pd
```

In [3]:

```
rho_f = 1.1466*1e-3
Drho_f = 0.0006e-3
Drho_k = 0.0025*1e-3
g = 9.81
R = 75e-3/2
```

In [4]:

```
def get_rho_k(radius):
    if 2*radius < 2e-3:
        return 1.3925e-3
    elif 2*radius <= 7.144e-3:
        return 1.3775e-3
    elif 2*radius <= 8e-3:
        return 1.3575e-3
    else:
        return 1.3625e-3

v_rho_k = np.vectorize(get_rho_k)
```

In [5]:

```
diameter = np.array([1.5, 2, 3, 4, 5, 6, 7.144, 8, 9])*1e-3
r = diameter/2
Dr = 1e-2*r
```

In [6]:

```
distance = np.array([20, 20, 20, 30, 30, 30, 30, 30, 30])*1e-2
```

In [7]:

```
rho_k = v_rho_k(r); print(rho_k)
```

```
[0.0013925 0.0013775 0.0013775 0.0013775 0.0013775 0.0013775 0.0013775
 0.0013575 0.0013625]
```

In [8]:

```
times = np.loadtxt('times.txt')
```

In [9]:

```
avg_time = np.average(times, axis = 1); print(avg_time)
Davg_time_std = 1/np.sqrt(5)*np.std(times, axis = 1)
Davg_time = Davg_time_std; print(Davg_time)
```

```
[124.48    78.68    39.694   34.45    22.656   16.53    12.234   11.116    8.692]
[2.98179543 0.63563512 0.32441455 0.23044305 0.14429414 0.10590562
 0.06762248 0.01930803 0.03615522]
```

In [10]:

```
v = distance/avg_time; print(v)
Dv = v*Davg_time/avg_time; print(Dv/v*100)
```

```
[0.00160668 0.00254194 0.00503854 0.00870827 0.01324153 0.01814882
 0.02452182 0.02698813 0.0345145 ]
[2.39540121 0.80787381 0.81728863 0.66892032 0.63689153 0.64068735
 0.55274221 0.17369584 0.41595974]
```

In [11]:

```
def get_ladenburg(radius):
    return 1 + 2.1*radius/R

ladenburg = get_ladenburg(r); print(ladenburg)
```

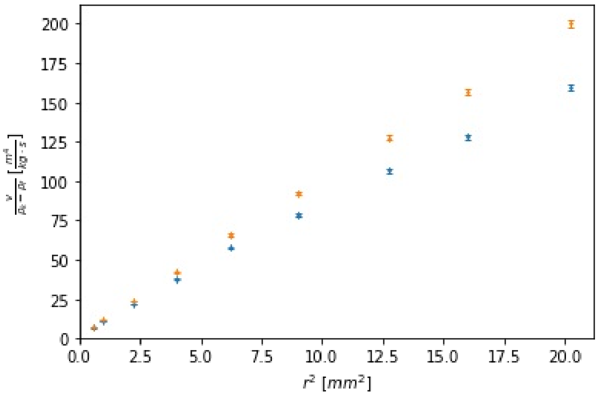
```
[1.042     1.056     1.084     1.112     1.14      1.168     1.200032 1.224
 1.252    ]
```

In [12]:

```
Dy = v/(rho_k - rho_f)*np.sqrt((Dv/v)**2 + (Drho_k/(rho_k - rho_f))**2)
```

In [13]:

```
plt.errorbar(r**2*1e6, v/(rho_k - rho_f), xerr = 0, yerr = Dy, marker = '.', linestyle = 'none', markersize = 3,
elinewidth = 1, capsize = 2)
plt.errorbar(r**2*1e6, ladenburg*v/(rho_k - rho_f), xerr = 0, yerr = ladenburg*Dy, marker = '.', linestyle = 'non
e', markersize = 3, elinewidth = 1, capsize = 2)
plt.xlabel(r"$r^2$ [${mm}^2$]")
plt.ylabel(r"$\frac{v}{\rho_k - \rho_f}$ [$\frac{m^4}{kg \cdot s}$]")
plt.xlim(left = 0)
plt.ylim(bottom = 0)
plt.savefig('figures/v_for_radii.pdf', format = 'pdf')
```



In [14]:

```
def fit_func_v(radius, visc):
    return 1/get_ladenburg(radius)*2/9*g*(v_rho_k(radius) - rho_f)/visc*radius**2
```

In [15]:

```
# number of radii for which Stoke's law seems to hold:
num_linear = 6
popt, pcov = curve_fit(fit_func_v, r[0:num_linear], v[0:num_linear], sigma = Dv[0:num_linear], p0 = [1])
eta = popt[0]
print("Viskosität: ", eta, " +- ", np.sqrt(pcov[0][0]))
```

```
('Viskosit\xc3\xa4t: ', 2.062107593436122e-07, ' +- ', 3.9655094031804O4e-09)
```

In [16]:

```
chi2_ = np.sum((fit_func_v(r[0:num_linear], *popt) - v[0:num_linear])**2/Dv[0:num_linear]**2)
dof = num_linear - 1
chi2_red = chi2_/dof
print("chi2 = ", chi2_)
print("chi2_red = ", chi2_red)
```

```
('chi2 = ', 190.83623818554094)
('chi2_red = ', 38.16724763710819)
```

In [17]:

```
def fit_func_odr(visc, radius):
    return 1/get_ladenburg(radius)*2/9*g*(v_rho_k(radius) - rho_f)/visc[0]*radius**2
```

In [18]:

```
stokes = Model(fit_func_odr)
data = RealData(r[0:num_linear], y = v[0:num_linear], sx = Dr[0:num_linear], sy = Dv[0:num_linear])
```

In [19]:

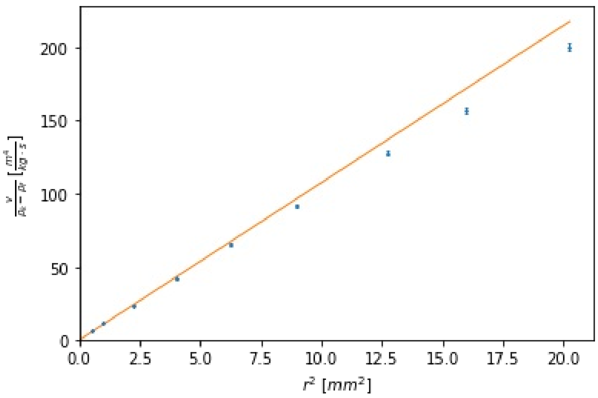
```
odr = ODR(data, stokes, beta0 = [1])
output = odr.run()
output.pprint()
eta_odr = output.beta[0]
Deta_odr = output.sd_beta[0]
```

```
Beta: [2.02949001e-07]
Beta Std Error: [5.02864962e-09]
Beta Covariance: [[3.17351616e-18]]
Residual Variance: 7.96823325624
Inverse Condition #: 1.0
Reason(s) for Halting:
  Sum of squares convergence
```

In [20]:

```
plt.errorbar(r**2*1e6, ladenburg*v/(rho_k - rho_f), xerr = 0, yerr = ladenburg*Dy, marker = '.', linestyle = 'non
e', markersize = 3, elinewidth = 1, capsize = 1)

r_cont = np.linspace(0, 4.5e-3, 100)
plt.plot(r_cont**2*1e6, get_ladenburg(r_cont)*fit_func_v(r_cont, eta_odr)/(v_rho_k(r_cont) - rho_f), linewidth =
1)
plt.xlabel(r"$r^2$ [${mm}^2$]")
plt.ylabel(r"$\frac{v}{\rho_k - \rho_f}$ [$\frac{m^4}{kg \cdot s}$]")
plt.xlim(left = 0)
plt.ylim(bottom = 0)
plt.savefig('figures/v_for_radii_fit.pdf', format = 'pdf')
```



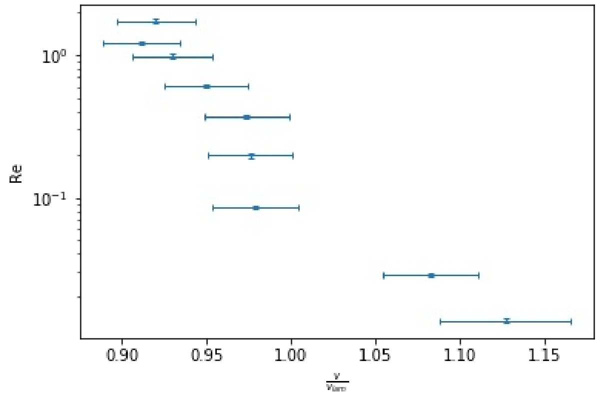
In [21]:

```
Re = rho_f*v*2*r/eta_odr; print(Re)
DRe = Re*np.sqrt((Dr/r)**2 + (Dv/v)**2 + (Deta_odr/eta_odr)**2); print(DRe/Re)
```

```
[0.01361591 0.0287224  0.08539873 0.19679635 0.37405291 0.61521182
 0.98973572 1.21979746 1.75496745]
[0.03588508 0.02791434 0.02794173 0.02754432 0.0274683  0.02747712
 0.02728547 0.02677613 0.02704157]
```

In [22]:

```
v_lam = fit_func_v(r, eta_odr)
Dv_lam = v_lam*Deta_odr/eta_odr
ratio = v/v_lam
Dratio = ratio*np.sqrt((Dv/v)**2 + (Dv_lam/v_lam)**2)
fig, ax = plt.subplots(1,1)
ax.errorbar(ratio, Re, xerr = Dratio, yerr = DRe, marker = '.', linestyle = 'none', markersize = 3, elinewidth = 
1, capsize = 2)
ax.set_yscale('log')
ax.set_xlabel(r"$\frac{v}{v_{lam}}$")
ax.set_ylabel("Re")
fig.savefig('figures/reynolds.pdf', format = 'pdf')
```



In [23]:

```
print("Kritische Reynoldszahl liegt zwischen ", Re[1], " und ", Re[2])
print("Re_krit = ", (Re[1] + Re[2])/2, " +- ", (Re[2] - Re[1])/2)
```

```
('Kritische Reynoldszahl liegt zwischen ', 0.028722395617467544, ' und ', 0.08539872854268957)
('Re_krit = ', 0.05706056208007856, ' +- ', 0.02833816646261101)
```

In [24]:

```
table_data = np.vstack((fit_func_v(r, eta_odr)*1e2, v*1e2, v/fit_func_v(r, eta_odr), Re))
pd.DataFrame(table_data, index = ["$v_{\text{lam}}$", "$v$", r"$\frac{v}{v_{\text{lam}}}$", "$Re$"])
```

Out[24]:

	0	1	2	3	4	5	6	7	8
v_{lam}	0.142588	0.234871	0.514810	0.892172	1.359780	1.911143	2.637082	2.961316	3.750964
v	0.160668	0.254194	0.503854	0.870827	1.324153	1.814882	2.452182	2.698813	3.451450
$\frac{v}{v_{\text{lam}}}$	1.126802	1.082271	0.978720	0.976075	0.973799	0.949632	0.929885	0.911356	0.920150
Re	0.013616	0.028722	0.085399	0.196796	0.374053	0.615212	0.989736	1.219797	1.754967

In [25]:

```
table_err = np.vstack((fit_func_v(r, eta_odr)*Deta_odr/eta_odr*1e2, Dv*1e2, DRe))
pd.DataFrame(table_err)
```

Out[25]:

	0	1	2	3	4	5	6	7	8
0	0.003533	0.005820	0.012756	0.022106	0.033692	0.047354	0.065341	0.073375	0.092941
1	0.003849	0.002054	0.004118	0.005825	0.008433	0.011628	0.013554	0.004688	0.014357
2	0.000489	0.000802	0.002386	0.005421	0.010275	0.016904	0.027005	0.032661	0.047457

Hagen-Poiseuille

In [26]:

```
h_A = 540e-3
h_E = 534e-3
h = (h_A + h_E)/2
Dh = 2e-3/np.sqrt(2); Dh/h
```

Out[26]:

```
0.0026335448088884448
```

In [27]:

```
rho_f = 1.1460e-3
```

In [28]:

```
# pressure difference:
p = h*rho_f*g
Dp = p*np.sqrt((Dh/h)**2 + (Drho_f/rho_f)**2); Dp/p
```

Out[28]:

```
0.0026850835281822176
```

In [29]:

```
# capillary:
L = 100e-3
DL = 0.5e-3
R = 1.5e-3/2
DR = 0.01e-3/2
```

In [30]:

```
vol = np.array([0, 5, 10, 15, 20, 25])*1e-6
time = np.array([0, 104, 236, 364, 498, 630])
Dvol = 0.5e-6
Dtime = 5.0
```

In [31]:

```
# check for errors:
flow = (np.concatenate((vol[1:], [0])) - np.concatenate(([0], vol[1:]))) / (np.concatenate((time[1:], [0])) - np.
concatenate(([0], time[1:])))
print(flow)
```

```
[4.80769231e-08 3.78787879e-08 3.90625000e-08 3.73134328e-08
 3.78787879e-08 3.96825397e-08]
```

In [32]:

```
total_time = time[5] - time[0]
total_vol = vol[5] - vol[0]
avg_flow = total_vol/total_time
print(avg_flow)

Dtotal_time = np.sqrt(2)*Dtime
Dtotal_vol = np.sqrt(2)*Dvol
Davg_flow = avg_flow*np.sqrt((Dtotal_time/total_time)**2 + (Dtotal_vol/total_vol)**2)
print(Davg_flow)
```

```
3.968253968253968e-08
1.2075340857971035e-09
```

In [33]:

```
eta_hp = np.pi*p*R**4 / (8*avg_flow*L)
Deta_hp = eta_hp * np.sqrt((4*DR/R)**2 + (Davg_flow/avg_flow)**2 + (DL/L)**2)
print("Vsicosity: ", eta_hp, " +- ", Deta_hp)
```

```
('Vsicosity: ', 1.8903115623357138e-07, ' +- ', 7.706555722928348e-09)
```

In [34]:

```
# Reynolds number:
Re_hp = rho_f*avg_flow*2*R / (np.pi*R**2*eta_hp)
print(Re_hp)
```

```
0.2042064924177512
```

In [35]:

```
# deviation:
combined_error = np.sqrt(Deta_odr**2 + Deta_hp**2)
sigma_val = abs(eta_hp - eta_odr) / combined_error
print(sigma_val)
```

```
1.512466882998919
```