Diffusion equation

Contents

List of Figures

1 Introduction

Diffusion is the net movement of anything (for example, atoms, ions, molecules, energy) generally from a region of higher concentration to a region of lower concentration ([1]). As defined in Wikipedia, the motion can happen in any medium and it can be used to define not only the motion of substances but also organisms like planton or bacteria. It is generally described by partial differential equation called *the diffusion equation*:

$$u_t(\mathbf{x}, t) = -\nabla \mathbf{J}(\mathbf{x}, t).$$

Where u is the density of the diffusing material at location $\mathbf{x}$ and time t and $\mathbf{J}$ is a flux representing the quantity and direction of transfer. Depending on $\mathbf{J}$ the equation is representing a different phenomena and solutions will differ.

2 Heat equation

The heat equation is modeling the distribution of heat in a n-dimensional object. It is one of the most widely studied subject in mathematics, used in variety of applicatations ([2]). In probability theory it is connected to the random walks and Brownian motions. This equation is a special case of diffusion where $\mathbf{J} = -d\nabla u(\mathbf{x}, t)$:

$$u_t(\mathbf{x}, t) = -\nabla(-d\nabla u(\mathbf{x}, t)) = d\Delta u(\mathbf{x}, t).$$

2.1 Starting and boundary conditions

To find the solution of the heat equation one needs starting and boundary conditions. Starting condition is a function that defines the distribution of heat at time $t = 0$:

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}).$$

The boundary conditions determine what is happening at the boundary of the set ($\partial\Omega$) that the heat equation is defined on. There are two types boundary conditions:

- Dirichlet boundary conditions:
$$u(\mathbf{x}, t) = f(\mathbf{x}, t), \quad \mathbf{x} \in \partial\Omega.$$
- Neumann boundary conditions.
$$\vec{\eta}(\mathbf{x}) \cdot \nabla u(\mathbf{x}, t) = f(\mathbf{x}, t), \quad \mathbf{x} \in \partial\Omega.$$
Where $\vec{\eta}$ is a normal vector of boundary (perpendicular to the boundary) at $\mathbf{x}$ and $\cdot$ is a dot (scalar) product.

One can mix those two types of conditions to formulate adequate problems.

2.2 Statement of the problem

We were given two problems, regarding to one and two dimensional cases.

2.2.1 Interval (1 dimension)

In this case $\Omega = [0, 1]$ can be thought of a 1-dimensional rod. For given conditions, what is the solution of the heat equation:

$$\begin{cases} u_t(x, t) = du_{xx}(x, t), \\ u_0(x) = \sin(3\pi x), \\ u(0, t) = -1, \\ u_x(1, t) = 1. \end{cases}$$

For which $d\frac{dt}{dx^2}$ numerical method diverges?

2.2.2 Square (2 dimensions)

In this case $\Omega = [0,1]^2$ can be thought of a 2-dimensional plate. For given conditions, what is the solution of the heat equation:

$$\begin{cases} u_t(x,y,t) = d(u_{xx}(x,t) + u_{yy}(x,t)), \\ u_0(x) = \sin(3\pi x)\sin(3\pi y), \\ u(x,1,t) = -1, \\ u(x,0,t) = 1, \\ u_{\vec{\eta}}(x,y,t) = 1, \quad x = 0 \vee x = 1. \end{cases}$$

For which $d\frac{dt}{dx^2}$ numerical method diverges?

3 Theoretical solutions

There exisist theoretical solutions for heat equations, but they are complex and heavy computationaly. One can solve problems stated in previous section (sec. 2.2) using Fourier series.

3.1 Interval (1 dimension)

Let's assume that function $u(x,t)$ is a product of functions $X(x)$ and $T(t)$. Then the heat equation and conditions have form:

$$\begin{cases} X(x)T'(t) = dX''(x)T(t) \\ u_0(x) = \sin(3\pi x), \\ X(0)T(t) = -1, \\ X'(1)T(t) = 1. \end{cases}$$

From the boundary conditions:

$$-X(0) = X'(1) = \frac{1}{T(t)}.$$

Dividing heat equation by $dX(x)T(t)$:

$$\frac{T'(t)}{dT(t)} = \frac{X''(x)}{X(x)} = \lambda.$$

These equations are equal to constant λ, because they depend on different variables so no other option is available. Solving for $T(t)$ using the theory of ODE's:

$$\begin{aligned} \frac{1}{d}(\log(T(t)))' &= \lambda \\ T(t) &= Ce^{d\lambda t} \end{aligned}$$

Solving for $X(x)$ using the theory of ODE's:

$$\begin{aligned} X''(x) - \lambda X(x) &= 0, \\ r^2 - \lambda &= 0, \\ r &= \pm\sqrt{\lambda}. \end{aligned}$$

There are 3 types of solutions, depending on λ:

(1) $\lambda > 0$

$$X(x) = c_1 e^{\sqrt{\lambda}x} + c_2 e^{-\sqrt{\lambda}x}$$

Given the boundary conditions:

$$\begin{aligned} -(c_1 + c_2) &= \frac{1}{T(t)} \\ c_1\sqrt{\lambda}e^{\sqrt{\lambda}} + c_2 - \sqrt{\lambda}e^{-\sqrt{\lambda}} &= \frac{1}{T(t)} \end{aligned}$$

(2) $\lambda = 0$

$$X(x) = c_1 x + c_2$$

Given the boundary conditions:

$$-c_2 = \frac{1}{T(t)}$$
$$c_1 = \frac{1}{T(t)}$$

(3) $\lambda < 0 \Rightarrow r = \pm i\sqrt{|\lambda|}$:

$$X(x) = c_1 \cos(\sqrt{|\lambda|}x) + c_2 \sin(\sqrt{|\lambda|}x)$$

Given the boundary conditions:

$$-c_1 = \frac{1}{T(t)}$$
$$-c_1 \sin(\sqrt{|\lambda|}) + c_2 \cos(\sqrt{|\lambda|}) = \frac{1}{T(t)}$$

I assume that given non-homogeneous boundary conditions one cannot find constants C, c_1, c_2. If there is possibility of achieving that, there will be known formulas for infinitely many functions $X_k(t), T_k(t)$ and hence infinitely many functions $u(x,t)$. Those functions would not satisfy the starting condition, so one can use sum of all those solutions and extract the weights using Fourier series:

$$u(x,t) = \sum_{k=1}^{\infty} b_k X_k(t) T_k(t) ???$$

$$\frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos\left(\frac{k\pi}{2}\right) + b_k \sin\left(\frac{k\pi}{2}\right) = u(x,0) = u_0(x) = \sin(3\pi x).$$

Where $a_k = 0$ because sine is is an odd function and b_k has form:

$$b_k = 2\int_0^1 u_0(x)\sin(k\pi x)dx$$

Now the function $u(x,t)$ satisfies all of the conditions.

3.2 Square (2 dimensions)

This case is analogous to the previous one, but now let's assume that function $u(x,y,t)$ is a product of functions $X(x), Y(y), T(t)$, then:

$$\begin{cases} X(x)Y(y)T'(t) = dX''(x)Y(y)T(t) + dX(x)Y''(y)T(t) \\ u_0(x,y) = \sin(3\pi x)\sin(3\pi y), \\ X(x)Y(1)T(t) = -1, \\ X(x)Y(0)T(t) = 1, \\ X(0)Y'(y)T(t) = -1, \\ X(1)Y'(y)T(t) = 1. \end{cases}$$

Dividing heat equation by $dX(x)Y(y)T(t)$:

$$\frac{T'(t)}{dT(t)} = \frac{X''(x)}{X(x)} + \frac{Y''(y)}{Y(y)} = \lambda.$$

The rest of calcutations are similar to calculations done with 1-dimensional rod. The problems arise again with non-homogeneity of conditions and some formulas cannot be computed. Those problems force us to work with numerical methods to obtain solutions.

4 Numerical methods

Since there is no way to obtain analitical solutions to problems stated (sec. 2.2), the only option left is to use numerical methods. Using finite differences method one would formulate one step of iteration:

- Interval (1 dimension)

$$u(x, t + dt) = u(x, t) + d\frac{dt}{dx^2}\left(u(x + dx, t) + u(x - dx, t) - 2u(x, t)\right).$$

- Square (2 dimensions)

$$u(x, y, t+dt) = u(x, y, t)+d\frac{dt}{dx^2}\left(u(x + dx, y, t) + u(x - dx, y, t) + u(x, y + dy, t) + u(x, y - dy, t) - 4u(x, y, t)\right).$$

Start of this iterative method is given by starting condition $u(\mathbf{x}, 0) = u_0(\mathbf{x})$. The boundaries are calculated using boundary conditions:

- Dirichlet boundary conditions:

$$u(\mathbf{x}, t) = f(\mathbf{x}, t), \quad \mathbf{x} \in \partial\Omega.$$

- Neumann boundary conditions:

$$u(\mathbf{x}, t) = u(x_1, x_2, \ldots, x_i \pm dx, x_{i+1}, \ldots, x_n) + dx \cdot f(\mathbf{x}, t), \quad \mathbf{x} \in \partial\Omega.$$

Where x_i is on the boundary and the $\pm$ sign depends on which boundary (dx must be substracted or added in such a way that $x_i \pm dx$ is in Ω).

5 Numerical solutions

Algorithm described above (sec. 4) was implemented in Python3, using NumPy package and calculations were done using double precision.

5.1 Solution for interval (1 dimension)

Solution obtained via finate differences method for interval is presented in fig. 5.1 ($d = 0.05$). Function $u(x, t)$ is continuous, starts with a sine function, at one boundary is always -1 and at the other the heat is flowing in. This means that every condition is satified.

The method loses it stability around $d = 0.055$ and gives no solution. This situation in presented in fig. 5.2.

5.2 Solution for square (2 dimensions)

Solution obtained via finate differences method for interval is presented in fig. 5.3 ($d = 0.025$). Function $u(x, y, t)$ is continuous, starts with a product of sine functions, at boundaries 1 and 3 is always -1 and 1 respectively and at the others the heat is (slowly) flowing in.

The method loses it stability around $d = 0.026$ and gives wrong solution. This situation in presented in fig. 5.4.

6 Conclusions

The diffusion and heat are complex phenomena and finding their solutions (formula for function) analyticaly is not always possible. The numeric methods give us a way to get close approximations. They also have drawbacks – for some values of d numeric methods fail. Visualising those solutions is also not possible when dimensions of objects exceed 2.

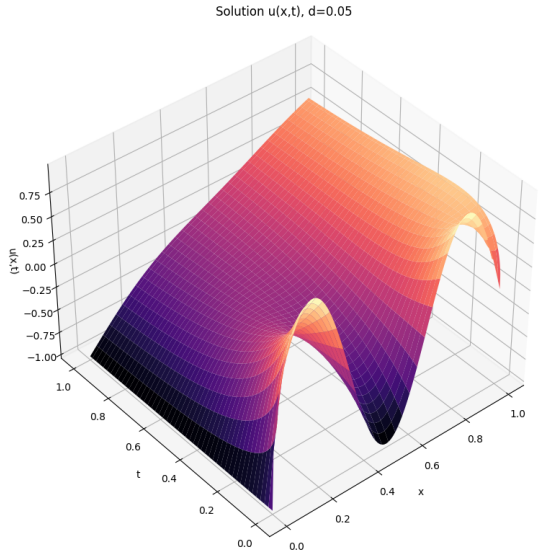

Figure 5.1: Solution for $u(x,t)$, $d = 0.05$

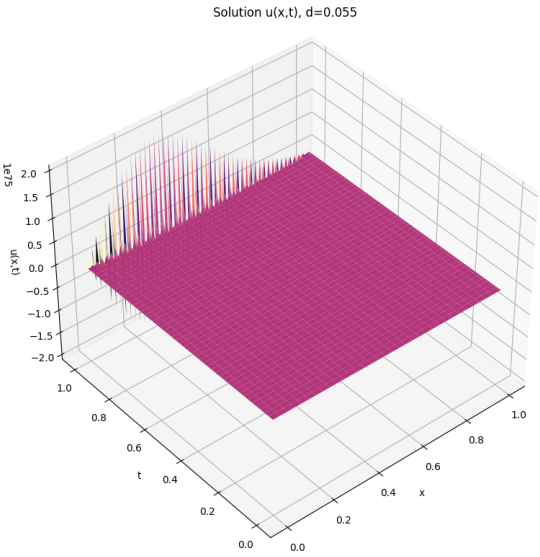

Figure 5.2: Solution for $u(x,t)$, $d = 0.055$

References

[1] Wikipedia contributors. Diffusion — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Diffusion&oldid=1060838939`, 2021. [Online; accessed 21-December-2021].

[2] Wikipedia contributors. Heat equation — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Heat_equation&oldid=1060098671`, 2021. [Online; accessed 21-December-2021].

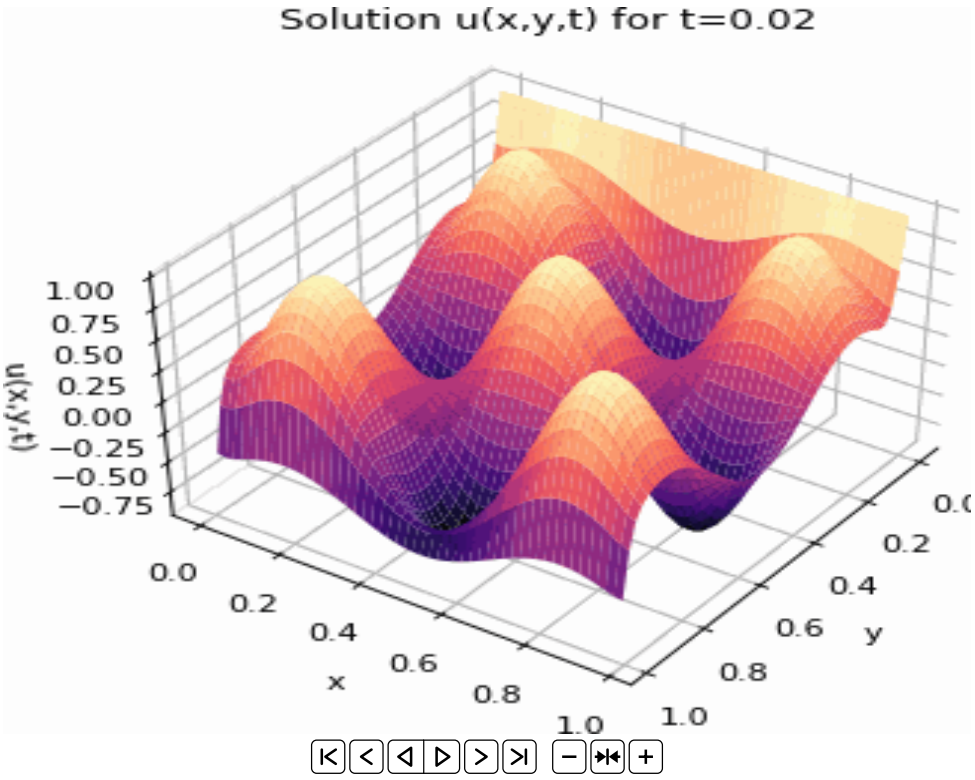

Figure 5.3: Solution for $u(x, y, t)$, $d = 0.025$

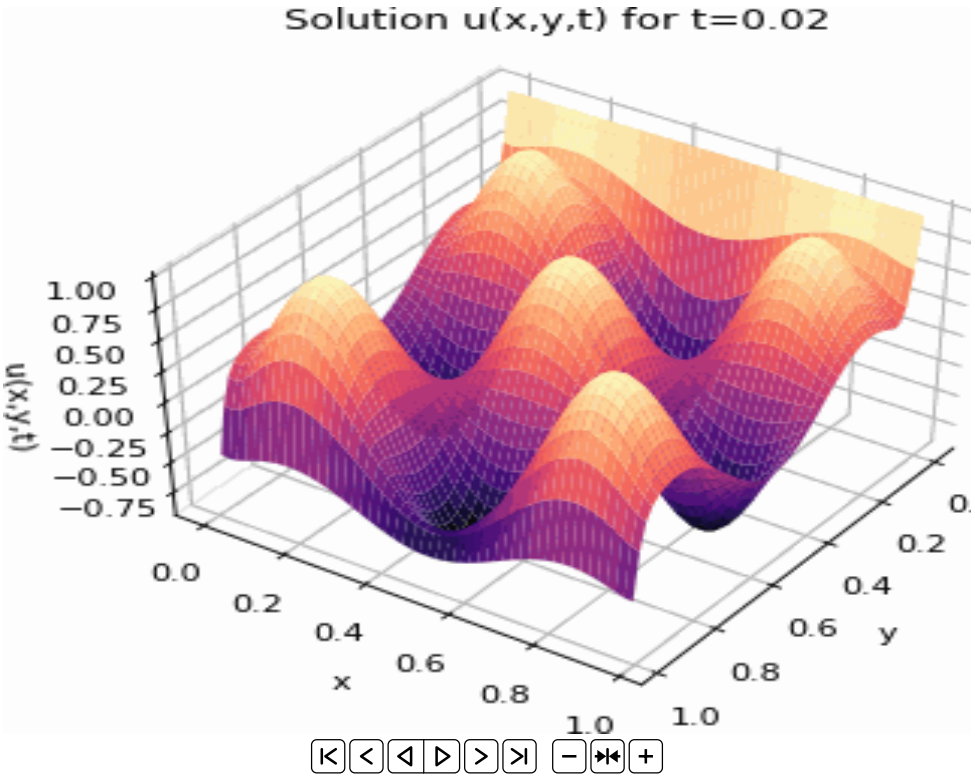

Figure 5.4: Solution for $u(x, y, t)$, $d = 0.026$

Appendices

Appendix 1 Python code

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

#Helpful functions
def ab_indicator(a=0, b=1, const=1):
    return lambda x: const * (a<=x) * (x<=b)

def f_const(c):
    return lambda *args: c

##################################################################
################SOLVING HEAT EQUATION#############################

#Rod/Interval (1-dimension)
def solve_heat_1dim(u_0, u_0t=None, u_1t=None, u_x_0t=None, u_x_1t=None, D
    =0.01, T_max=10, x_min=0, x_max=1, dt=0.01, dx=0.1):
    x = np.arange(x_min, x_max+dx, step=dx)
    t = np.arange(0, T_max + dt, step=dt)
    U = np.zeros((t.size, x.size))
    U[0, :] = u_0(x)
    if(u_0t is not None):
        U[:, 0] = u_0t
    if(u_1t is not None):
        U[:, -1] = u_1t

    for i in range(0, t.size-1):
        for j in range(1, x.size-1):
            U[i+1, j] = U[i, j] + D * (dt/dx**2) * ( U[i, j-1] + U[i, j+1]
                - 2* U[i, j] )

        if(u_x_0t is not None):
            U[i+1, 0] = U[i+1, 1] + dx*u_x_0t
        if(u_x_1t is not None):
            U[i+1, -1] = U[i+1, -2] + dx*u_x_1t
    return t, x, U

#Plate/Square (2-dimensions)
def solve_heat_2dim(u_0, u_1=None, u_2=None,  u_3=None, u_4=None, u_x_1=
    None, u_x_2=None, u_x_3=None, u_x_4=None, D=0.01, T_max=10, x1_min=0,
    x1_max=1, x2_min=0, x2_max=1, dt=0.01, dx=0.1):
    x1 = np.arange(x1_min, x1_max+dx, step=dx)
    x2 = np.arange(x2_min, x2_max+dx, step=dx)
    t = np.arange(0, T_max + dt, step=dt)
    U = np.zeros((t.size, x1.size, x2.size))
    X1, X2 = np.meshgrid(x1, x2)
    U[0, :, :] = u_0(X1, X2)

    if(u_1 is not None):
        U[:, -1, :] = u_1(X1[-1,:], X2[-1, :])
    if(u_2 is not None):
        U[:, :, -1] = u_2(X1[:, -1], X2[:, -1])
```

```
    if(u_3 is not None):
        U[:, 0, :] = u_3(X1[0, :], X2[0, :])
    if(u_4 is not None):
        U[:, :, 0] = u_4(X1[:, 0], X2[:, 0])

    for i in range(0, t.size-1):
        #for j in range(1, x1.size-1):
            #for k in range(1, x2.size-1):
                #U[i+1, j, k] = U[i, j, k] + D * (dt/dx**2) * ( U[i, j-1, k
                    ] + U[i, j+1, k] + U[i, j, k-1] + U[i, j, k+1] - 4* U[i,
                     j, k] )
        U[i+1, 1:-1, 1:-1] = U[i, 1:-1, 1:-1] + D * (dt/dx**2) * ( U[i,
            :-2, 1:-1] + U[i, 2:, 1:-1] + U[i, 1:-1, :-2] + U[i, 1:-1, 2:] -
             4* U[i, 1:-1, 1:-1] )

        if(u_x_1 is not None):
            U[i+1, -1, 1:-1] = U[i+1, -2, 1:-1] +  dx*u_x_1(t[i+1], X1
                [-1,:], X2[-1, :])
        if(u_x_2 is not None):
            U[i+1, 1:-1, -1] = U[i+1, 1:-1, -2] +  dx*u_x_2(t[i+1], X1[:,
                -1], X2[:, -1])
        if(u_x_3 is not None):
            U[i+1, 0, 1:-1] = U[i+1, 1, 1:-1] +  dx*u_x_3(t[i+1], X1[0, :],
                 X2[0, :])
        if(u_x_4 is not None):
            U[i+1, 1:-1, 0] = U[i+1, 1:-1, 1] +  dx*u_x_4(t[i+1], X1[:, 0],
                 X2[:, 0])

    return X1, X2, U

##########################################################################
########################PLOTS#############################################

#Rod/Interval [0,1], T=1 (1-dimension)
'''
u_0 = lambda x: np.sin(3*np.pi*x)

t, x, U = solve_heat_1dim(u_0=u_0, u_0t=-1, u_x_1t=1, dt=0.001, dx=0.01, D
    =0.05, T_max=1) #D=0.055
X, T = np.meshgrid(x, t)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, T, U, cmap='magma', edgecolor='none')
ax.set_title('Solution u(x,t), d=0.055')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u(x,t)')
ax.view_init(40, -130)
plt.show()
'''

#Using plotly
'''
fig = go.Figure(go.Surface(z=U))
fig.show()
'''
```

```
#Plate/Square [0,1]^2, T=1, (2-dimensions)
'''
X1, X2, U = solve_heat_2dim(u_0=u_0, u_1=f_const(-1), u_3=f_const(1), u_x_2
    =f_const(1), u_x_4=f_const(1), dt=0.001, dx=0.01, T_max=1)
#3d plots
w = go.Surface(z=U[0,:,:])
fig = go.Figure(w)
fig.show()
w = go.Surface(z=U[200,:,:])
fig = go.Figure(w)
fig.show()
w = go.Surface(z=U[500,:,:])
fig = go.Figure(w)
fig.show()
w = go.Surface(z=U[1000,:,:])
fig = go.Figure(w)
fig.show()
'''

############################################################
######################ANIMATIONS############################

def heat_animation(u_0, u_1=None, u_2=None,  u_3=None, u_4=None, u_x_1=None
    , u_x_2=None, u_x_3=None, u_x_4=None, D=0.01, T_max=10, x1_min=0, x1_max
    =1, x2_min=0, x2_max=1, dt=0.01, dx=0.1, dx_skip=2, dt_skip=10):
    X1, X2, U = solve_heat_2dim(u_0, u_1, u_2, u_3, u_4, u_x_1, u_x_2,
        u_x_3, u_x_4, D, T_max, x1_min, x1_max, x2_min, x2_max, dt, dx)
    fig = plt.figure()
    ax = plt.axes(projection='3d')

    ax.view_init(40, 35)#ax.view_init(40, -130)#
    ind = np.arange(int(1/dx), step=dx_skip)
    def heat_animate(t):
        ax.collections.clear()
        ax.plot_surface(X1[ind, :][:, ind], X2[ind, :][:, ind], U[:, ind,
            :][t, :, ind], cmap='magma', edgecolor='none')
        ax.set_title('Solution u(x,y,t) for t='+str(round(t*dt, 2)))
        ax.set_xlabel('y')
        ax.set_ylabel('x')
        ax.set_zlabel('u(x,y,t) ')
        return ax

    anim = FuncAnimation(fig, heat_animate, frames=np.arange(int(1/dt),
        step=dt_skip), interval=10)
    anim.save('MD_proj2_sol_2dim_d=0026.gif', dpi=80, writer='Pillow', fps
        =60)
    plt.show()

#u_0 = lambda x1, x2: np.sin(3*np.pi*x1) * np.sin(3*np.pi*x2)
#heat_animation(u_0=u_0, u_1=f_const(-1), u_3=f_const(1), u_x_2=f_const(1),
    u_x_4=f_const(1), dt=0.001, dx=0.01, T_max=1, D=0.026, dt_skip=10)
#u_0 = lambda x1, x2: x1*(1-x1)*x2*(1-x2)
#heat_animation(u_0=u_0, u_1=f_const(0.1), u_3=f_const(0.1), u_x_2=f_const
    (-1), u_x_4=f_const(-1), dt=0.001, dx=0.01, T_max=1)
```