

Partial Differential Equations for Science and Engineering

Final Report

November 27, 2017

Thumwanit Napat

16B00133

Contents

[Problem 1 Diffusion Equation 2](#_Toc498265223)

[A. Effect of Initial Condition (By Neumann) 3](#_Toc498265224)

[B. Observation of Conditions 5](#_Toc498265225)

[- Desired Initial 8](#_Toc498265226)

[- Cyclic 9](#_Toc498265227)

[- Dirichlet 10](#_Toc498265228)

[- Neumann (G=0) 11](#_Toc498265229)

[- Neumann (G=20) 13](#_Toc498265230)

[C. Effect of d’s value 14](#_Toc498265231)

Table of Figures

[Figure 1 Initial condition set to 100 and only to regions aa 4](#_Toc497723333)

# Problem 1 Diffusion Equation

Construct a diffusion model for a 2-D heat plate with dimensions 100 m. by 100 m given the equation,



Discuss the following (add figures if necessary):

1. Influence of the initial condition. Test various initial conditions or distributions.
2. Investigate various boundary conditions:
   1. Dirichlet Boundary condition
   2. Neumann Boundary condition
3. Investigate what is the influence of by testing various values for *d*. What are the threshold values for *d*?
4. Show three time steps (start, middle, and almost steady-state). Steady-state means the variations with time are almost negligible.

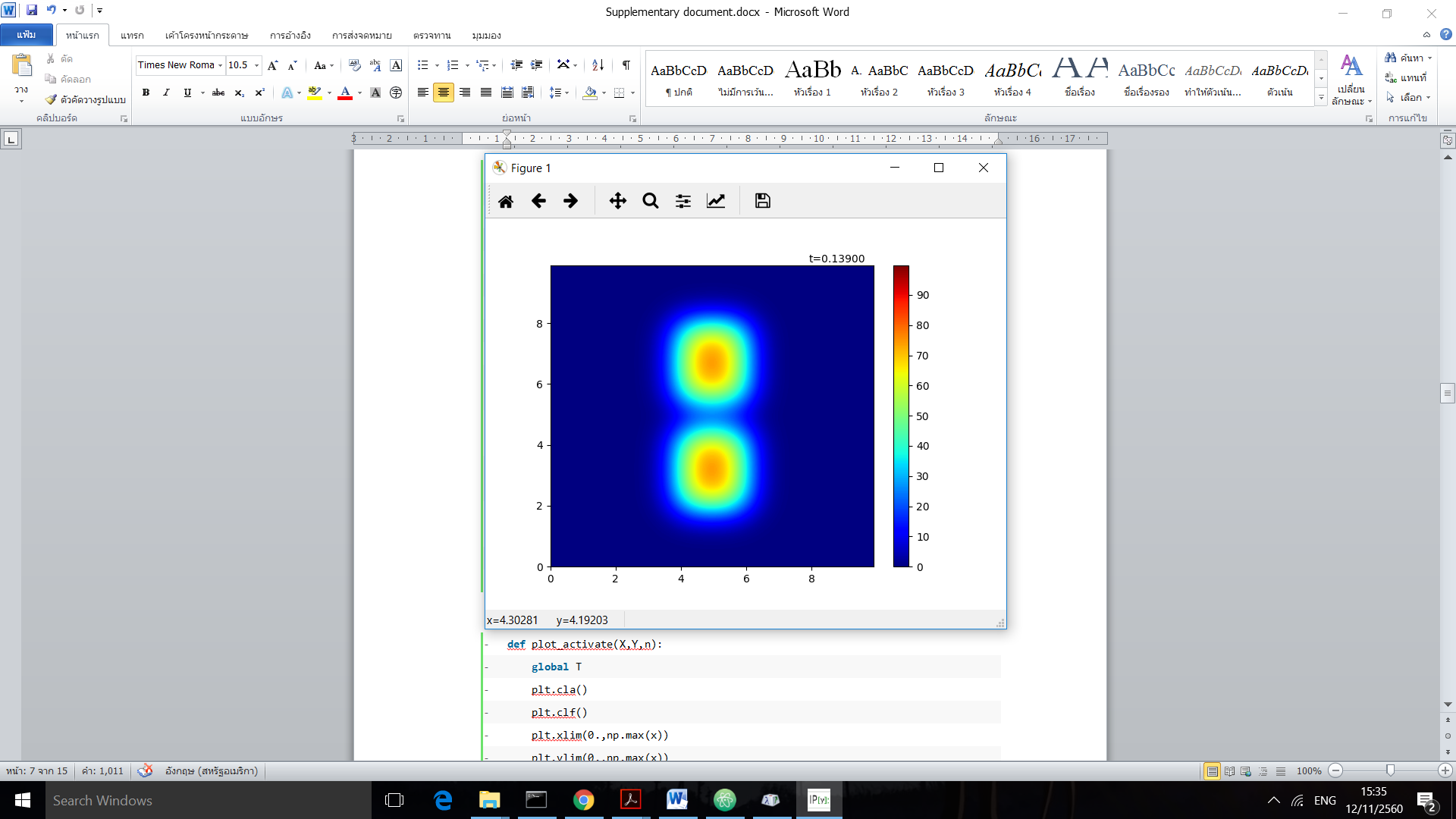
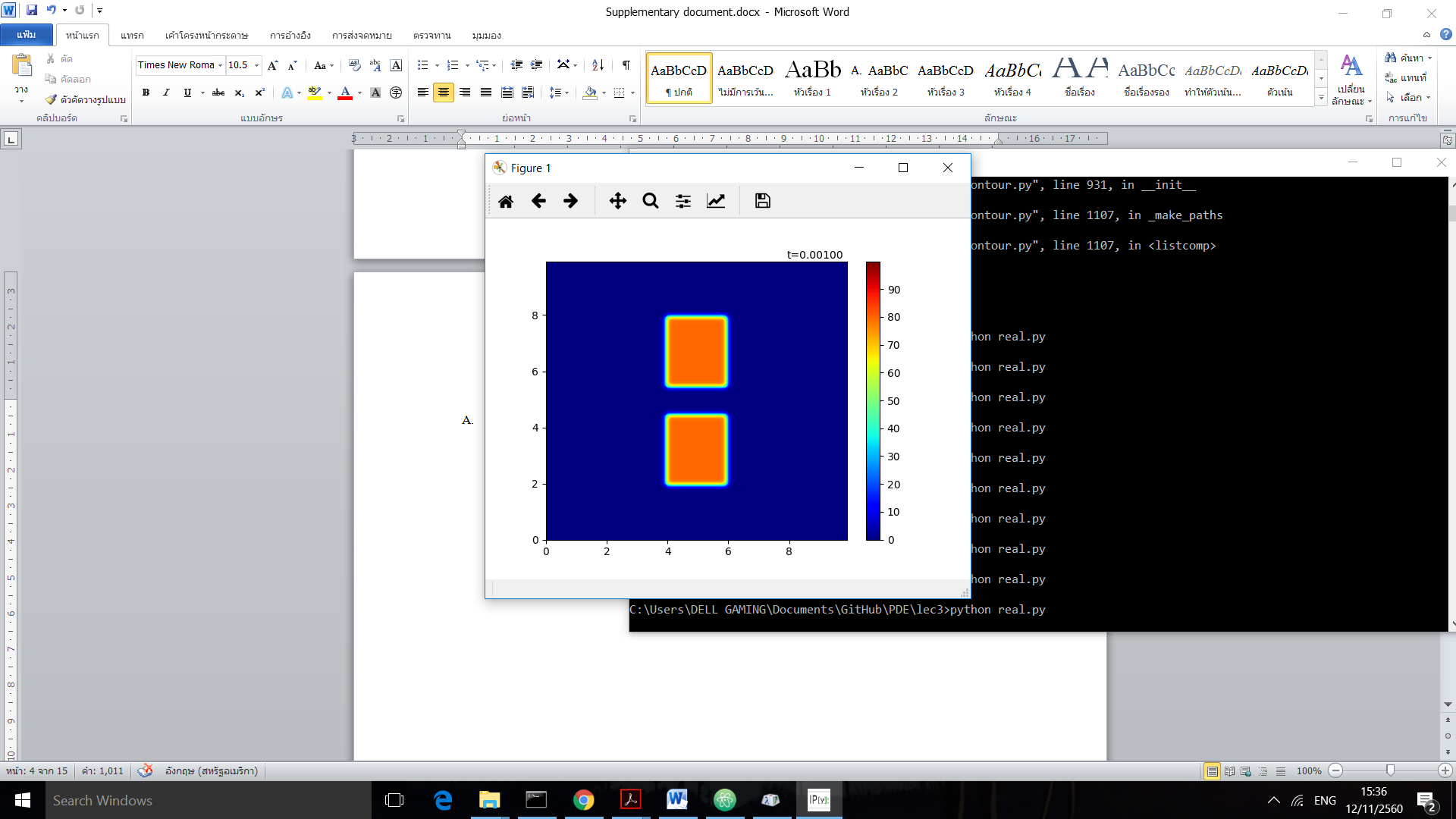
## Effect of Initial Condition (By Neumann) [Q.1]

Initial Condition

1. dx = 0.1
2. dy = dx
3. alpha = 1.
4. grid\_x = 100
5. grid\_y = 100
6. nt = 100
7. d = 0.1
8. dt = d \* (dx\*\*2)/alpha
10. **def** init():
11. T = np.zeros((grid\_x,grid\_y))
12. T[70:grid\_x, 70:grid\_y] = 80
13. **return** T
14. T = init()

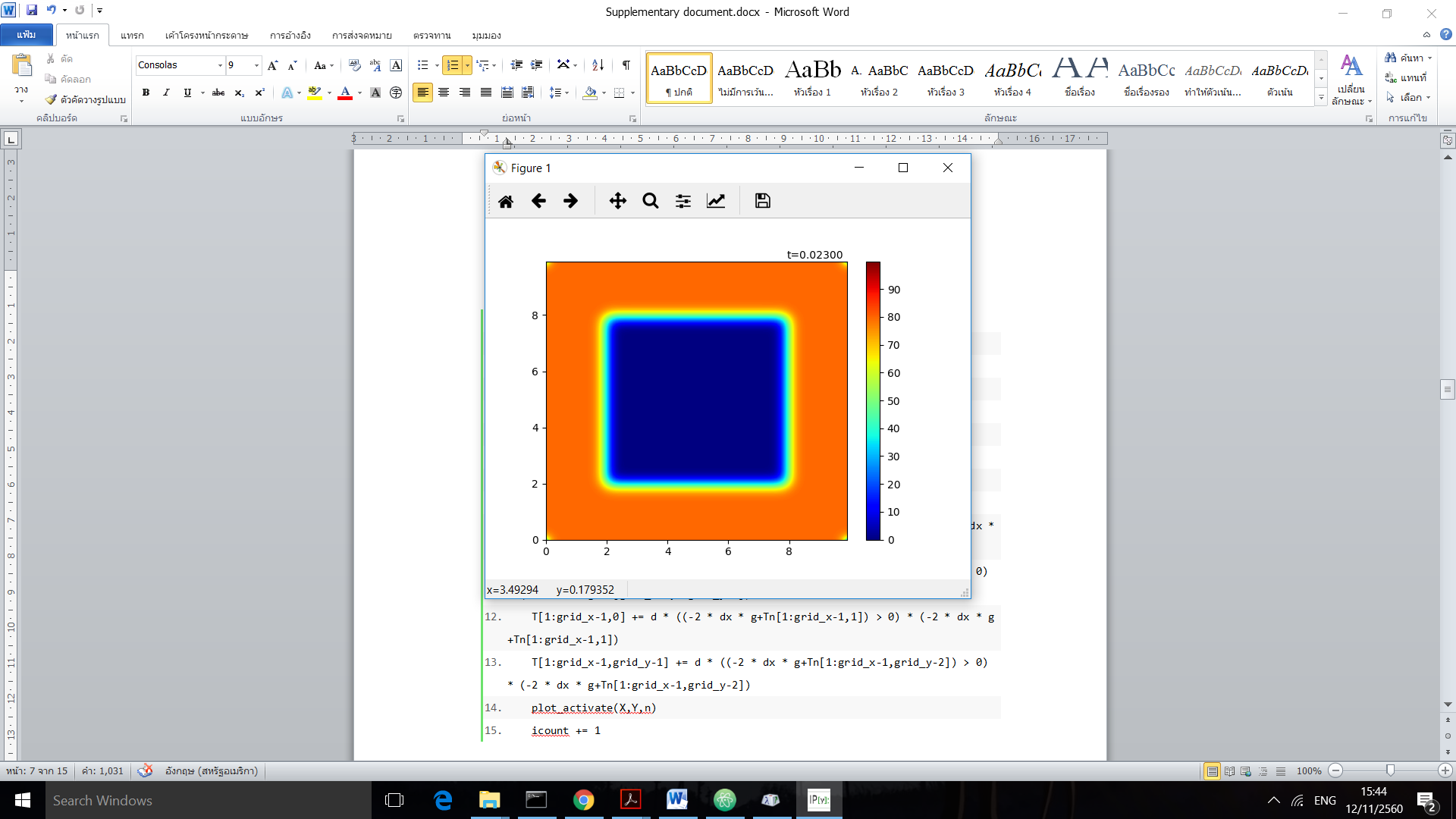
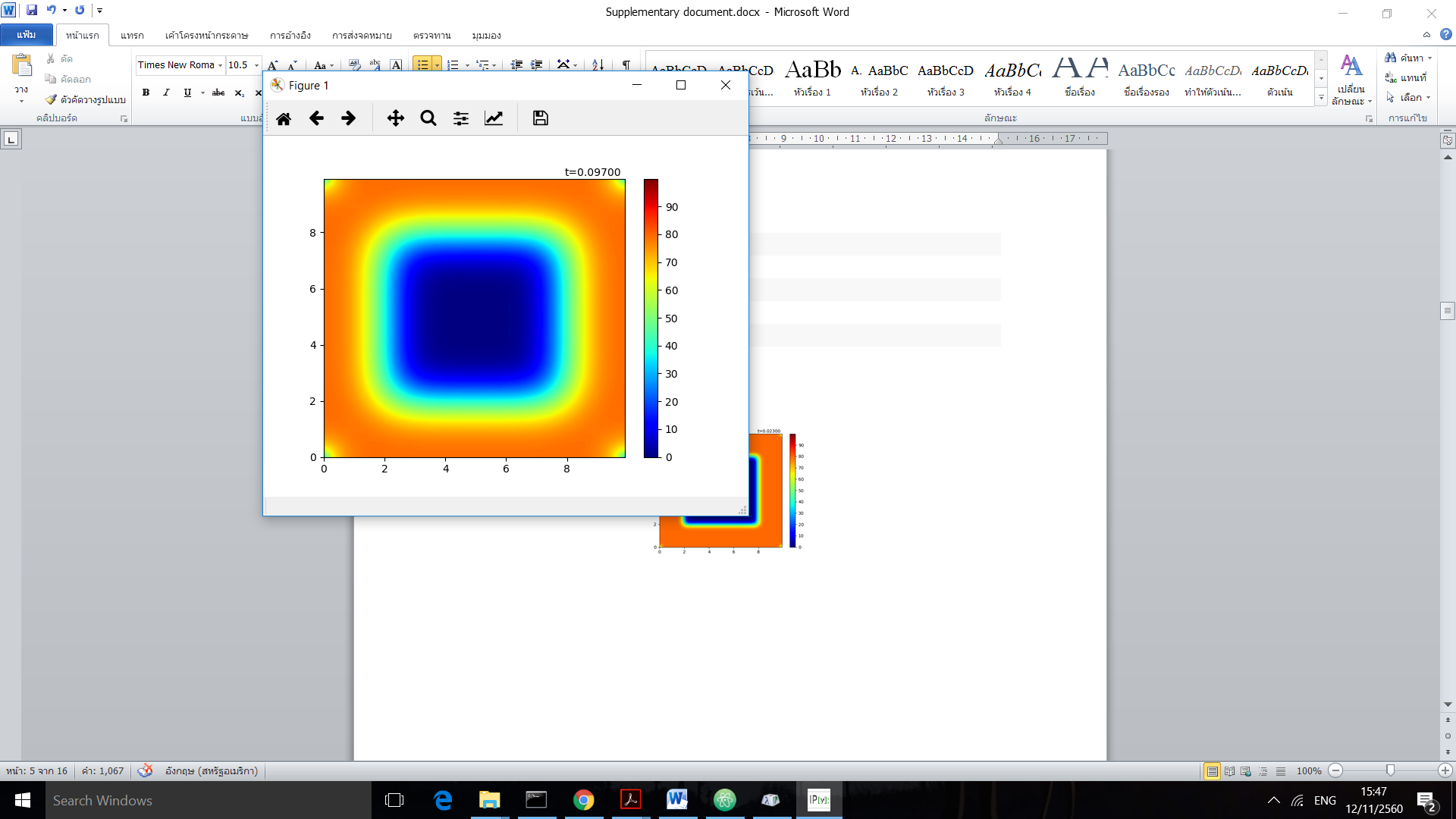
* couple blobs

1. **def** init():
2. T = np.zeros((grid\_x,grid\_y))
3. T[20:45, 40:60] = 80.
4. T[55:80, 40:60] = 80.
5. **return** T



* Heat wall

1. **def** init():
2. T = np.zeros((grid\_x,grid\_y))
3. T[0:20, :] = 80.
4. T[grid\_x-20:grid\_x, :] = 80.
5. T[:,0:20] = 80.
6. T[:,grid\_y-20:grid\_y] = 80.
7. **return** T

## Observation of Conditions [Q.2+4]

Cyclic

1. **def** cyc(n, plot=True):
2. **global** T, icount
3. Tn = T.copy()
4. T = Tn+d\*(np.roll(Tn,1,axis=0)+np.roll(Tn,-1,axis=0)+np.roll(Tn,1,axis=1)+np.roll(Tn,-1,axis=1)-4\*Tn)
5. **if** plot:
6. plot\_activate(X,Y,n)
7. icount += 1

Dirichlet

1. **def** dir(n,plot=True):
2. **global** T, icount
3. T[0,:], T[:,0], T[grid\_x-1,:], T[:,grid\_y-1] = 0,0,0,0
4. Tn = T.copy()
5. T = Tn+d\*(np.roll(Tn,1,axis=0)+np.roll(Tn,-1,axis=0)+np.roll(Tn,1,axis=1)+np.roll(Tn,-1,axis=1)-4\*Tn)
6. **if** plot:
7. plot\_activate(X,Y,n)
8. icount += 1

Neumann

1. **def** neu(n, g=0,plot=True):
2. **global** T, icount
3. Tn = T.copy()
4. left = np.roll(Tn,1,axis=0)
5. right = np.roll(Tn,-1,axis=0)
6. up = np.roll(Tn,-1,axis=1)
7. down = np.roll(Tn,1,axis=1)
8. left[0,:],right[grid\_x-1,:],up[:,grid\_y-1],down[:,0] = 0,0,0,0
9. T = Tn+d\*(up+down+right+left-4\*Tn)
10. T[0,1:grid\_y-1] += d \* ((-2 \* dx \* g + Tn[1,1:grid\_y-1]) > 0) \* (-2 \* dx \* g + Tn[1,1:grid\_y-1])
11. T[grid\_x-1,1:grid\_y-1] += d \* ((-2 \* dx \* g+Tn[grid\_x-2,1:grid\_y-1]) > 0) \* (-2 \* dx \* g+Tn[grid\_x-2,1:grid\_y-1])
12. T[1:grid\_x-1,0] += d \* ((-2 \* dx \* g+Tn[1:grid\_x-1,1]) > 0) \* (-2 \* dx \* g+Tn[1:grid\_x-1,1])
13. T[1:grid\_x-1,grid\_y-1] += d \* ((-2 \* dx \* g+Tn[1:grid\_x-1,grid\_y-2]) > 0) \* (-2 \* dx \* g+Tn[1:grid\_x-1,grid\_y-2])
14. **if** plot:
15. plot\_activate(X,Y,n)
16. icount += 1

Plot function

1. **def** plot\_activate(X,Y,n):
2. **global** T
3. plt.cla()
4. plt.clf()
5. plt.xlim(0.,np.max(x))
6. plt.ylim(0.,np.max(x))
7. cl = plt.contourf(X,Y,T,levels,cmap=cmap)
8. plt.colorbar(cl)
9. plt.text(np.max(x)\*0.8,np.max(y)+dy,"t=%01.5f"%(dt\*n))

### Desired Initial

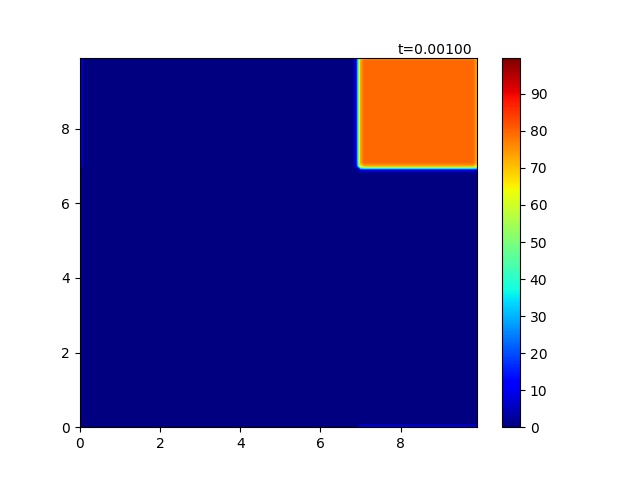


Figure 1 initial condition for observation

Here is the visualization for the initial values and init() function above. We initiate the blob of heat at the corner to observe each method.

### Cyclic

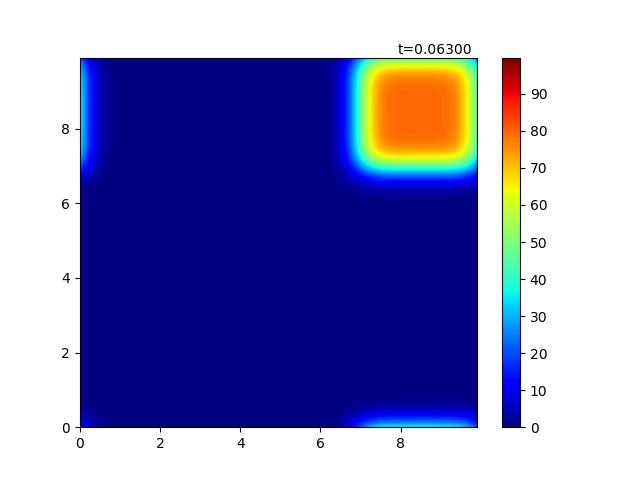


Figure 2 middle state of cyclic condition

Unfortunately, our np.roll() function rolls over the matrix and the temperature somehow leaks to the other side. However, the dispersion can be observed cleary.

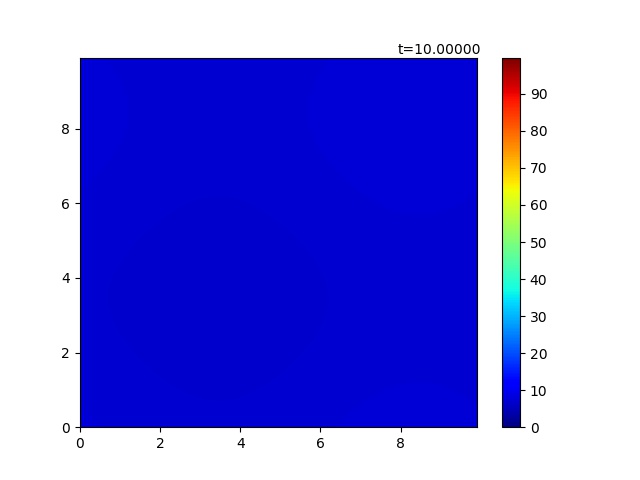


Figure 3 steady state of cyclic condition

The above figure is the steady state where the heat uniformly distributed through the plane. We will compare it later with Dirichlet and Neumann.

### Dirichlet

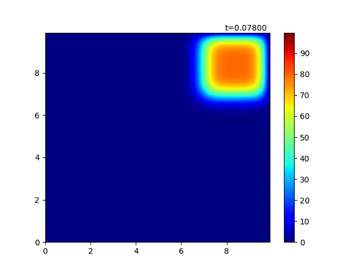


Figure 4 middle state of Dirichlet condition

Dirichlet dispersed in the same way as Cyclic did but not leak to the other side. Due to the wall condition, the wall itself acted like “black hole” that heat suddenly lost over there.

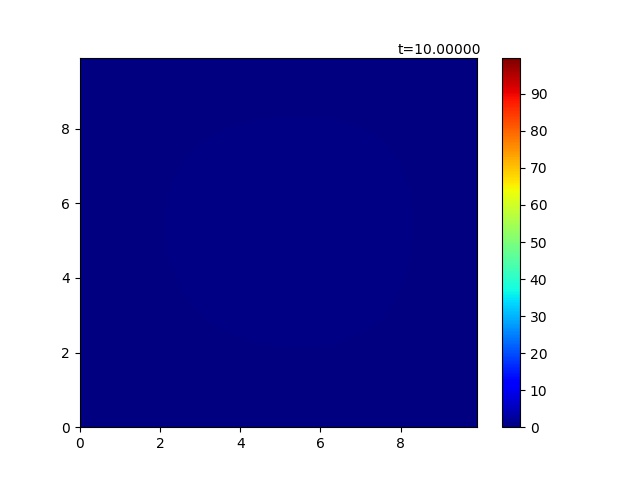
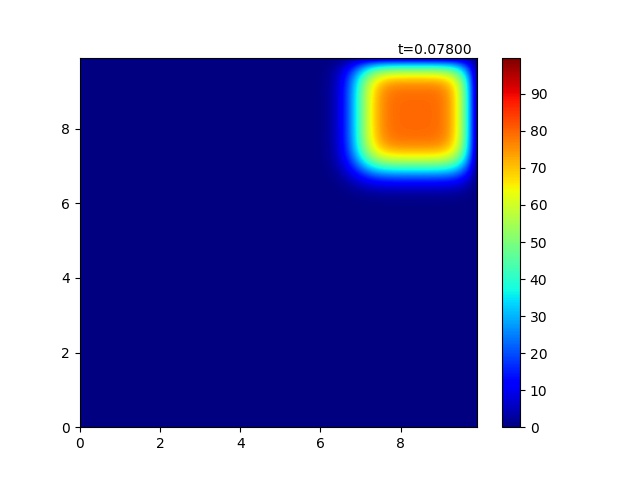


Figure 5 final state of Dirichlet condition

If we compare steady state with the cyclic condition, it is somehow darker, to be said, lower heat. This is because the wall condition did not conserve the heat.

### Neumann (G=0)



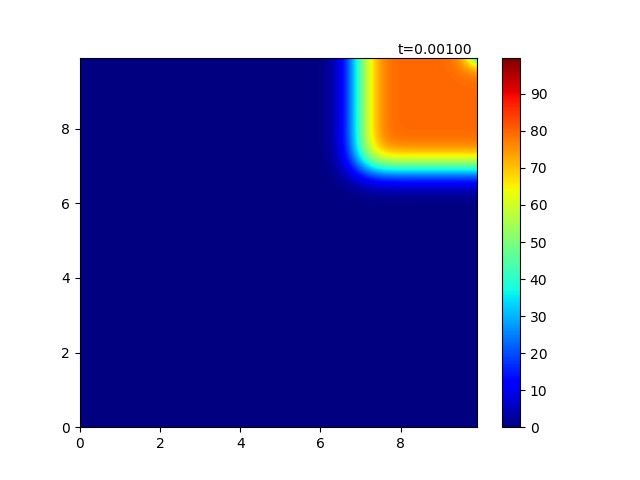


Figure 6 middle state of Neumann condition

If we compare with derichlet, the wall and the one beside the wall have almost the same amount of heat. Notice that np.roll() is implemented; however, we set the values of those rows or columns that went across the plane to zero. When g=0 the effect from the ghost point is going to be as following.

T[0,1:grid\_y-1] += d \* Tn[1,1:grid\_y-1]

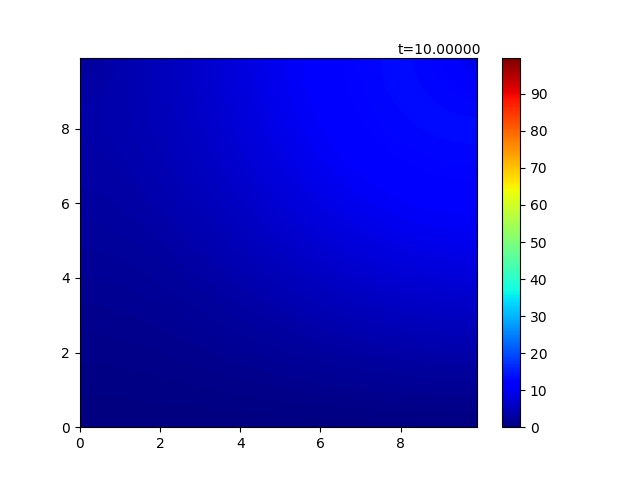


Figure 7 almost final state of Neumann condition

It is pretty sure that steady state should be uniformly distributed. However, with same time (10 seconds or 10000 loops) as cyclic and dirichlet condition, more heat is conserve than dirichlet but not well distributed as cyclic because heat cannot go through the wall.

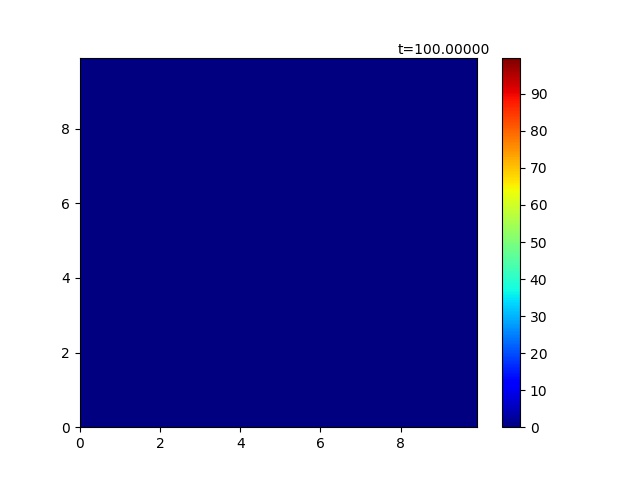


Figure 8 final state of Neumann condition

In order to observe the final state of Neumann condition, we loop it 10 times more (100000) then we reach a steady heat plane which looks as same as Dirichlet because heat lost through time.

### Neumann (G=20)

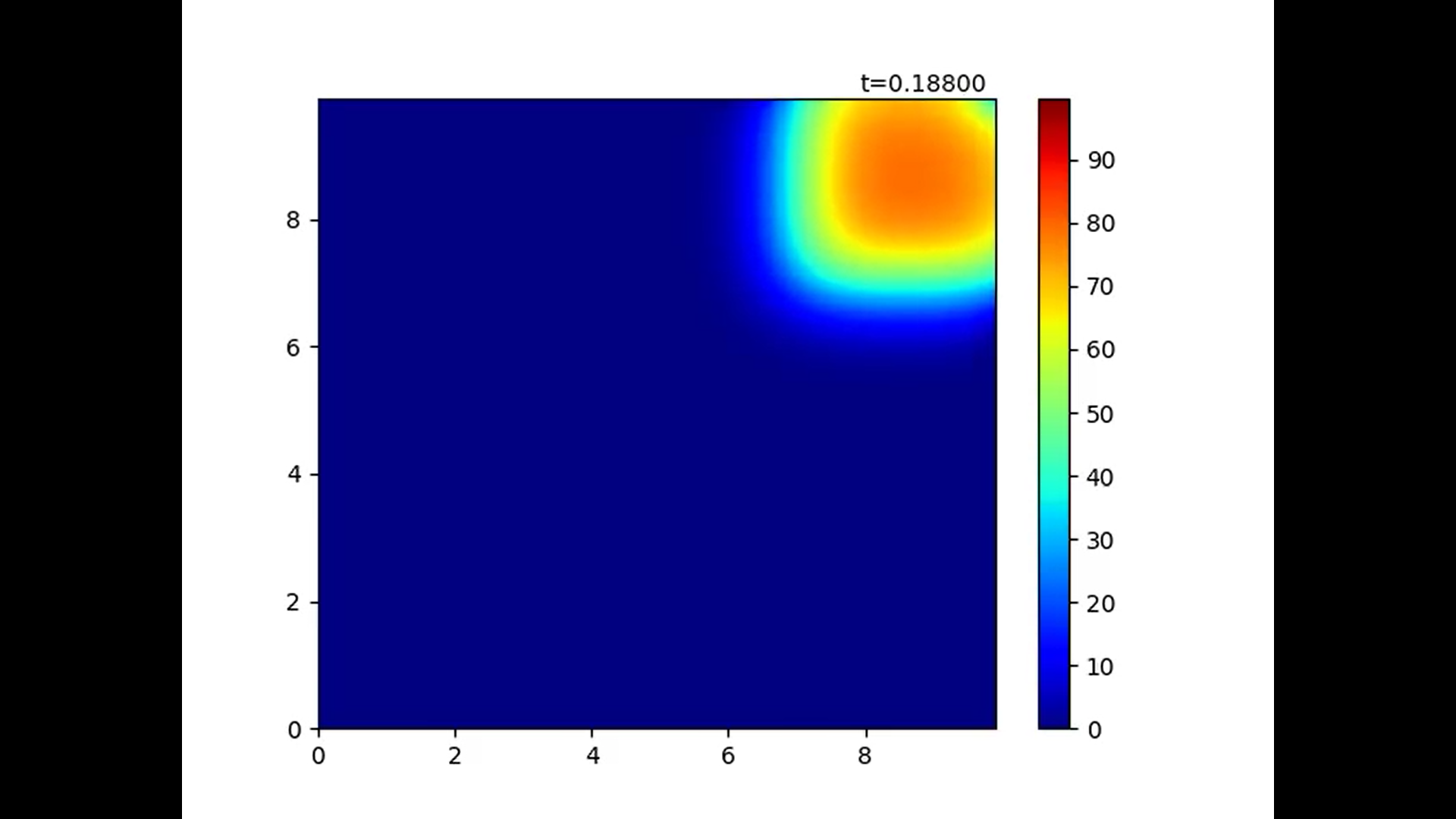


Figure 9 middle state of Neumann condition (G≠0)

With G ≠0 the length of the dx affected the heat of the ghost point. We can see that there is some gradient near the wall.

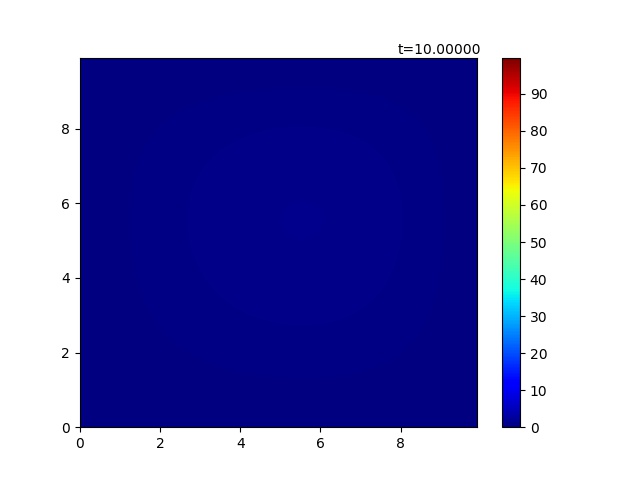


Figure 10 final state of Neumann condition (G≠0)

Within the system, heat loss by dx so the final state of this condition looks similar to Dirichlet condition.

## Effect of d’s value [Q.3]

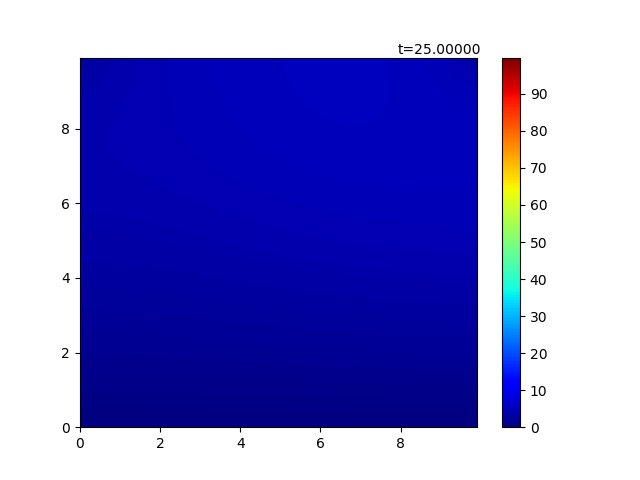


Figure 11 d=0.25 10000 loops

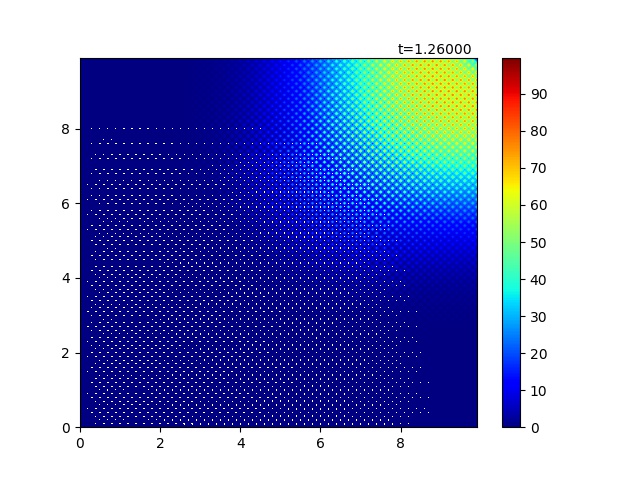


Figure 12 d=0.252 500 loops

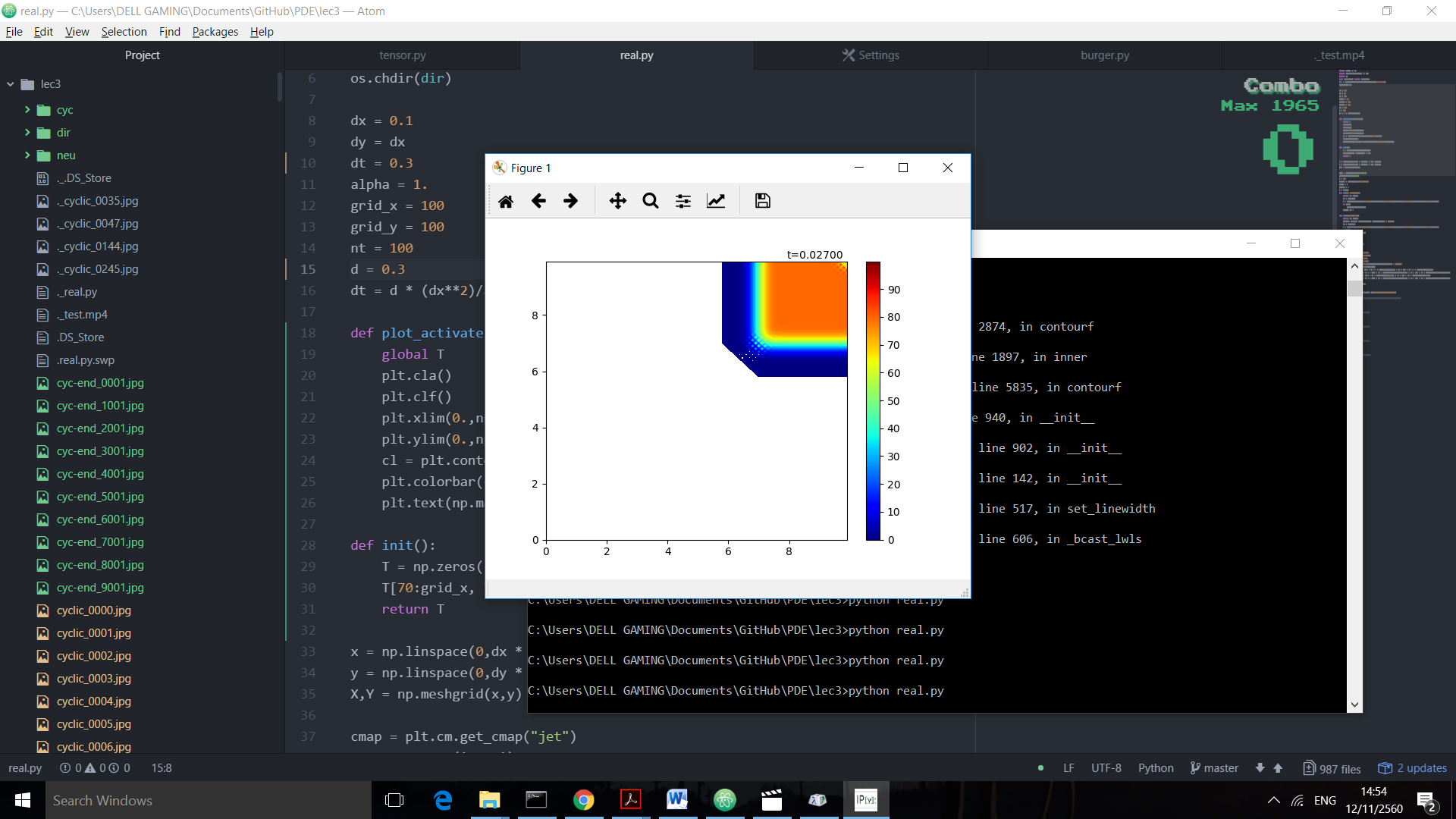


Figure 13 d = 0.3

According to Von Neumann Stability Analysis, for 2-dimensional data, it should be that d ≤0.25. It was verified by setting d=0.25 and passed it through 10000 loops which still sustained its stability; in the other hand, a small change to 0.252 with only 500 loops can destruct the plot by overflow of error. We tried d=0.3. Within 0.027 second the whole animation collapses because the error surges and exceed the heat range which displayed by white color.

## Python Code

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **import** os
4. **from** matplotlib **import** animation
5. dir = os.path.dirname(os.path.realpath(\_\_file\_\_))
6. os.chdir(dir)
8. dx = 0.1
9. dy = dx
10. alpha = 1.
11. grid\_x = 100
12. grid\_y = 100
13. nt = 100
14. d = 0.1
15. dt = d \* (dx\*\*2)/alpha
17. **def** plot\_activate(X,Y,n):
18. **global** T
19. plt.cla()
20. plt.clf()
21. plt.xlim(0.,np.max(x))
22. plt.ylim(0.,np.max(x))
23. cl = plt.contourf(X,Y,T,levels,cmap=cmap)
24. plt.colorbar(cl)
25. plt.text(np.max(x)\*0.8,np.max(y)+dy,"t=%01.5f"%(dt\*n))
27. **def** init():
28. T = np.zeros((grid\_x,grid\_y))
29. T[0:20, :] = 80.
30. T[grid\_x-20:grid\_x, :] = 80.
31. T[:,0:20] = 80.
32. T[:,grid\_y-20:grid\_y] = 80.
33. **return** T
35. x = np.linspace(0,dx \* (grid\_x - 1), grid\_x)
36. y = np.linspace(0,dy \* (grid\_y - 1), grid\_y)
37. X,Y = np.meshgrid(x,y)
39. cmap = plt.cm.get\_cmap("jet")
40. cmap.set\_over('grey')
41. g = 20
42. levels = np.arange(0.,100.,0.2)
43. count = 1
44. icount = 0
45. T = init()
46. **def** cyc(n, plot=True):
47. **global** T, icount
48. Tn = T.copy()
49. T = Tn+d\*(np.roll(Tn,1,axis=0)+np.roll(Tn,-1,axis=0)+np.roll(Tn,1,axis=1)+np.roll(Tn,-1,axis=1)-4\*Tn)
50. **if** plot:
51. plot\_activate(X,Y,n)
52. icount += 1
54. **def** dir(n,plot=True):
55. **global** T, icount
56. T[0,:], T[:,0], T[grid\_x-1,:], T[:,grid\_y-1] = 0,0,0,0
57. Tn = T.copy()
58. T = Tn+d\*(np.roll(Tn,1,axis=0)+np.roll(Tn,-1,axis=0)+np.roll(Tn,1,axis=1)+np.roll(Tn,-1,axis=1)-4\*Tn)
59. **if** plot:
60. plot\_activate(X,Y,n)
61. icount += 1
63. #FIX WALL
64. **def** neu(n, g=0,plot=True):
65. **global** T, icount
66. Tn = T.copy()
67. left = np.roll(Tn,1,axis=0)
68. right = np.roll(Tn,-1,axis=0)
69. up = np.roll(Tn,-1,axis=1)
70. down = np.roll(Tn,1,axis=1)
71. left[0,:],right[grid\_x-1,:],up[:,grid\_y-1],down[:,0] = 0,0,0,0
72. T = Tn+d\*(up+down+right+left-4\*Tn)
73. T[0,1:grid\_y-1] += d \* ((-2 \* dx \* g + Tn[1,1:grid\_y-1]) > 0) \* (-2 \* dx \* g + Tn[1,1:grid\_y-1])
74. T[grid\_x-1,1:grid\_y-1] += d \* ((-2 \* dx \* g+Tn[grid\_x-2,1:grid\_y-1]) > 0) \* (-2 \* dx \* g+Tn[grid\_x-2,1:grid\_y-1])
75. T[1:grid\_x-1,0] += d \* ((-2 \* dx \* g+Tn[1:grid\_x-1,1]) > 0) \* (-2 \* dx \* g+Tn[1:grid\_x-1,1])
76. T[1:grid\_x-1,grid\_y-1] += d \* ((-2 \* dx \* g+Tn[1:grid\_x-1,grid\_y-2]) > 0) \* (-2 \* dx \* g+Tn[1:grid\_x-1,grid\_y-2])
77. **if** plot:
78. plot\_activate(X,Y,n)
79. icount += 1
80. fig = plt.figure()
81. a = animation.FuncAnimation(fig, neu, fargs=(g,),frames=200,interval=10)
82. plt.show()