

Partial Differential Equations for Science and Engineering

Final Report

November 27, 2017

VARQUEZ, Alvin C.G.

14B00000

Contents

[Problem 1 Diffusion Equation 2](#_Toc498357664)

[A. Influence of Initial Condition 3](#_Toc498357665)

[Program 3](#_Toc498357666)

[Discussion 4](#_Toc498357667)

[Problem 2 Burger’s Equation 5](#_Toc498357668)

[Problem 3 1-D Advection Equation 6](#_Toc498357669)

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.

## Influence of Initial Condition

### Program

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **import** os
5. directory = os.path.dirname(os.path.realpath(\_\_file\_\_))
6. os.chdir(directory)
8. **def** get\_value(it):
9. y = t
10. **return** t
12. **def** main():
13. t = np.arange(0.,10.,0.1)
14. x = np.linspace(0.,10.,num=100)
15. y = np.copy(x)
16. y[:] = 0.
17. icount = 1
18. **for** it **in** t:
19. **for** i **in** range(0,y.shape[0]):
20. y[i] = get\_value(it)
21. plt.clf()
22. plt.plot(x,y)
23. plt.xlim(0,10)
24. plt.ylim(0,10)
25. plt.savefig("test\_%03di.jpg"%(icount))
26. icount+=1
28. **if** \_\_name\_\_ == "\_\_main\_\_":
29. main()

### Discussion

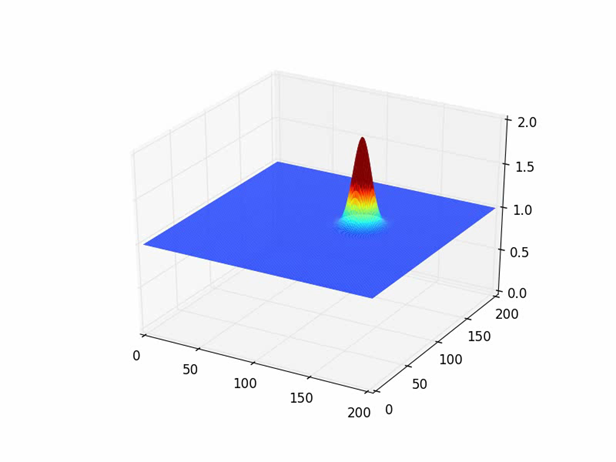


Figure Initial condition set to 100 and only to regions aa

The initial condition plays an important role.. In this report, I investigated the following conditions… All blah blah… as shown in Figure 1

# Problem 2 Burger’s Equation

Using forward-in-time and backward-in-space for the 1st derivative, and centered difference for the 2nd derivative, construct a numerical model for the Burger’s equation. Decide on your own initial conditions and values for *v*.



Discuss the following (add figures if necessary):

1. Derive and write the discretization as an algebraic equation.
2. Investigate by modelling the differences when *a* (linear) is a constant and when *a* is *u* (non-linear) for a cyclic condition.
3. What happens when you set a Dirichlet boundary?
4. List potential applications for Burger’s equation. Properly write the reference (e.g. T.J. Chung, 2002).

# Problem 3 1-D Advection Equation

Given the following 1-dimensional equation



At *t*=0,



and with a cyclic boundary, discuss using the following parameters: . To answer the questions, decide on appropriate *C* values. Construct a model using (1) Upwind scheme, (2) Leith’s Method, (3) CIP Method, and (4) analytical solution.

1. Construct *f* plots along x for *t=*100, 300, 500, 700. Compare the results of each method and discuss the errors accompanied by each method.

Note: Make sure the code is constructed neatly. Place comments using “#” character.

## Initial Condition

Plot Function

1. **def** plot\_line(n, folder): #Usual Plot Function
2. **global** model,f
3. plt.clf()
4. plt.cla()
5. plt.ylim(-.5,1.5)
6. plt.plot(model,f)
7. plt.text(80.,1.51,'t=%05.2f'%(n))
8. plt.text(10.,1.51,folder) #type of analysis
9. plt.savefig('%s/timestep\_%04i.jpg'%(folder, n))

## Initial Value

1. f = np.zeros\_like(model)
2. g = np.zeros\_like(model)
3. f[40:60] = 1.
4. g[40], g[60] = 1./dx, -1./dx
5. dx = 1.
6. C = 0.9
7. u = 1.
8. dt = np.abs(C \* dx/u)

## Functions

Upwind scheme

1. **def** upwind(n, toplot=True):
2. **global** f
3. fn = f.copy()
4. usign = int(np.sign(u))
5. f = fn + C \* (np.roll(fn,usign,0)-fn)
6. **if** toplot:
7. plot\_line(n, 'upwind') #save file in folder 'upwind'

Leith’s or Lax-Wendroff method

1. **def** lex(n, toplot=True):
2. **global** f
3. fn = f.copy()
4. c = fn
5. b = 1/(2\*dx) \* (np.roll(fn,-1,0) - np.roll(fn,1,0))
6. a = 1/(2\*dx\*\*2) \* (np.roll(fn,-1,0) - 2\*fn + np.roll(fn,1,0))
7. f = a \* (u\*dt)\*\*2 - b\*(u\*dt) + c
8. **if** toplot:
9. plot\_line(n, 'lex')

CIP Method

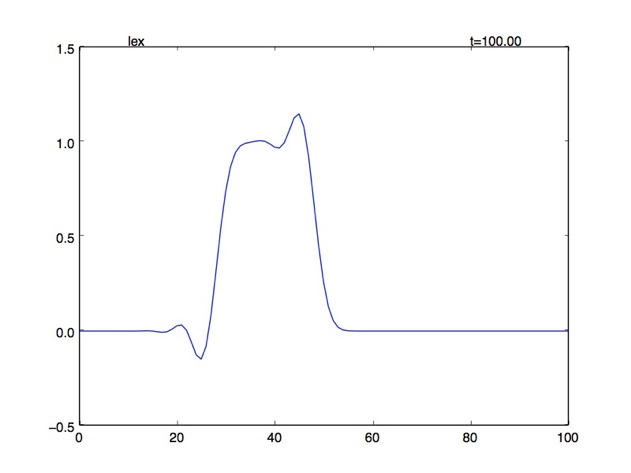
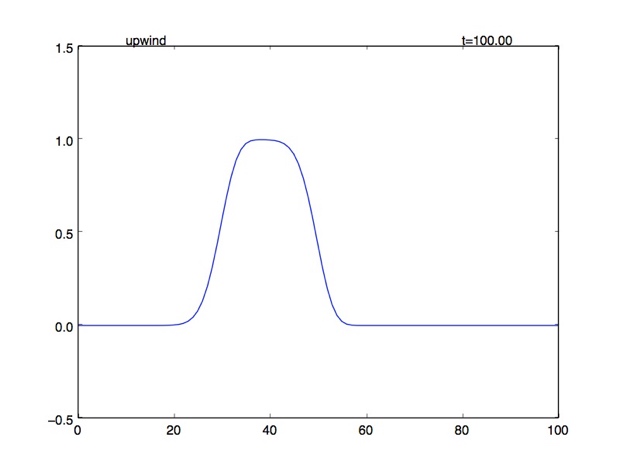
1. **def** cip(n, toplot=True):
2. **global** f,g
3. fn = f.copy()
4. gn = g.copy()
5. usign = np.int(np.sign(u))
6. x\_iiup = (-usign\*dx) #x\_iup - x\_i sign deped on the sgn of stream
7. a = -2\*(np.roll(fn,usign,0)-fn)/x\_iiup\*\*3 + (gn + np.roll(gn,usign,0))/x\_iiup\*\*2
8. b = -3\*(-np.roll(fn,usign,0)+fn)/x\_iiup\*\*2 - (2\*gn + np.roll(gn,usign,0))/x\_iiup
9. eps = -u\*dt
10. f = a \* eps\*\*3 + b\*eps\*\*2+gn\*eps+fn
11. g = (3\*a\*eps\*\*2+2\*b\*eps+gn)
12. **if** toplot:
13. plot\_line(n, 'cip')

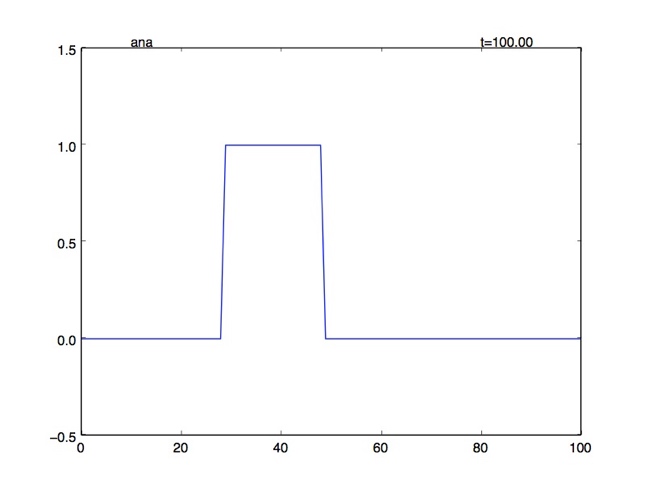
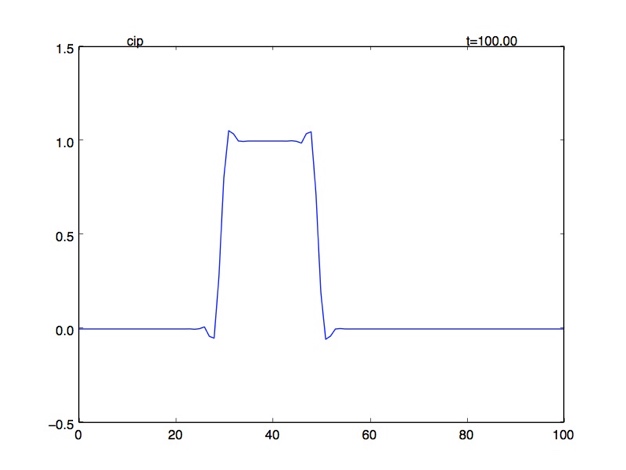
Analytical Solution

1. fana = f.copy()
2. **def** ana(n, toplot=True):
3. **global** f
4. f = np.roll(fana,np.int(np.floor(u\*dt/dx)%fana.shape[0])) #move function according to the stream
5. **if** toplot:
6. plot\_line(n, 'ana')

## Time Step

### t = 100

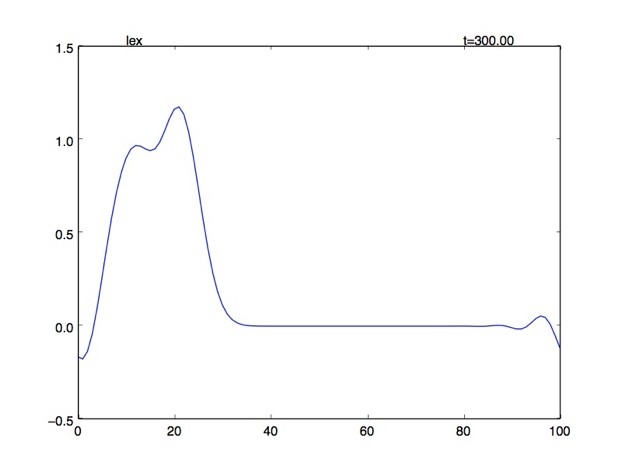
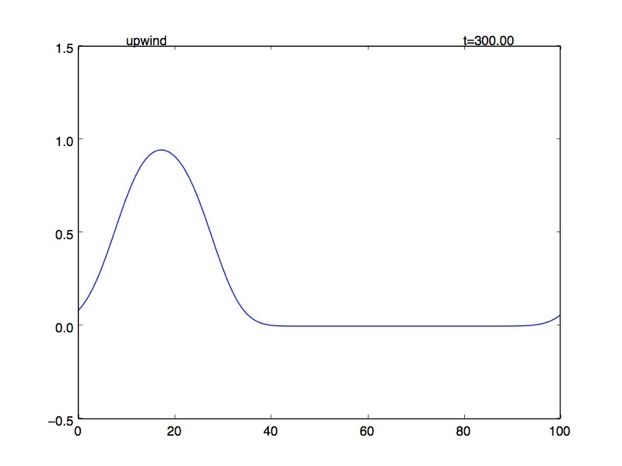


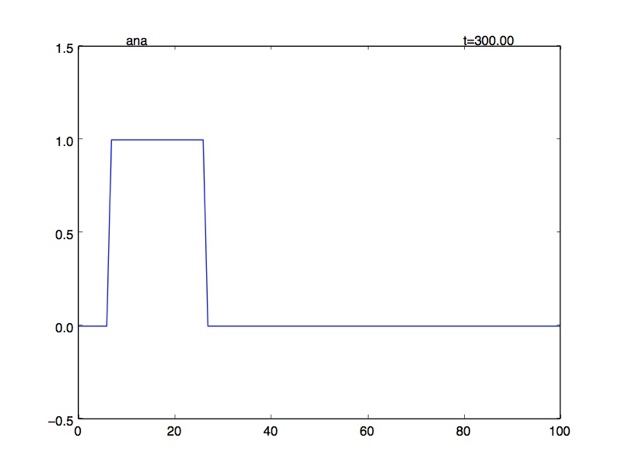
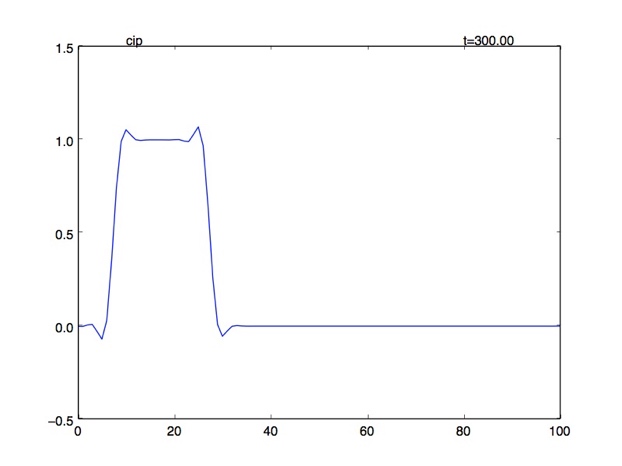


Upwind scheme: lost its shape and maximum altitude.

Leith’s method: some shooting appears at the border.

CIP method: the shape is conserved with a small overshooting around the corner.





Upwind scheme: kept losing its shape, maximum altitude and clearly became flat.

Leith’s method: more wavy shooting appears at the corners though the altitude is not that much changed.

CIP method:

