

PhD Diary Week Beginning 12th November

Nathan Hughes

November 14, 2018

Contents

1	TODO 1D Diffusion (Update to latest Equation)	3
1.1	Diffusion from centre	4
1.2	Diffusion from one side	4
1.3	Diffusion over time	5
1.4	Odd Behaviour	6
1.4.1	Data	6
1.4.2	3D	7
1.5	TODO Fully Investigate all parameters and their function	7
2	Diffusion 2D	8
2.1	Initial Equation	8
2.2	Odd results (repeated in 2D)	10

1 TODO 1D Diffusion (Update to latest Equation)

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

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2} \quad (1)$$

Where u can be numerically solved with:

$$u = u(x_i, t_n) \quad (2)$$

Implementing the solved equation for 1D diffusion:

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

```

1  %matplotlib inline
2  import seaborn as sns
3  import matplotlib.pyplot as plt
4  import numpy as np
5  sns.set()
6  plt.close('all')
7  def diffuse_1D(nx,dx,nt,D,dt, IC=None, slow=False):
8      u = np.zeros(nx)
9      mid = int(nx/2)
10     dx2 = dx**2
11     if IC is None:
12         u[mid-int(mid/4):mid+int(mid/4)] = 1
13     elif IC is 'start':
14         u[:mid-int(mid/4)] = 1
15     for n in range(nt):
16         un = u.copy() # Update previous values
17         if slow:
18             for i in range(1, nx-1):
19                 u[i] = un[i] + D * dt / dx**2 * (un[i+1] - 2 * un[i] + un[i-1])
20         else:
21             #u[1:-1] = un[1:-1] + D * dt / dx**2 * (un[0:-2] - 2 * un[1:-1] + un[2:])
22             u[1:-1] = un[1:-1] + D * (un[0:-2] - 2 * un[1:-1] + un[2:])/dx2
23     return un
24
25 nx = 100 # Number of x measurements
26 dx = 5 #2 / (nx-1) # Change in X
27 nt = 1 # Number of timesteps to make in calculation
28 D = 0.3 # Diffusion constant
29 dt = 0.001 # change in time
30
31 fig, axes = plt.subplots(2,2, sharex=True, sharey=True)
32 dts = {nt: diffuse_1D(nx,dx,nt,D,dt) for nt in np.logspace(0,3,4,base=10, dtype=int)}
33
34 for idx, d in enumerate(dts.keys()):
35     axes[idx//2, idx%2].plot(np.linspace(0,1,nx), dts[d])
36     axes[idx//2, idx%2].set_title('TS: {0}'.format(d))
37
38 plt.suptitle('Diffusion in 1d')

```

1.1 Diffusion from centre

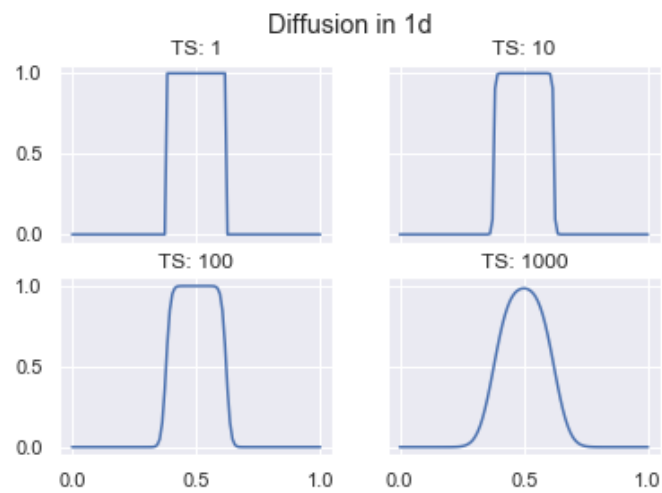


Figure 1: Diffusion initial condition from centre

1.2 Diffusion from one side

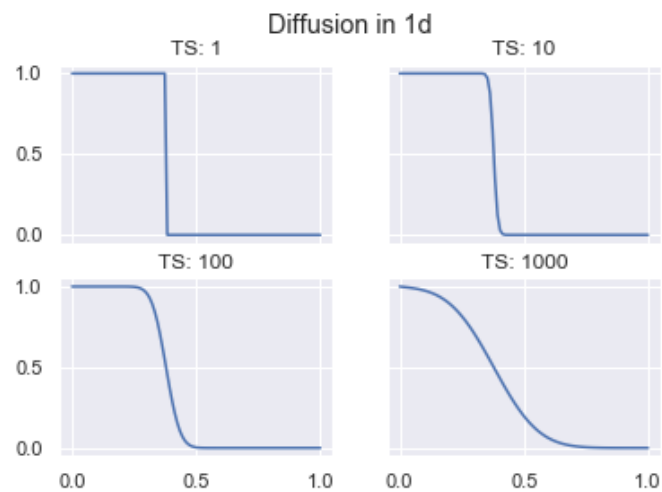


Figure 2: Diffusion initial condition from left quarter

1.3 Diffusion over time

```

1 plt.close('all')
2 from mpl_toolkits.mplot3d import Axes3D
3 import pandas as pd
4 from matplotlib import cm
5 from scipy.interpolate import griddata
6
7 N = 200
8 data = {nt: diffuse_1D(nx,dx,int(nt),D*2,dt, IC='start') for nt in np.linspace(1,1000,N)}
9 y = np.arange(0,len(data[1]))
10
11 def make_3d_points_df(A, n):
12     x = np.full(len(A), n)
13     z = A
14     return pd.DataFrame({'x':x,'y':y,'z':z})
15
16 df = pd.concat([make_3d_points_df(v,k) for k,v in data.items()])
17
18 x1 = np.linspace(df['x'].min(), df['x'].max(), len(df['x'].unique()))
19 y1 = np.linspace(df['y'].min(), df['y'].max(), len(df['y'].unique()))
20 X, Y = np.meshgrid(x1,y1)
21 Z = griddata((df['x'], df['y']), df['z'], (X,Y), method='cubic')
22
23 fig = plt.figure()
24 for idx, deg in enumerate(np.linspace(0,350,4)):
25     ax = fig.add_subplot(2,2,idx+1, projection='3d')
26     ax.plot_surface(X,Y,Z, cmap='plasma', linewidth=0)
27     ax.view_init(30,int(deg))
28 fig.tight_layout()

```

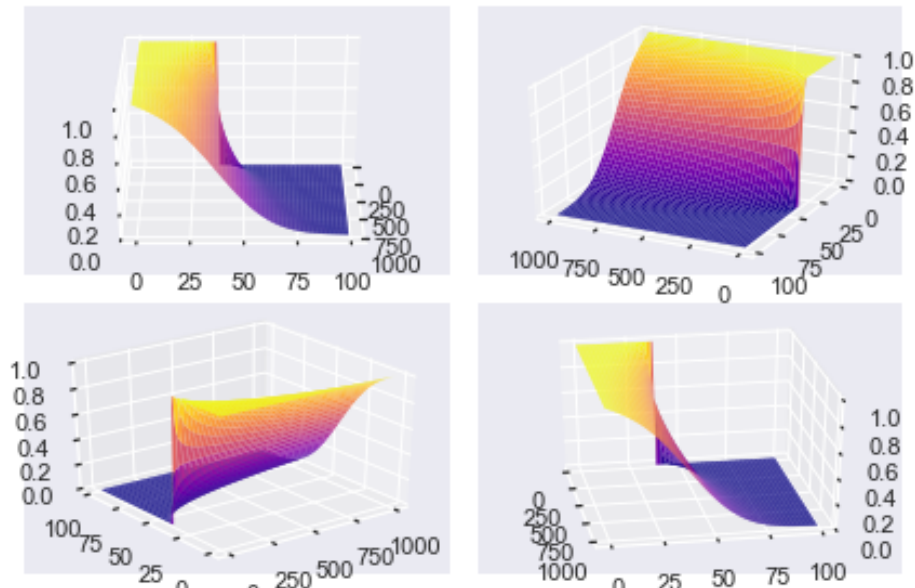


Figure 3: Diffusion over time

1.4 Odd Behaviour

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

```

1  nx = 100 # Number of x measurements
2  dx = 0.05 #2 / (nx-1) # Change in X
3  nt = 0.1 # Number of timesteps to make in calculation
4  D = 0.7 # Diffusion constant
5  dt = 0.001 # change in time
6
7  fig, axes = plt.subplots(2,2)
8  dts = {nt: diffuse_1D(nx,dx,nt,D,dt) for nt in np.logspace(0,3,4,base=10, dtype=int)}
9
10 for idx, d in enumerate(dts.keys()):
11     axes[idx//2, idx%2].plot(np.linspace(0,1,nx), dts[d])
12     axes[idx//2, idx%2].set_title('TS: {0}'.format(d))
13 plt.tight_layout()

```

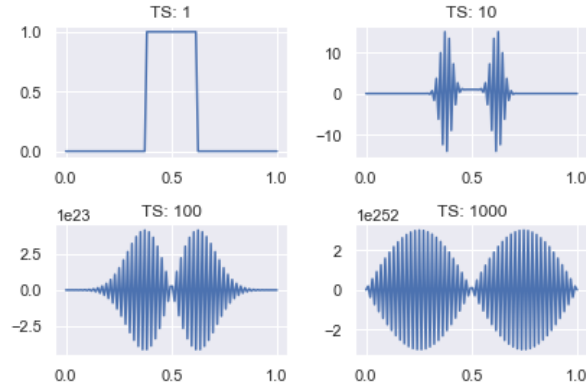


Figure 4: Diffusion with different parameters

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

```

1 plt.close('all')
2 nx = 100 # Number of x measurements
3 dx = 0.05 # 2 / (nx-1) # Change in X
4 nt = 0.1 # Number of timesteps to make in calculation
5 D = .7 # Diffusion constant
6 dt = 0.001 # change in time
7 N = 10
8 data = {nt: diffuse_1D(nx,dx,int(nt),D,dt) for nt in np.linspace(1,1000,N)}
9 y = np.arange(0,len(data[1]))
10 def make_3d_points_df(A, n):
11     x = np.full(len(A), n)
12     z = A
13     return pd.DataFrame({'x':x,'y':y,'z':z})
14 df = pd.concat([make_3d_points_df(v,k) for k,v in data.items()])
15 x1 = np.linspace(df['x'].min(), df['x'].max(), len(df['x'].unique()))
16 y1 = np.linspace(df['y'].min(), df['y'].max(), len(df['y'].unique()))
17 X, Y = np.meshgrid(x1,y1)
18 Z = griddata((df['x'], df['y']), df['z'], (X,Y), method='cubic')
19 fig = plt.figure()
20 for idx, deg in enumerate(np.linspace(0,350,4)):
21     ax = fig.add_subplot(2,2,idx+1, projection='3d')
22     ax.plot_surface(X,Y,Z, cmap='plasma', linewidth=0)
23     ax.view_init(30,int(deg))
24 fig.tight_layout()

```

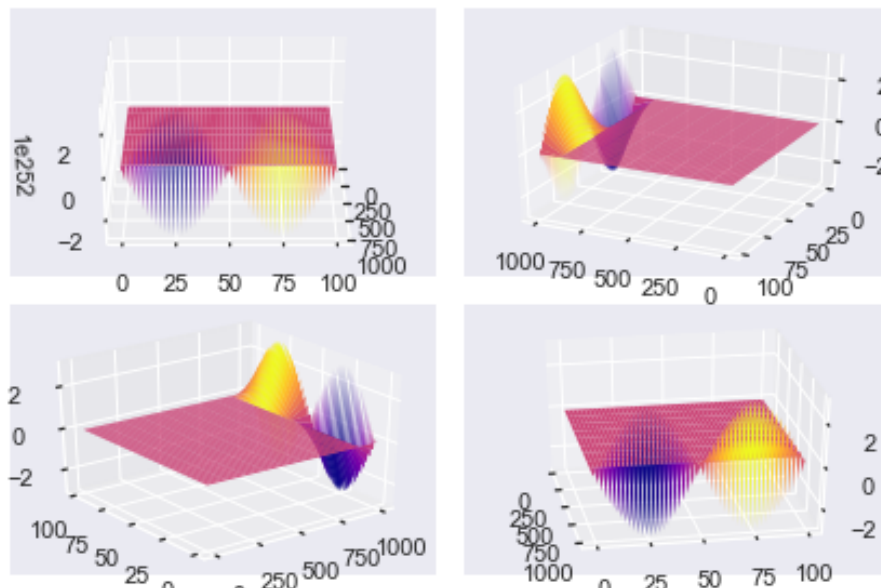


Figure 5: Diffusion with different parameters

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

2 Diffusion 2D

2.1 Initial Equation

Adapted from Rossant (2013); Hill (2018)

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

Which becomes:

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

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 %matplotlib inline
5 sns.set()
6 plt.close('all')
7
8 def diffuse_2D(nx, dx, dy, nt, D, dt):
9     dx2 = dx**2
10    dy2 = dy**2
11    u = np.zeros((nx, nx))
12    mid = int(nx/2)
13
14    # Assuming a square shape!
15    # Initial Condition for diffusion
16    u[int(mid-(mid/4)):int(mid+(mid/4)),
17      int(mid-(mid/4)):int(mid+(mid/4))] = 1
18
19    for n in range(nt):
20        un = u.copy() # Update previous values
21        u[1:-1, 1:-1] = un[1:-1, 1:-1] + D * \
22            (((un[2:, 1:-1] - 2 * un[1:-1, 1:-1] + un[:-2, 1:-1])/dx2) +
23             ((un[1:-1, 2:] - 2 * un[1:-1, 1:-1] + un[1:-1, :-2])/dy2))
24    return un
25
26 nx = 100 # Number of x measurements
27 dx, dy = 5, 5 # Change in X & Y
28 nt = 1 # Number of timesteps to make in calculation
29 D = 0.3 # Diffusion constant
30 dt = 0.001 # change in time
31 fig, axes = plt.subplots(2, 3, sharex=True, sharey=True)
32 nts = np.around([nt for nt in np.linspace(1, 10000, 6)])
33 dts = {nt: diffuse_2D(nx, dx, dy, int(nt), D, dt) for nt in nts}
34
35 for idx, d in enumerate(nts):
36     axes[idx//3, idx % 3].imshow(dts[d], cmap='gray', vmin=0, vmax=1)
37     axes[idx//3, idx % 3].set_axis_off()
38     axes[idx//3, idx % 3].set_title('TS: {0}'.format(d))
39
40 plt.tight_layout()

```

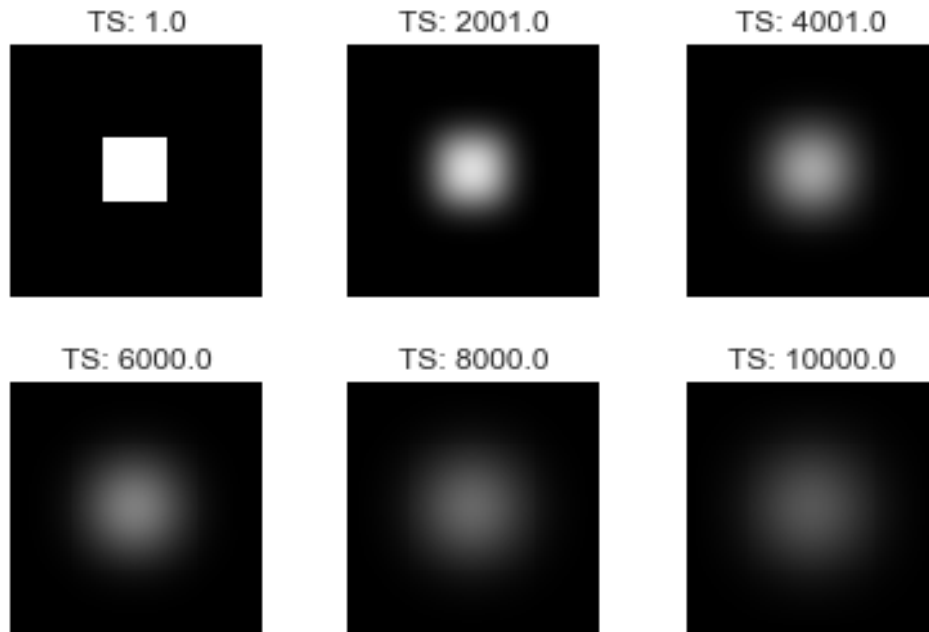


Figure 6: Diffusion in 2D

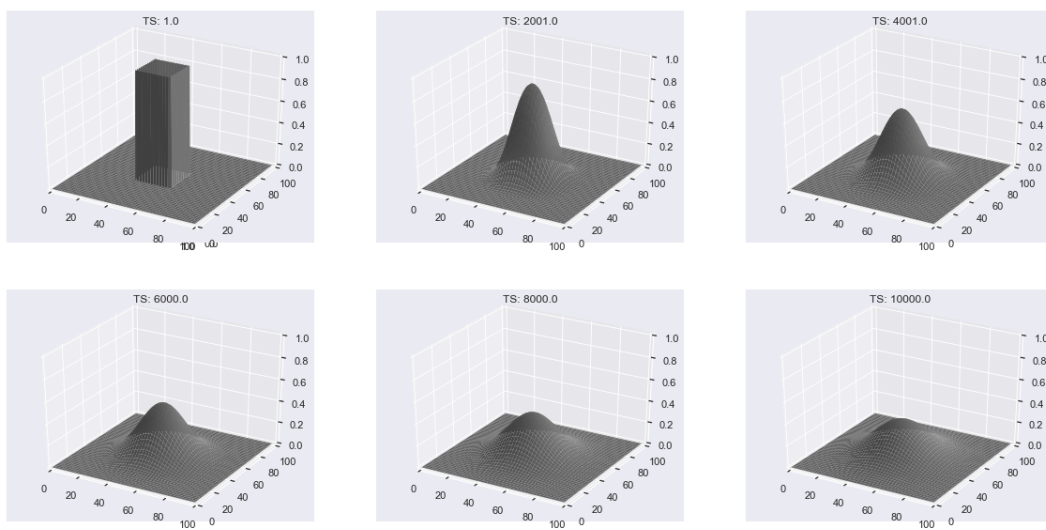


Figure 7: 3D projection of 2D diffusion

2.2 Odd results (repeated in 2D)

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 %matplotlib inline
5 sns.set()
6 plt.close('all')
7
8 def diffuse_2D(nx, dx, dy, nt, D, dt):
9     dx2 = dx**2
10    dy2 = dy**2
11    u = np.zeros((nx, nx))
12    mid = int(nx/2)
13
14    # Assuming a square shape!
15    # Initial Condition for diffusion
16    u[int(mid-(mid/4)):int(mid+(mid/4)),
17      int(mid-(mid/4)):int(mid+(mid/4))] = 1
18
19    for n in range(nt):
20        un = u.copy() # Update previous values
21        u[1:-1, 1:-1] = un[1:-1, 1:-1] + D * \
22            (((un[2:, 1:-1] - 2 * un[1:-1, 1:-1] + un[:-2, 1:-1])/dx2) +
23             ((un[1:-1, 2:] - 2 * un[1:-1, 1:-1] + un[1:-1, :-2])/dy2))
24    return un
25
26 nx = 100 # Number of x measurements
27 dx, dy = .05, .05 # Change in X & Y
28 nt = 0.1 # Number of timesteps to make in calculation
29 D = 0.7 # Diffusion constant
30 dt = 0.001 # change in time
31 fig, axes = plt.subplots(2, 3, sharex=True, sharey=True)
32 nts = np.around([nt for nt in np.linspace(1, 80, 6)])
33 dts = {nt: diffuse_2D(nx, dx, dy, int(nt), D, dt) for nt in nts}
34
35 for idx, d in enumerate(nts):
36     axes[idx//3, idx % 3].imshow(dts[d], cmap='gray', vmin=0, vmax=1)
37     axes[idx//3, idx % 3].set_axis_off()
38     axes[idx//3, idx % 3].set_title('TS: {0}'.format(d))
39
40 plt.tight_layout()

```

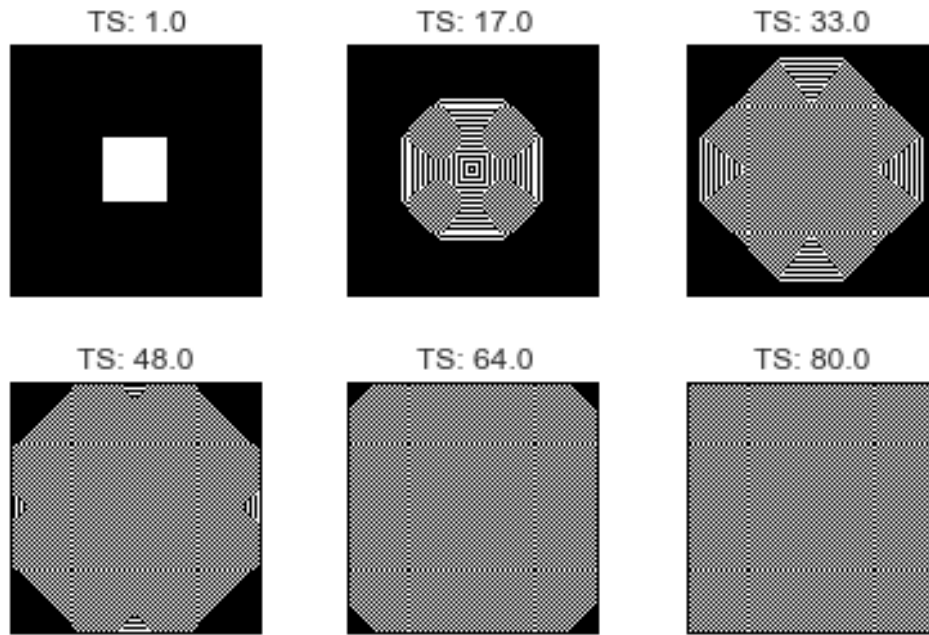


Figure 8: Diffusion in 2D

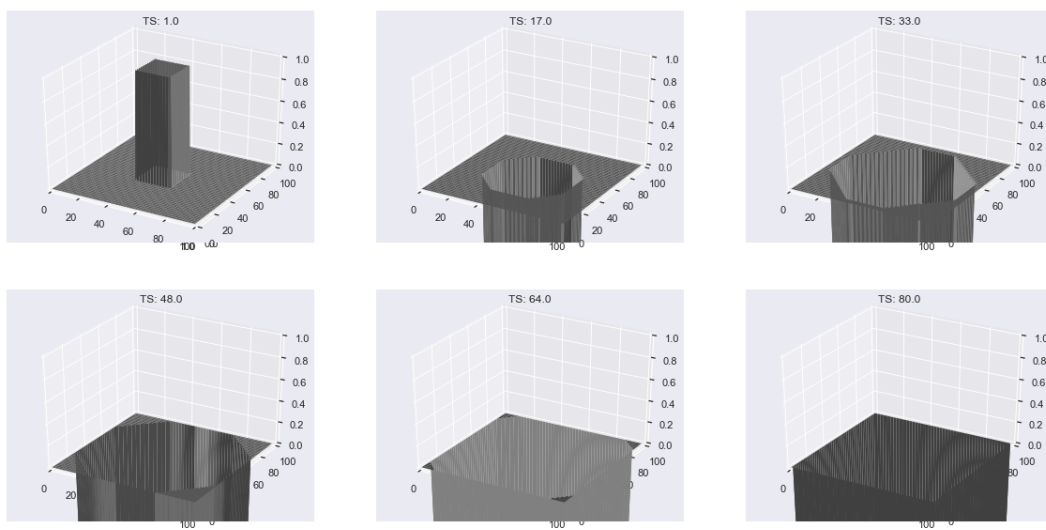


Figure 9: 3D projection of 2D diffusion

References

- Christian Hill. The two-dimensional diffusion equation. <https://scipython.com/book/chapter-7-matplotlib/examples/the-two-dimensional-diffusion-equation/>, 2018.
- 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.
- Cyrille Rossant. *Learning IPython for Interactive Computing and Data Visualization*. Packt Publishing Ltd, 2013.