Numerical Visualization of Fick's Law of Diffusion

Jesse Anderson

Ramiro Roman

Dr. Jan Verschelde

5/2/2023

Abstract

This paper presents a comprehensive study on numerical experimentation to develop efficient computational methods to solve diffusion problems in complex media.The paper provides a derivation of Fick's law of Diffusion and two example codes to model diffusion at various time steps and steady state values of heated walls. The results showcase the implementation of the codes to solve problems with varying Diffusion coefficients and temperatures.Finally, the paper discusses the implications of the findings for future research directions and potential applications in diverse fields that require an improved understanding of mass transport phenomena.

Table of Contents

Nomenclature

D	Diffusivity constant
J	Mass flux per unit time per unit time
n	The number of unit molecules or the number density of molecules
m	Particle mass
k_B	Maxwell-Boltzman constant
T	Thermodynamic temperature of system
$lcos(\theta)$	Distance above or below gas boundary layer
v	Velocity vector of a molecule as represented in spherical coordinates
θ	The azimuthal angle in spherical coordinates
$\emptyset$	The polar angle in spherical coordinates

Introduction

Fick's laws of diffusion have been the cornerstone of our understanding of mass transport phenomena in various fields, including chemical engineering, materials science, and biology. As the demand for high-performance materials and simulating complex biological systems grows, the need for accurate and efficient numerical models to predict diffusion processes becomes increasingly vital. This paper presents a comprehensive study on numerical experimentation, which aims to advance our understanding of diffusion phenomena and develop robust, reliable, and efficient computational methods to solve diffusion problems in heterogeneous and anisotropic media.

The background of this work is rooted in the extensive literature surrounding Fick's laws of diffusion, as well as recent advances in numerical methods and computational resources. We build upon the well-established theoretical framework of Fick's laws, which describe the relationship between diffusion flux and concentration gradients in isotropic and homogeneous media. However, real-world systems are often more complex, necessitating the development of sophisticated numerical models that can accommodate the intricacies of heterogeneous and anisotropic materials. Our research extends the application of Fick's laws to such complex systems, and critically examines the suitability and accuracy of various numerical methods in predicting diffusion processes.

In the following sections, we provide a detailed account of our research methodology, including the selection of numerical methods, their implementation, and the construction of benchmark problems that serve as test cases for the evaluation of each method. We then present our results, comparing and contrasting the performance of different numerical approaches in terms of accuracy, computational efficiency, and adaptability to various diffusion problems. Finally, we discuss the implications of our findings for future research directions, and the potential applications of our work in diverse fields that require an improved understanding of mass transport phenomena.

Analysis

The derivation of Fick's Law of diffusion has its roots in the kinetic model when derived for mass diffusivity, assuming a steady diffusion, where acceleration is zero and diffusion is the only driving force. Consider a slab with 2 regions of the same gas, one with a high density and another of a lower density in parallel plates to each other. The number density *n with* distance *y* to each other. Let n_0 be the number density at the boundary between the two regions and dA be the differential area for a certain number of molecules moving with velocity *v* and when the angle θ is normal to *dA* in a time interval *dt* [2]. This number can be expressed as

$$nv\cos cos\,(\theta)\,dAdt\times\left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} e^{-\left(\frac{(mv^2)}{2k_B T}\right)}(v^2\sin sin\,(\theta)\,dvd\theta d\emptyset)$$

The above expression is derived from the Maxwell-Boltzmann distribution function, which is a probability distribution function representing the fraction of particles with a certain velocity in an *M*-dimensional velocity space. In the above formula, 3 dimensions with respect to velocity, angle normal to *dA* and angle ∅ in spherical coordinates.

The number of molecules at $lcos(\theta)$ from the boundary, or median, is expressed as

$$n\pm = n_0 \pm lcos(\theta)\frac{dn}{dy}$$

The positive sign indicates the number of molecules above the boundary, while negative indicates below. Assuming that the differential $\left(\frac{dn}{dy}\right)$ is constant over the mean free path, the path before change in direction of a molecule, the above expressions can be integrated over the following intervals:

$$\{v > 0\ 0 < \theta < \pi/2\ 0 < \emptyset < 2\pi$$

This yields the molecular transfer per unit time per unit area, or the mass flux:

$$J = \ -\frac{1}{3}\bar{v}l\frac{dn}{dy}$$

The expression $\frac{1}{3}\bar{v}l$ is known as the diffusivity constant *D*, which results in Ficks first law of diffusion:

$$J = -\ D\frac{dn}{dy}$$

In two or more dimensions, the gradient operator, del, ∇ is introduced and

$$J = -\ D\nabla n$$

Fick's second law relates the concentration gradient to time. Assuming a steady-state mass balance where all mass in the system is conserved and does not leave the boundaries, we have

$$\frac{\partial n}{\partial t} = -\frac{\partial}{\partial y}J \rightarrow \frac{\partial n}{\partial t} + \frac{\partial}{\partial y}J = 0$$

$$\frac{\partial n}{\partial t} - \frac{\partial}{\partial y}\left(D\frac{\partial}{\partial y}n\right) = 0$$

Assuming the driving force for the concentration gradient is random motion (diffusion only), the diffusivity constant *D* remains constant:

$$\frac{\partial}{\partial y}\left(D\frac{\partial}{\partial y}n\right) = D\frac{\partial}{\partial y}\frac{\partial}{\partial y}n = D\frac{\partial^2 n}{\partial y^2}$$

For 2 or more dimensions Fick's second law of diffusion becomes

$$\frac{\partial n}{\partial t} = D\nabla^2 n$$

$$\frac{\partial n}{\partial t} = D(\frac{\partial^2 n}{\partial x^2} + \frac{\partial^2 n}{\partial y^2})$$

In cartesian coordinates, the discretization of Fick's second law of diffusion is as follows:

$$\frac{n_{i,j}^{k+1} - n_{i,j}^{k}}{\Delta t} = D\frac{\left(n_{i+1,j}^{k} - 2n_{i,j}^{k} + n_{i-1,j}^{k}\right)}{\Delta x^2} + \frac{\left(n_{i,j+1}^{k} - 2n_{i,j}^{k} + n_{i,j-1}^{k}\right)}{\Delta y^2}$$

Solving for the next iteration of the mass concentration through a kernel in time:

$$n_{i,j}^{k+1} = D\frac{\Delta t}{\Delta x \Delta y}\left(n_{i+1,j}^{k} + n_{i-1,j}^{k} + n_{i,j+1}^{k} + n_{i,j-1}^{k} - 4n_{i,j}^{k}\right) + n_{i,j}^{k}$$

Fick's law of diffusion of particles states that the flux of particles is proportional to the gradient of their concentration in space. On the other hand, the heat equation describes the transport of heat through a medium and states that the rate of change of temperature is proportional to the second derivative of temperature with respect to space. The heat equation can be derived from Fick's law of diffusion by considering that the flux of heat is proportional to the gradient of temperature, and that the flux of heat is related to the rate of change of temperature through Fourier's law of heat conduction. Thus, Fick's law of diffusion and the heat equation are intimately related.

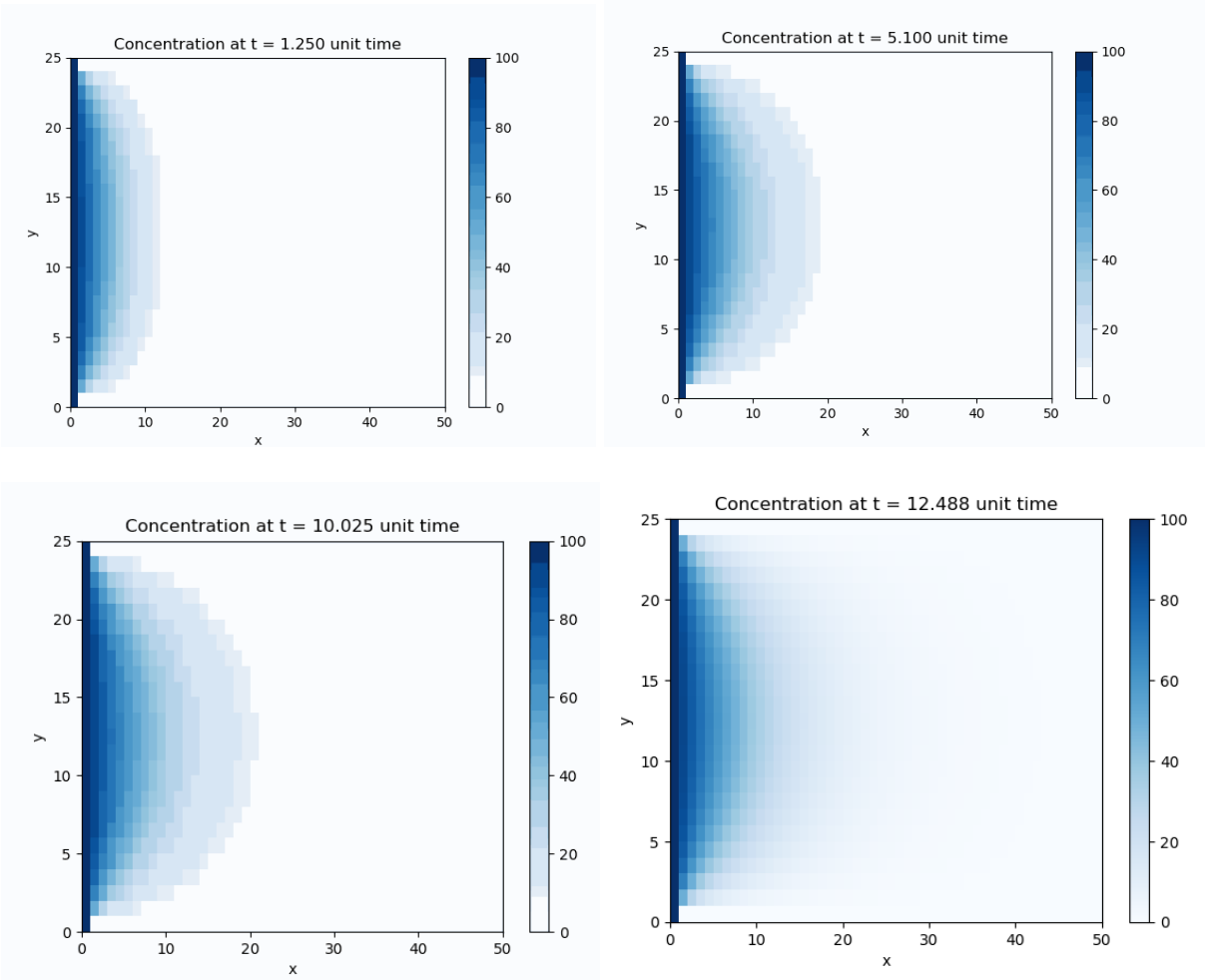
Note that for the current study our explicit scheme to arrive at the solution is only conditionally stable. That is that if our step size in time is too large in time compared to the step size in space we will encounter a divergence. The best practice will be to set a tolerance of 1E-6 between iterations to ensure that this does not occur. Note that this may dramatically slow run time.

Experimental Equipment and Procedure

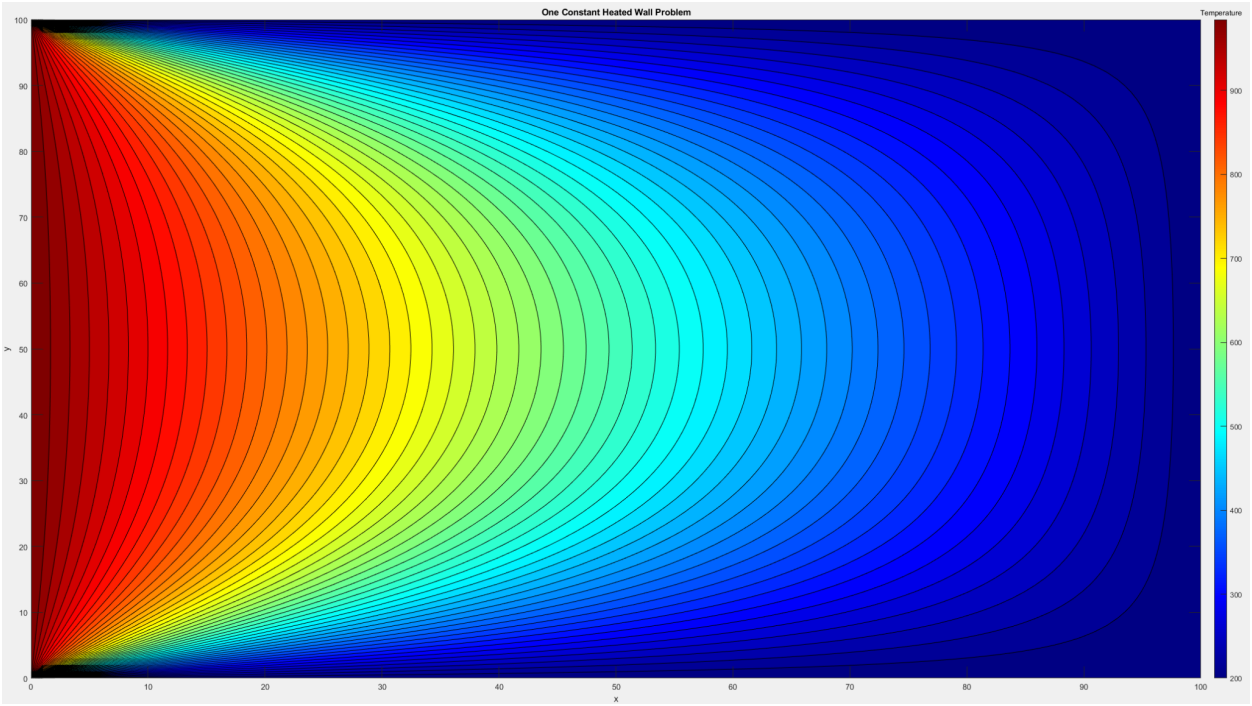
The code was run on an Acer Aspire A515-54G-70TZ with an Intel Core i7-8565U, 20GB of 2400 Mhz RAM, and a 512GB PCIe 3.0 SSD, on Windows 10. For the code created in python, a .gif file was saved to the local folder. For the MATLAB code, the figure was saved using the snipping tool in Windows 10.

Results

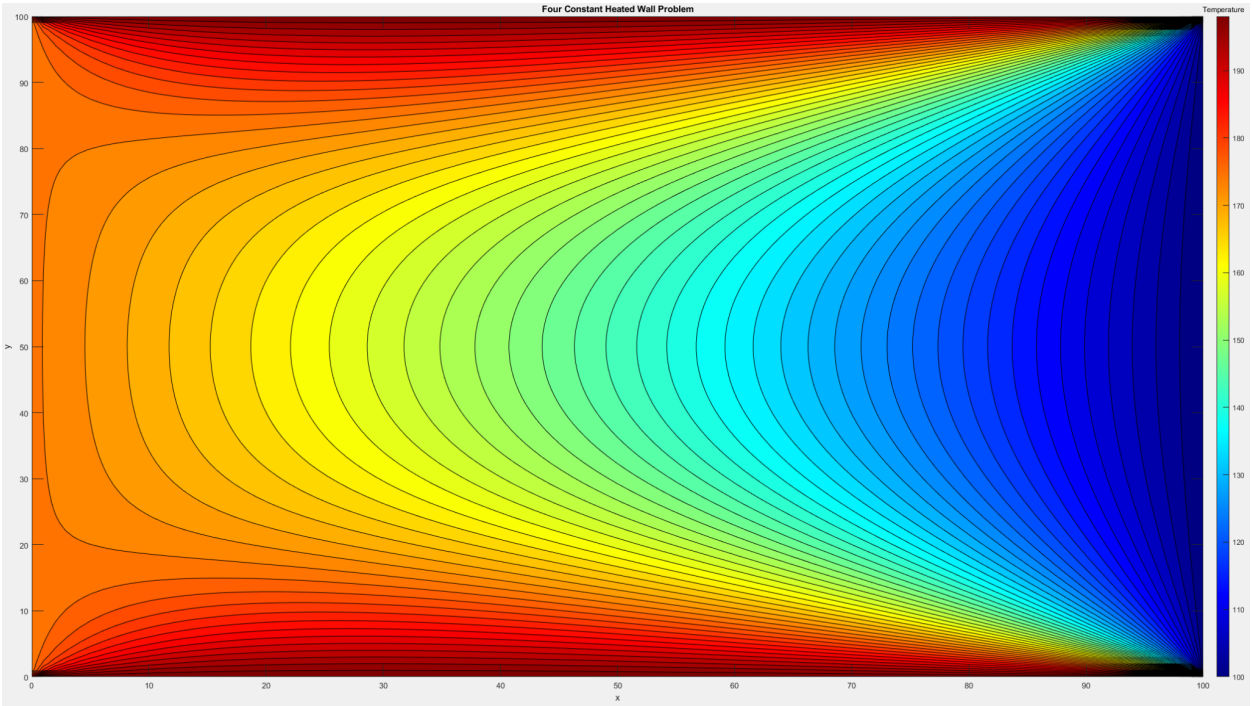
The animation for using finite differences to display Fick's law of diffusion was successfully created at different time steps seen below:



MATLAB was used to assess similarities between Fick's Law of Diffusion and heat transfer. The Figure below shows one heated wall at 1000C and the others at 200C in the same fashion as the above image.



Extending the problem to multiple heated walls or perhaps particles diffusing from different walls into a space we can arrive at the below image with heated walls of 200C for the top and bottom and 175C and 100C for the left and right walls respectively.



Discussion

For ease of viewing, a high diffusion constant was used. Furthermore, vectorization of calculating the next discrete point in time was used over multiple nested for loops, this was used for a faster, more efficient code. In order to create a cleaner animation, such as the one on Wikipedia's page [1], different libraries should be explored within the Python language. MATLAB code was written to assess if the second by second diffusion of particles would be similar to that of the heat equation and it was found that the heat equation and the diffusion of particles is very similar.

Conclusions

In conclusion, utilizing discretization and finite difference is one solution to solving partial differential equations in 3 dimensions. For the case applied above, x-direction, y-direction and time were the dimensions under consideration. Another solution is utilization of the Crank-Nicholson method which leads to a system of band-diagonal equations rather than tridiagonal ones. This was explored in another programming language, Excel VBA. but was ultimately abandoned as convergence issues hampered arriving at a solution. Transport phenomena such as diffusion and heat transfer are subjects which demand the utmost attention in the realm of engineering and physics and as such the development of more rigorous computational solutions will always be required. This paper sought to showcase implementations of the diffusion of particles and heat in two programming languages to model complex physical phenomena such as heat transfer in a bioreactor or COVID virus particles diffusing into a room.

References

[1] Wikimedia Foundation. (2023, April 19). *Fick's laws of diffusion*. Wikipedia. Retrieved April 26, 2023, from https://en.wikipedia.org/wiki/Fick%27s_laws_of_diffusion

[2] Philibert J (2005). "One and a Half Centuries of Diffusion: Fick, Einstein, before and beyond" (PDF). *Diffusion Fundamentals*. **2**: 1.1–1.10. Archived from the original (PDF) on 5 February

[3] Williams FA (1985). "Appendix E". *Combustion Theory*. Benjamin/Cummings.

[4] Nosek TM. "Section 3/3ch9/s3ch9_2". *Essentials of Human Physiology*. Archived from the original on 24 March 2016.

[5] Incropera, F. P. (2011). *Fundamentals of heat and mass transfer*. Wiley.

[6] Bird, R. B., Stewart, W. E., & Lightfoot, E. N. (2007). *Transport phenomena*. Wiley.

Appendices

Python Code for Diffusion Animation

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.animation import FuncAnimation

plate_length_x = 25
plate_length_y = int(plate_length_x*2)

max_iter_time = 1000#750

alpha = 20
delta_x = 1

delta_t = (delta_x ** 2)/(4 * alpha)
gamma = (alpha * delta_t) / (delta_x ** 2)

# Initialize solution: the grid of u(k, i, j)
u = np.empty((max_iter_time, plate_length_x, plate_length_y))

# Initial condition everywhere inside the grid
u_initial = 0

# Boundary conditions
u_top = 0.0 #100.0
u_left = 100
```

```
u_bottom = 0.0
u_right = 0.0

# Set the initial condition
u.fill(u_initial)

# Set the boundary conditions
u[:, (plate_length_y-1):, :] = u_top
u[:, :, :1] = u_left
u[:, :1, 1:] = u_bottom
u[:, :, (plate_length_x-1):] = u_right



def calculate(u):
    # Vectorization step
    for k in range(0, max_iter_time-1, 1):
        u[k+1, 1:-1, 1:-1] = gamma * (u[k, 2:, 1:-1] + u[k, :-2, 1:-1] + u[k, 1:-1, 2:] + u[k, 1:-1, :-2] -
4*u[k, 1:-1, 1:-1]) + u[k, 1:-1, 1:-1]

    return u

def plotheatmap(u_k, k):
    # Clear the current plot figure
    plt.clf()

    plt.title(f"Concentration at t = {k*delta_t:.3f} unit time")
    plt.xlabel("x")
    plt.ylabel("y")
```

```
    # This is to plot u_k (u at time-step k)
    plt.pcolormesh(u_k, cmap='Blues', vmin=0, vmax=100)
    plt.colorbar()

    return plt

# Do the calculation here
u = calculate(u)

def animate(k):
    plotheatmap(u[k], k)

anim = animation.FuncAnimation(plt.figure(), animate, interval=1, frames=max_iter_time,
repeat=False)
anim.save("diffusion_equation_solution.gif")

print("Done!")
```

Matlab Code for Heat Transfer

```
%%
% Code is for steady state, different from the diffusion animation but
% wanted to show how we could relate diffusion and the heat equation and
% the steady state final result of heat and each frame of diffusion
clear all
close all
clc
%steady state
numx=100; %number of points calculated left to right
numy=100;% up and down
```

```
len=100; %total length of block
wid=100; %width
x=linspace(0,len,numx); %points 0 to 1, nx num steps
y=linspace(0,wid,numy);
deltx=x(3)-x(1); %delta X
delty=y(3)-y(2);
%boundary conditions
temp=300*ones(numx,numy); %X Start
temp(1,:)=200;%bottom wall
a1 = temp(1,:);
a1 = a1(1,1); %easier to swap
temp(numy,:)=200;%top wall
b1 = temp(numy,:);
b1 = b1(1,1);
temp(:,1)=175;%left wall
a2 = temp(:,1);
a2 = a2(1,1);
temp(:,numx)=100;%right wall
b2 = temp(:,numx);
b2 = b2(1,1);
temp(1,1)=(a1+a2)/2; %simple calc
temp(1,end)=(a1+b2)/2;
temp(end,1)=(a2+b1)/2;
temp(end,end)=(b1+b2)/2;
[xx,yy]=meshgrid(x,y); %make mesh grid of linspace before
Thold=temp; %hold
termCalc1=1/(2*((1/deltx^2)+(1/delty^2))); %calc
errorCalc=1e+9;
tol=1e-6; %change this as needed to run
counter=1; %ssee iterations
while(errorCalc>tol) %go until we get to tolerance
for i=2:numx-1
```

```
  for j=2:numy-1
     termCalc2=(Thold(i-1,j)+Thold(i+1,j))/(deltx^2); %calcs
     termCalc3=(Thold(i,j-1)+Thold(i,j+1))/(delty^2);
     temp(i,j)=termCalc1*(termCalc2+termCalc3);



  end
end
   errorCalc=max(max(abs(Thold-temp))) %get error
   Thold=temp;
   counter=counter+1 %couintMe



end
   [c,h]=contourf(xx,yy,temp,50);
   hhh = colorbar; % give z axis, heat intensity, labeling
   colormap(jet) % jet is always best, no matter what
   %clabel(c,h) % if you wanted to label things you can do so here, but
   %this is best for really simple heat plots.
xlabel('x')
ylabel('y')
title('One Constant Heated Wall Problem')
title(hhh,'Temperature')
```