

ESTIMATING LAG TIME AND PERMEABILITY IN MEMBRANES

ELYSE FOSSE

1. INTRODUCTION

The study of membrane permeation has two main goals: finding a membrane that allows solutes to permeate rapidly, and finding a membrane that will act as a barrier and force solutes to permeate slowly. The research performed in Prof. Nancy Lape's Engineering lab pertains to the latter category. The lab has studied different polymers with the objective of finding polymers that enhance the barrier between solutes. An example of one such polymer is Saran Wrap. Saran Wrap acts as a barrier between outside air particles and whatever food one wishes to preserve. The lab is searching for polymers that have consumer-friendly characteristics similar to Saran Wrap, which is lightweight and clear, and have a smaller permeation rate, or flux than Saran Wrap[4].

Matlab code was created in order to aid Prof. Lape's lab in their analyses. It will be shown that estimates for permeability as well as for lag time can be obtained via experiments and statistical analysis. It is the purpose of the Matlab code to estimate the desired unknowns.

2. THEORY

In order to fully understand flux, diffusion must first be defined. "Diffusion is caused by random molecular motion that leads to complete mixing." [1] It follows then that flux can be described as "the rate per unit area at which mass moves." [1] Thomas Graham worked on the problem of diffusion during the years 1828-1833 and was one of the first to consider that diffusion must be decreasing with respect to time. Furthermore, his experiments led him to postulate, "flux caused by diffusion is proportional to the concentration difference of the diffused substance." [1]

In 1855 Adolf Fick furthered the work of Graham by speculating that diffusion could be described using the same mathematical models as Fourier's law for heat conduction and Ohm's law for electrical conduction. Both Fourier's law and Ohm's law are proportions between the rate of energy transfer (heat and electricity, respectively) to the distances the energies travel [6][7]. By relating these models to the particulars of diffusion, Fick's first and second laws of diffusion were formed.

Fick's first law relates the one-dimensional flux to the change in concentration. Let A be the area across which the diffusion occurs, J_i be the flux per unit area, D be the diffusion coefficient, c_l be the concentration of solution l and z be the distance in the direction the diffusion is occurring. Fick's first law of diffusion states that the total one-dimensional

flux, J_1 is,

$$(1) \quad J_1 = A_{j1} = -AD \left(\frac{\partial c_l}{\partial z} \right).$$

In other words, the rate of diffusion is proportional to the area and coefficient of the diffusion multiplied by the portion of solution that has permeated through the diffusion distance, or the thickness of the membrane[1].

Fick's second law of diffusion can be obtained by first calculating a conservation equation with respect to volume,

$$(2) \quad \frac{\partial C_l}{\partial t} = D \left(\frac{\partial^2 C_l}{\partial z^2} + \frac{1}{A} \frac{\partial A}{\partial z} \frac{C_l}{\partial z} \right).$$

By setting A to be constant, Eq.(2) describes unsteady state diffusion, which is Fick's second law of diffusion. Together, Fick's first and second laws describe how much solute moves across the film (diffusion flux) and reveal how the solute concentration changes within the film (concentration profile)[4].

To apply Fick's laws, certain assumptions must be made. Let C_{1l} be the concentration of downstream solution and let C_{10} be the concentration of upstream solution. If H is the partition coefficient which relates pressure in gas to concentration in the film, then $c_{10} = HC_{10}$.

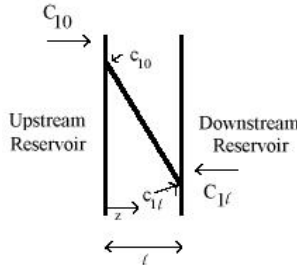


FIGURE 1. Steady-state concentration profile in a thin membrane. C_{10} is proportional c_{10} and the partition coefficient H. Infinite reservoirs on both sides are assumed.

Initially, the downstream solution is assumed to be free of upstream solution. At time zero, the membrane is also assumed to be free of upstream solution. For $t > 0$ the amount of upstream solution in the membrane is c_{10} at $z = 0$. Furthermore, where $z = l$ the amount of upstream solution is said to be zero. The boundary conditions restrict the downstream solution to be free of the upstream solution. In experiments it has been found that the concentration is negligible. These boundary conditions allow Eq.(2) to be transformed into an ordinary differential equation, which will describe a pseudo-steady state in the system.

In 1920 H. A. Daynes wrote a paper discussing the characteristic of lag time in the diffusion model. Daynes used separation of variables on Fick's second law and imposed the

aforementioned conditions to obtain,

$$(3) \quad c_l = c_{l0} - \frac{c_{l0}z}{l} - \frac{2c_{l0}}{\pi} \sum_{n=1}^{\infty} \sin\left(\frac{n\pi z}{l}\right) \exp\left(\frac{-Dn^2\pi^2 t}{l^2}\right).$$

Daynes also calculated the limit of the above equation at large times:

$$(4) \quad \frac{c_l}{c_{l0}} = \frac{AD}{Vl} \left(t - \frac{l^2}{6D} \right),$$

where V is the ratio of the volume in the air chamber to the area of the fabric. This limiting equation revealed that there is a lag time for the system to reach the desired pseudo-steady state. Namely, the system can not be described by an ordinary differential equation when $t < \frac{l^2}{6D}$ [2]. The limiting equation also reveals that the permeability of the membrane can be found experimentally by calculating the best-fit slope of the equation. Similarly the lag time can be found by experimentally calculating the x-intercept of the best-fit line of the pseudo-steady state data points.

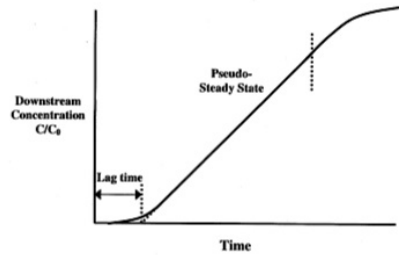


FIGURE 2. Typical permeation/lag time curve: normalized downstream concentration versus time in a thin membrane initially free of solute. The code created estimates the pseudo-steady state section

3. EXPERIMENT

Lape et. al performed a series of experiments measuring the permeation of the upstream solution through the membrane to the downstream solution. The data revealed a certain amount of noise related to the points. This meant that the x-intercept and slope of the pseudo-steady state could not be easily read.

The group realized that obtaining the x-intercept and slope of the pseudo-steady state could be computed using a best fit statistical algorithm implemented in Matlab. The codes created were not giving reliable results since occasionally they would return a negative slope, which is impossible in the physical system. The problem appeared to be how the sections of pseudo-steady state data were being cut. The code was cutting small amounts of data and finding maximum slopes between different “cut” data sets. The problem is

that the noise associated with the data could in fact lead to decreasing slopes since the majority of data sets in a “cut” data set could be decreasing due to the noise.[5]

4. DISCUSSION

For this project, the desired estimates were calculated by performing linear regression analysis on only the data points that contribute to the pseudo-steady state. The model implied the regression should be linear and the general form used was $Y = \beta_0 + \beta_1 * X$. Thus β_0 and β_1 were estimated by $\hat{\beta}_0$ and $\hat{\beta}_1$, respectively. There is no heuristic way to find the beginning and ending of the pseudo-steady state region via the data points. Matlab code was written to facilitate the finding of the beginning and ending of the data set that described the pseudo-steady state.

This new code was based on the idea that if there were data points that actually contributed to the non-linear portion of the state rather than the linear portion, the best-fit slope that would be created would be less steep than the true best-fit slope. Following on this idea, the new `regress(data)` function concentrates on finding the upper cut off of data points. The user inputs the raw data obtained from experiment and the function performs an iterated loop in which data points that were taken at larger times are deleted from the set. Specifically, at first the full data set is linearly regressed upon and that slope is compared to the slope obtained from the first half of the data set. That slope is then compared to the first half of the first half of the data set, or first quarter of the original set, and so on.

The linear regression was performed with Matlab’s `polyfit` function. Regression coefficients, $\hat{\beta}_0$ and $\hat{\beta}_1$, are stored and compared to the next iteration which deletes data points as described above. Once the next iteration outputs a slope that is less than the existing maximum slope, the function ceases the iterative steps and returns the estimated regression coefficients as well as a plot of the original data set and the estimated regression line.

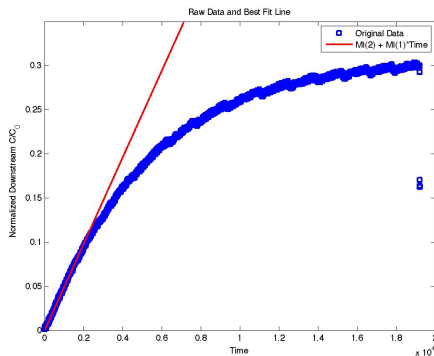


FIGURE 3. Graphical output from `regress(data)` function

It is worthy to note that the analysis would provide incorrect estimates if not enough data points are used. If an abundance of the data points that contribute to the linearity of the pseudo-steady state are deleted the slope will become flat, since some of the influential points may have been deleted. With this in mind a second Matlab function, `regress2(data)`, was created. This function performs the above iteration five times and plots those “best-fit” lines with the original data points. This plot is a good visual check to see what happens if further iterations are performed.

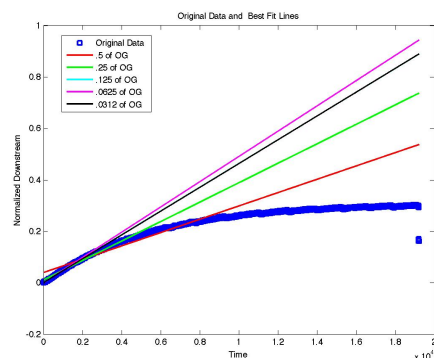


FIGURE 4. Graphical output from `regress2(data)` function

The final Matlab function, `regress3(data)`, incorporates the deletion of data points from both the beginning and the end of the data set. This function is useful if lag time proves to be significant. It first finds the steepest slope by deleting data points from larger times. That maximum slope is then compared to the maximum slope obtained from deleting small sections of data from the beginning of the data set. It is assumed that if lag time is significant, it will not be very large. Therefore, only five data points are deleted at a time, but if not enough data points is deleted, the slope will remain the same and if the iterations are terminated the maximum estimated slope would not be obtained. Therefore, the function performs this cut of 5 data points 200 times and compares the estimated slope of each of those 200 sets of data points to the maximum estimated slope obtained from deleting data points from the other end of the set.

5. CONCLUSIONS

The code is currently being reviewed by the lab. Once the code is shown to be sufficient the group can then use the estimated slopes and x-intercepts as the permeability and the lag time, respectively to describe the polymer membrane. This use of the estimations will in turn allow for the diffusion coefficient to be estimated and the equation for permeation will be fully solved.

This project is a great example of how different areas of mathematics are interwoven when performing research. Fick described a system by a partial differential equation.

Daynes was able to impose assumptions and transform the PDE into an ordinary differential equation. Even better, he was able to find what happens at large times which enabled the use of experimentation to solve for the permeation.

Knowledge of the chemical background to this model was minimal, and the problem was solved by using statistical knowledge to estimate the solutions to an ordinary differential equation. An original aspect to this project was to perform the analysis using R. The functions were first created in R but problems were encountered when trying to extract the large data sets. Since Prof. Lape's lab works primarily with Matlab the decision was made to convert the codes to Matlab and perform the calculations as such.

Further work could be done using R or any other statistical software. The code already created could also be re-configured to estimate the sum of residuals, which would show the accuracy of the estimated regression coefficients. The objective to be able to accurately calculate the permeability of different membranes does seem to be accomplished.

6. APPENDIX

The following are the three functions described in the paper as well as their output for a specific data set, 04102008_2.lvm.

```

%Elyse Fosse
%April 18, 2008
%This function assumes there is no lag time. It looks for the cut off
%point data where the slope is maximized. It does this by cutting off
%data points from the upper end of the set. It then compares the
%respective slopes and recursively performs the cutting again.

function regress = regress(data)

Time = data(:,1); %Creates a vector of Time
Down = data(:,3); %Creates a vector of Down
Up = data(:,2); %Creates a vecotr of Up
ND = Down./Up; %Creates a normalized vector of downstream
MS = 0; %MS = maximum slope to compare
MI = 0; %MI = regression matrix of max. slope

%Do initial comparison to create a base max slope to compare future
%iterations to.

l1=floor(length(ND)/2); %Creates initial lengths
l2=floor(length(ND)/4);
d1=ND(1:l1); %Creates time and down vectors to
d2=ND(1:l2); %regress on initially
t1=Time(1:l1);
t2=Time(1:l2);
LC1 = polyfit(t1,d1,1); %Perform regression on the data sets
LC2 = polyfit(t2,d2,1);
S1 = LC1(1); %Extract slopes from regression
S2 = LC2(1);

if S1 > S2 %Sets first max slope and first length
MS = S1;
l1 = length(d1);
else MS = S2;
l1 = length(d2);
end

MC = MS - 1; %Sets slope to compare to be less than

```

```

%MS to ensure the iteration is done at least
%once.

```

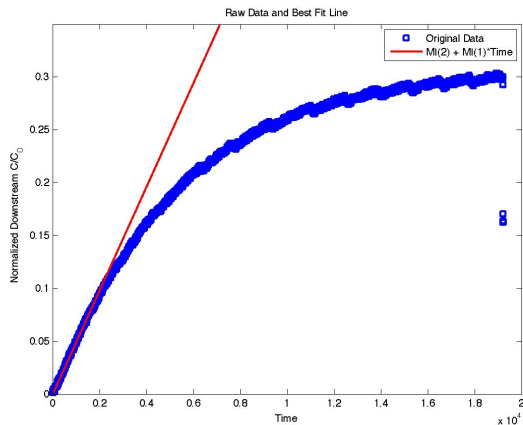
```

while MS > MC
    MC = MS;
    d1 = ND(1:l1);
t1 = Time(1:l1);
    l2 = floor(length(d1)/2);
    d2 = ND(1:l2);
t2 = Time(1:l2);
LC1 = polyfit(t1,d1,1); %Perform regression on the data sets
LC2 = polyfit(t2,d2,1);
    S1 = LC1(1) ; %Extract slopes from regression
    S2 = LC2(1);
    if S1 > S2
MS = S1; %Sets max slope to compare
l1 = length(d1);
MI = LC1;
else MS = S2;
l1 = length(d2);
MI = LC2;
    end
end

%Plot Original Data with new regressed line
Lfit = polyval(MI, Time);
plot(Time, ND, 's', Time, Lfit, 'r-', 'LineWidth', 2)
legend('Original Data', 'MI(2) + MI(1)*Time')
axis([0 20000 0 .35])
title('Raw Data and Best Fit Line')
%Returns the regression matrix. 1st element is slope, second element is
%x-intercept. Also returns the length of the data set that gave the
%maximum slope.
Regression_Coefficients = MI
Length_of_Data_Set = l1

OUTPUT:
Regression_Coefficients =
1.0e-03 *
0.0491 -0.1690
Length_of_Data_Set =
1185

```



%Elyse Fosse

%April 18, 2008

%This function can be used to check if the regress function is providing
 %accurate information. It plots multiple best fit lines for multiple sets
 %of data. The one of the steepest slope should always correspond to the
 %best fit line outputted by the regress function.

```
function regress2 = regress2(data)
```

```
Time = data(:,1);           %Creates a vector of Time
Down = data(:,3);          %Creates a vector of Down
Up = data(:,2);            %Creates a vector of Up
ND = Down./Up;
l1=floor(length(ND)/2)    %Creates initial lengths
l2=floor(length(ND)/4)
l3 = floor(length(ND)/8)
l4 = floor(length(ND)/16)
l5 = floor(length(ND)/32)

d1=ND(1:l1);              %Creates time and down vectors to regress
d2=ND(1:l2);
d3 = ND(1:l3);
d4 = ND(1:l4);
d5 = ND(1:l5);
t1=Time(1:l1);
t2=Time(1:l2);
t3=Time(1:l3);
t4 = Time(1:l4);
t5 = Time(1:l5);
```

```

LC1 = polyfit(t1,d1,1)           %Performs regression on the data sets
LC2 = polyfit(t2,d2,1)
LC3 = polyfit(t3,d3,1)
LC4 = polyfit(t4,d4,1)
LC5 = polyfit(t5,d5,1)

S1 = LC1(1);                    %Extracts slopes from regression
S2 = LC2(1);
S3 = LC3(1);
S4 = LC4(1);
S5 = LC5(1);

Lfit1 = polyval(LC1, Time);
Lfit2 = polyval(LC2, Time);
Lfit3 = polyval(LC3, Time);
Lfit4 = polyval(LC4, Time);
Lfit5 = polyval(LC5, Time);
plot(Time, ND, 's',Time, Lfit1, 'r-', Time, Lfit2, 'g-',...
      Time, Lfit3,'c-',Time, Lfit4, 'm-',Time, Lfit5, 'k-', 'LineWidth',2)
legend('Original Data','.5 of OG','.25 of OG',...
      '.125 of OG','.0625 of OG', '.0312 of OG')
title('Original Data and Best Fit Lines')

```

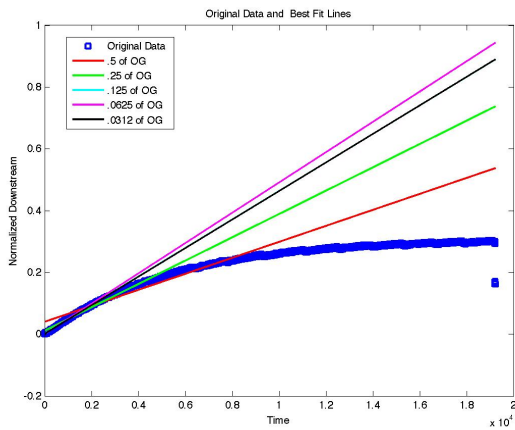
OUTPUT

```

11 =
  9487
12 =
  4743
13 =
  2371
14 =
  1185
15 =
  592
LC1 =
  0.0000    0.0393
LC2 =
  0.0000    0.0116
LC3 =
  0.0000    0.0017
LC4 =
  1.0e-03 *
  0.0491   -0.1690
LC5 =
  1.0e-03 *

```

0.0462 0.5085



%Elyse Fosse

```
function regress3 = regress3(data)
```

```
Time = data(:,1); %Creates a vector of Time
Down = data(:,3); %Creates a vector of Down
Up = data(:,2);
ND = D./U;
MS = 0; %MS = maximum slope to compare
MI = 0; %MI = regression matrix of max. slope
```

```
%Do initial comparison to create a base max slope to compare future
%iterations to.
```

```
l1=floor(length(ND)/2); %Creates initial lengths
l2=floor(length(ND)/4);
d1=ND(1:l1); %Creates time and down vectors to
d2=ND(1:l2); %regress on initially
t1=Time(1:l1);
t2=Time(1:l2);
LC1 = polyfit(t1,d1,1); %Perform regression on the data sets
LC2 = polyfit(t2,d2,1);
S1 = LC1(1); %Extract slopes from regression
S2 = LC2(1);

if S1 > S2 %Sets first max slope and first length
MS = S1;
```

```

l1 = length(d1);
    else MS = S2;
        l1 = length(d2);
end

MC = MS - 1;                                %Sets slope to compare to be less than
                                             %MS to ensure the iteration is done at least
                                             %once.

while MS > MC
    MC = MS;
    d1 = ND(1:l1);
    t1 = Time(1:l1);
    l2 = floor(length(d1)/2);
    d2 = ND(1:l2);
    t2 = Time(1:l2);
    LC1 = polyfit(t1,d1,1); %Perform regression on the data sets
    LC2 = polyfit(t2,d2,1);
    S1 = LC1(1) ;                                %Extract slopes from regression
    S2 = LC2(1);
    if S1 > S2
    MS = S1; %Sets max slope to compare
    l1 = length(d1);
    MI = LC1;
    else MS = S2;
    l1 = length(d2);
    MI = LC2;
    end
end

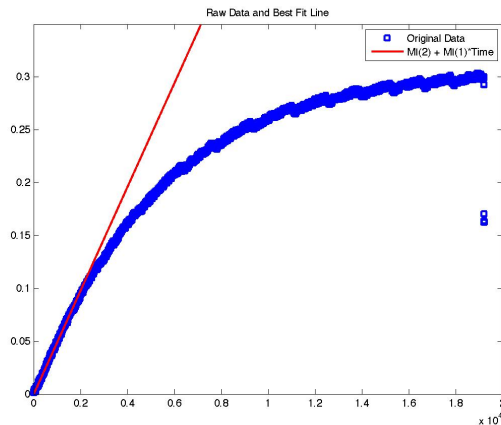
%Now we subtract from beginning to get max. slope
bmax = 0;
b = [1:5:1000];
MC = MS - 1;
for i = 1:200;
while MS > MC
    MC = MS;
    d1 = ND(b(i):l1);
    t1 = Time(b(i):l1);
    d2 = ND(b(i):l1);
    t2 = Time(b(i):l1);
    LC1 = polyfit(t1,d1,1); %Perform regression on the data sets
    LC2 = polyfit(t2,d2,1);

```

```
S1 = LC1(1) ; %Extract slopes from regression
S2 = LC2(1);
if S1 > S2
MS = S1; %Sets max slope to compare
l1 = length(d1);
    bmax = b(i);
    MI = LC1;
else MS = S2;
l1 = length(d2);
    bmax = b(i);
MI = LC2;
end
end
end

%Plot Original Data with new regressed line
Lfit = polyval(MI, Time);
plot(Time, Down, 's', Time, Lfit, 'r-', 'LineWidth', 2)
legend('Original Data', 'MI(2) + MI(1)*Time')
axis([0 20000 0 40])
title('Raw Data and Best Fit Line')
%Returns the regression matrix. 1st element is slope, second element is
%x-intercept. Also returns the length of the data set that gave the
%maximum slope.
Regression_Coefficients = MI
Data_Set_Begin = bmax
Data_Set_End = l1

OUTPUT:
Regression_Coefficients =
1.0e-03 *
0.0491 -0.1690
Data_Set_Begin =
1
Data_Set_End =
1185
```



REFERENCES

- [1] E. L. Cussler. *Diffusion: Mass Transfer in Fluid Systems*. Cambridge University Press, 1997. [The sections in this book lay out the ground work for models of diffusion. It started with a basic diffusion through a membrane and then added complexities, such as diffusion through a porous membrane and diffusion through which the solutions on each side are not the same. This information combined with the ideas in Dayne’s article combine to create the ordinary differential equation of the pseudo-steady state that will be studied.]
- [2] H. A. Daynes. The process of diffusion through a rubber membrane. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 97(685):286–307, 1920. [This article discussed the lag time that occurs when diffusing through membranes. When the information in this article is combined with the information in Cussler’s text, the pseudo-steady state ordinary differential equation is created.]
- [3] Kurt Hornik. *The R FAQ*, 2008. ISBN 3-900051-08-9.
- [4] Nancy Lape. Diffusion in heterogeneous membranes. [Prof. Lape’s thesis provides a real world motivation for why the x-intercept and slope of the pseudo-steady state of the system should be found. Different polymers will lead to different x-intercepts. The polymer with the greatest x-intercept will result in the longest lagtime before a permeation between the two compartments. Since the goal is to prevent premeation, the longer the lag-time the better. This idea coupled with the real-world consumer constraints, such as appearance and thickness, leads to the discussion of a polymer that is dispersed with flakes.]
- [5] Lape Students. pdms. [This program compiles the date and forms a best fit line using by ”smoothing” the date through taking cuts of the data and finding the slopes at those cuts. This program did not require the user to input lag time.]
- [6] Wikipedia. Ohm’s law — wikipedia, the free encyclopedia, 2002. [Online; accessed 15-April-2008].
- [7] Wikipedia. Heat conduction — wikipedia, the free encyclopedia, 2008. [Online; accessed 15-April-2008].