# 2025-10-17-file-1

October 27, 2025

## 1 Data Analysis Problems

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import math
```

[4]:
```
#Some of the code in the cells below were taken and adapted from AI
```

## 2 Q1)

[5]:
```
# calculating vaues of v, then calculating the mean values for d,t and v

# defining d(m) and t(s)
d = np.array([5.97, 5.95, 5.96, 5.95, 5.94, 5.95, 6.00, 5.96, 5.96, 5.97, 5.94,␣
 ↪5.96])
t = np.array([1.75, 1.75, 1.76, 1.74, 1.76, 1.75, 1.76, 1.75, 1.75, 1.73, 1.77,␣
 ↪1.75])

v = d/t #v values

#calculate mean values

d_mean = (5.97+ 5.95+ 5.96+ 5.95+ 5.94+ 5.95+ 6.00+ 5.96+ 5.96+ 5.97 +5.94+ 5.
 ↪96)/12
t_mean = (1.75+ 1.75+ 1.76+ 1.74+ 1.76+ 1.75+ 1.76+ 1.75+ 1.75+ 1.73+ 1.77+ 1.
 ↪75)/12
v_mean = np.sum(v)/12

#printing alues
print('The velocity values are:',v ,'m/s')
print('The mean time values are:',t_mean, 's')
print('The mean distance values are:',d_mean, 'm')
print('The mean velocity values are:',v_mean, 'm/s')
```

```
The velocity values are: [3.41142857 3.4        3.38636364 3.41954023 3.375
3.4
```

```
 3.40909091 3.40571429 3.40571429 3.45086705 3.3559322  3.40571429] m/s
The mean time values are: 1.7516666666666667 s
The mean distance values are: 5.959166666666666 m
The mean velocity values are: 3.402113788276999 m/s
```

[6]:
```
#calculating the uncertainties on d,t and v

#calculating standard deviations of d,t and v

d_std = np.std(d,ddof=1)
t_std = np.std(t,ddof=1)
v_std = np.std(v,ddof=1)

#calculating the standard error on the mean d,t and v
dd = d_std/np.sqrt(len(d))
dt = t_std/np.sqrt(len(t))
dv = v_std/np.sqrt(len(v))

#mean time, velocity and distance with errors
print('The mean time values are:',f"{t_mean} ± {dt}", 's')
print('The mean distance values are:',f"{d_mean} ± {dd}", 'm')
print('The mean velocity values are:',f"{v_mean} ± {dv}", 'm/s')
```

```
The mean time values are: 1.7516666666666667 ± 0.0029729419500528183 s
The mean distance values are: 5.959166666666666 ± 0.004680445027619198 m
The mean velocity values are: 3.402113788276999 ± 0.006730638631330154 m/s
```

[7]:
```
#Calculate the mean value of v, from the mean values of d and t

v_mean2 = d_mean/t_mean

#finding uncertainty of new velocity
dv2 = np.sqrt(v_mean2*((dd/d_mean)**2 + (dt/t_mean)**2))

#printing new velocity with error
print('The mean velocity values are:',f"{v_mean2} ± {dv2}", 'm/s')
```

```
The mean velocity values are: 3.4019980970504275 ± 0.0034493678546870044 m/s
```

As we are using a normal approximation, the t-distribution for a 95% confidence level is aproximately equal to 1.96, so the error is the standard error mean multiplied by 1.96.

[8]:
```
#Calculate the 95% confidence level limits on v using the uncertainty on the␣
 ↪mean estimated in (iv.)

print('The mean velocity values are:',f"{v_mean2} ± {dv2*1.96}", 'm/s')
```

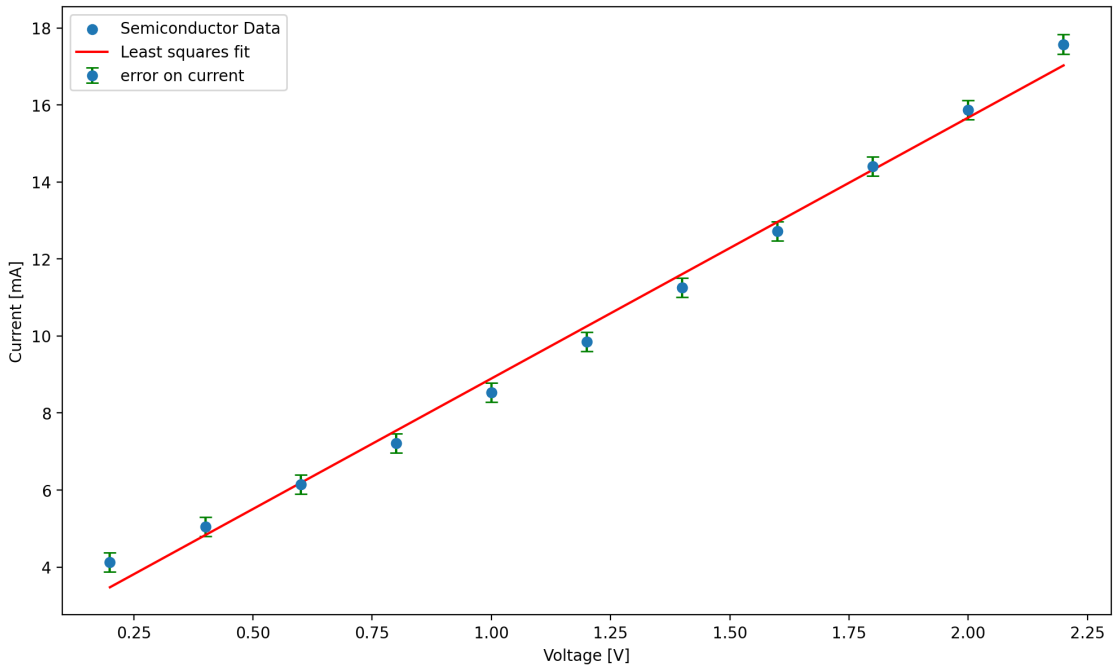
```
The mean velocity values are: 3.4019980970504275 ± 0.006760760995186529 m/s
```

## 3 Q2)

Not all of the data points with error bars touch the line of best fit suggesting overfitting or underestimated error bars. The chi-squared value is 22, so our fit may be an overestimate. However, the reduced chi-squared is only a value of 2 (quite close to 1), suggesting that there may be some overfitting, but not as much as first anticipated when just looking at the chi squared (as the reduced chi-squared unlike the chi-squared takes the number of fit parameters used into account). Therefore I think that our error bars may be underestimated.

[9]:
```
#least squares fitting plot

# defining voltage in V and current in mA
volt = np.array([0.20, 0.40, 0.60, 0.80, 1.00, 1.20, 1.40, 1.60, 1.80, 2.00, 2.
 ↪20])
current = np.array([4.12, 5.05, 6.15, 7.21, 8.54, 9.85, 11.26, 12.73, 14.41, 15.
 ↪87, 17.58])
dc = np.array([0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.
 ↪25]) #error on current

# the code below was taken and adapted from AI

#parameters for linear fit
m, c = np.polyfit(volt,current,deg=1)

#error plot
error = 0.25
plt.errorbar(volt, current, yerr=error, fmt='o', ecolor='green', capsize=4,␣
 ↪label='error on current')

#plotting data and fit
plt.scatter(volt, current, label="Semiconductor Data")
plt.plot(volt, m*volt + c, color='red', label="Least squares fit")
plt.ylabel("Current [mA]")
plt.xlabel("Voltage [V]")
plt.legend()
plt.show()

#calculating reduced chi squared
current_fit = m*volt + c
chi2 = np.sum((current - current_fit)**2 / dc**2)
dof = len(volt) - 2      # two fit parameters: m and c
chi2_reduced = chi2 / dof

#printing the chi-squared
print('The chi-squared is:',chi2)
print('The reduced chi-squared is:',chi2_reduced)
```

[9]:




```
The chi-squared is: 22.23150545454541
The reduced chi-squared is: 2.470167272727268
```

# 4 Q3)

[10]:
```
#Mean number of goals/std/variance (u for ucl, i for imperial)

#data
goals  = np.array([0, 1, 2, 3, 4, 5, 6, 7]) # number goals
u_freq = np.array([2, 3, 9, 8, 6, 2, 2, 1]) # frequency of ucl goals
i_freq = np.array([1, 4, 6, 8, 5, 4, 1, 0]) # frequency of imperial goals

#mean goals
u_mean = np.sum((goals)*(u_freq))/33
i_mean = np.sum((goals)*(i_freq))/29

#standard deviation
u_data = np.repeat(goals, u_freq) #expanded frequency table into full data set␣
 ↪for ucl
i_data = np.repeat(goals, i_freq) #expanded frequency table into full data set␣
 ↪for imperial
u_std = np.std(u_data,ddof=1) #standard deviation for ucl
i_std = np.std(i_data,ddof=1) #standard deviation for imperial
```

```
#variance
u_var = u_std**2 #variance for ucl
i_var = i_std**2 #variance for imperial

print('The mean number of goals scored for UCL is:',u_mean)
print('The standard deviation number of goals scored for UCL is:',u_std)
print('The variance of  number of goals scored for UCL is:',u_var)
print('The mean number of goals scored for Imperial is:',i_mean)
print('The standard deviation number of goals scored for UCL is:',i_std)
print('The variance of number of goals scored for UCL is:',i_var)
```

```
The mean number of goals scored for UCL is: 2.9696969696969697
The standard deviation number of goals scored for UCL is: 1.6485760614248377
The variance of  number of goals scored for UCL is: 2.7178030303030303
The mean number of goals scored for Imperial is: 2.9655172413793105
The standard deviation number of goals scored for UCL is: 1.4755812080254453
The variance of number of goals scored for UCL is: 2.1773399014778327
```

[11]:
```
#poisson frequencies

#ucl poisson frequencies
r=np.arange(0,7) #0 to 7 goals
Nu= 33 #total observations
u_p = (((u_mean)**r)*np.exp(-u_mean))/np.array([math.factorial(r) for r in r])␣
 ↪#probabillity
u_p = np.append(u_p, 1 - u_p.sum())  # fixes array length to 8 (to makethe␣
 ↪shape match with u_expect)
u_expect = Nu*u_p # expected number of goals

#imperial poisson frequencies
r=np.arange(0,7) #0 to 7 goals
Ni= 29 #total observations
i_p = (((i_mean)**r)*np.exp(-i_mean))/np.array([math.factorial(r) for r in r])␣
 ↪#probabillity
i_p = np.append(i_p, 1 - i_p.sum())  # fixes array length to 8
i_expect = Ni*i_p # expected number of goals

print('The expected poisson frequencies for UCL:',u_expect)

print('The expected poisson frequencies for Imperial:',i_expect)
```

```
The expected poisson frequencies for UCL: [1.69352235 5.0292482  7.46767156
7.39224054 5.48817858 3.25964546
 1.61335987 1.05613343]
The expected poisson frequencies for Imperial: [1.4944804  4.4319074  6.5714489
6.495915   4.81593698 2.85634883
 1.41175862 0.92220387]
```

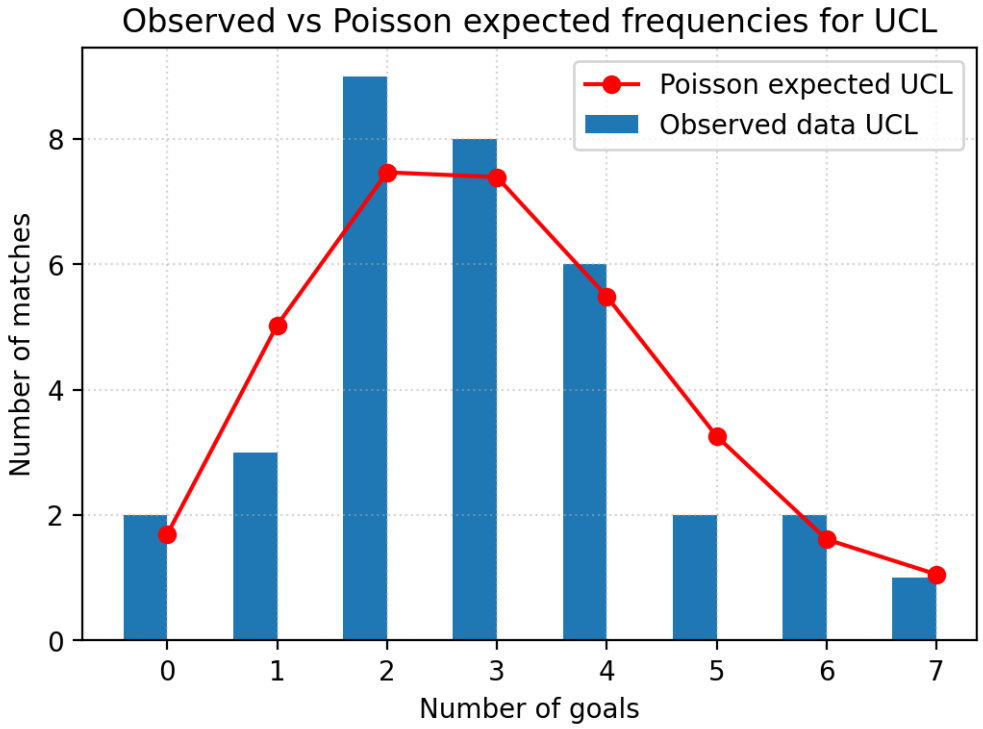
[12]:
```
import scipy.stats as stats


#chi-squared
chi2_u = np.sum((u_freq - u_expect)**2 / u_expect) #ucl
chi2_i = np.sum((i_freq - i_expect)**2 / i_expect) #imperial

# Degrees of freedom : bins - 1 - number of fitted parameters (1)
u_dof = len(u_freq) - 1 - 1  #ucl
i_dof = len(i_freq) - 1 - 1  #imperial

# Chi-square probability (p-value)
pu_value = stats.chi2.sf(chi2_u, u_dof) #ucl
pi_value = stats.chi2.sf(chi2_i, i_dof) #imperial

print("Chi-square statistic for UCL:", chi2_u)
print("Chi-square statistic for Imperial:", chi2_i)
print("p-value for UCL:", pu_value)
print("p-value for Imperial:", pi_value)
```

```
Chi-square statistic for UCL: 1.8687831158305848
Chi-square statistic for Imperial: 2.11089313788261
p-value for UCL: 0.9313622139743601
p-value for Imperial: 0.9092223258122328
```

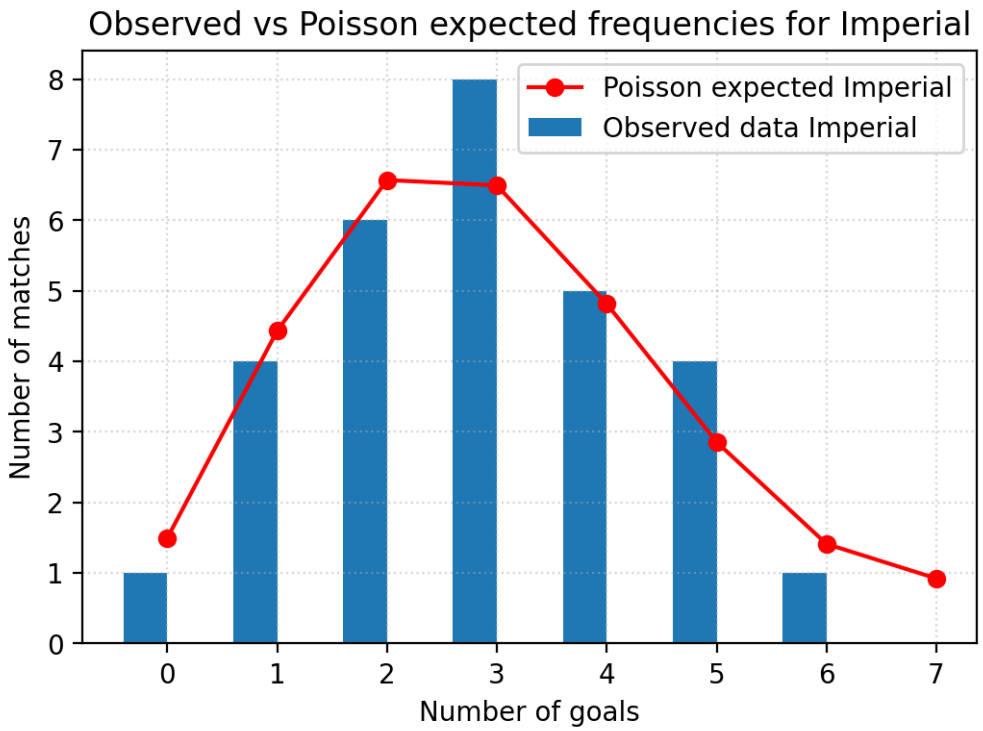
[21]:
```
#plotting probaillities and exerimental data for UCL
plt.figure(figsize=(6,4))
rs=np.arange(0,8) #0 to 7 goals

# Observed: bar chart
plt.bar(rs - 0.2, u_freq, width=0.4, label="Observed data UCL")

# Theoretical Poisson: line plot or markers
plt.plot(rs, u_expect, 'o-', color='red', label="Poisson expected UCL")
plt.xlabel("Number of goals")
plt.ylabel("Number of matches")
plt.title("Observed vs Poisson expected frequencies for UCL")
plt.xticks(rs)
plt.legend()
plt.grid(True, linestyle=':', alpha=0.5)
plt.show()


#plotting probaillities and exerimental data for imperial
plt.figure(figsize=(6,4))
rs=np.arange(0,8) #0 to 7 goals

# Observed: bar chart
```

```
plt.bar(rs - 0.2, i_freq, width=0.4, label="Observed data Imperial")

# Theoretical Poisson: line plot or markers
plt.plot(rs, i_expect, 'o-', color='red', label="Poisson expected Imperial")
plt.xlabel("Number of goals")
plt.ylabel("Number of matches")
plt.title("Observed vs Poisson expected frequencies for Imperial")
plt.xticks(rs)
plt.legend()
plt.grid(True, linestyle=':', alpha=0.5)
plt.show()
```

[21]:




[21]:




## 5 Q4)

[14]:
```
#calculating the chi-squared probabillities for both coefficients and␣
 ↪refractive index of perspex n_2

#defining data
angle = np.array([10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85]) #degrees
r_par = np.array([0.030,0.029,0.027,0.026,0.022,0.017,0.013,0.007,0.003,0.001,0.
 ↪003,0.013,0.036,0.100,0.221,0.466]) #R^2 parrallel
r_per = np.array([0.035,0.038,0.041,0.046,0.051,0.062,0.074,0.091,0.112,0.141,0.
 ↪183,0.245,0.325,0.447,0.602,0.824]) #R^2 perepndicular

#uncertainties
sigma_a = 0.5 #uncertainty on angle
sigma_r = 0.002 #uncertainty on coefficients
sigma_a = np.deg2rad(sigma_a) #converts angle uncertainties into radians

def fresnel_Rs(theta, n):
    """Perpendicular reflection coefficient squared (R ²)."""
    n0 = 1
    sin_t2 = (n0/n)*np.sin(theta)
    sin_t2 = np.clip(sin_t2, -1, 1)
```

```
    theta_t = np.arcsin(sin_t2)
    rs = (n0*np.cos(theta) - n*np.cos(theta_t)) / (n0*np.cos(theta) + n*np.
 ↪cos(theta_t))
    return rs**2

def fresnel_Rp(theta, n):
    """Parallel reflection coefficient squared (R ²)."""
    n0 = 1.0
    sin_t2 = (n0 / n) * np.sin(theta)
    sin_t2 = np.clip(sin_t2, -1, 1)
    theta_t = np.arcsin(sin_t2)
    rp = (n*np.cos(theta) - n0*np.cos(theta_t)) / (n*np.cos(theta) + n0*np.
 ↪cos(theta_t))
    return rp**2

#caluclating chi-squared
def chi2_fresnel(R_obs, theta, model_func, n):
    """Return  ² and combined uncertainties."""
    R_model = model_func(theta, n) #theoretical model
    h = 1e-6  #steps of the derivative
    dR_dtheta = (model_func(theta + h, n) - model_func(theta - h, n)) / (2*h)

    sigma_total = np.sqrt((sigma_r)**2 + (dR_dtheta *sigma_a)**2) #errors
    chi2 = np.sum(((R_obs - R_model) / sigma_total)**2) #chi squared formulae
    return chi2

#brewster agnle relationship
theta_b = 57 #brewster angle = 57 degrees
n_2 = np.tan(np.deg2rad(theta_b)) #brewster angle relationship and brewster␣
 ↪angle in radians and finding refractive inex of material 2
theta = np.deg2rad(angle) #converts our angles in radians

#plugging in experimental data and theoretical model to caluclate chi-squared
chi2_par = chi2_fresnel(r_par, theta, fresnel_Rp, n_2)
chi2_per = chi2_fresnel(r_per, theta, fresnel_Rs, n_2)

# Degrees of freedom (no fitted parameters, n fixed)
dof_par = len(r_par)
dof_per = len(r_per)

# p-values
pval_par = stats.chi2.sf(chi2_par, dof_par)
pval_per = stats.chi2.sf(chi2_per, dof_per)

print("The P( ) for the parralel coefficient is: ",pval_par)
print("The P( ) for the perpendicular coefficient is: ",pval_per)
print("The   2 for the parralel coefficient is: ",chi2_par)
```

```
print("The  2 for the parralel coefficient is: ",chi2_per)
print("The  refractive index for perspex is: ",n_2)
```

```
The P( ) for the parralel coefficient is:  1.2986573733560927e-24
The P( ) for the perpendicular coefficient is:  3.590596981671584e-52
The  2 for the parralel coefficient is:  153.94752099607126
The  2 for the parralel coefficient is:  289.6159797558089
The  refractive index for perspex is:  1.539864963814583
```

[15]:
```
#minimizing chi-squared to find best fits for both coefficients sepertely

def chi2_n(n,R_obs, theta, model_func):
    """Return  ² and combined uncertainties."""
    R_model = model_func(theta, n) #theoretical model
    h = 1e-6  #steps of the derivative
    dR_dtheta = (model_func(theta + h, n) - model_func(theta - h, n)) / (2*h)

    sigma_total = np.sqrt((sigma_r)**2 + (dR_dtheta *sigma_a)**2) #errors
    chi2 = np.sum(((R_obs - R_model) / sigma_total)**2) #chi squared formulae
    return chi2

from scipy.optimize import minimize_scalar

# Parallel polarization mimization
res_par = minimize_scalar(chi2_n, args=(r_par, theta, fresnel_Rp), bounds=(1.4,␣
 ↪1.7), method='bounded')
n_best_par = res_par.x     # the best refractive index fit
chi2_min_par = res_par.fun  # the minimum chi-squared value at that n

# Perpendicular polarization minimization
res_per = minimize_scalar(chi2_n, args=(r_per, theta, fresnel_Rs), bounds=(1.4,␣
 ↪1.7), method='bounded')
n_best_per = res_per.x   # the best refractive index fit
chi2_min_per = res_per.fun #the minimum chi-squared value at that n

# Degrees of freedom = number of data points - number of fitted parameters (1)
dof_par = len(r_par) - 1
dof_per = len(r_per) - 1

#probabillity values
pval_par = stats.chi2.sf(chi2_min_par, dof_par)
pval_per = stats.chi2.sf(chi2_min_per, dof_per)

print("Parallel: n_best =", n_best_par, "chi2_min =", chi2_min_par, "p-value␣
 ↪=", pval_par)
print("Perpendicular: n_best =", n_best_per, "chi2_min =", chi2_min_per,␣
 ↪"p-value =", pval_per)
```

```
Parallel: n_best = 1.4487016758022877 chi2_min = 10.238247438186333 p-value =
0.8044918393792495
Perpendicular: n_best = 1.491437709499198 chi2_min = 110.12333435719174 p-value
= 1.5269403973801925e-16
```

[16]:
```
#minimizing chi-squared to find best fits for both coefficients together

#cobined data
R_obs_combined = np.concatenate([r_par, r_per])
theta_combined = np.concatenate([theta, theta])


#chi-squared for combined data
def chi2_combined(n):
    """Return  ² and combined uncertainties."""
    R_model_combined = np.concatenate([fresnel_Rp(theta, n), fresnel_Rs(theta,␣
 ↪n)]) #theoretical
    h = 1e-6  #steps of the derivative
    dR_dtheta_combined = (np.concatenate([fresnel_Rp(theta+h, n),␣
 ↪fresnel_Rs(theta+h, n)])
                          - np.concatenate([fresnel_Rp(theta-h, n),␣
 ↪fresnel_Rs(theta-h, n)])) / (2*h)
    sigma_total = np.sqrt((sigma_r)**2 + (dR_dtheta_combined *sigma_a)**2)␣
 ↪#errors
    chi2 = np.sum(((R_obs_combined - R_model_combined) / sigma_total)**2) #chi␣
 ↪squared formulae
    return chi2

#minimizing combined data
res_combined = minimize_scalar(chi2_combined, bounds=(1.4,1.7),␣
 ↪method='bounded')
n_best_combined = res_combined.x    # the best refractive index fit
chi2_min_combined = res_combined.fun #the minimum chi-squared value at that n

#dof
dof_combined = len(R_obs_combined) - 1

## combined probabillity values
pval_combined = stats.chi2.sf(chi2_min_combined, dof_combined)

print("Combined: n_best =", n_best_combined, "chi2_min =", chi2_min_combined,␣
 ↪"p-value =", pval_combined)
```

```
Combined: n_best = 1.4841593094005379 chi2_min = 145.14922811996044 p-value =
1.0784466604250294e-16
```

## 6 Comment on results:

The actual refractive index of perspex is on average 1.491, and is measured in the range of 1.46-1.56. The value of n obtained by the perpendicular coefficient was the most accurate value, however all the values obtained lie within the range mentioned above. The p-values obtained should lie in the range of 0-1, however only the p-value for the parrallel coefficient is below 1. This is due to a calulcation error when finding the p-values. There were some anomalous data points that i did not remove from analysis, which could also be a cause for the p-value being greater than 1. For a good fit, the p-value should be between 0.5 and 1, so the fit for the parralell coefficient seams to be a good fit, however could suggest uncertainties may be overestimated.

The refractive index for the combined data set was different compared to the individual ones, suggesting experimental inconsistencies when measuring the parralell and perpendicuar coefficients. This could be due to perspex not being smooth or containing impurities, or maybe the light used to refract into the perspex is not perfectly polarised. So the errors used in this experiment may be understimates. I should also point out that the combined data has a really high chi-squared, further proving the point that error bars might be too small.

The chi-squared being high suggests that experimental data differs from theory. The fresnal equations are approximations though, so other factors (such as impure perspex) could play a part in creating this high chi-squared. But the chi-squared may not be a true representation of how good the fit ws, as it does not take into account how many degrees of freedom that were used , unlike the reduced chi-squared.

## 7 Refferences

The true value of refractive index of perspex was found from this paper:

Determination of Refractive Index Using Snell's Law

Jamie Lee Somers

https://jamiesomers.com/reports/determination-of-refractive-index-using-snells-law.pdf

[0]: