PhD Diary Week Beginning 12th November

Nathan Hughes

November 16, 2018

Contents

1 TODO 1D Diffusion

Initial Equation from Fick's first law (Kaufmann, 1998):

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} \tag{1}$$

Where u can be numerically solved with:

$$u = u(x_i, t_n) \tag{2}$$

Implementing the solved equation for 1D diffusion:

$$u_i^{n+1} = u_i^n + D\frac{(u_{i+1}^n - 2u_i^n + u_{i-1}^n)}{\Delta x^2} \tag{3}$$

```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
sns.set()
plt.close('all')
def diffuse_1D(nx,dx,nt,D,dt, IC=None, slow=False):
    u = np.zeros(nx)
    mid = int(nx/2)
    dx2 = dx**2
    if IC is None:
        u[mid-int(mid/4):mid+int(mid/4)] = 1
    elif IC is 'start':
        u[:mid-int(mid/4)] = 1
    for n in range(nt):
        un = u.copy() # Update previous values
        if slow:
            for i in range(1, nx-1):
                u[i] = un[i] + D  (un[i+1] -2 * un[i] + un[i-1])/dx2
        else:
            u[1:-1] = un[1:-1] + D  * (un[0:-2] -2 * un[1:-1] + un[2:])/dx2
    return un

nx = 100 # Number of x measurements
dx = 1 #2 / (nx-1) # Change in X
nt = 1 # Number of timesteps to make in calculation
D = 0.3 # Diffusion constant
dt = 0.001 # change in time

fig, axes = plt.subplots(2,2, sharex=True, sharey=True)
dts = {nt: diffuse_1D(nx,dx,nt,D,dt) for nt in np.logspace(0,3,4,base=10, dtype=int)}

for idx, d in enumerate(dts.keys()):
    axes[idx//2, idx%2].plot(np.linspace(0,1,nx), dts[d])
    axes[idx//2, idx%2].set_title('TS: {0}'.format(d))

plt.suptitle('Diffusion in 1d')
```

1.1 Diffusion from centre

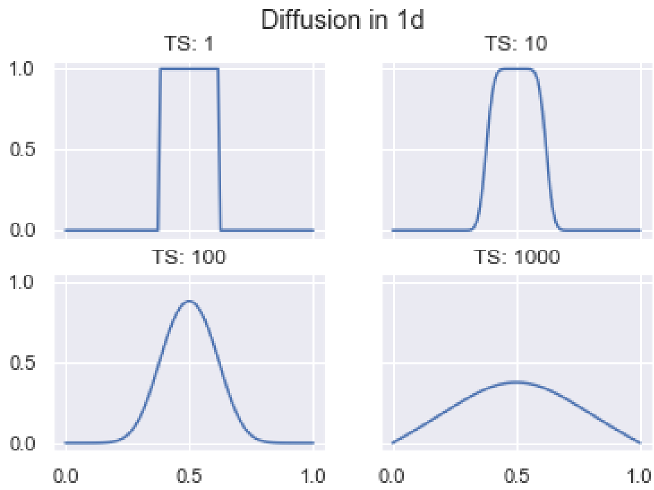

Figure 1: Diffusion initial condition from centre

1.2 Diffusion from one side

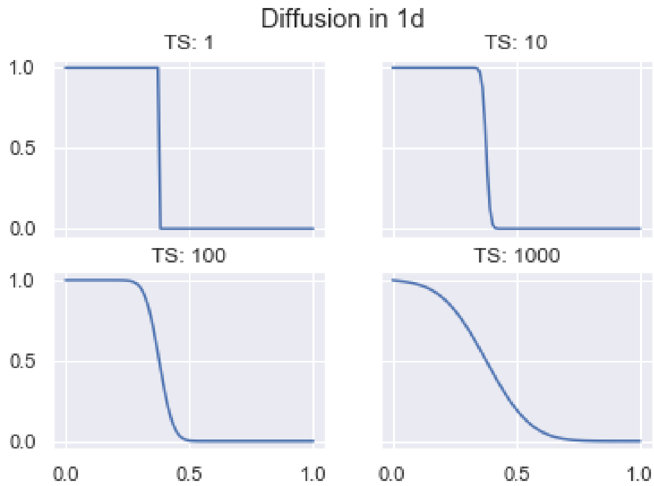

Figure 2: Diffusion initial condition from left quarter

1.3 Diffusion over time

```
plt.close('all')
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
from matplotlib import cm
from scipy.interpolate import griddata

N = 100
data = {nt: diffuse_1D(nx,dx,int(nt),D,dt, IC='start') for nt in np.linspace(1,100,N)}
y = np.arange(0,len(data[1]))

def make_3d_points_df(A, n):
    x = np.full(len(A), n)
    z = A
    return pd.DataFrame({'x':x,'y':y,'z':z})

df = pd.concat([make_3d_points_df(v,k) for k,v in data.items()])

x1 = np.linspace(df['x'].min(), df['x'].max(), len(df['x'].unique()))
y1 = np.linspace(df['y'].min(), df['y'].max(), len(df['y'].unique()))
X, Y = np.meshgrid(x1,y1)
Z = griddata((df['x'], df['y']), df['z'], (X,Y), method='cubic')

fig = plt.figure()
for idx, deg in enumerate(np.linspace(0,350,4)):
    ax = fig.add_subplot(2,2,idx+1, projection='3d')
    ax.plot_surface(X,Y,Z, cmap='plasma', linewidth=0)
    ax.view_init(30,int(deg))
fig.tight_layout()
```

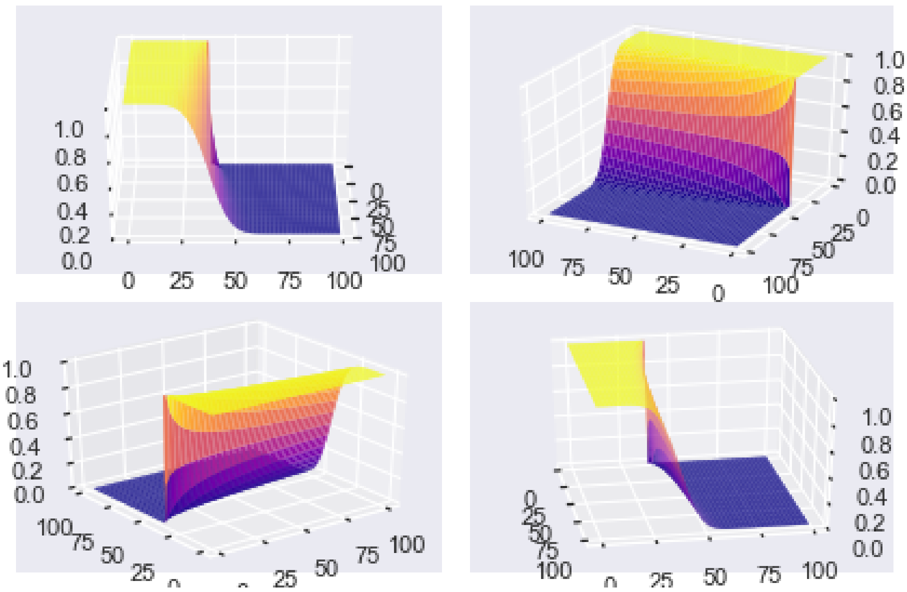

Figure 3: Diffusion over time

1.4 Odd Behaviour

- Of particular note is the negative numbers
  - Is there something wrong with the input variables and increasing the constant to $D = 1.5$?

```
nx = 100 # Number of x measurements
dx = 1 # Change in X
nt = 0.1 # Number of timesteps to make in calculation
D = 0.7 # Diffusion constant
dt = 0.01 # change in time

fig, axes = plt.subplots(2,2)
dts = {nt: diffuse_1D(nx,dx,nt,D,dt) for nt in np.logspace(0,3,4,base=10, dtype=int)}

for idx, d in enumerate(dts.keys()):
    axes[idx//2, idx%2].plot(np.linspace(0,1,nx), dts[d])
    axes[idx//2, idx%2].set_title('TS: {0}'.format(d))
plt.tight_layout()
```

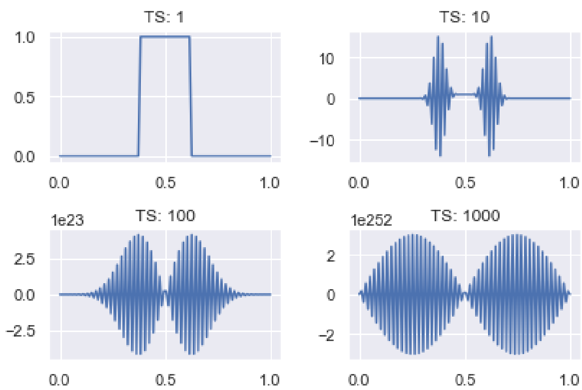

Figure 4: Diffusion with different parameters

1.4.1 Data

$dts[100]$ =:

Table 1: Values of $dts[100]$

$0.00000000 \times 10^{00}$	$1.28601728 \times 10^{09}$	$-2.79546811 \times 10^{09}$	$4.77701928 \times 10^{09}$
$1.89080824 \times 10^{12}$	$-2.60579910 \times 10^{12}$	$3.28001602 \times 10^{12}$	$-3.90268288 \times 10^{12}$
$4.46379536 \times 10^{12}$	$-4.95439275 \times 10^{12}$	$5.36685401 \times 10^{12}$	$-5.69519737 \times 10^{12}$
$5.93535752 \times 10^{12}$	$-6.08541216 \times 10^{12}$	$6.14573169 \times 10^{12}$	$-6.11903005 \times 10^{12}$
$6.01030308 \times 10^{12}$	$-5.82665017 \times 10^{12}$	$5.57698563 \times 10^{12}$	$-5.27165600 \times 10^{12}$
$4.92198811 \times 10^{12}$	$-4.53979827 \times 10^{12}$	$4.13689571 \times 10^{12}$	$-3.72461294 \times 10^{12}$
$3.31339182 \times 10^{12}$	$-2.91244828 \times 10^{12}$	$2.52953079 \times 10^{12}$	$-2.17077915 \times 10^{12}$
$1.84068222 \times 10^{12}$	$-1.54212556 \times 10^{12}$	$1.27651462 \times 10^{12}$	$-1.04395535 \times 10^{12}$
$8.43472362 \times 10^{11}$	$-6.73245295 \times 10^{11}$	$5.30845569 \times 10^{11}$	$-4.13459088 \times 10^{11}$
$3.18083897 \times 10^{11}$	$-2.41695821 \times 10^{11}$	$1.81378769 \times 10^{11}$	$-1.34419617 \times 10^{11}$
$9.83700764 \times 10^{10}$	$-7.10797163 \times 10^{10}$	$5.07053099 \times 10^{10}$	$-3.57019986 \times 10^{10}$
$2.48015740 \times 10^{10}$	$-1.69825919 \times 10^{10}$	$1.14362164 \times 10^{10}$	$-7.53076662 \times 10^{09}$
$4.77701928 \times 10^{09}$	$-2.79546811 \times 10^{09}$	$1.28601728 \times 10^{09}$	$0.00000000 \times 10^{00}$

1.4.2 3D

```
plt.close('all')
nx = 100 # Number of x measurements
dx = 1 # Change in X
nt = 0.1 # Number of timesteps to make in calculation
D = .7 # Diffusion constant
dt = 0.001 # change in time
N = 10
data = {nt: diffuse_1D(nx,dx,int(nt),D,dt) for nt in np.linspace(1,1000,N)}
y = np.arange(0,len(data[1]))
def make_3d_points_df(A, n):
    x = np.full(len(A), n)
    z = A
    return pd.DataFrame({'x':x,'y':y,'z':z})
df = pd.concat([make_3d_points_df(v,k) for k,v in data.items()])
x1 = np.linspace(df['x'].min(), df['x'].max(), len(df['x'].unique()))
y1 = np.linspace(df['y'].min(), df['y'].max(), len(df['y'].unique()))
X, Y = np.meshgrid(x1,y1)
Z = griddata((df['x'], df['y']), df['z'], (X,Y), method='cubic')
fig = plt.figure()
for idx, deg in enumerate(np.linspace(0,350,4)):
    ax = fig.add_subplot(2,2,idx+1, projection='3d')
    ax.plot_surface(X,Y,Z, cmap='plasma', linewidth=0)
    ax.view_init(30,int(deg))
fig.tight_layout()
```

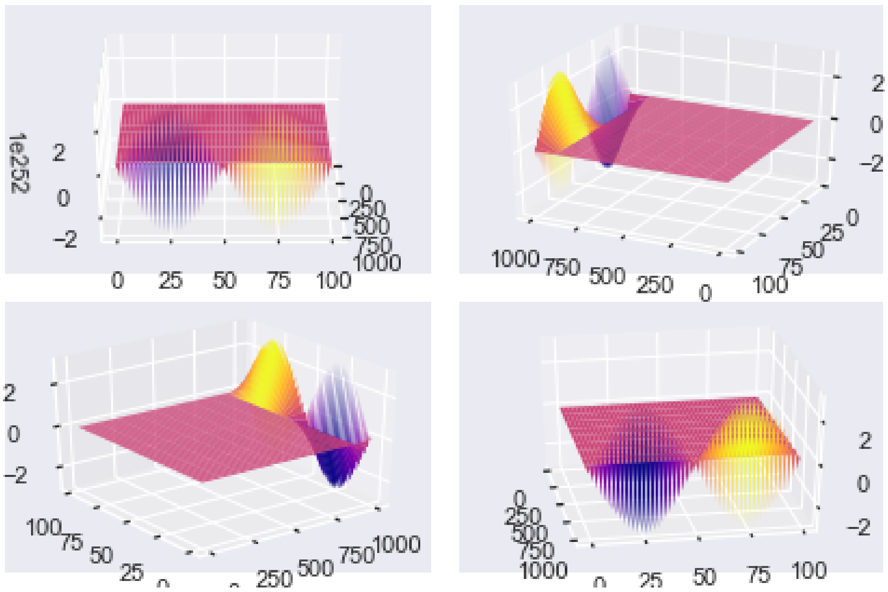

Figure 5: Diffusion with different parameters

1.5 **TODO** Fully Investigate all parameters and their function

2 Diffusion 2D

2.1 Initial Equation

Adapted from Rossant (2013); Hill (2018)

$$\frac{\partial u}{\partial t} = D(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \tag{4}$$

Which becomes:

$$u_{i,j}^{n+1} = u_{i,j}^{n} + D(\frac{(u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n})}{\Delta x^2} + \frac{(u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n})}{\Delta y^2}) \tag{5}$$

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()
plt.close('all')

def diffuse_2D(nx, dx, dy, nt, D, dt):
    dx2 = dx**2
    dy2 = dy**2
    u = np.zeros((nx, nx))
    mid = int(nx/2)

    # Assuming a square shape!
    # Initial Condition for diffusion
    u[int(mid-(mid/4)):int(mid+(mid/4)),
      int(mid-(mid/4)):int(mid+(mid/4))] = 1

    for n in range(nt):
        un = u.copy()  # Update previous values
        u[1:-1, 1:-1] = un[1:-1, 1:-1] + D * \
            (((un[2:, 1:-1] - 2 * un[1:-1, 1:-1] + un[:-2, 1:-1])/dx2) +
             ((un[1:-1, 2:] - 2 * un[1:-1, 1:-1] + un[1:-1, :-2])/dy2))
    return un

nx = 100  # Number of x measurements
dx, dy = 1, 1   # Change in X & Y
nt = 1  # Number of timesteps to make in calculation
D = 0.01  # Diffusion constant
dt = 0.01  # change in time
fig, axes = plt.subplots(2, 3, sharex=True, sharey=True)
nts = np.around([nt for nt in np.linspace(1, 10000, 6)])
dts = {nt: diffuse_2D(nx, dx, dy, int(nt), D, dt) for nt in nts}

for idx, d in enumerate(nts):
    axes[idx//3, idx % 3].imshow(dts[d], cmap='gray', vmin=0, vmax=1)
    axes[idx//3, idx % 3].set_axis_off()
    axes[idx//3, idx % 3].set_title('TS: {0}'.format(d))

plt.tight_layout()
```

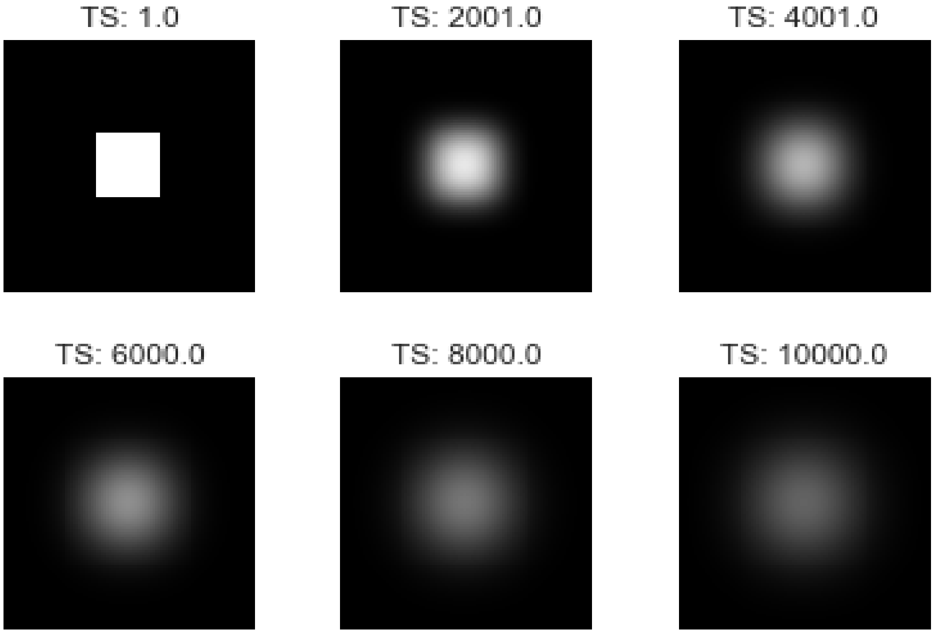

Figure 6: Diffusion in 2D

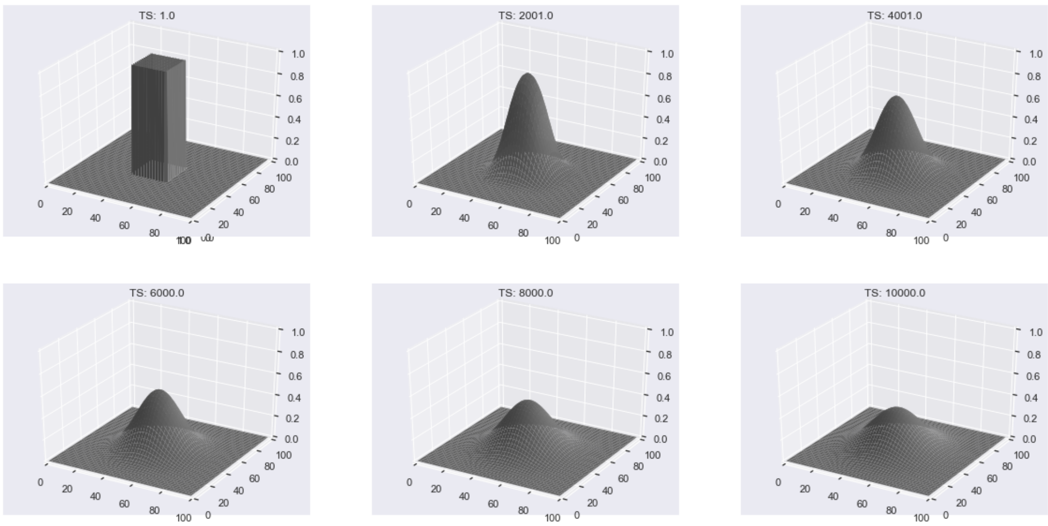

Figure 7: 3D projection of 2D diffusion

2.2 Odd results (repeated in 2D)

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()
plt.close('all')

def diffuse_2D(nx, dx, dy, nt, D, dt):
    dx2 = dx**2
    dy2 = dy**2
    u = np.zeros((nx, nx))
    mid = int(nx/2)

    # Assuming a square shape!
    # Initial Condition for diffusion
    u[int(mid-(mid/4)):int(mid+(mid/4)),
      int(mid-(mid/4)):int(mid+(mid/4))] = 1

    for n in range(nt):
        un = u.copy()  # Update previous values
        u[1:-1, 1:-1] = un[1:-1, 1:-1] + D * \
            (((un[2:, 1:-1] - 2 * un[1:-1, 1:-1] + un[:-2, 1:-1])/dx2) +
             ((un[1:-1, 2:] - 2 * un[1:-1, 1:-1] + un[1:-1, :-2])/dy2))
    return un

nx = 100  # Number of x measurements
dx, dy = 1,1   # Change in X & Y
nt = 0.1  # Number of timesteps to make in calculation
D = 0.7  # Diffusion constant
dt = 0.001  # change in time
fig, axes = plt.subplots(2, 3, sharex=True, sharey=True)
nts = np.around([nt for nt in np.linspace(1, 80, 6)])
dts = {nt: diffuse_2D(nx, dx, dy, int(nt), D, dt) for nt in nts}

for idx, d in enumerate(nts):
    axes[idx//3, idx % 3].imshow(dts[d], cmap='gray', vmin=0, vmax=1)
    axes[idx//3, idx % 3].set_axis_off()
    axes[idx//3, idx % 3].set_title('TS: {0}'.format(d))

plt.tight_layout()
```

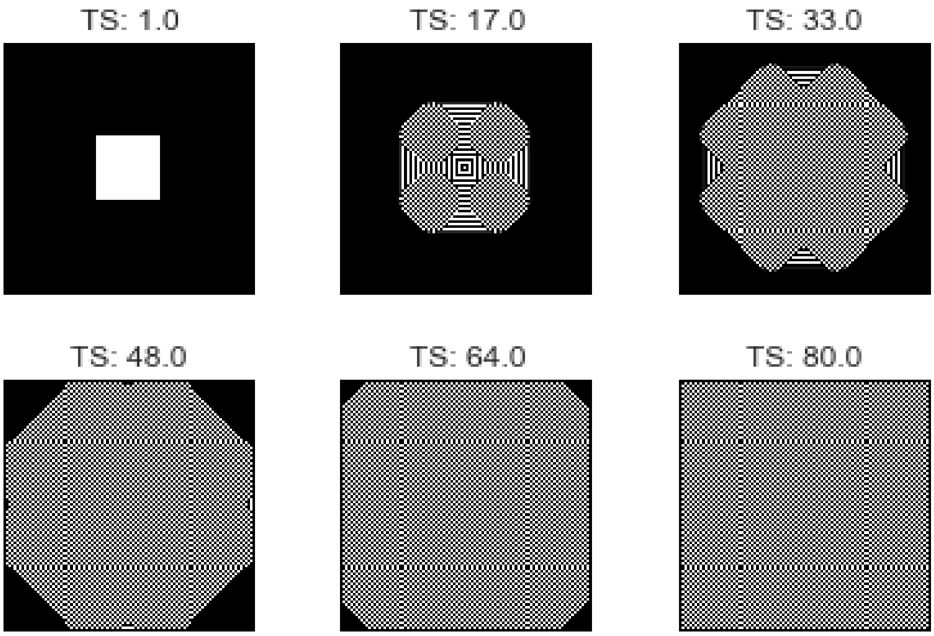

Figure 8: Diffusion in 2D

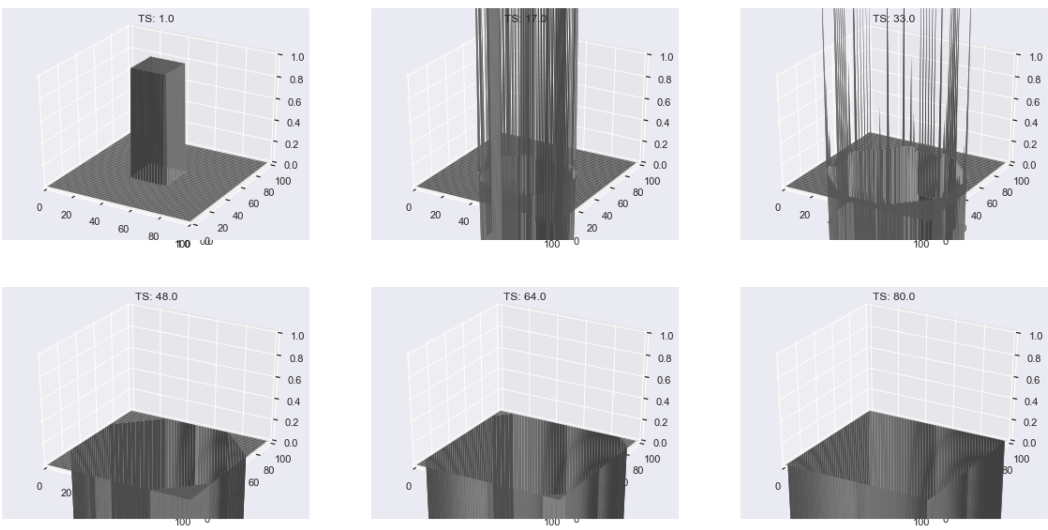

Figure 9: 3D projection of 2D diffusion

References

Christian Hill. The two-dimensional diffusion equation. https://scipython.com/book/chapter-7-matplotlib/examples/the-two-dimensional-diffusion-equation/, 2018. 00000.

Ronald S. Kaufmann. Fick's lawFick's law. In *Geochemistry*, pages 245–246. Springer Netherlands, Dordrecht, 1998. ISBN 978-1-4020-4496-0. doi: 10.1007/1-4020-4496-8_123. 00000.

Cyrille Rossant. *Learning IPython for Interactive Computing and Data Visualization*. Packt Publishing Ltd, 2013. 00021.