Department of Physics - Ariel University
Computational Physics

Dimensionless 3D Diffusion Equation C++

Noach Detwiler
Tal Aharon

June 14, 2020

Contents

0.1 Problem Definition . . . 1
0.2 Finite Difference methods . . . 4
0.3 initial and Boundary conditions . . . 5
0.4 Numerical Results . . . 7
0.5 Analytical solution . . . 11
0.6 Source code, matlab . . . 13
0.7 Conclusion . . . 19

Theory

0.1 Problem Definition

Governing equation

The diffusion equation is a partial differential equation which describes density fluctuations in a material undergoing diffusion. The equation is written as:

$$\frac{\partial C}{\partial t} + \mathbf{V} \cdot \nabla C = \nabla \cdot (\bar{K} \otimes \nabla C)$$

Where $C_{(x,y,z,t)}$ is the Scalar rate of diffusion at (x, y, z) in Cartesian Coordinates at time t. $\mathbf{V}$ is the 'Average'[1] flow speed (in any given direction). and $\bar{K}$ is the eddy-diffusivity or dispersion tensor.

which can be rewritten as:

$$\frac{\partial C}{\partial t} + \mathbf{V}\frac{\partial C}{\partial x} = \bar{K}\frac{\partial^2 C}{\partial x^2}$$

In three dimensions:

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} + v\frac{\partial C}{\partial y} + w\frac{\partial C}{\partial z} = D_x\frac{\partial^2 C}{\partial x^2} + D_y\frac{\partial^2 C}{\partial y^2} + D_z\frac{\partial^2 C}{\partial z^2} \quad [1]$$

With u, v, w representing a natural flow speed in the directions x, y, z respectively, furthermore, D_x, D_y, D_z is the diffusibility in each respected direction. For example the diffusibility officiant of Acetone (dis) -in- water (l) at

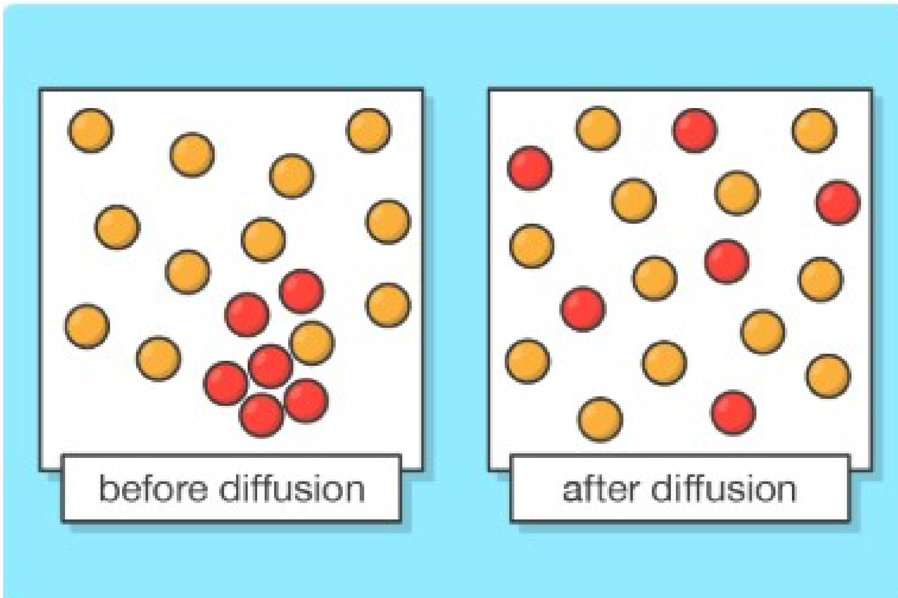
$$T = 25^\circ \ \ [Celsius] \quad is \ \ D = 1.16 \cdot 10^5 \ \ [\frac{cm^2}{s}]$$

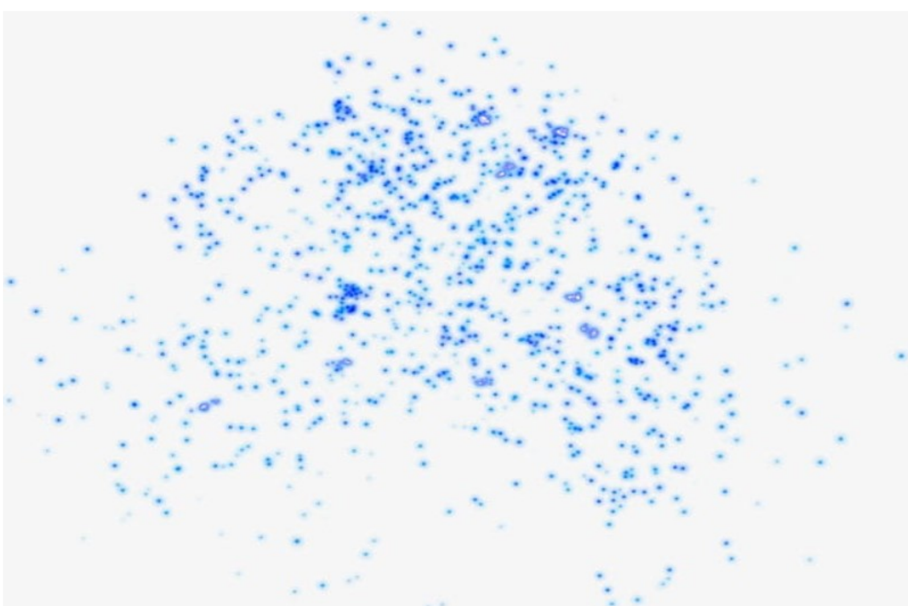
In this study, we solve a dimentionless 3-D diffusion equation by using the Forward in Time, Center in Space (FTCS) finite difference method. The domain of this study covers two points; firstly in a closed environment, we study diffusion equation and compare analytical results to our numerical.

and secondly we study the affects of a natural flow(u, v, w),on the diffusion equation.

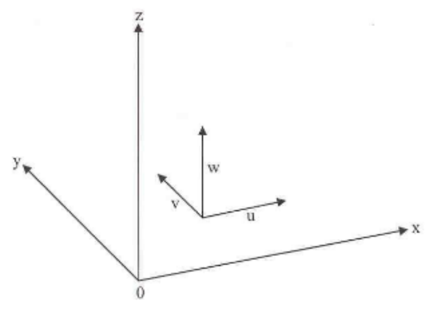
[1] we did not calculate a fluctuating speed, 'average' is used to represent the idea of a varying speed, in other words $\mathbf{V}$ is a constant.

Our goal is to Create a 3D grid domain, a rate of diffusion was set at (x, y, z) coordinate at time t=0. We will diffuse it while keeping the wall (reflecting boundary's) as a box $1x1x1cm^3$. We also aim to plot the diffusion in 2D with the 3d data of the pollution .Great example of diffusion in 2D.

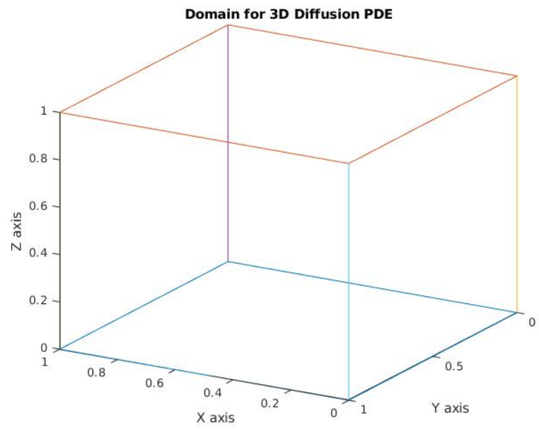




— flow direction for 3D model —



— Domain of PDE —



For our problem, the domain was chosen as an arbitrary box of $\{(x, y, z) \in 0 \rightarrow 1\}$ where the variables are placed as a uniformly equal grid-like domain:

$$x_i = i\Delta x,\ i = 0, 1, 2, 3...Nx$$

$$y_j = j\Delta y,\ j = 0, 1, 2, 3...Ny$$

$$z_k = k\Delta z,\ k = 0, 1, 2, 3...Nz$$

$$t_l = l\Delta t,\ l = 0, 1, 2, 3...Nz$$

Approximations $C^n_{(x,y,z,t)}$ to $C_{(i\Delta x, j\Delta y, k\Delta z, l\Delta t)}$ are calculated at the point of intersection of these lines according to the (i,j,k,l) grid points. The uniform spatial and temporal grid spacing's are:

$$\Delta x = \frac{1}{Nx - 1}$$

$$\Delta y = \frac{1}{Ny - 1}$$

$$\Delta z = \frac{1}{Nz - 1}$$

$$\Delta t = \frac{t_f}{Nt - 1}$$

0.2 Finite Difference methods

The Forward in Time, Center in Space (FTCS) finite difference method was chosen for increased accuracy and simplicity.
this scheme approximates the original equation with errors of first order in the time interval and second order in spatial coordinate grid spacing.

Equation [1] can be written as:

$$\frac{C^{n+1}_{i,j,k}-C^{n}_{i,j,k}}{\Delta t}+u\left(\frac{C^{n}_{i+1,j,k}-C^{n}_{i-1,j,k}}{2\Delta x}\right)+v\left(\frac{C^{n}_{i,j+1,k}-C^{n}_{i,j-1,k}}{2\Delta y}\right)+w\left(\frac{C^{n}_{i,j,k+1}-C^{n}_{i,j,k-1}}{2\Delta z}\right)=$$
$$=D_x\left(\frac{C^{n}_{i+1,j,k}-2C^{n}_{i,j,k}+C^{n}_{i-1,j,k}}{(\Delta x)^2}\right)+D_y\left(\frac{C^{n}_{i,j+1,k}-2C^{n}_{i,j,k}+C^{n}_{i,j-1,k}}{(\Delta y)^2}\right)+D_z\left(\frac{C^{n}_{i,j,k+1}-2C^{n}_{i,j,k}+C^{n}_{i,j,k-1}}{(\Delta z)^2}\right)$$

Rearrangement of the Equation to separate $C^{n+1}_{i,j,k}$:

$$C^{n+1}_{i,j,k}=C^{n}_{i,j,k}-\frac{u\cdot\Delta t}{2\Delta x}(C^{n}_{i+1,j,k}-C^{n}_{i-1,j,k})-\frac{v\cdot\Delta t}{2\Delta y}(C^{n}_{i,j+1,k}-C^{n}_{i,j-1,k})-\frac{u\cdot\Delta t}{2\Delta z}(C^{n}_{i,j,k+1}-C^{n}_{i,j,k-1})$$
$$+\frac{D_x\cdot\Delta t}{\Delta x^2}(C^{n}_{i+1,j,k}-2C^{n}_{i,j,k}+C^{n}_{i-1,j,k})+\frac{D_y\cdot\Delta t}{\Delta y^2}(C^{n}_{i,j+1,k}-2C^{n}_{i,j,k}+C^{n}_{i,j-1,k})+\frac{D_z\cdot\Delta t}{\Delta z^2}(C^{n}_{i,j,k+1}-2C^{n}_{i,j,k}+C^{n}_{i,j,k-1})$$

Defining constant values:

$$D_x\frac{\Delta t}{\Delta x^2}=S_x$$
$$D_y\frac{\Delta t}{\Delta y^2}=S_y$$
$$D_z\frac{\Delta t}{\Delta z^2}=S_z$$

$$\frac{u\Delta t}{\Delta x}=C_x$$
$$\frac{v\Delta t}{\Delta y}=C_y$$
$$\frac{w\Delta t}{\Delta z}=C_z$$

The equation can simply be written as:

$$C^{n+1}_{i,j,k}=(S_x+\frac{C_x}{2})C^{n}_{i-1,j,k}+(S_y+\frac{C_y}{2})C^{n}_{i,j-1,k}+(S_z+\frac{C_z}{2})C^{n}_{i,j,k-1}+$$
$$(S_x-\frac{C_x}{2})C^{n}_{i+1,j,k}+(S_y-\frac{C_y}{2})C^{n}_{i,j+1,k}+(S_z-\frac{C_z}{2})C^{n}_{i,j,k+1}+(1-2[S_x+S_y+S_z])C^{n}_{i,j,k}$$

For stability we use two couriants:

$Couriant-1$

$$S_x+S_y+S_z\leq\frac{1}{2}$$

$Couriant-2$

$$\frac{C_x^2}{S_x}+\frac{C_y^2}{S_y}+\frac{C_z^2}{S_z}\leq 3$$

0.3 initial and Boundary conditions

To preform a numeric calculation of the diffusion equation we need to set boundary and initial condition. Using two arrays 1.C 2.C_t to preform calculation over time. The initial condition to initialize the grid at any given coordinates and set the concentration at time t=0 to be zero and x,y,z axis at point 0 to be also 0 at any given time. this insures the stability of a closed room. The Initial conditions follow by a nested loop will be:

$$[\,1\,]\;\; C^0_{i,j,k} = 0.0$$

$$[\,2\,]\;\; (C_t)^0_{i,j,k} = 0.0$$

$$[\,3\,]\;\; C^n_{0,j,k} = 0.0$$

$$[\,4\,]\;\; C^n_{i,0,k} = 0.0$$

$$[\,5\,]\;\; C^n_{i,j,0} = 0.0$$

$$[\,6\,]\;\; C^n_{Nx,j,k} = 0.0$$

$$[\,7\,]\;\; C^n_{i,Ny,k} = 0.0$$

$$[\,8\,]\;\; C^n_{i,j,Nz} = 0.0$$

The main problem is to calculate the first step while using the Forward in Time, Center in Space (FTCS) finite difference method , The first step of any given loop is -0- and to calculate i=0,j=0,k=0 in the derivatives we face $[i-1]$ or any other integer in the same form. To solve this scheme we will follow the equations:

1.) Using forward in time, forward in space condition for i=0,j=0,k=0 we get:

$$\frac{\partial C}{\partial x}(0,j,k) = -\frac{C_{0,j,k} + 4C_{1,j,k} - 3C_{2,j,k}}{2\Delta x}$$
$$\frac{\partial C}{\partial y}(i,0,k) = -\frac{C_{i,0,k} + 4C_{i1,k} - 3C_{i,2,k}}{2\Delta y}$$
$$\frac{\partial C}{\partial z}(i,j,0) = -\frac{C_{i,j,0} + 4C_{i,j,1} - 3C_{i,j,2}}{2\Delta z}$$

2.) Using forward in time backwards in space condition for the first derivative in the end of the cubical dimension we get:

$$\frac{\partial C}{\partial x}(Nx,j,k) = \frac{3C_{Nx,j,k} - 4C_{Nx-1,j,k} + 3C_{Nx-2,j,k}}{2\Delta x}$$

$$\frac{\partial C}{\partial y}(i,Ny,k) = \frac{3C_{i,Ny,k} - 4C_{i,Ny-1,k} + 3C_{i,Ny-2,k}}{2\Delta y}$$

$$\frac{\partial C}{\partial z}(i,j,Nz) = \frac{3C_{i,j,Nz} - 4C_{i,j,Nz-1} + 3C_{i,j,Nz-2}}{2\Delta z}$$

Preforming the initial boundary condition to the second order derivatives we use forward in time, forward in space:

$$\frac{\partial^2 C}{\partial x^2}(0, j, k) = \frac{C_{0,j,k} - 2C_{1,j,k} + C_{2,j,k}}{\Delta x^2}$$

$$\frac{\partial^2 C}{\partial y^2}(i, 0, k) = \frac{C_{i,0,k} - 2C_{i,1,k} + C_{i,2,k}}{\Delta y^2}$$

$$\frac{\partial^2 C}{\partial z^2}(i, j, 0) = \frac{C_{i,j,0} - 2C_{i,j,1} + C_{i,j,2}}{\Delta z^2}$$

Preforming the initial boundary condition to the second order derivatives we use forward in time, backwards in space:

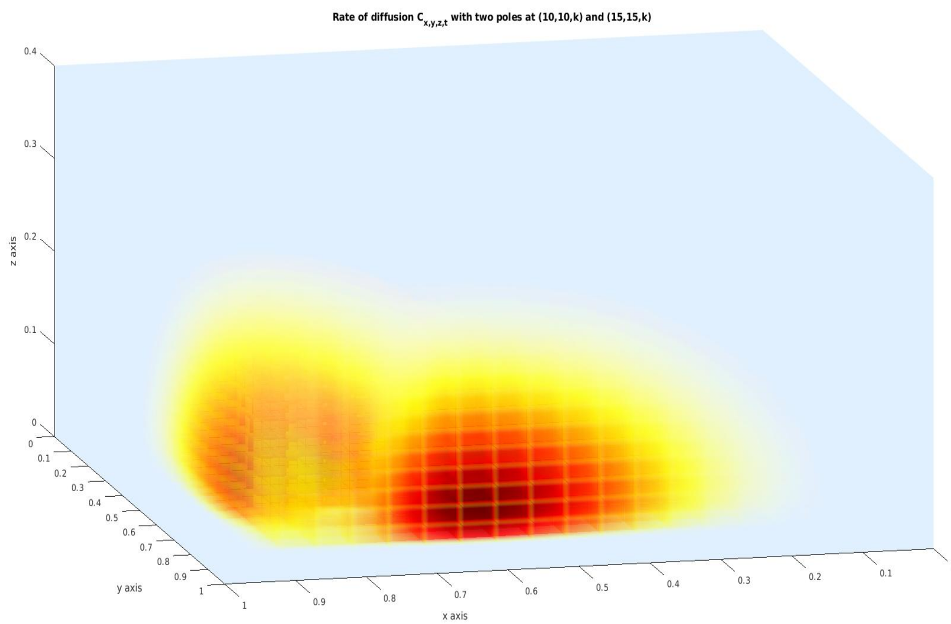
$$\frac{\partial^2 C}{\partial x^2}(Nx, j, k) = \frac{C_{Nx,j,k} - 2C_{Nx-1,j,k} + C_{Nx-2,j,k}}{\Delta x^2}$$
$$\frac{\partial^2 C}{\partial y^2}(i, Ny, k) = \frac{C_{i,Ny,k} - 2C_{i,Ny-1,k} + C_{i,Ny-2,k}}{\Delta y^2}$$
$$\frac{\partial^2 C}{\partial z^2}(i, j, Nz) = \frac{C_{i,j,Nz} - 2C_{i,j,Nz-1} + C_{i,j,Nz-2}}{\Delta z^2}$$

Contributing all the above to the code insures us the stability of dispersion through time in the domain.

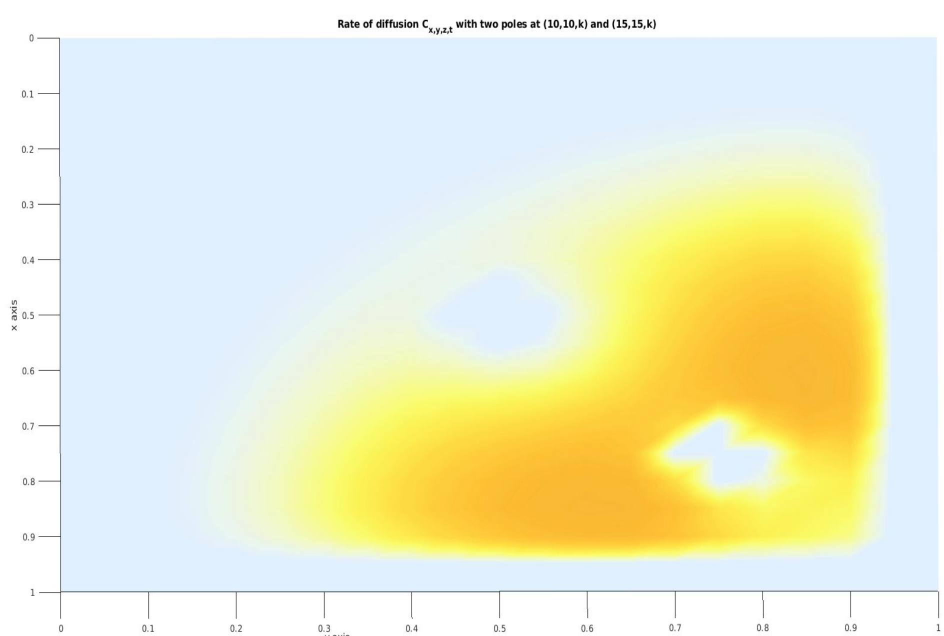
0.4 Numerical Results

Example 1

Considering diffusion in 3D and in 2D with linear speed of $u_x = 0.05, v_y = 0.05, w_z = -0.05$.
initial pollution at t=0 - $C_{(0.01,0.01,0.9,0)} = 200$. the domain is 1x1x1 cm^3, 2000 time steps and final time 20 *sec.* also we add two columns in (0.5,0.5,z) and in (0.75,0.75,z) at all height to create a unique flow of matter.

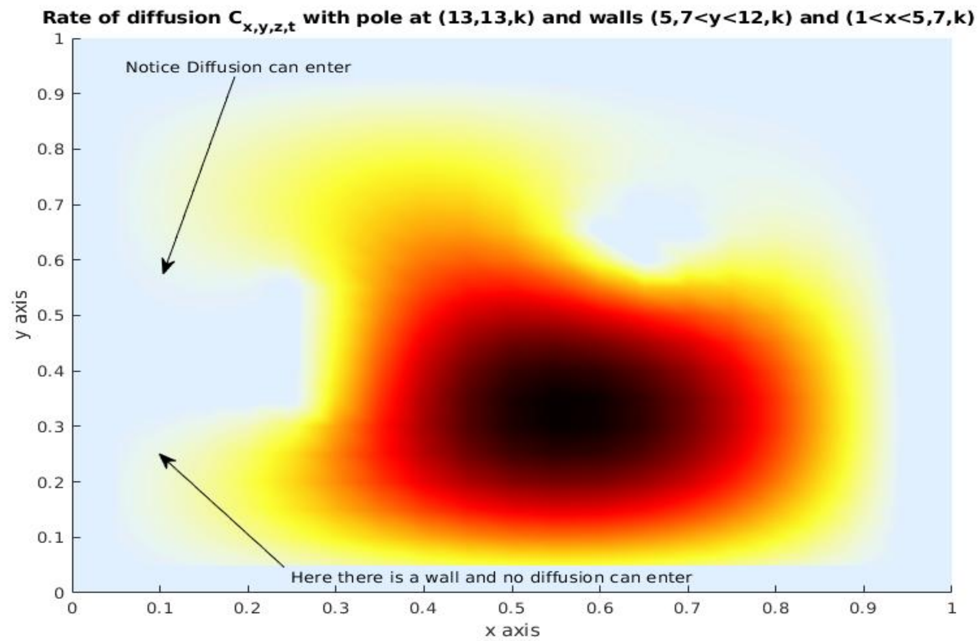


Looking at the 2d slice (x,y) of the diffusion we get.

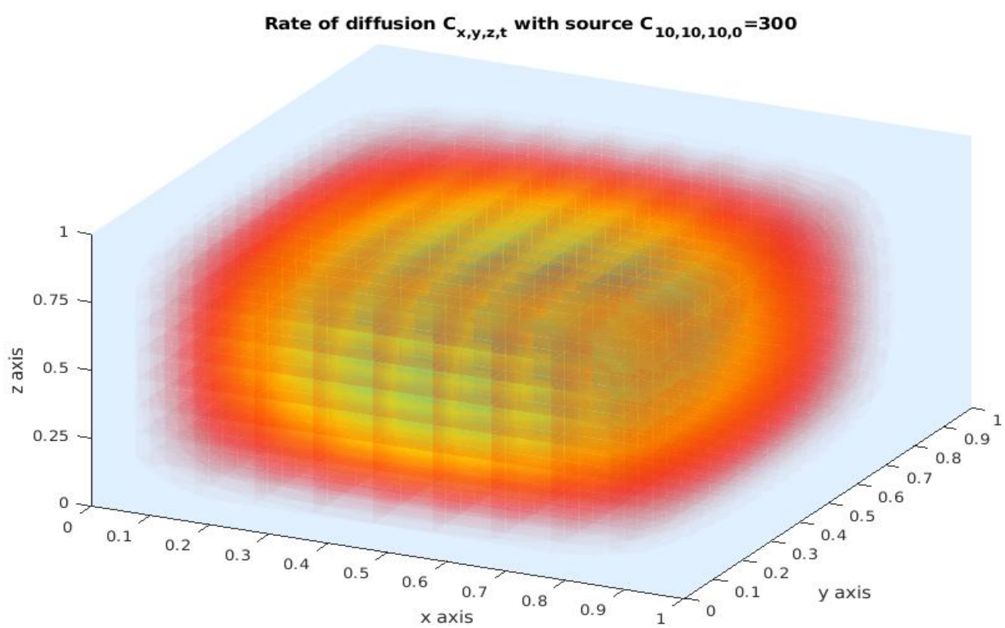


Example 2

Considering diffusion in 3D and in 2D with no linear speed at all $u_x = 0.0\frac{m}{s}, v_y = 0.0\frac{m}{s}, w_z = 0.0\frac{m}{s}$.
initial pollution at t=0 - $C_{(0.5,0.5,0.5,0)} = 300$. the domain is 1x1x1 cm^3, 2000 time steps and final time 100 s.
also we add one column in (0.65,0.65,z) and one box in (0.01<=x<=0.25,0.35<=y<=0.6,z) to create a unique flow of matter.

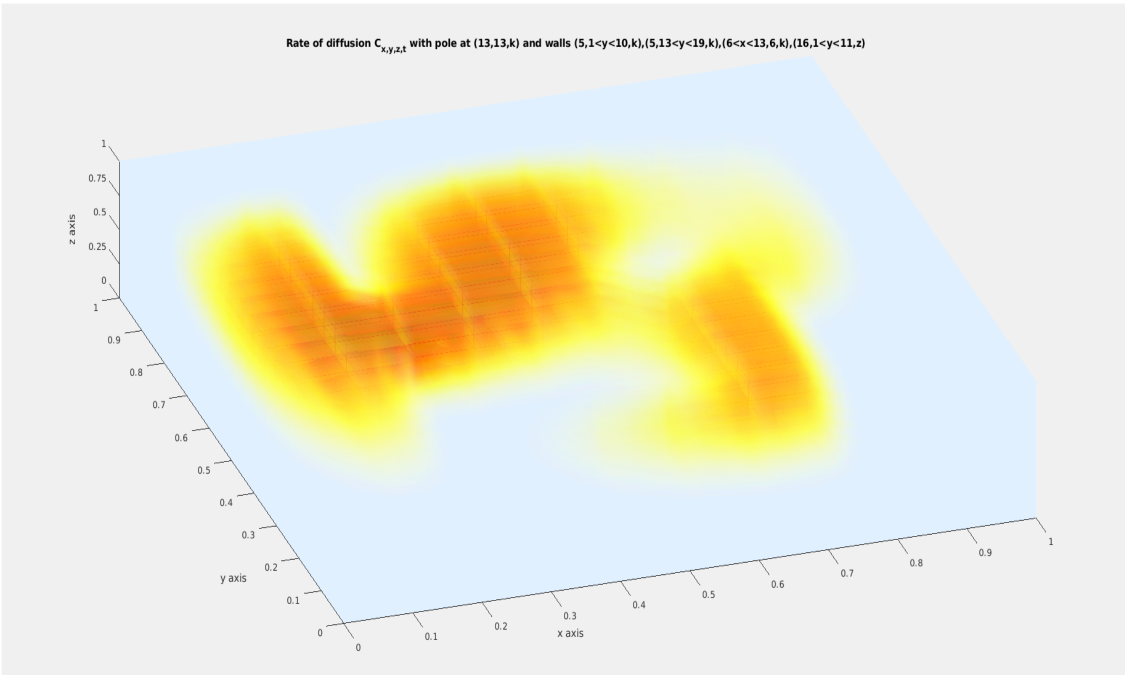


Looking in the center of the grid with normal diffusion means, no currents and ideal conditions we get a sphere.

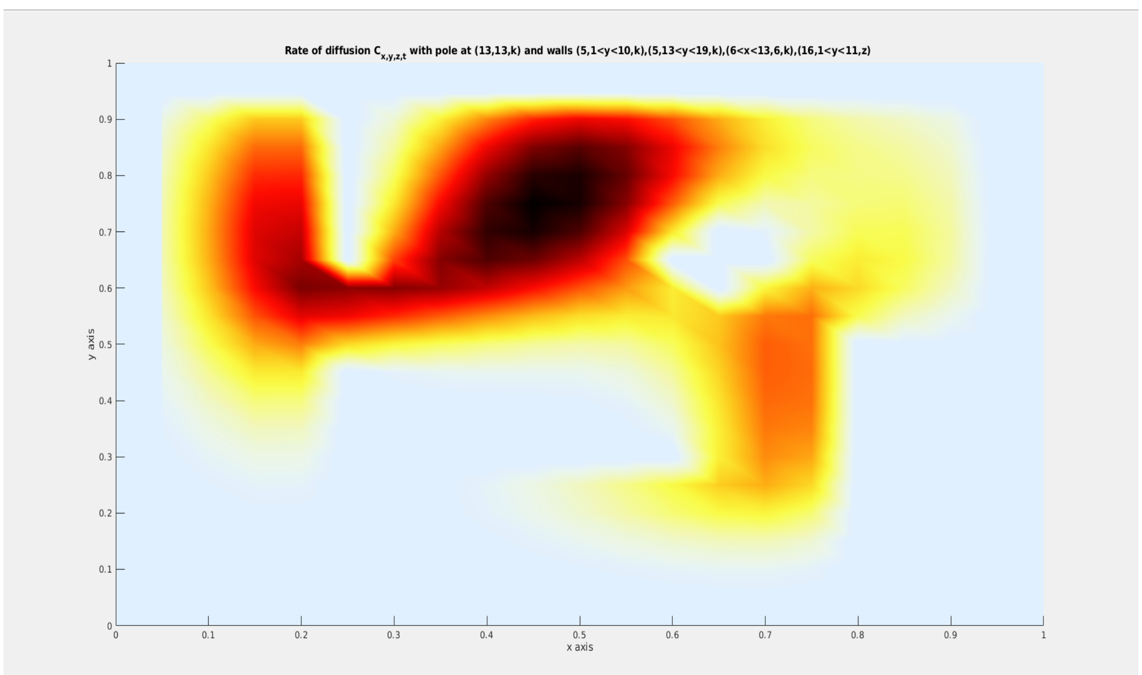


Example 3

Considering diffusion in 3D and in 2D with linear speed $u_x = 0.05\frac{m}{s}, v_y = 0.07\frac{m}{s}, w_z = 0.0\frac{m}{s}$.
initial pollution in two points at t=0 - $C_{(0.3,0.1,0.5,0)} = 300$, $C_{(0.1,0.1,0.5,0)} = 300$ the domain is 1x1x1 cm^3, 2000 time steps and final time 20 *sec*. The image represents t=7.5 s also we add one columns in (0.65,0.65,z) and walls in (0.25,0.01<=y<=0.5,z),(0.25,0.65<=y<=0.95,z),(0.3<=x<=0.65,0.3,z),(0.8,0.01<=y<=0.55,z). to create a unique flow of matter.

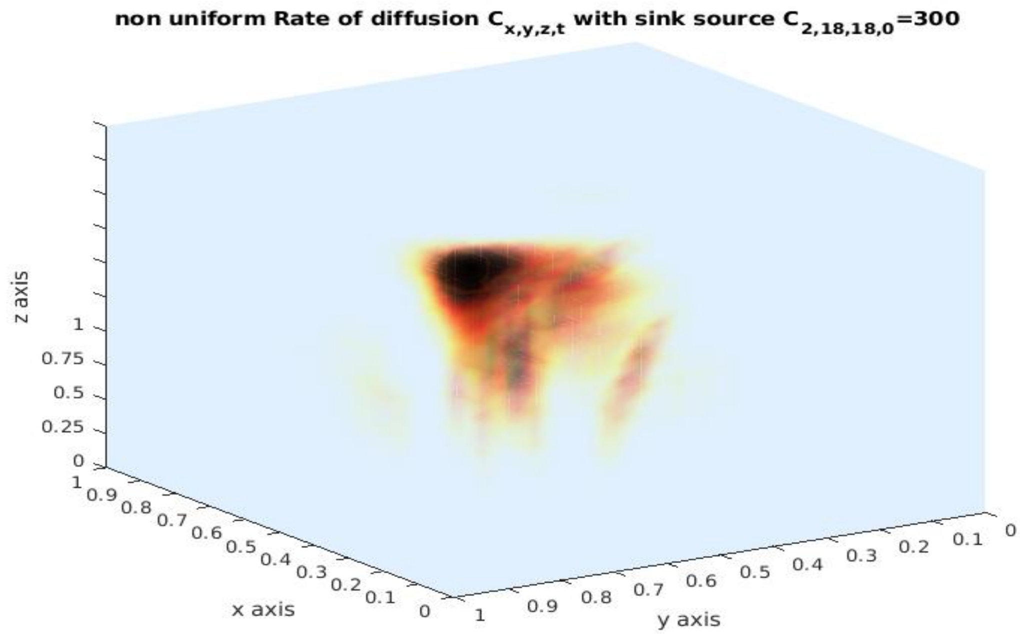


Looking at the 2d slice (x,y) of the diffusion we get.

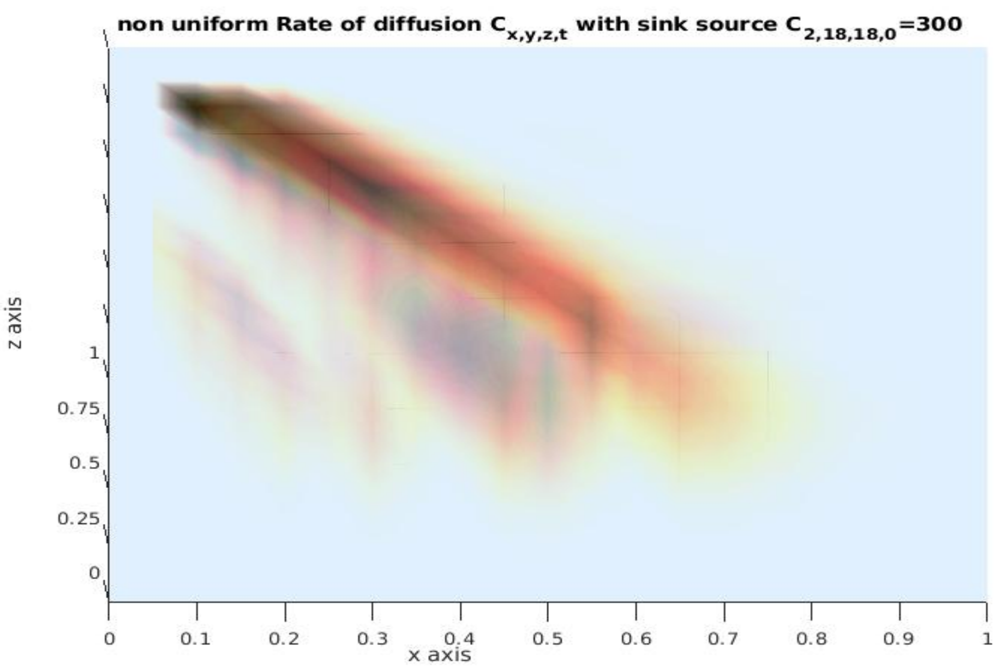


Example 4

Considering diffusion in 3D and in 2D with linear speed of $u_x = 0.05\frac{m}{s}, v_y = 0.05\frac{m}{s}, w_z = -0.05\frac{m}{s}$.
initial pollution at $C_{(0.01,0.98,0.98)} = 20$, in all times. the domain is 1x1x1 cm^3, 2000 time steps and final time 15 *sec.* creating clouds distribution.



Looking at the 2d slice (y,z) of the diffusion we get.



0.5 Analytical solution

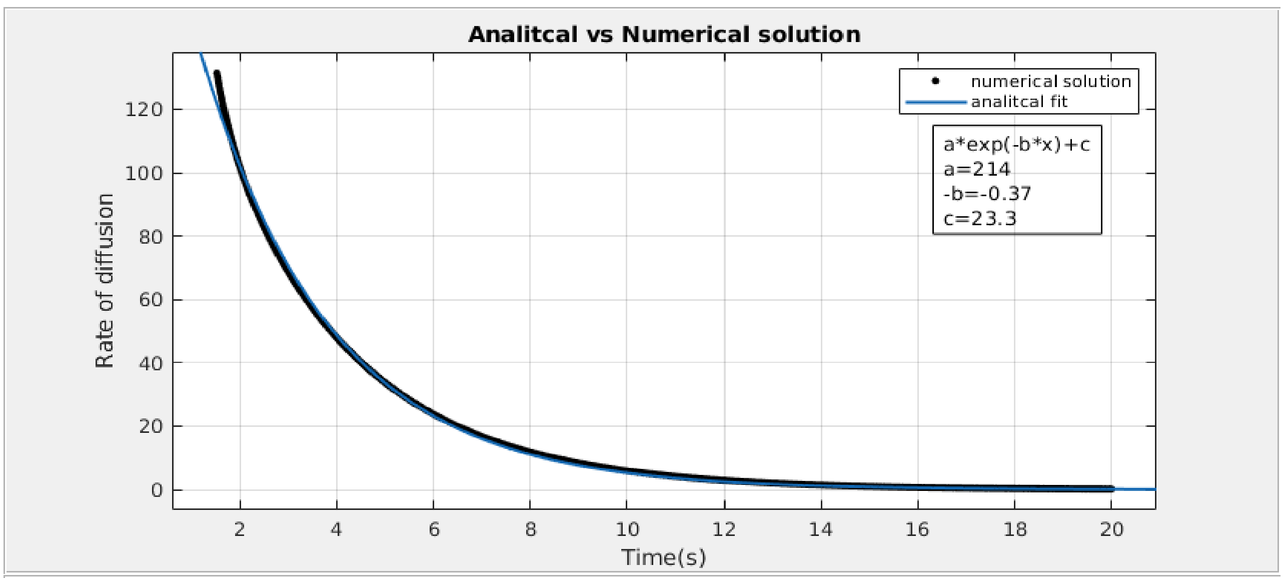
Considering the analytical solution for the 3D diffusion equation:

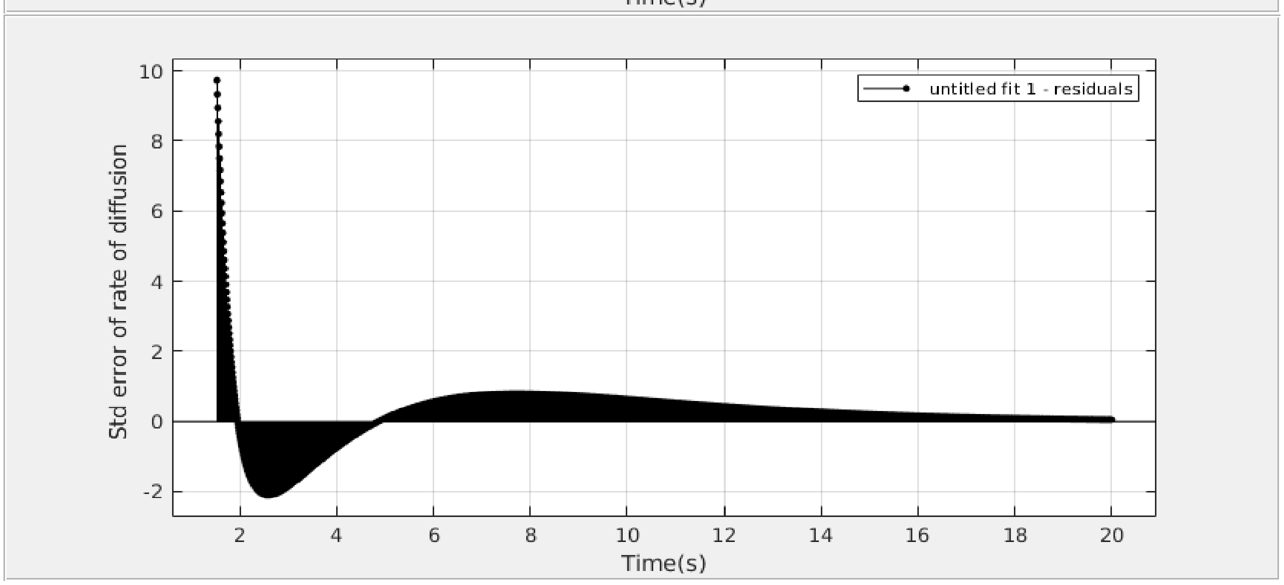
$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial t} + v\frac{\partial C}{\partial t} + w\frac{\partial C}{\partial t} = D_x\frac{\partial^2 C}{\partial x^2} + D_y\frac{\partial^2 C}{\partial y^2} + D_z\frac{\partial^2 C}{\partial z^2} \quad [1]$$

Equals to the expiration in the form of:

$$C_{(x,y,z,t)} = \frac{M}{(4\pi t)^{\frac{3}{2}}\sqrt{D_x D_y D_z}}\mathbf{e}^{-\frac{x^2}{4D_x t}-\frac{y^2}{4D_y t}-\frac{z^2}{4D_z t}}$$

While -M- is the mass realised in the point of grid (x,y,z) at t=0. D_x, D_y, D_z are the diffusibility constant in the domain. The function above describes the exponential concentration decay over time. Compering our results to the analytic results we expect to get the same decay and the same graph with an estimated error. when compairing with our numerical solutions at $tf = 20s$ and $M(x,y,z) = 130$ at a single given point $C(10,10,10,0) = 130$; As you can see the error decays with time:





YouTube for all experiments

here:

https://youtu.be/unfME-Jeg8U

0.6 Source code, matlab

—————— C++ SOURCE CODE ——————

```
//------------------------------------ 3D Diffusion equation c++------------------------------
// This code solves the three-dimensional diffusion equation using numerical
// tools for the patron of differential equations and numerical derivatives.
// The domain where the diffusiom occur is in 1x1x1 cubical box, Begining
// by defining the time and place interval, and the stability corient.
// Set initial conditions for concentration through all the domain (i,j,k)
// at time t = 0 to be -0- to initialize diffusion.
// Creating temporery arrey and initializing it to.
// Defining pollution at some point in the domain.
// Creating the numeric derivatives to itterat
// over them using nested loops.
// Printing the results to a data file for later use.
// Finishing with the loop over time.
//------------------------------------------------------------------------------------------

#include <iostream>          // Including Standard Input / Output Streams Library.
#include <fstream>           // Including Input/output file stream class iostream.
#include <cstdlib>           // Including several general purpose functions stdlib.
#include <string>            // Including string to represent sequences of characters.
#include <cmath>             // Including cmath to compute common mathematical operations.

using namespace std;

int main(){                  // Opening int main to preform calculations.

//------------------------------------- Decleration of variables.

const int P = 55;           // Size of wanted arrey.

long double C[P][P][P];     // Creating the inital arrey of the diffusion.

long double C_t[P][P][P];   // Creating a temporery arrey to itterat over time on it.

int Ny,Nx,Nz,Nt;            // Declering the the number of steps in every axis and time.

int i,j,k,l;                       // Declering the loop integers.

long double dx,dy,dz,dt;           // Differentials of x,y,z,t axis.

long double Sx,Sy,Sz;              // Defining constant values.

long double Cx,Cz,Cy;              // Defining constant values.

long double x,y,z,t;               // 3D coardinates.

long double u,v,w;                 // Currents speed in the u,v,w coardinates.

long double coreant;               // Stability coreant.

```

```
long double coreant2;              // Stability coreant.

long double Difx,Dify,Difz;        // Diffusibility constants.

//———————————————————————————— Variables calculation.

ofstream  myfile("diff.dat");      // Opening data file for later use.

myfile.precision(17);              // Defining our precision.

string buf;                        // String stream buffer.

cout <<" Enter final time:"<<endl; // Asking the user for the final time.

cin >>t;getline(cin,buf);          // Entering the value of time to the memory.

Nx=21;                             // Creating number of steps in the 3D axis.

 Ny=21;                            // Creating number of steps in the 3D axis.

 Nz=21;                            // Creating number of steps in the 3D axis.

 Nt=2001;                          // Creating number of steps in the 3D axis.

Difx= 0.00001;                     // The Diffusibility constant in the x direction.

Dify= 0.00001;                     // The Diffusibility constant in the y direction.

Difz= 0.0001;                      // The Diffusibility constant in the z direction.

dx=1.0/(Nx-1);                     // Calculating axis Differentials.

dy=1.0/(Ny-1);                     // Calculating axis Differentials.

dz=1.0/(Nz-1);                     // Calculating axis Differentials.

dt=t/(Nt-1);                       // Calculating time Differentials.

u=0.0; v=0.0; w=0.0;               // Currents speeds in the u,v,w grid.


Sx=(Difx*dt)/(pow(dx,2)); // Creating the constant parameters of the numeric calculation.

Sy=(Dify*dt)/(pow(dy,2)); // Creating the constant parameters of the numeric calculation.

Sz=(Difz*dt)/(pow(dz,2)); // Creating the constant parameters of the numeric calculation.

Cx=(u*dt)/(dx);          // Creating the constant parameters of the numeric calculation.

Cy=(v*dt)/(dy);           // Creating the constant parameters of the numeric calculation.

Cz=(w*dt)/(dz);           // Creating the constant parameters of the numeric calculation.

```

```
coreant=Sx+Sy+Sz;           // Calculating coreantS for stability.

coreant2 = (pow(Cx,2)/Sx) +(pow(Cy,2)/Sy)+(pow(Cz,2)/Sz);

if(coreant2 > 3.0){cout << "cnt2 =" << coreant2 << endl; return(0);}  // coreant<3.

if(coreant > 0.5){cout << "Cnt=" << coreant << endl;return(0);}        // coreant<0.5.

cout <<"  dt=  "<<dt<<"  dx= "<<dx<<"  dy= "<<dy<<"  dz= "<<dz<< endl;

cout << "cnt2 = " <<coreant2<<" cnt = " <<coreant<< endl;

//------------------------------------------------- Inital conditions and bounderies

                                              // Opening the nested for loops.

for(k=1;k<=Nz-1;k++)                          // Loop for z axis.

    for(i=1;i<=Nx-1;i++)                      // Loop for x axis.

        for(j=1;j<=Ny-1;j++){                 // Loop for y axis.


C[i][j][k]=0.0;                               // Inital condition to preset all the grid.

C_t[i][j][k]=0.0;                             // Doing the same with the temporery arrey.

// ------------    DIFFUSION RATE AT T=0     -------------

C[7][7][18]=150;                              //generating diffusion rate in the grid at t=0.
// -------------------------------------------------------------
}                                             // For loop {}

l=0;                                          // Initalizing l to be zero for the loop.

do{                                           // Opening do loop to preform time.

     for(i=1;i<Nx-1;i++)                      // Opening nested for loop to calculat values.

          for(j=1;j<Nx-1;j++)                 // Same for j.

               for(k=1;k<Nz-1;k++){           // Same for j.

C_t[i][j][k]=                                 // Governing equation.
(Sx+(Cx/2.0))*C[i-1][j][k]
+(Sy+(Cy/2.0))*C[i][j-1][k]
+(Sz+(Cz/2.0))*C[i][j][k-1]
+(Sx-(Cx/2.0))*C[i+1][j][k]
+(Sy-(Cy/2.0))*C[i][j+1][k]
+(Sz-(Cz/2.0))*C[i][j][k+1]
+(1.0-2.0*(coreant))*C[i][j][k];
```

```
C[i][j][k]=C_t[i][j][k];            // Updating the teporal arrey for time t+dt.

//------------------------------------------------------- boundery conditions.


//---------------------------------- Forward in time backward in space for the end of x axis.

C_t[Nx-1][j][k]=C[Nx-1][j][k]
-(Cx/2)*(3.0*C[Nx-1][j][k]-4.0*C[Nx-2][j][k]+3.0*C[Nx-3][i][k])
-(Cy/2)*(C[Nx-1][j+1][k]-C[Nx-1][j-1][k])
-(Cz/2)*(C[Nx-1][j][k+1]-C[Nx-1][j][k-1])
+Sx*(C[Nx-1][j][k]-2.0*C[Nx-2][j][k]+C[Nx-3][j][k])
+Sy*(C[Nx-1][j+1][k]-2.0*C[Nx-1][j][k]+C[Nx-1][j-1][k])
+Sz*(C[Nx-1][j][k+1]-2.0*C[Nx-1][j][k]+C[Nx-1][j][k-1]);

//---------------------------------- Forward in time backward in space for the end of y axis.

C_t[i][Ny-1][k]=C[i][Ny-1][k]
-(Cx/2)*(C[i+1][Ny-1][k]-C[i-1][Ny-1][k])
-(Cy/2)*(3.0*C[i][Ny-1][k]-4.0*C[i][Ny-2][k]+3.0*C[i][Ny-2][k])
-(Cz/2)*(C[i][Ny-1][k+1]-C[i][Ny-1][k-1])
+Sx*(C[i+1][Ny-1][k]-2.0*C[i][Ny-1][k]+C[i-1][Ny-1][k])
+Sy*(C[i][Ny-1][k]-2.0*C[i][Ny-2][k]+C[i][Ny-3][k])
+Sz*(C[i][Ny-1][k+1]-2.0*C[i][Ny-1][k]+C[i][Ny-1][k-1]);

//---------------------------------- Forward in time backward in space for the end of z axis.

C_t[i][j][Nz-1]=C[i][j][Nz-1]
-(Cx/2)*(C[i+1][j][Nz-1]-C[i-1][j][Nz-1])
-(Cy/2)*(C[i][j+1][Nz-1]-C[i][j-1][Nz-1])
-(Cz/2)*(3.0*C[i][j][Nz-1]-4.0*C[i][j][Nz-2]+3.0*C[i][j][Nz-3])
+Sx*(C[i+1][j][Nz-1]-2.0*C[i][j][Nz-1]+C[i-1][j][Nz-1])
+Sy*(C[i][j+1][Nz-1]-2.0*C[i][j][Nz-1]+C[i][j-1][Nz-1])
+Sz*(C[i][j][Nz-1]-2.0*C[i][j][Nz-2]+C[i][j][Nz-3]);

//---------------------------------- Forward in time Forward in space for the end of x axis.

C_t[1][j][k]=C[1][j][k]
-(Cx/2)*(C[1][j][k]-4.0*C[2][j][k]+3.0*C[3][i][k])
-(Cy/2)*(C[1][j+1][k]-C[1][j-1][k])
-(Cz/2)*(C[1][j][k+1]-C[1][j][k-1])
+Sx*(C[1][j][k]-2.0*C[2][j][k]+C[3][j][k])
+Sy*(C[1][j+1][k]-2.0*C[1][j][k]+C[1][j-1][k])
+Sz*(C[1][j][k+1]-2.0*C[1][j][k]+C[1][j][k-1]);

//---------------------------------- Forward in time Forward in space for the end of y axis.

C_t[i][1][k]=C[i][1][k]
-(Cx/2)*(C[i+1][1][k]-C[i-1][1][k])
-(Cy/2)*(C[i][1][k]-4.0*C[i][2][k]+3.0*C[i][3][k])
-(Cz/2)*(C[i][1][k+1]-C[i][1][k-1])
+Sx*(C[i+1][1][k]-2.0*C[i][1][k]+C[i-1][1][k])
+Sy*(C[i][1][k]-2.0*C[i][2][k]+C[i][3][k])
```

```
+Sz*(C[i][1][k+1]-2.0*C[i][1][k]+C[i][1][k-1]);

//———————————————— Forward in time Forward in space for the end of z axis.

C_t[i][j][1]=C[i][j][1]
-(Cx/2)*(C[i+1][j][1]-C[i-1][j][1])
-(Cy/2)*(C[i][j+1][1]-C[i][j-1][1])
-(Cz/2)*(C[i][j][1]-4.0*C[i][j][2]+3.0*C[i][j][3])
+Sx*(C[i+1][j][1]-2.0*C[i][j][1]+C[i-1][j][1])
+Sy*(C[i][j+1][1]-2.0*C[i][j][1]+C[i][j-1][1])
+Sz*(C[i][j][1]-2.0*C[i][j][2]+C[i][j][3]);

//———————————————— Updating edge points.

C[0][j][k]=0.0;         // For x axis.

C[i][0][k]=0.0;         // For y axis.

C[i][j][0]=0.0;         // For z axis.

C[Nx][j][k]=0.0;        // For x axis.

C[i][Ny][k]=0.0;        // For y axis.

C[i][j][Nz]=0.0;        // For z axis.

}                       // Closing for loop {}

l++;                    // Updating l to be +1 every time in the do loop.

//———————————————Creating data file.

for(k=1;k<=Nz-1;k++)                                   // Nested for loop.

    for(i=1;i<=Nx-1;i++)                               // Nested for loop.

        for(j=1;j<=Nx-1;j++){                          // Nested for loop.



myfile  << C[i][j][k] << '\n';  // Inserting data.

}                                                      // Closing for loop{}

}                                                      // Closing do loop{}

while(l<Nt-1);                                         // End of the do loop.

myfile.close();                                        // Closing file.

}                                                      // Closing main.
```

———— MATLAB SCRIPT ————

```matlab
Data=load('diff.dat');    %uploading data to matlab
C=Data(:,1)';
% -------------------------------------------------------------------------------

c=num2cell(reshape(C, 19*19*19, 2000 ),1); % in c++ we have a grid of 19x19x19,
%with 2000 steps in t
%
%so we start sorting by placing each step in time as a seperate cell
% 2000 cells of vectorized 19x19x19 grids, {1:2000}(1x6859) double

for(i=1:length(c))
    SUMPLOT(i)=abs(sum(c{i}));    % vector of the rate of diffusion to
                                   %compare to analitical
end

%% 2D-----------------------------------------------------------------------------
for(i=1:length(c))
 C1{i}=reshape(c{i},19, 19, 19);% again reshaping from
                                 %a 1x6859 vector to 19x19x19 grid
end

%%


for(i=1:length(c)/10) % animation for loop
   C1{i*10} = double(squeeze(C1{i*10})); % dimention removal for 'slice' func
   h =     slice(max(C1{i*10},1.e-15)*100, 1:2:19 , 1:1:19 , 1:2:19); % plot

    % basic settings for plot---------------------------------------------------
    set(h, 'EdgeColor','none','FaceColor','interp')
    colormap(flipud(jet))
    alpha(h,'color') % sets transparacy
    set(gca,'Color','[0.88 0.94 1]')
    shading interp %smooth shading
    grid off
    campos([  (i*100)/length(c) 7 1.4673 ]) %rotation of the video
    %campos([ 10  10   180] )            %2d veiw
    title('non uniform Rate of diffusion C_{x,y,z,t} with source C_{10,10,10,0}=300')
    xlabel('x axis')
    ylabel('y axis')
    zlabel('z axis')
     xticklabels({'0', '0.1' , '0.2', '0.3','0.4','0.5','0.6','0.7','0.8','0.9','1'})
      yticklabels({'0', '0.1' , '0.2', '0.3','0.4','0.5','0.6','0.7','0.8','0.9','1'})
        zticklabels({'0', '0.25' , '0.5', '0.75','1'})
    hold off
    pause(0.000000005) % as pause goes to 0 the movie goes faster
end
```

0.7 Conclusion

In this paper we investigated the diffusion equation considering the 3D and 2D FTCS scheme. We developed the numeric solution to the analytical differential equation and wrote it in a C++ code extracting the data of the pollution and using Matlab graphical functions to display the rate of diffusion over time in our domain. Using spacial boundary conditions we managed to keep the rate of diffusion as if enclosed in a room and to diffuse while remaining in the boarders of the domain. We have learned how to deal with complected numerical schemes and equations and how to simplify it to the software we use. To conclude the results received are satisfying to our desire and to the project demands with estimated error within our grid and time spatial while comparing them to the analytic solution(s).

Reaction-diffusion equation is applied in many areas of specialization for example, in developmental biology (Ref Alan Turing Work), neuroscience (Ref Hodgkin-Huxley work on Action Potential Propagation), Calcium dynamics (Ref James Sneyd)

Bibliography

[1] [Thai Journal of MathematicsVolume 5(2007) Number 1 : 91–108]
"Numerical Solution of a 3-D Advection-DispersionModel for Pollutant Transport"
http://thaijmath.in.cmu.ac.th/index.php/thaijmath/article/viewFile/188/183

[2] "Boundary Value Problems: The Finite Difference Method"
http://web.mit.edu/10.001/Web/Course_Notes/Differential_Equations_Notes/node9.html

[3] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.2962&rep=rep1&type=pdf