# PST Exercise 10

## Problem 1

1) $r(\theta) = \frac{p}{1+\varepsilon\cos(\theta)} \Rightarrow r(\theta)(1+\varepsilon\cos(\theta)) = p$

$\Rightarrow r(\theta) + r(\theta)\varepsilon\cos(\theta) = p$

$\Rightarrow r(\theta) = p - r(\theta)\varepsilon\cos(\theta)$

$\Rightarrow y = \beta_1\phi_1(\theta) - \beta_2\phi_2(\theta)$

$\Rightarrow y = r(\theta)$

$\beta_1 = p \qquad \phi_1 = 1$

$\beta_2 = r(\theta)\varepsilon \quad \phi_2 = \cos(\theta)$

2) $N = N_0 e^{-\lambda t} \Rightarrow \ln(N) = \ln(N_0) - \lambda t$

$\Rightarrow y = \beta_1\phi(t) + \beta_2\phi_2$

$\Rightarrow y = \ln(N) \qquad \beta_1 = \ln(N_0) \quad \phi_1 = 1$

$\beta_2 = -\lambda \qquad \phi_2(t) = t$

3) $r(\alpha) = k\, C_a^{\alpha_1} C_b^{\alpha_2}$

$\ln(r) = \ln(k\, C_a^{\alpha_1} C_b^{\alpha_2})$

$= \ln(k) + \alpha_1\ln(C_A) + \alpha_2\ln(C_b)$

$\Rightarrow y = \beta_1\phi_1 + \beta_2\phi_2(\alpha) + \beta_3\phi_3(\alpha)$

$y = \ln(r(\alpha))$

$\beta_1 = \ln(k)$ $\qquad \phi_1 = 1$

$\beta_2 = \ln(C_A)$ $\qquad \phi_2(\alpha) = \alpha_1$

$\beta_3 = \ln(C_B)$ $\qquad \phi_2(\alpha) = \alpha_2$

## Problem 2

Part 1) The normal equation arises from the setting up the matrix equation with all measurements and doing matrix derivations with respect to $\beta$ to minimize the square of the error giving $X^T X \beta = X^T y$.

This can be solved by multiplying by the inverse of $X^T X$

$\Rightarrow \quad B = (X^T X)^{-1} X^T y$

Part 2)

a) $P = \exp(\beta_1 + \beta_2 T)$

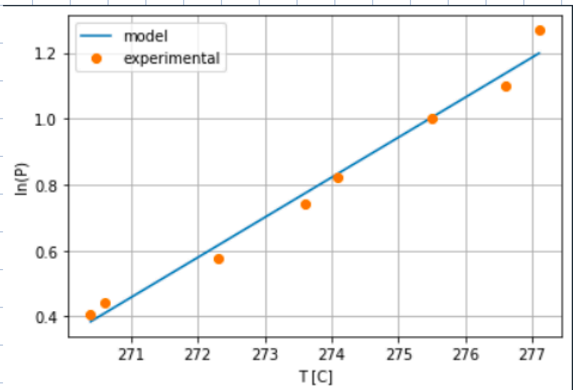
$\Rightarrow \ln(P) = \beta_1 + \beta_2 T$

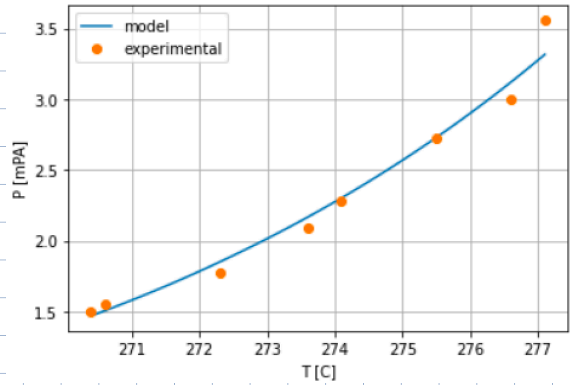
b - d)

PST10_1.py

```
def f(x,B):
    B0,B1 =B
    return B0 + B1*x


# b
T = np.array([270.4, 270.6, 272.3, 273.6, 274.1, 275.5, 276.6, 277.1]) #K
exP= np.array([1.502, 1.556, 1.776, 2.096, 2.281, 2.721, 3.001, 3.556]) # mPa
y = np.log(exP)

# c
A= np.column_stack((np.ones_like(T),T))
AT = np.transpose(A)
ATA = np.dot(AT,A)
ATy = np.dot(AT,y)

Beta = lg.solve(ATA, ATy)

#d
x= np.linspace(np.min(T),np.max(T),100)

sol_y_pts=f(x,Beta)
y_calc_trans = np.exp(sol_y_pts)

plt.figure(1)
plt.plot(x,sol_y_pts, label="model")
plt.plot(T, y, "o" ,label = "experimental")
plt.legend()
plt.xlabel("T [C]")
plt.ylabel("ln(P)")
plt.grid()

plt.figure(2)
plt.plot(x,y_calc_trans, label="model")
plt.plot(T, exP, "o" ,label = "experimental")
plt.legend()
plt.xlabel("T [C]")
plt.ylabel("P [mPA]")
plt.grid()
```

gives us



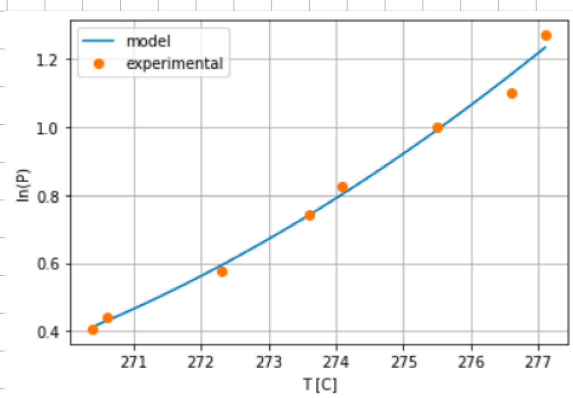



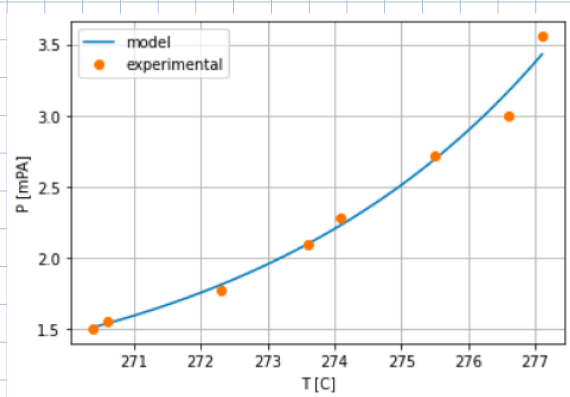

e) This is because we don't know the relationship between p & T outside this range and can't validate the result. The model could also become more innacurate the further away from the range of experimentation we extrapolate

## Part 3

a) PST 10_2.Ps

```
def f(x,B):
    B0,B1,B2 =B
    return B0 + B1*x + B2*x**(-2)


# b
T = np.array([270.4, 270.6, 272.3, 273.6, 274.1, 275.5, 276.6, 277.1]) #K
exP= np.array([1.502, 1.556, 1.776, 2.096, 2.281, 2.721, 3.001, 3.556]) # mPa
y = np.log(exP)

# c
A= np.column_stack((np.ones_like(T),T,T**(-2)))
AT = np.transpose(A)
ATA = np.dot(AT,A)
ATy = np.dot(AT,y)

Beta = lg.solve(ATA, ATy)

#d
x= np.linspace(np.min(T),np.max(T),100)

sol_y_pts=f(x,Beta)
y_calc_trans = np.exp(sol_y_pts)

plt.figure(1)
plt.plot(x,sol_y_pts, label="model")
plt.plot(T, y, "o" ,label = "experimental")
plt.legend()
plt.xlabel("T [C]")
plt.ylabel("ln(P)")
plt.grid()

plt.figure(2)
plt.plot(x,y_calc_trans, label="model")
plt.plot(T, exP, "o" ,label = "experimental")
plt.legend()
plt.xlabel("T [C]")
plt.ylabel("P [mPA]")
plt.grid()
```



b)

```
res= exP-np.dot(A,Beta)
norm= lg.norm(res, ord=2)
```

```
2-norm of model 1: 4.44205142493076
```

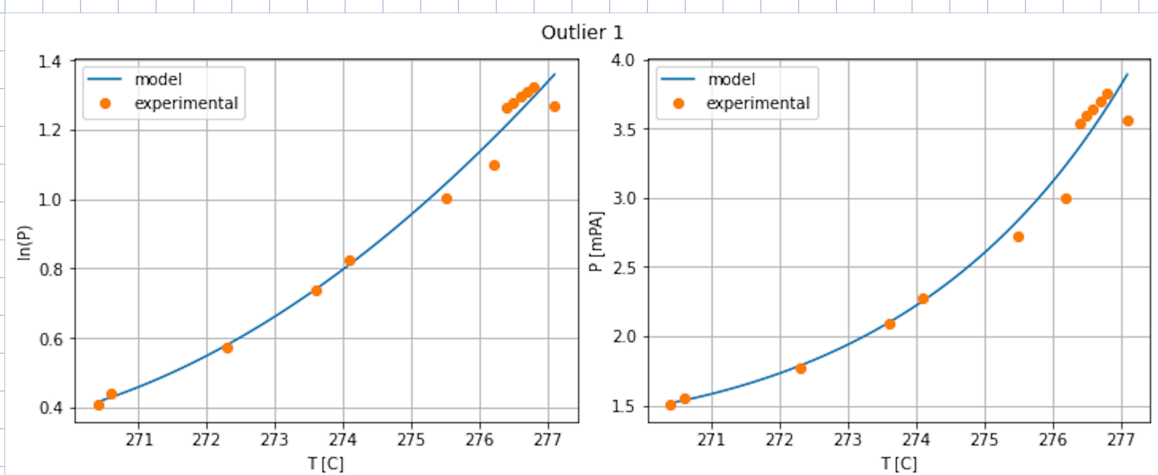
```
2-norm of model 2: 4.436181610708281
```

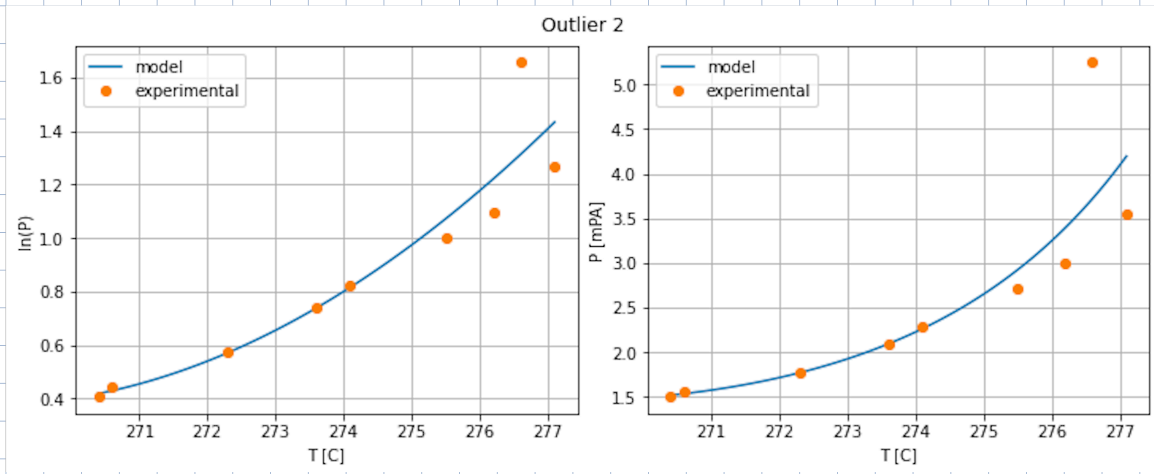
c) produces the lowest norm

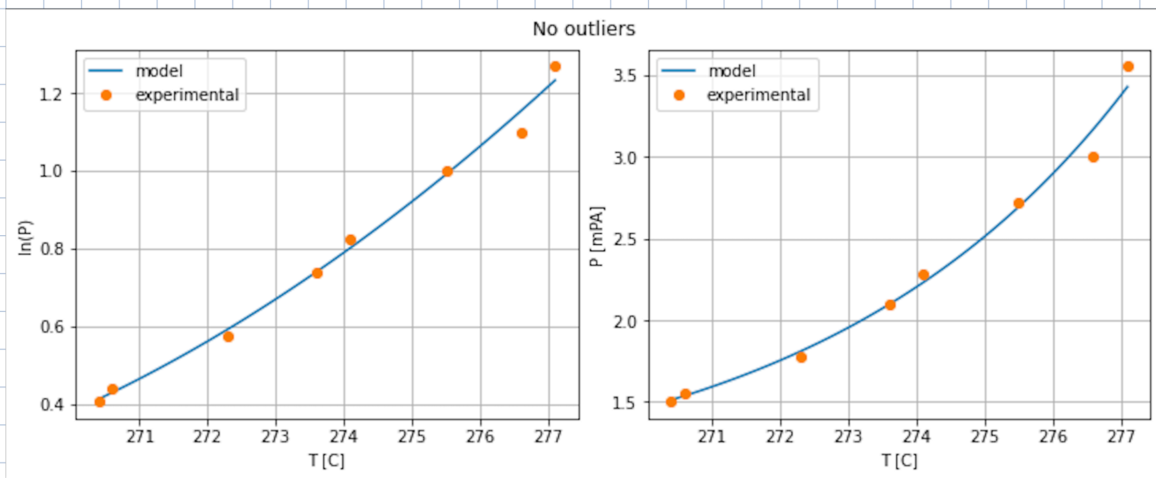
e) PST10_3.PY

```
def OLS(T,exP,y,title):

    def f(x,B):
        B0,B1,B2 =B
        return B0 + B1*x + B2*x**(-2)



    A= np.column_stack((np.ones_like(T),T,T**(-2)))
    AT = np.transpose(A)
    ATA = np.dot(AT,A)
    ATy = np.dot(AT,y)

    Beta = lg.solve(ATA, ATy)
    print(title +f' beta:{Beta}')


    x= np.linspace(np.min(T),np.max(T),100)

    sol_y_pts=f(x,Beta)
    y_calc_trans = np.exp(sol_y_pts)


    fig,(ax1,ax2)=plt.subplots(constrained_layout = True, ncols=2,
                               figsize=(10,4))

    ax1.plot(x,sol_y_pts, label="model")
    ax1.plot(T, y, "o" ,label = "experimental")
    ax1.legend()
    ax1.set_xlabel("T [C]")
    ax1.set_ylabel("ln(P)")
    ax1.grid()

    ax2.plot(x,y_calc_trans, label="model")
    ax2.plot(T, exP, "o" ,label = "experimental")
    ax2.legend()
    ax2.set_xlabel("T [C]")
    ax2.set_ylabel("P [mPA]")
    ax2.grid()
    fig.suptitle(title)


OLS(ol1T,ol1exP,ol1exlnP,"Outlier 1")
OLS(ol2T,ol2exP,ol2exlnP,"Outlier 2")
OLS(T_o, exP_o, y_o, "No outliers")
```

δ)






```
Outlier 1 beta:[-8.78774147e+02  2.18906032e+00  2.10039426e+07]
Outlier 2 beta:[-1.14217087e+03  2.83418129e+00  2.75082419e+07]
No outliers beta:[-4.60805857e+02  1.16492812e+00  1.06912233e+07]
```

as the points are at the end of the range it is mostly the end wich is affected

we see that the models gives more attention to a single point with large error.

This makes sense as $(y-C(a))^2 + (y-C(b))^2$ is less than $(y-C(a+b))^2$