530.767 CFD
Spring 2024
HW 1–Haobo Zhao

February 9, 2024

This report could be divided two parts: For first part, we used different order UPwind scheme to simulate the 1-D wave equation, compared them with exact solution. By analyzing the result, we could find their modified wavenumber's real part influence dissipation error could contribute to FDE amplitude error, and its imaginary part influence dispersion error which could contribute to FDE phase error. For iteration time increase or condition frequency increase, these error occur obvious. For long time iteration, UPwind 3 scheme is not stable.

For the second part, we examined aliasing error by changing grid size, and plot the result use FFT. It shows for full aliased (raw grid), the amplitude for its wavenumber is different from the unaliased one (finer grid), while the result is also different.

Contents

1 Question Review

1. We will explore the finite-difference solution to the simple 1-D wave equation. Consider the following equation:

$$u_t + u_x = 0; \quad \text{on} \quad 0 \leq x \leq 2\pi \tag{1}$$

with

$$u(0,t) = u(2\pi,t)$$
$$\frac{\partial^n u}{\partial x^n}(0,t) = \frac{\partial^n u}{\partial x^n}(2\pi,t), \quad n > 0 \quad \text{(i.e. periodic BC)}$$
$$u(x,0) = \sin(mx)$$

(i) Discretize the above equation with an explicit scheme in time on a mesh with $\Delta x = \frac{2\pi}{20}$ and $\Delta t = 0.001$. Use the following spatial discretization schemes:

- First-order upwind
- Second-order upwind (using $i, i-1, i-2$)
- Third order upwind (using $i, i-1, i-2, i-3$)
- Third-order upwind biased (using $i+1, i, i-1$ and $i-2$)

and obtain the numerical solution for $m = 2, 4, 6,$ and 8. You should integrate the discretized equations long enough in time so that the effects of the truncation error are apparent.

(ii)Derive the exact solution of the PDE.

(iii)Derive and plot the modified wavenumber curves (check the lecture notes) for the four schemes; use the modified wavenumber analysis to explain the observed behavior of the numerical solution and its comparison to the exact solution.

2. Examination of aliasing error in solutions of time-dependent differential equations. Consider a domain of size 2π and a grid with $\Delta x = \frac{2\pi}{20}$ and the following two equations:

$$u_t + u^2 = 0$$

and

$$u_t + \sin(3x)u_x = 0$$

Let the initial condition be $u(x) = \sin(x) + 0.5\sin(4x)$ and use the Forward Euler scheme with a time step of 0.10.

For both these equations plot and compare the unaliased (assuming that the aliasing error is zero) and fully aliased energy spectra for the resolved scales at time steps from 1 to 10. Explain the key characteristics of the results that you observe. Spectra can be obtained by running a FFT on your simulation results.

You can obtain an unaliased spectra by running the same simulation on a finer grid; you might also need a small time-step size for the finer grid.

2 1. Finite Difference for 1D Wave Equations.

2.1 Finite Difference Schemes

The PDE is showing below:

$$u_t + u_x = 0; \quad \text{on} \quad 0 \le x \le 2\pi \tag{2}$$

Using forward difference in time, transform u_x to different finite difference method, the equation is showing below:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + FD(u_i^n) = 0 \tag{3}$$

Then the iteration formula is showing below:

$$u_i^{n+1} = u_i^n - \Delta t \cdot FD(u_i^n) \tag{4}$$

2.1.1 1st Upwind Scheme

For the 1st order UPwind scheme, u_x becomes as follows:

$$FD(u_i') = \frac{u_i^n - u_{i-1}^n}{\Delta x} \tag{5}$$

$$u_i^{n+1} = u_i^n + \frac{\Delta t}{\Delta x}(-u_i^n + u_{i-1}^n) \tag{6}$$

2.1.2 2nd Upwind Scheme

For the 2nd order UPwind scheme, u_x becomes as follows:

$$FD(u_i') = -\frac{3u_i - 4u_{i-1} + u_{i-2}}{2\Delta x} \tag{7}$$

$$u_i^{n+1} = u_i^n + \Delta t \cdot \frac{3u_i^n - 4u_{i-1}^n + u_{i-2}^n}{2\Delta x} \tag{8}$$

2.1.3 3rd Upwind Scheme

For the 3rd order UPwind scheme, u_x becomes as follows:

$$FD(u_i') = \frac{\frac{11}{6}u_i^n - 3u_{i-1}^n + \frac{3}{2}u_{i-2}^n - \frac{1}{3}u_{i-3}^n}{\Delta x} \tag{9}$$

$$u_i^{n+1} = u_i^n - \Delta t \cdot \frac{\frac{11}{6}u_i^n - 3u_{i-1}^n + \frac{3}{2}u_{i-2}^n - \frac{1}{3}u_{i-3}^n}{\Delta x} \tag{10}$$

2.1.4 3rd biased Upwind Scheme

For the 3rd order biasd UPwind scheme, u_x becomes as follows:

$$u_i^{n+1} = u_i^n - \Delta t \cdot \frac{2u_{i+1}^n + 3u_i^n - 6u_{i-1}^n + u_{i-2}^n}{6\Delta x} \tag{11}$$

2.2 Solver Algorithm

The solver architecture is showing below:

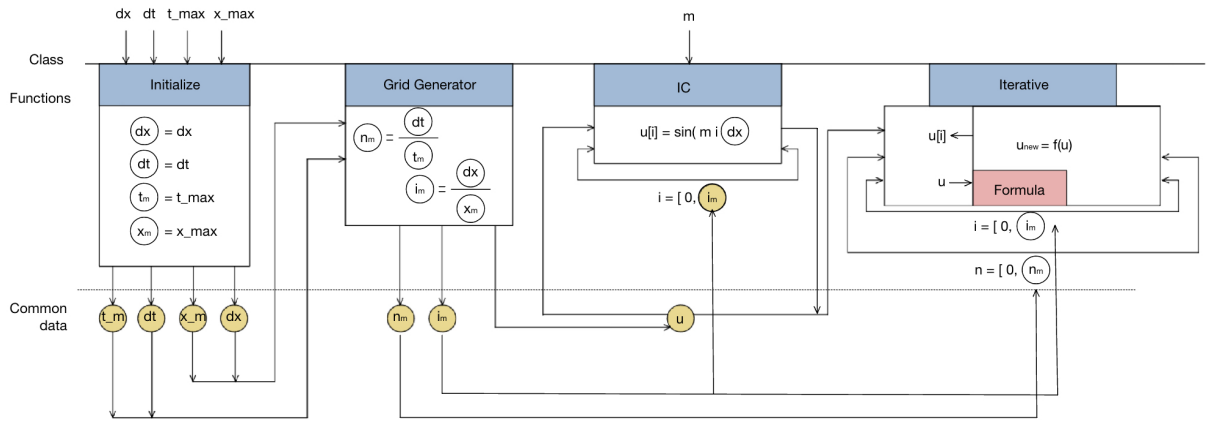


Figure 1: Solver architecture

As we put all the function we need into same class, the data transfer between functions become much easier, the variables in color yellow means common parameters in the class. To find the numerical solution on certain time, we only need to call each functions in sequence. If we need to use different solver scheme, we only need to inheritance, and define new Iterative function (for BC and parameter passing), Formula function (different scheme), and IC function (initial condition).

2.3 (i) Result–Solutions for different schemes

For this section, we compared the different results with m from two time: t=0.1s, and t=1s. It shows as m increase, the difference between our FDE scheme and the exact solution makes difference, not only amplitude difference, but also the phase difference.

For the solution, the result is showing below:

For $t_{max} = 0.1$, the result for different m is showing below:

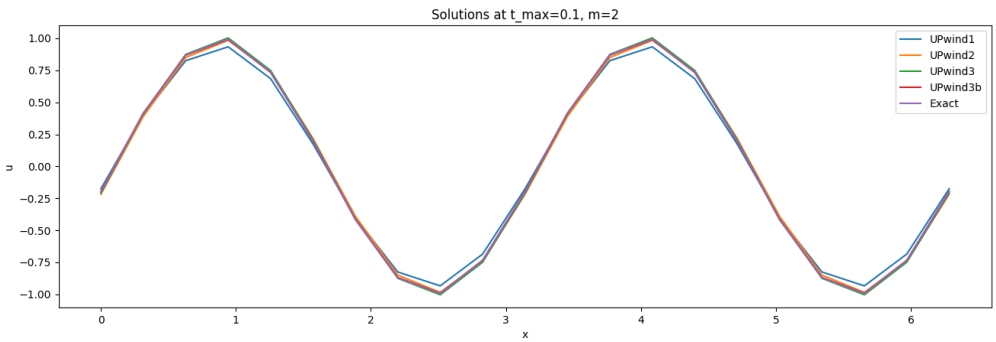


Figure 2: result comparison of t=0.1, m=2

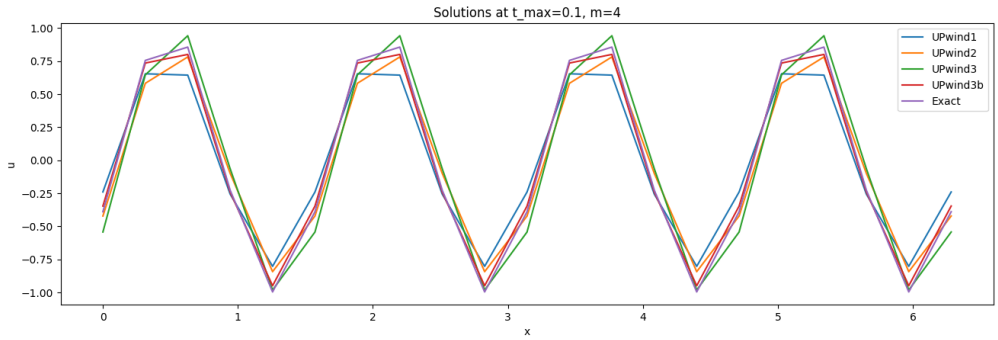


Figure 3: result comparison of t=0.1, m=4

It could seen although is not enormous, the result difference of FDE of PDE is occur: UPwind 1 and UPwind 3 scheme's difference with exact solution is obvious.

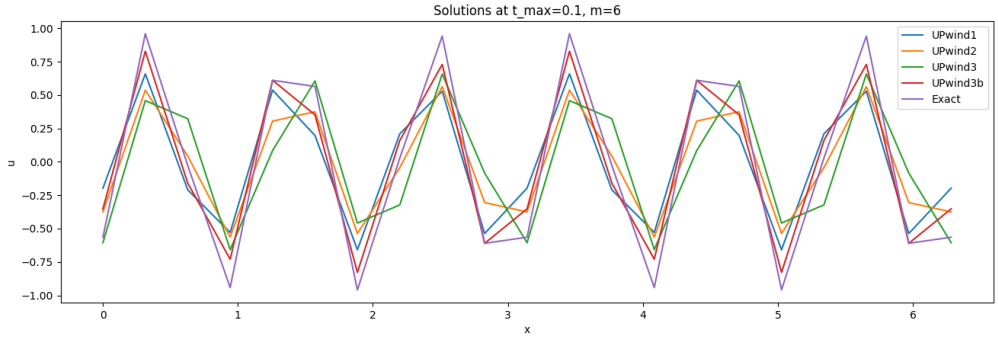


Figure 4: result comparison of t=0.1, m=6

At m=6, the difference is much more obvious.

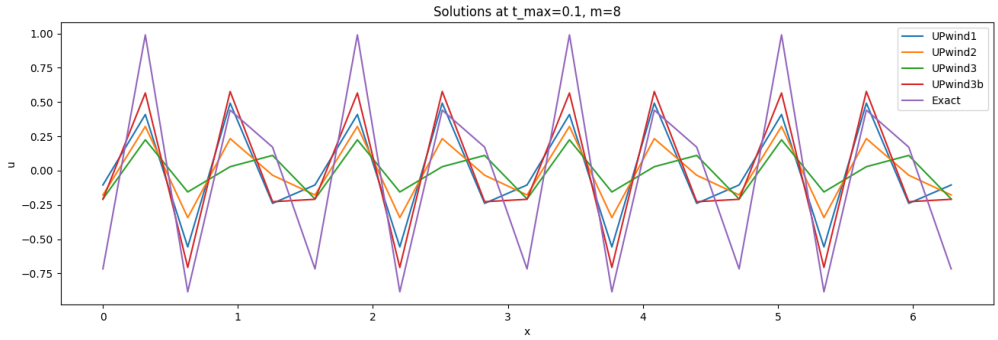


Figure 5: result comparison of t=0.1, m=8

At m=8, could seen the difference of wave amplitiude is showing.

For $t_{max} = 1$, the results for different m is showing below:

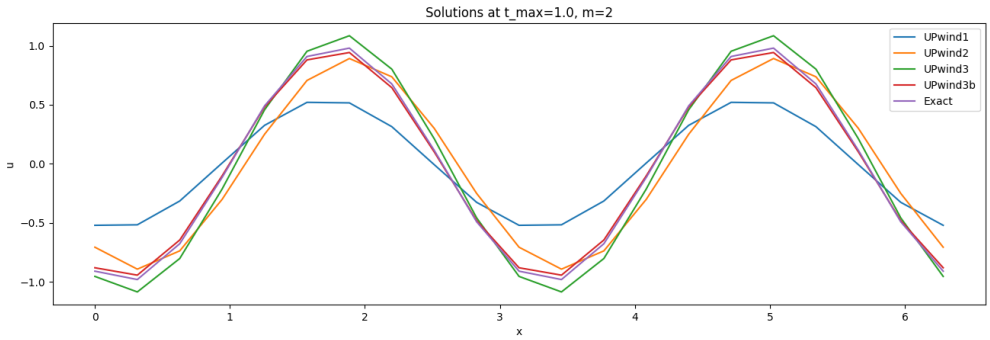


Figure 6: result comparison of t=1, m=2

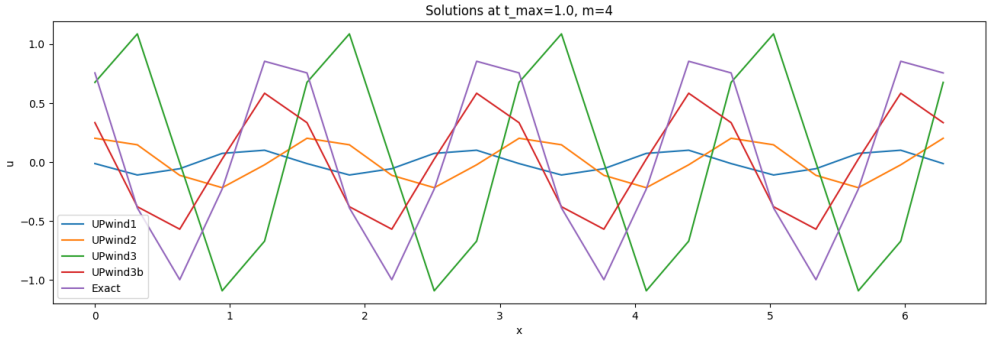


Figure 7: result comparison of t=1, m=4

It is showing for this time, not only amplitude could be observed, but also the phase difference is showing up.

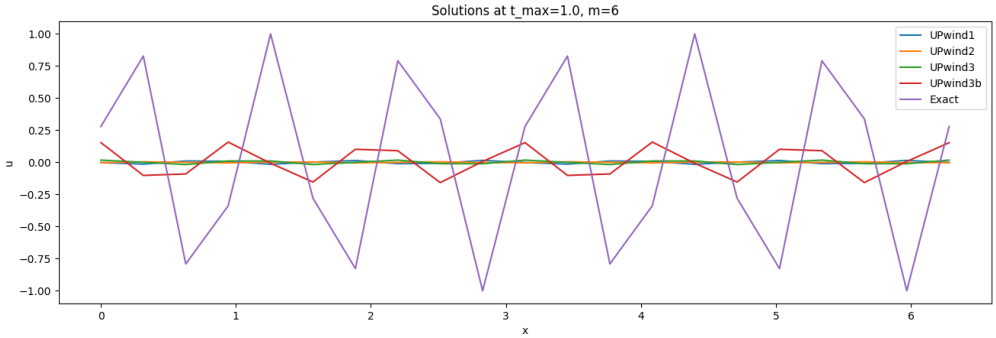


Figure 8: result comparison of t=1, m=6

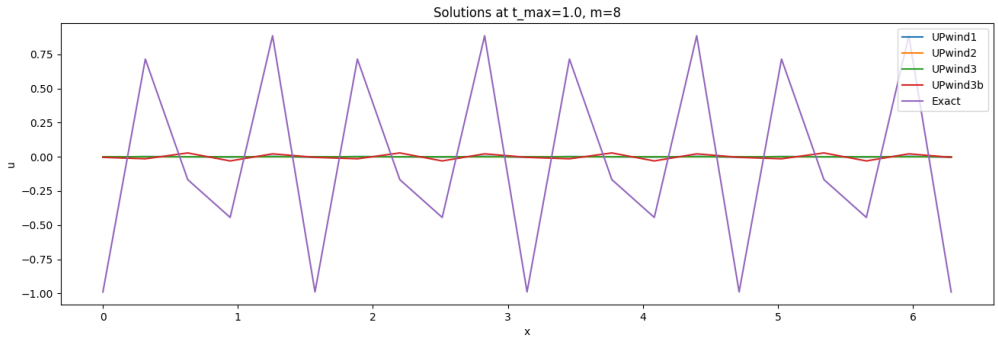


Figure 9: result comparison of t=1, m=8

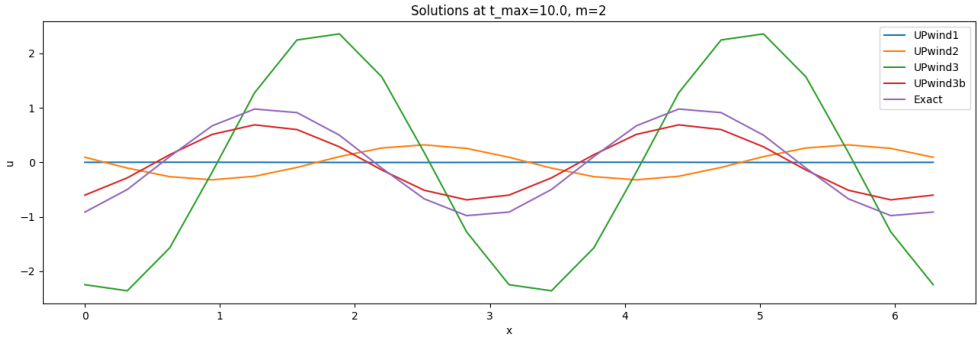


Figure 10: result comparison of t=10, m=2

For t=10, which is pretty long time iteration, it shows UPwind 3 is not stable.

2.4 (ii) Exact Solution

The 1-D wave equation is:

$$u_t + u_x = 0$$

Use method of separation of varible, we could write u as a combination of a function of t and a function of x:

$$u = A(t)B(x)$$

Then, could get:

$$\frac{1}{A}\frac{dA}{dt} = -\frac{1}{B}\frac{dB}{dx} = h$$

Could find the result in the general form:

$$y = ce^{-ht}e^{hx} == A\cos(ht)e^{-ht} - iB\sin(ht)e^{-ht}$$

As the initial condition:

$$y(0,t) = \sin(mx)$$

could get the solution:

$$y = \sin m(x - t)$$

The exact solution for the problem is:

$$u(x,t) = \sin(mx - mt)$$

2.5 (iii) Modified Wavenumber Curves and Analysis

2.5.1 Modified Wavenumbers

For the wave number analysis, the solution at certain time step could be represented as following:

$$u_j = ue^{ikx} \tag{12}$$

Take derivative f u, could get wave number k out of u:

$$\text{PDE:} \quad u'_j = iku \tag{13}$$

For the Finite difference operation, it is different, but could be modified as below:

1st UPwind Scheme modified wavenumber

$$\text{FDE}(u_j) = ik'u = \frac{u_j - u_{j-1}}{\Delta x} = \frac{ue^{ikx} - ue^{ik(x-\Delta x)}}{\Delta x} \tag{14}$$

$$ik'u = i(-i\left(\frac{1 - e^{-ik\Delta x}}{\Delta x}\right))u \tag{15}$$

Then, we could get the modified wavenumber which contain two parts: Real part and Imaginary part:

$$k' = \frac{\sin(k\Delta x)}{\Delta x} + i\frac{-1 + \cos(k\Delta x)}{\Delta x} = \text{Re}(k') + \text{Im}(k') \tag{16}$$

2nd UPwind Scheme modified wavenumber

$$\text{FDE}(u_j) = ik'u = 3u_j - 4u_{j+1} + u_{j+2} \tag{17}$$

$$= \frac{3ue^{ikx} - 4ue^{ik(x+\Delta x)} + ue^{ik(x+2\Delta x)}}{\Delta x} \tag{18}$$

$$ik'u = u\left(\frac{3 - 4e^{ik\Delta x} + e^{2ik\Delta x}}{\Delta x}\right) \tag{19}$$

$$k' = -\frac{4\sin(k\Delta x) - \sin(2k\Delta x)}{2\Delta x} + i\frac{4\cos(k\Delta x) - \cos(2k\Delta x) - 3}{2(\Delta x)^2} \tag{20}$$

3rd UPwind Scheme modified wavenumber

$$\text{FDE}(u_j) = ik'u = \frac{\frac{11}{6}u_j - 3u_{j-1} + \frac{3}{2}u_{j-2} - \frac{1}{3}u_{j-3}}{\Delta x} \tag{21}$$

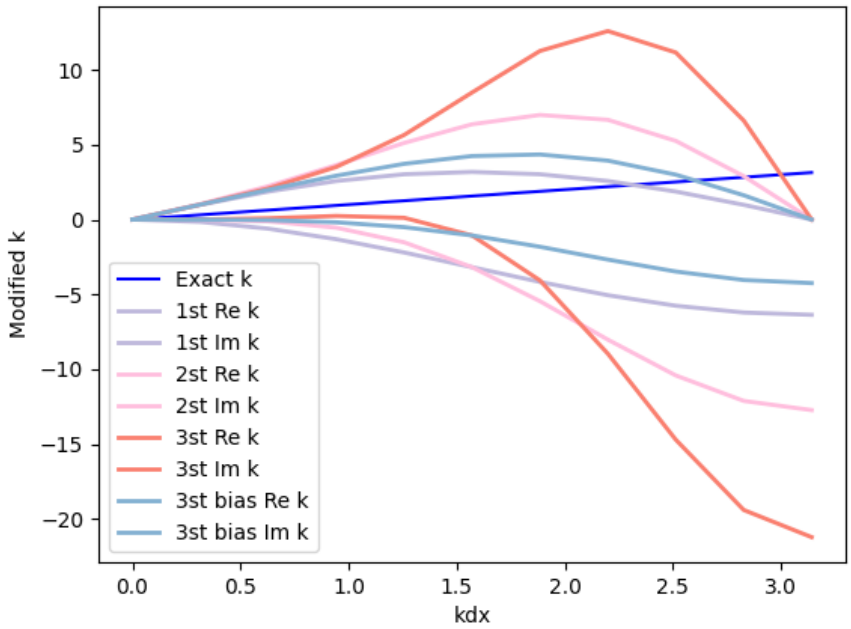
$$k' = \frac{3\sin(k\Delta x) - \frac{3}{2}\sin(2k\Delta x) + \frac{1}{3}\sin(3k\Delta x)}{\Delta x} + i\left(\frac{3\cos(k\Delta x) - \frac{3}{2}\cos(2k\Delta x) + \frac{1}{3}\cos(3k\Delta x) - \frac{11}{6}}{\Delta x}\right) \tag{22}$$

$$k' = -\frac{4\sin(k\Delta x) - 3\sin(2k\Delta x) + \sin(3k\Delta x)}{(\Delta x)^2} + i\frac{4(1-\cos(k\Delta x)) - 3(1-\cos(2k\Delta x))}{(\Delta x)^2} \tag{23}$$

3rd UPwind Scheme modified wavenumber

$$k' = \frac{6\sin x - \sin 2\Delta x + 2\sin 3\Delta x}{6\Delta x} + i\frac{6\cos\Delta x - \cos 2\Delta x + 2\cos 3\Delta x - 3}{6\Delta x} \tag{24}$$

2.5.2 Modified Wavenumber Curves and Result Analysis



For the modified wavenumber, the result is showing above. The Exact wavenumber is named "Exact k" in the picture, which don't have imaginary part. For the modified wavenumbers for upwind schemes, each scheme's wavenumber have two parts in the same color, the real part named "Re k" in the chart, while the imaginary part named "Im k" for each scheme.

For the modified wavenumber's real part (Re k) difference could cause dissipation error, makes wave decrease faster or slower, which is shwn in the chart t=1 m=2, UPwind 1 scheme's

wave is much smaller than the exact solution, also chart t=1 m=6, and t=1 m=8, where numerical simulation wave is much smaller than the exact solution. Its imaginary part (Im k) could cause dispersion error, which could let the simulation phase difference with the exact solution, which is pretty obvious in chart t=1 m=4, where UPwind 2 scheme's phase is obviously difference than others, and all numerical simulation wave have phase difference with the exact solution.

It is easily to be observed that, for the modified wavenumber's real part, the 3rd UPwind scheme is much larger than others, which is also far away from the exact k, which could explain it cannot well-simulate the equation, espically on the lone time rigion. On the other hand, 3rd biased UPwind scheme is much closer to the exact k than the unbiased one, which could explain its simulation result is closer to the exact solution.

For the modified wavenumber's imaginary part, the 3rd biased scheme's Im k is much smaller than the others, which could explain its good simulation result.

3 2. Aliased Analysis with FFT

3.1 2.1 Function 1 analysis

Function 1 is

$$u_t + u^2 = 0$$

its iteration formula is:

$$u_{\text{new}} = u - \Delta t \cdot u^2$$

we use original grid $dx = \frac{2\pi}{20}$, $dt = 0.1$, the following charts are the result compare (left), and spatial frequancy domain compare (right) of raw grid (Fully aliased) and finer grid (un-aliased):

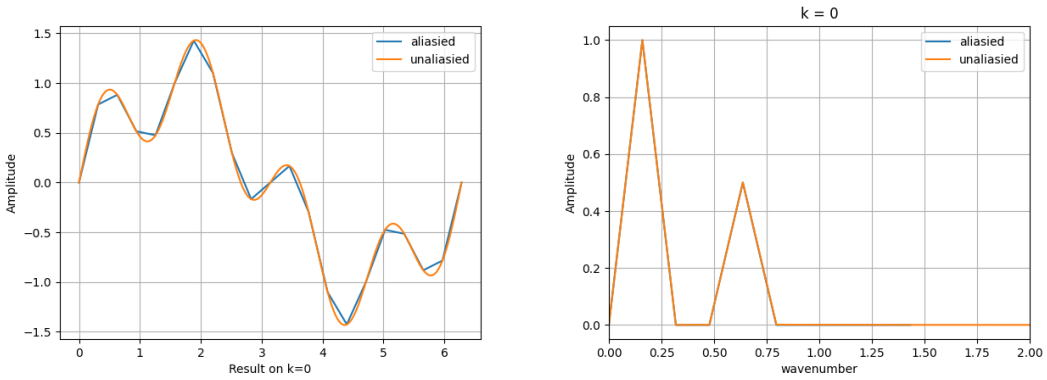


Figure 11: k=0, result comparison and frequency comparison

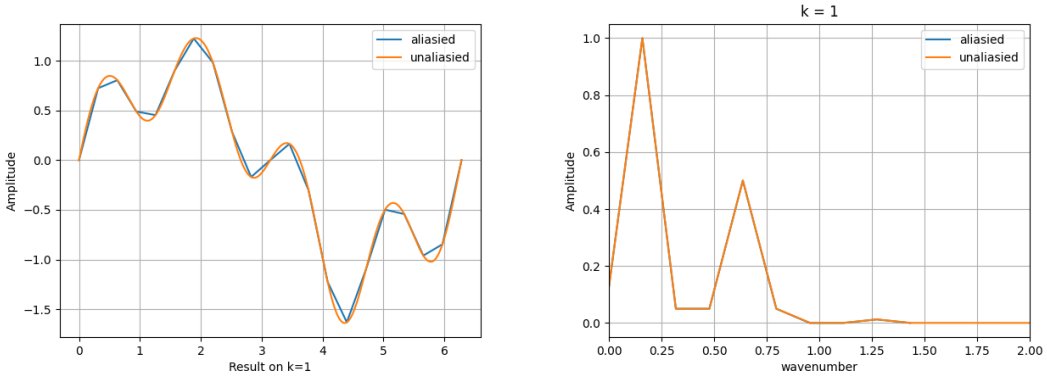


Figure 12: k=1, result comparison and frequency comparison

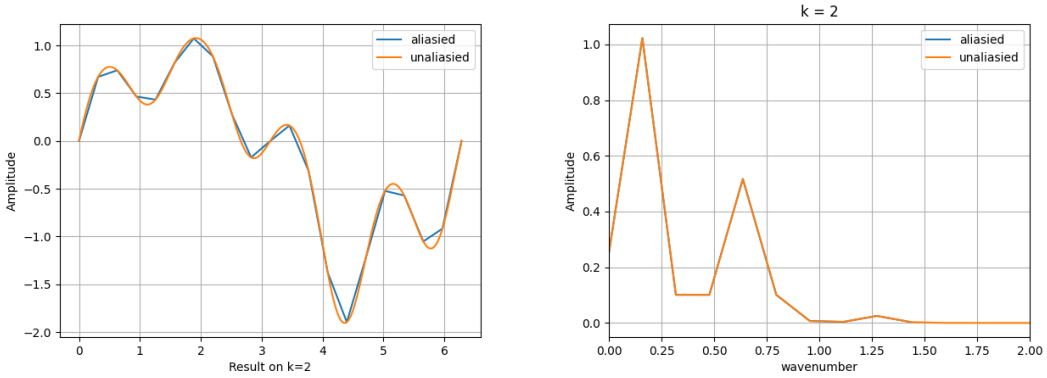


Figure 13: k=2, result comparison and frequency comparison

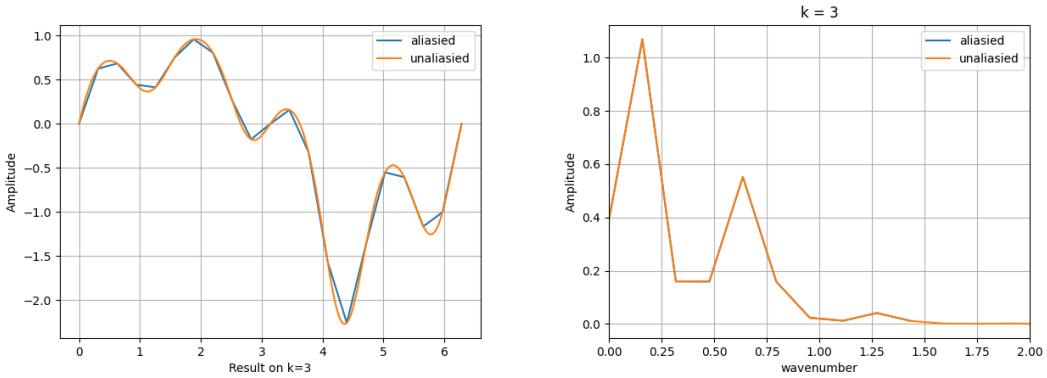


Figure 14: k=3, result comparison and frequency comparison

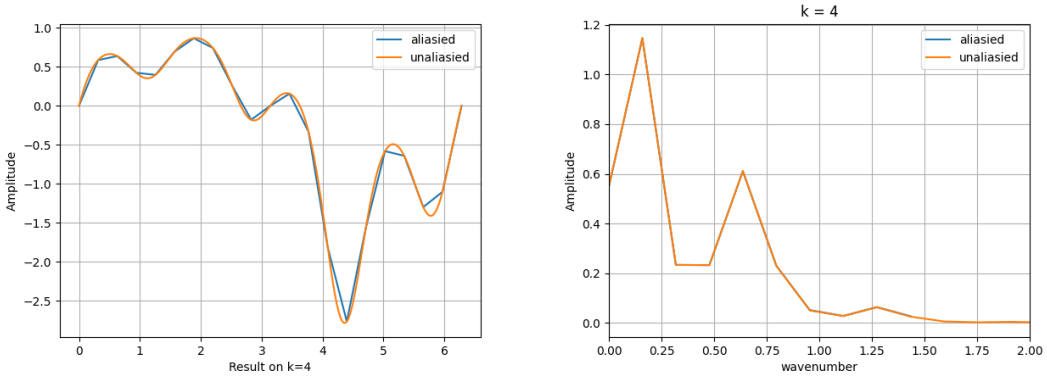


Figure 15: k=4, result comparison and frequency comparison

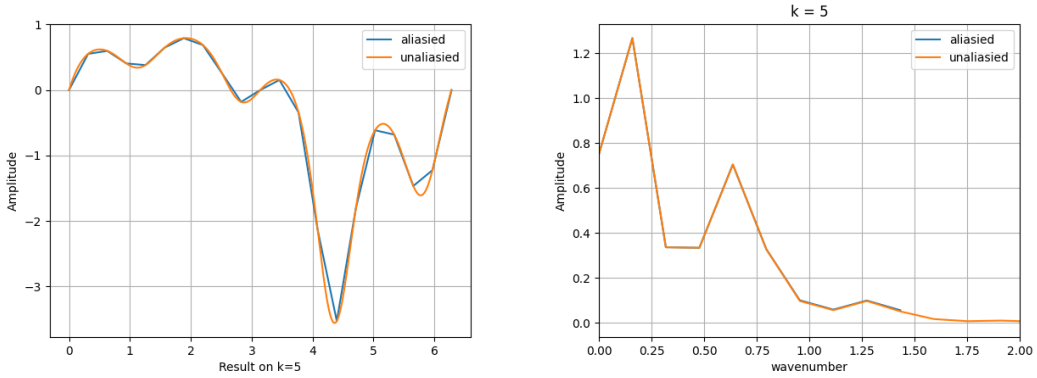


Figure 16: k=5, result comparison and frequency comparison

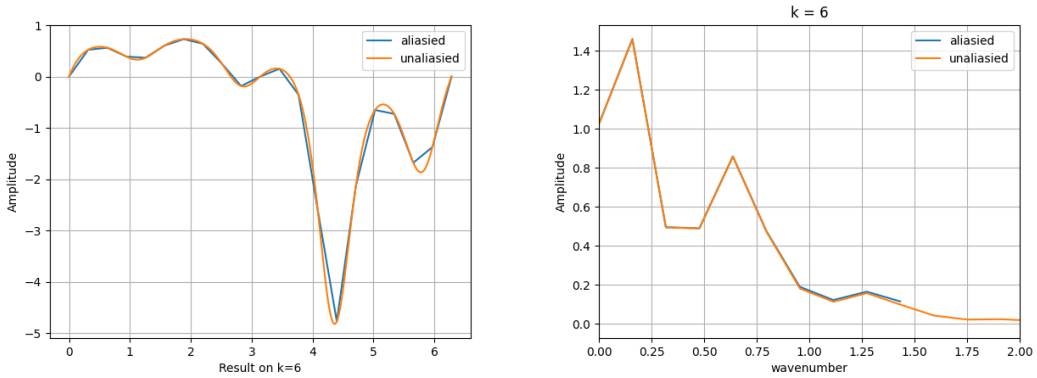


Figure 17: k=6, result comparison and frequency comparison

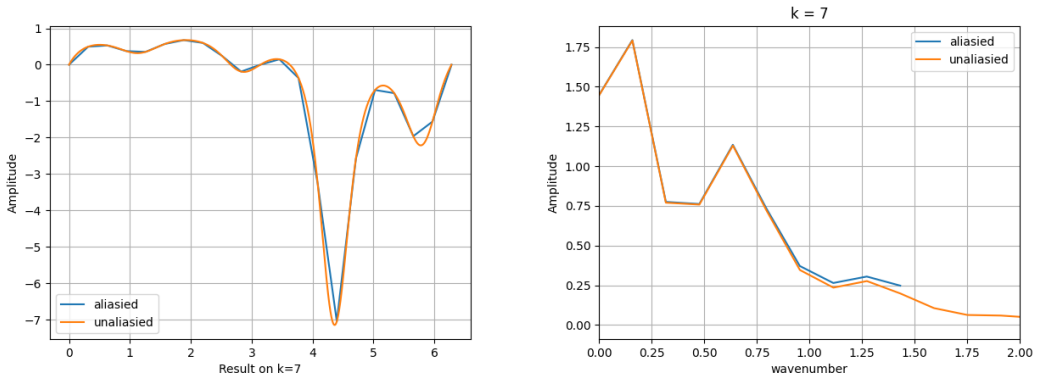


Figure 18: k=7, result comparison and frequency comparison

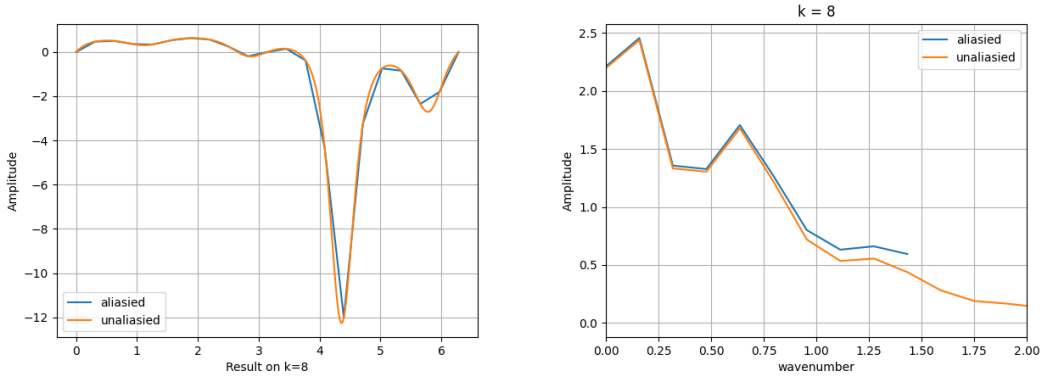


Figure 19: k=8, result comparison and frequency comparison

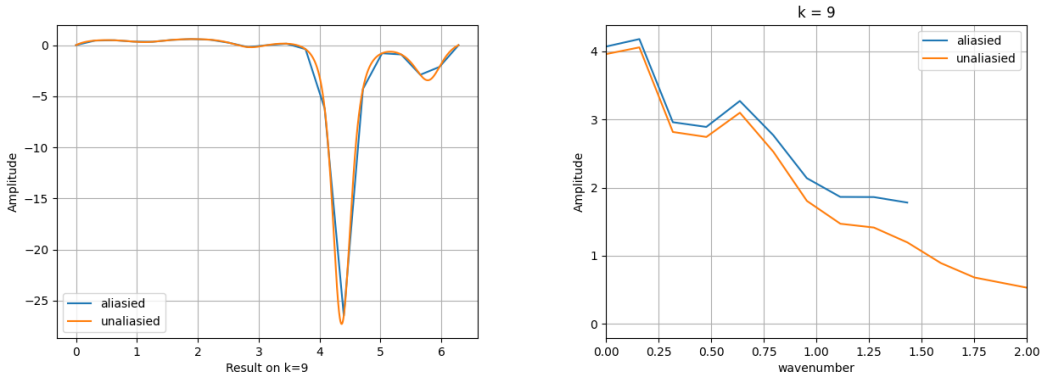


Figure 20: k=9, result comparison and frequency comparison

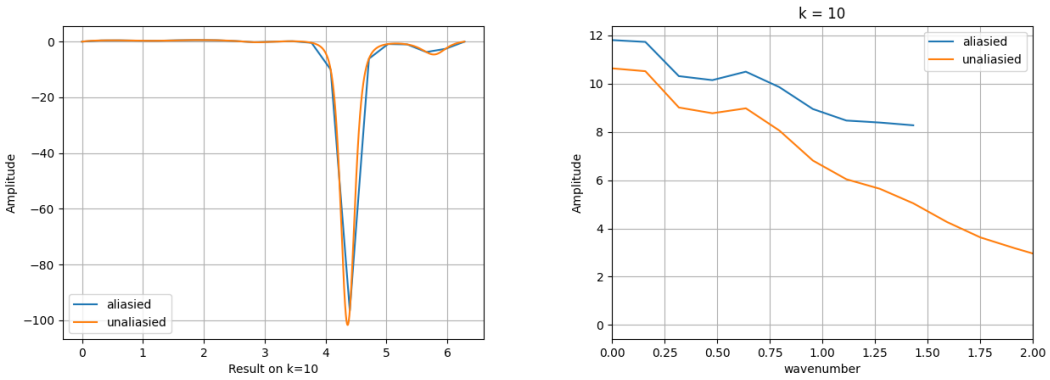


Figure 21: k=10, result comparison and frequency comparison

It could be shown, as iteration running, the aliasing error occur, makes high wavenumber larger than k_{max}, and mirrored back to influence the low wavenumebr, makes difference between the original grid and the finer grid.

In these pictures, as iteration times k increase, the difference of low wavenumber amplitude between fully aliased and un-aliased result is increase, shows the energy is 'abnormal' increase due to the aliasing error.

3.2 2.2 Function 2 analysis

The function 2 equation is showing below:

$$u_t + \sin(3x)u = 0$$

Its iteration formula is:

$$u_{\text{new}} = u - \Delta t \cdot \sin(3x) \cdot u$$

The results is showing below:

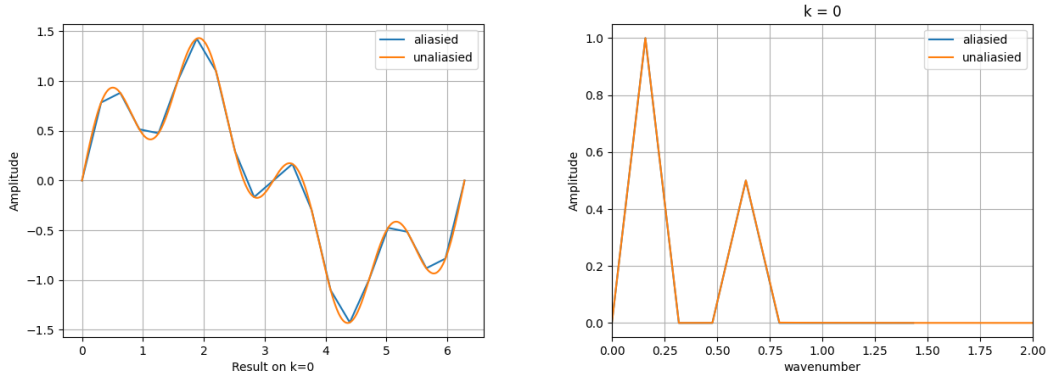


Figure 22: k=0, result comparison and frequency comparison

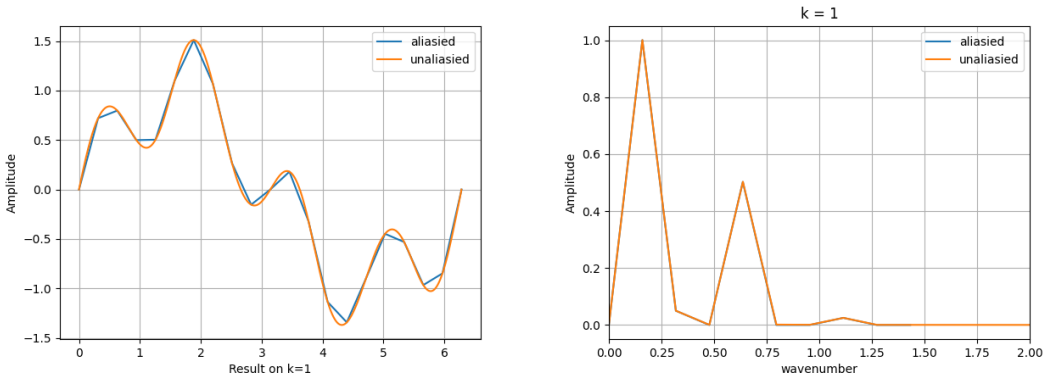


Figure 23: k=1, result comparison and frequency comparison

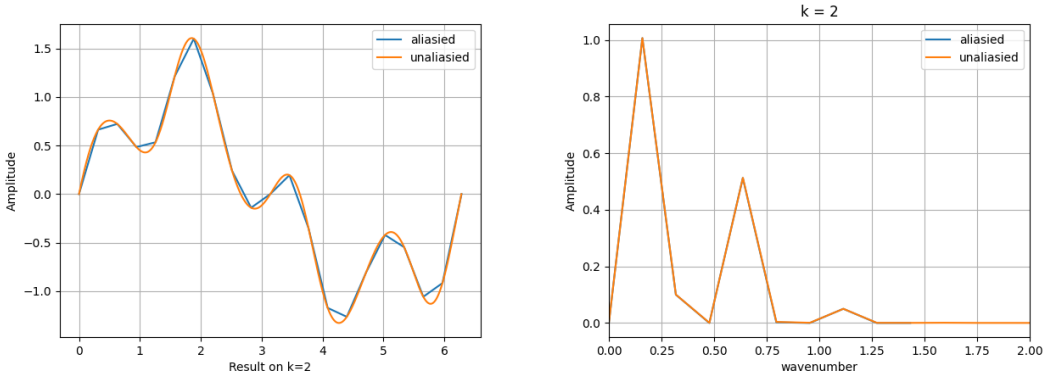


Figure 24: k=2, result comparison and frequency comparison

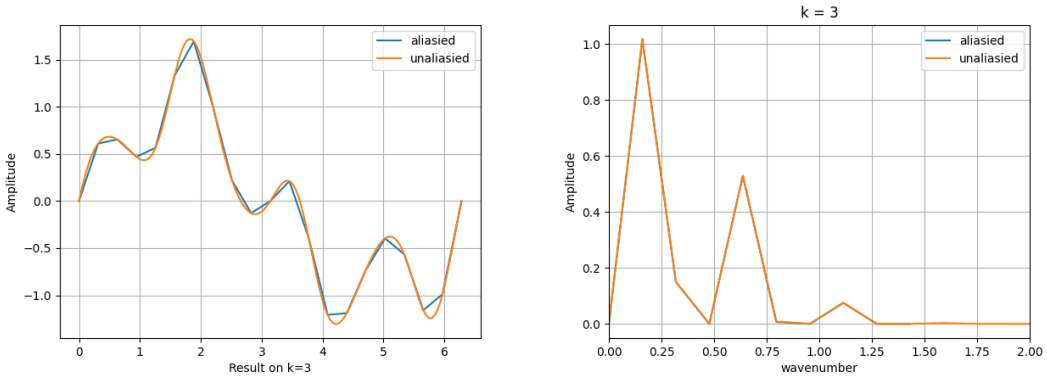


Figure 25: k=3, result comparison and frequency comparison

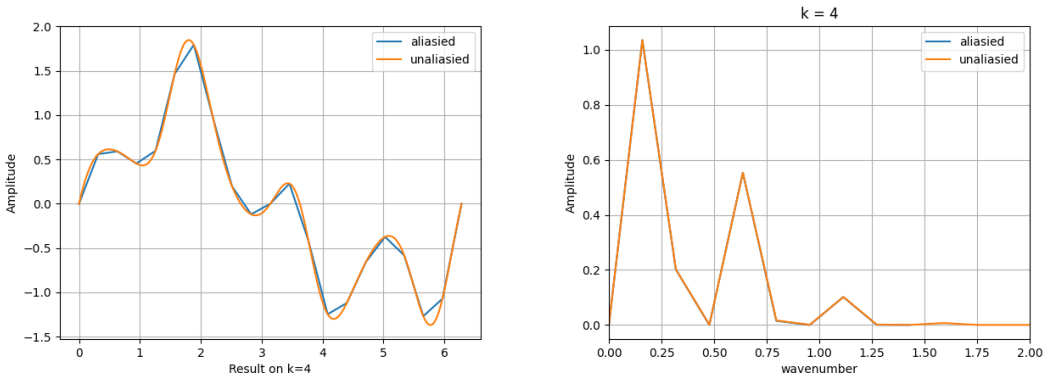


Figure 26: k=4, result comparison and frequency comparison

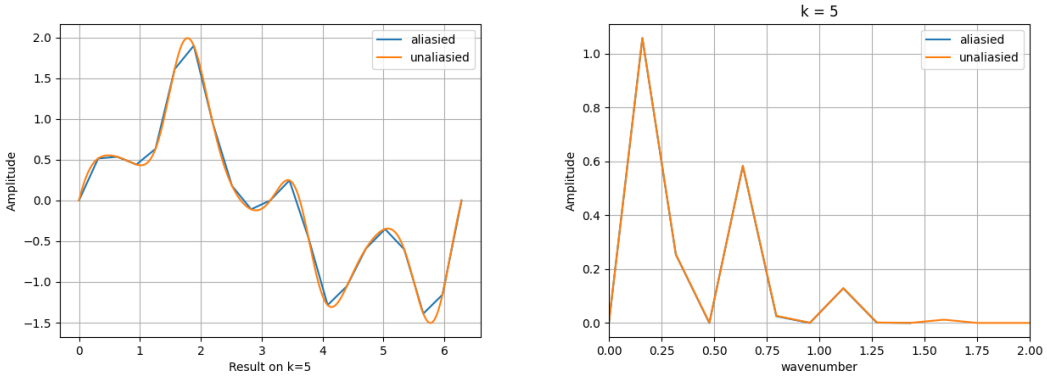


Figure 27: k=5, result comparison and frequency comparison

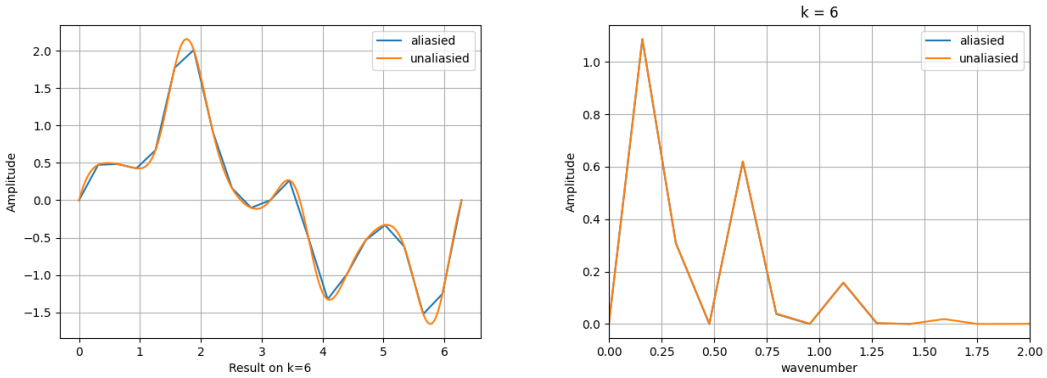


Figure 28: k=6, result comparison and frequency comparison

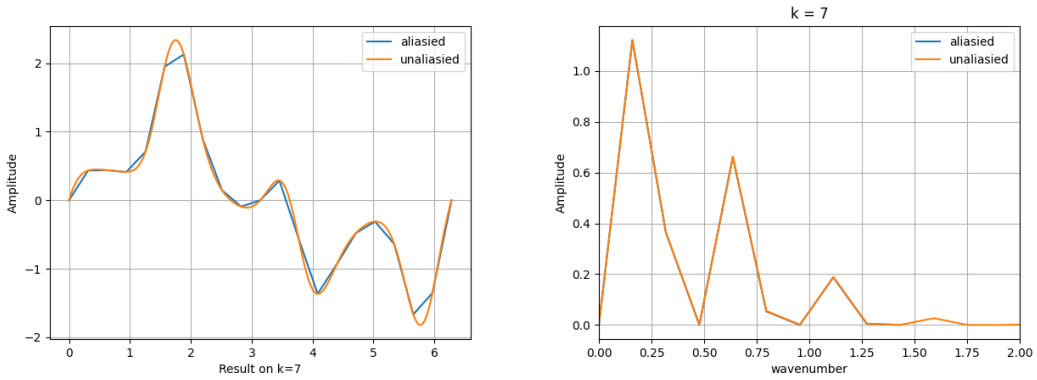


Figure 29: k=7, result comparison and frequency comparison

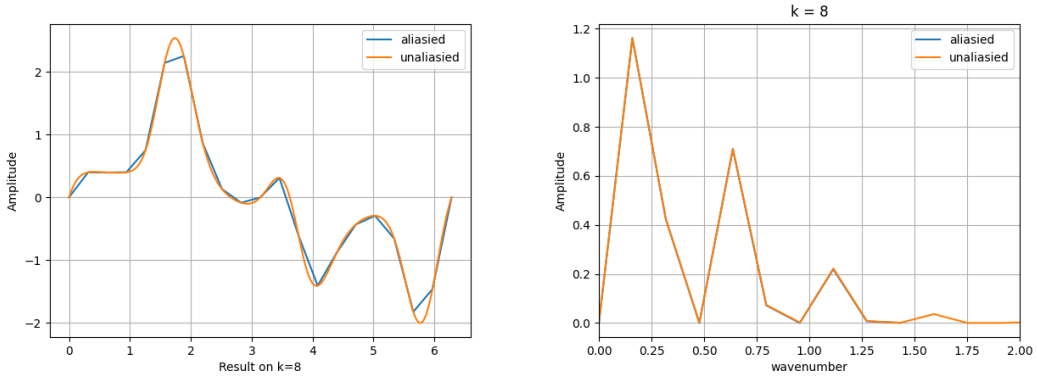


Figure 30: k=8, result comparison and frequency comparison

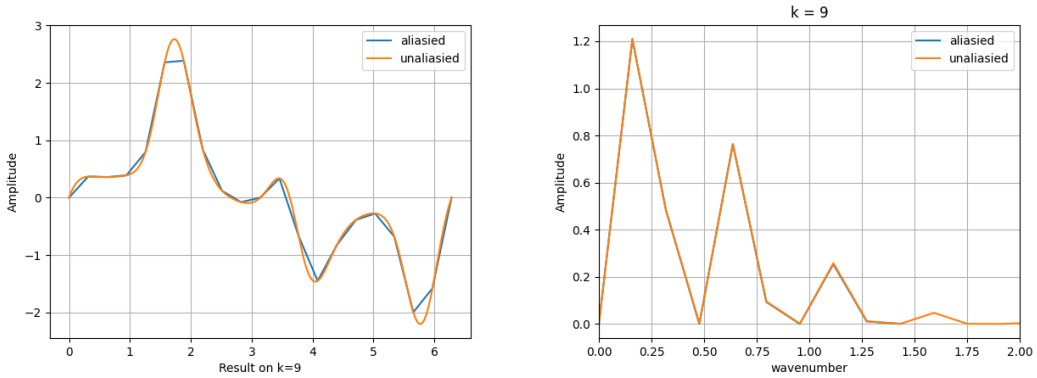


Figure 31: k=9, result comparison and frequency comparison

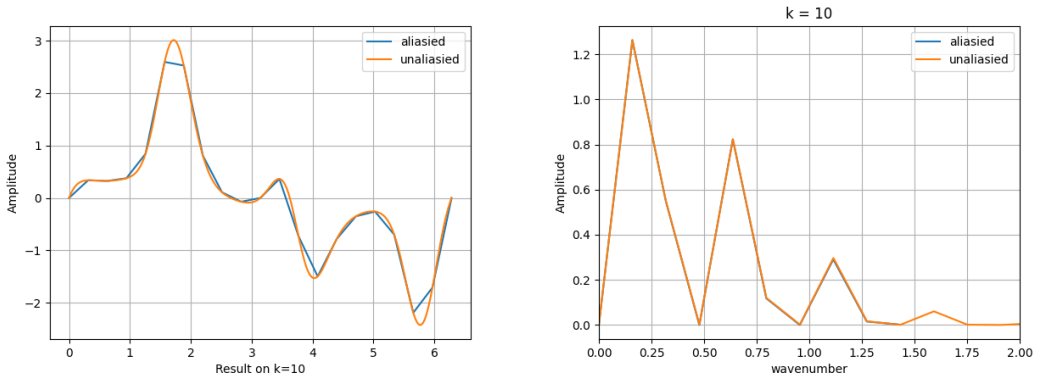


Figure 32: k=10, result comparison and frequency comparison

We could noticed that the amplitude difference of low wave number between the raw grid (Fully aliased) and finer grid (un-aliased) is not obvious in first 10 iteration, which may because the non-liner effect of the control equation is not obvious:

For each iteration, the wavenumber increase in not as much as Function 1:
For our iteration formula for Function 1 and Function 2, can be shown as:

$$u_{new} = u - \Delta t f(u)$$

in Function 1:

$$f_1(u) = u^2 == \left(\sum A_k \sin(kx)\right)^2 == B_k \cos(2kx) + \sum \text{others}$$

In Function 2:

$$f_2(u) = \sin(3x) \cdot u == \sin(3x)\left(\sum A_k \sin(kx)\right) == \sum A_k \left[\sin((3-k)x) - \sin((3+k)x)\right]$$

Compare their wavenumber generate due to the nonlinear term in their equation, it could found in Function 1, wavenumbers are generated in order of 2, where in Function 2, wavenumbers are only generated increase 3 each iteration, which could explain why is not obvious for the aliasing effect in first 10 steps iteration of Function 2.

However, if we run iteration long enough (100 iteration steps), the aliasing effect is still obvious:

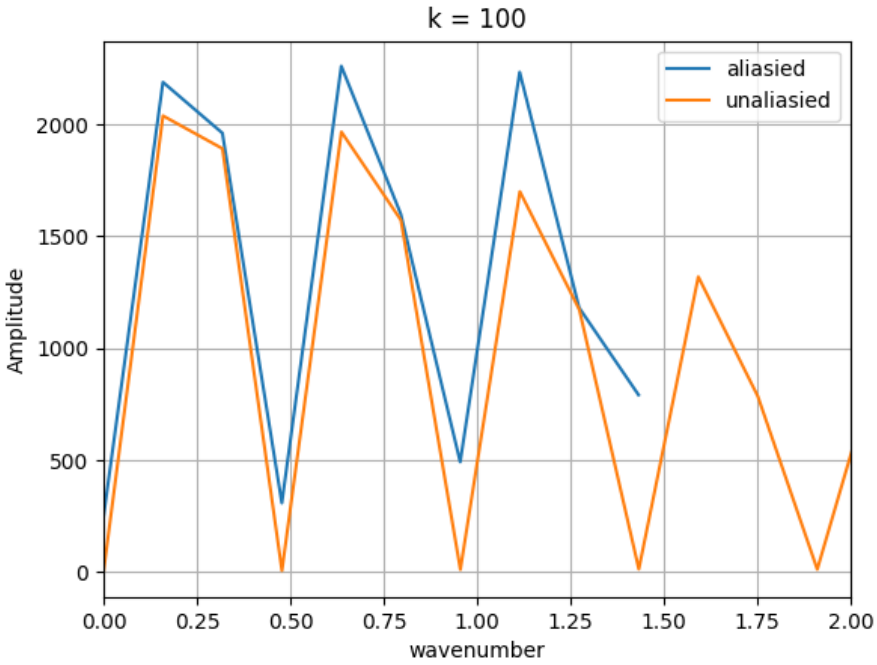


Figure 33: k=10, result comparison and frequency comparison

Appendix

Listing 1: Problem1, Py code for Solvers

```
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import copy

class UPwind1st_Solver:
    def __init__(self, dx, dt, x_max, t_max,m):
        self.dx = dx
        self.dt = dt
        self.x_max = x_max
        self.t_max = t_max


        self.m = m  #added condition input

    def grid_generate(self):
        self.i_max = int(self.x_max/self.dx)
        self.n_max = int(self.t_max/self.dt)

        self.u = np.zeros(( self.i_max  ))


    def IC(self,m):
        for i in range(0, self.i_max):
            self.u[i] = math.sin(m*i*self.dx)

    def Iteration_Formula(self, u_W, u):
        u_new = u + (self.dt/(self.dx))*(u_W-u)
        return u_new

    def Iterative(self):
        u_next = copy.deepcopy(self.u)

        for n in range(1, self.n_max+1):
            for i in range(0,self.i_max):
                u_next[i] = self.Iteration_Formula(self.u[i-1], self.u[i])
            self.u[:] = u_next[:]
        self.u_Full = self.getTHElastBACK(self.u)
        return self.u_Full


    def getTHElastBACK(self,u_lost):
        u_Fullget = np.copy(u_lost)
        u_Fullget = np.append(u_Fullget, [u_lost[0]])
        return u_Fullget

    def plot_result(self):
        x = np.arange(0,self.i_max+1)
        plt.plot(x*(2*math.pi/20),self.u_Full)
        plt.show()


    def exactsoln(self):
        u_e = np.zeros(self.i_max+1)
        for i in range(self.i_max+1):
            x = i * self.dx
            u_e[i] = math.sin(self.m * (x - self.t_max))
        return u_e

    def plot_result_compare(self):
        x = np.arange(0,self.i_max+1)
        u_e =  self.exactsoln()
        plt.plot(x*(2*math.pi/20),self.u_Full)
        plt.plot(x*(2*math.pi/20),u_e)
```

```
        print(u_e - self.u_Full)

        plt.show()


class UPwind2nd_Solver(UPwind1st_Solver):
    def Iteration_Formula(self, u, u_W, u_WW):
        u_new = u - (3*u -4*u_W +u_WW)*self.dt/(2*self.dx)

        return u_new

    def Iterative(self):
        u_next = np.copy(self.u)

        for n in range(1, self.n_max+1):
            for i in range(0,self.i_max):
                u_next[i] = self.Iteration_Formula(self.u[i],self.u[i-1],self.u[i-2])
            self.u[:] = u_next[:]
        self.u_Full = self.getTHElastBACK(self.u)
        return self.u_Full


class UPwind3rd_Solver(UPwind1st_Solver):
    def Iteration_Formula(self, u, u_W, u_WW,u_WWW):
        u_new = u - ((11/6)*u -3*u_W +(3/2)*u_WW -(1/3)*u_WWW)*self.dt/(self.dx)

        return u_new

    def Iterative(self):
        u_next = np.copy(self.u)

        for n in range(1, self.n_max+1):
            for i in range(0,self.i_max):
                u_next[i] = self.Iteration_Formula(self.u[i],self.u[i-1],self.u[i-2],self.u[i-3])
            self.u[:] = np.copy(u_next)

        self.u_Full = self.getTHElastBACK(np.copy(self.u))
        return self.u_Full



class UPwind3rdBias_Solver(UPwind1st_Solver):
    def Iteration_Formula(self, u, u_W, u_WW,u_E):
        u_new = u - ((3)*u -6*u_W +1*u_WW +2*u_E)*self.dt/(6*self.dx)

        return u_new

    def Iterative(self):
        u_next = copy.deepcopy(self.u)

        for n in range(1, self.n_max+1):
            for i in range(0,self.i_max):
                u_next[i] = self.Iteration_Formula(self.u[i],self.u[i-1],self.u[i-2],self.u[(i+1)%(self.i_
            self.u[:] = u_next[:]
        self.u_Full = self.getTHElastBACK(self.u)
        return self.u_Full



class Total_Compare(UPwind1st_Solver):
    def __init__(self, dx, dt, x_max, t_max, m, UPwind1, UPwind2, UPwind3, UPwind3b):
        super().__init__(dx, dt, x_max, t_max, m)
        self.UPwind1 = UPwind1
        self.UPwind2 = UPwind2
        self.UPwind3 = UPwind3
        self.UPwind3b = UPwind3b

    def plot_result_compare(self):
        x = np.arange(0,self.i_max+1)

        u_e =  self.exactsoln()
```

```
        plt.plot(x*(2*math.pi/20),self.UPwind1, label = "UPwind1")
        plt.plot(x*(2*math.pi/20),self.UPwind2, label = "UPwind2")
        plt.plot(x*(2*math.pi/20),self.UPwind3, label = "UPwind3")
        plt.plot(x*(2*math.pi/20),self.UPwind3b, label = "UPwind3b")
        plt.plot(x*(2*math.pi/20),u_e, label = "Exact")

        plt.title('Solutions at t_max=%f'%self.t_max)
        plt.xlabel('x')
        plt.ylabel('u')
        plt.legend()
        plt.show()


    def Error_Compute(self, u, u_e):
        Error = np.zeros(self.i_max+1)
        for i in range(0,self.i_max+1):
                Error[i]+= abs(u[i]-u_e[i])
        return Error

    def plot_Error_compare(self):
        x = np.arange(0,self.i_max+1)
        u_e =  self.exactsoln()

        EUPwind1 = self.Error_Compute(self.UPwind1, u_e)
        EUPwind2 = self.Error_Compute(self.UPwind2, u_e)
        EUPwind3 = self.Error_Compute(self.UPwind3, u_e)
        EUPwind3b = self.Error_Compute(self.UPwind3b, u_e)

        plt.plot(x*(2*math.pi/20),EUPwind1, label = "UPwind1")
        plt.plot(x*(2*math.pi/20),EUPwind2, label = "UPwind2")
        plt.plot(x*(2*math.pi/20),EUPwind3, label = "UPwind3")
        plt.plot(x*(2*math.pi/20),EUPwind3b, label = "UPwind3b")

        plt.title('Solutions at t_max=%f'%self.t_max)
        plt.xlabel('x')
        plt.ylabel('u')
        plt.legend()
        plt.show()


def main():

    x_max = 2*math.pi
    t_max = .1

    dx = 2*math.pi/20
    dt = 0.001

    m = 2


    ####### 2st Upwind Iteration
    upwind1st = UPwind1st_Solver(dx, dt, x_max, t_max,m)
    upwind1st.grid_generate()
    upwind1st.IC(m)
    U1 = upwind1st.Iterative()

    ####### 2nd Upwind Iteration
    upwind2nd = UPwind2nd_Solver(dx, dt, x_max, t_max,m)
    upwind2nd.grid_generate()
    upwind2nd.IC(m)
    U2 = upwind2nd.Iterative()


    ####### 3rd Upwind Iteration
    upwind3 = UPwind3rd_Solver(dx, dt, x_max, t_max, m)
    upwind3.grid_generate()
    upwind3.IC(m)
    U3 = upwind3.Iterative()


    ####### 3rd bias Iteration
    upwind3b = UPwind3rdBias_Solver(dx, dt, x_max, t_max,m)
```

```
        upwind3b.grid_generate()
        upwind3b.IC(m)
        U3b = upwind3b.Iterative()


        #### Total Compare
        Total = Total_Compare(dx, dt, x_max, t_max, m, U1, U2, U3, U3b)
        Total.grid_generate()
        Total.IC(m)
        Total.plot_result_compare()
        #Total.plot_Error_compare()


    if __name__ == '__main__':
        main()
```

Listing 2: Problem1.2, Py code for modified wavenumbers

```
import math

import numpy as np


import matplotlib.pyplot as plt
import matplotlib as mpl



def main():

    x_max = 2*math.pi
    t_max = 60

    dx = 2*math.pi/20
    dt = 0.001

    m = 2

    def sin(x):
        return np.sin(x)

    def cos(x):
        return np.cos(x)

    x = np.arange(0,math.pi+dx, dx)
    #k = 2*math.pi/dx
    y_exact = x

    y_1Re = sin(x)/dx
    y_1Im = (cos(x)-1)/dx

    y_2Re = (4*sin(x)-sin(2*x))/(2*dx)
    y_2Im = (4*cos(x)-cos(2*x)-3)/(2*dx)

    y_3Re = (3*sin(x)-3/2*(sin(2*x))+1/3*sin(3*x))/dx
    y_3Im = (3*cos(x)-3/2*(cos(2*x))+1/3*cos(3*x)-11/6)/dx


    y_3bRe = (6*sin(x)-sin(2*x)+2*sin(x))/(6*dx)
    y_3bIm = (6*cos(x)-cos(2*x)-2*cos(x)-3)/(6*dx)


    plt.plot(x, y_exact, color = 'blue', label = 'Exact k')

    plt.plot(x, y_1Re, color = (190/255,184/255,220/255), label = '1st Re k',linewidth = 2)
    plt.plot(x, y_1Im, color = (190/255,184/255,220/255), label = '1st Im k',linewidth = 2)

    plt.plot(x, y_2Re, color = (255/255,190/255,222/255), label = '2st Re k',linewidth = 2)
    plt.plot(x, y_2Im, color = (255/255,190/255,222/255), label = '2st Im k',linewidth = 2)
```

```
    plt.plot(x, y_3Re, color = (250/255,127/255,111/255), label = '3st Re k',linewidth = 2)
    plt.plot(x, y_3Im, color = (250/255,127/255,111/255), label = '3st Im k',linewidth = 2)

    plt.plot(x, y_3bRe, color = (130/255,176/255,210/255), label = '3st bias Re k',linewidth = 2)
    plt.plot(x, y_3bIm, color = (130/255,176/255,210/255), label = '3st bias Im k',linewidth = 2)

    #plt.annotate('1st Re k', xy=(3, y_1Re[3]), xytext=(3 +0.1, y_1Re[3]+0.1))
    plt.xlabel('kdx')
    plt.ylabel('Modified k')
    plt.legend()
    plt.show()



if __name__ == '__main__':
    main()
```

Listing 3: Problem2.1, Function 1 analysis

```
import math

import numpy as np

import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy.fftpack import fft, fftfreq

import copy



class UPwind1st_Solver:
    def __init__(self, dx, dt, x_max, t_max,m):
        self.dx = dx
        self.dt = dt
        self.x_max = x_max
        self.t_max = t_max
        self.m = m  #added condition input

    def grid_generate(self):
        self.i_max = int(self.x_max/self.dx)
        self.n_max = int(self.t_max/self.dt)

        self.u = np.zeros(( self.i_max  ))


    def IC(self,m):
        for i in range(0, self.i_max):
            self.u[i] = math.sin(m*i*self.dx)

    def Iteration_Formula(self, u_W, u):
        u_new = u + (self.dt/(self.dx))*(u_W-u)
        return u_new

    def Iterative(self):
        u_next = copy.deepcopy(self.u)

        for n in range(1, self.n_max+1):
            for i in range(0,self.i_max):
                u_next[i] = self.Iteration_Formula(self.u[i-1], self.u[i])
            self.u[:] = u_next[:]
        self.u_Full = self.getTHElastBACK(self.u)
        return self.u_Full


    def getTHElastBACK(self,u_lost):
        u_Fullget = np.copy(u_lost)
        u_Fullget = np.append(u_Fullget, [u_lost[0]])
        return u_Fullget
```

```
    def plot_result(self):
        x = np.arange(0,self.i_max+1)
        print(self.u_Full) #TESTING ##############DELETE AFT TEST
        plt.plot(x*(2*math.pi/20),self.u_Full)
        plt.show()


    def exactsoln(self):
        u_e = np.zeros(self.i_max+1)
        for i in range(self.i_max+1):
            x = i * self.dx
            u_e[i] = math.sin(self.m * (x - self.t_max))
        return u_e

    def plot_result_compare(self):
        x = np.arange(0,self.i_max+1)
        u_e =  self.exactsoln()

        plt.plot(x*(2*math.pi/20),self.u_Full)
        plt.plot(x*(2*math.pi/20),u_e)
        print(u_e - self.u_Full)
        plt.show()




class Fn1(UPwind1st_Solver):

    def IC(self):
        for i in range(0, self.i_max):
            self.u[i] = math.sin(i*self.dx) +0.5*math.sin(4*i*self.dx)

    def Iteration_Formula(self, u):
        u_new = u - self.dt*(u**2)
        return u_new

    def Iterative(self):
        u_next = copy.deepcopy(self.u)

        for n in range(1, self.n_max+1):
            for i in range(0,self.i_max):
                u_next[i] = self.Iteration_Formula(self.u[i])
            self.u[:] = u_next[:]
        self.u_Full = self.getTHElastBACK(self.u)
        return self.u_Full


class Fn2(UPwind1st_Solver):

    def IC(self):
        for i in range(0, self.i_max):
            self.u[i] = math.sin(i*self.dx) +0.5*math.sin(4*i*self.dx)

    def Iteration_Formula(self, u,i):
        u_new = u - self.dt * math.sin(3*i*self.dx)*u
        return u_new

    def Iterative(self):
        u_next = copy.deepcopy(self.u)

        for n in range(1, self.n_max+1):
            for i in range(0,self.i_max):
                u_next[i] = self.Iteration_Formula(self.u[i],i)
            self.u[:] = u_next[:]
        self.u_Full = self.getTHElastBACK(self.u)
        return self.u_Full
```

```
class Post_op():
    def __init__(self, Function):
        self.Function = Function
    def append(self):
        return 0


    def FFT(self, k, u_analysis, delta_x):
        dx = copy.deepcopy(delta_x)
        u = copy.deepcopy(u_analysis)
        N = self.Function.i_max
        yf = fft(u[:-1])
        kx = fftfreq(N, dx)[:N // 2]
        y_plot = 2.0 / N * np.abs(yf[:N // 2])
        return kx, y_plot




def Aliasing_grid_rearrange(dx, dt):
    t = copy.deepcopy(dt)
    x = copy.deepcopy(dx)
    t = t
    x= x/100
    return x, t



def resultcompare(k, x_max, dx, dxa, y1, y1a):
    i_max = int(x_max/dx)
    i_maxa = int(x_max/dxa)
    x = np.arange(0,i_max+1)
    xa = np.arange(0,i_maxa+1)


    plt.plot(x*dx, y1, label = 'aliasied')
    plt.plot(xa*dxa, y1a, label = 'unaliasied')
    plt.grid()
    plt.xlabel('Result on k=%d'%k)
    plt.ylabel('Amplitude')
    plt.legend()
    plt.show()









def main():

    x_max = 2*math.pi
    dx = 2*math.pi/20
    dt = 0.1



    k = 10                      ## Contorl Iteration counts

    t_max = dt*k
    m = 2


    Function1 = copy.deepcopy(Fn1(dx, dt, x_max, t_max, m))
    Function1.grid_generate()
    Function1.IC()
    u1 = Function1.Iterative()
```

```
    Post1 = Post_op(Function1)
    xf, y1  = Post1.FFT(k, u1, dx)



    dxa, dta = Aliasing_grid_rearrange(dx, dt)

    Function1a = copy.deepcopy(Fn1(dxa, dta, x_max, t_max, m))
    Function1a.grid_generate()
    Function1a.IC()

    u1a = Function1a.Iterative()
    Post1a = Post_op(Function1a)
    xfa, y1a  = Post1a.FFT(k, u1a, dxa)
    resultcompare(k, x_max, dx, dxa, u1, u1a) # Plot compare in real space



###################### Plot compare in frequence space #############

    plt.plot(xf, y1, label = 'aliasied')
    plt.plot(xfa, y1a, label = 'unaliasied')
    plt.grid()
    plt.xlim(0,2)
    plt.xlabel('wavenumber')
    plt.ylabel('Amplitude')
    plt.legend()
    plt.title('k = %d'%k)
    plt.show()

######################  ###################### ######################



if __name__ == '__main__':
    main()
```

Listing 4: Problem2.2, Function 2 analysis

```
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy.fftpack import fft, fftfreq
import copy



class UPwind1st_Solver:
    def __init__(self, dx, dt, x_max, t_max,m):
        self.dx = dx
        self.dt = dt
        self.x_max = x_max
        self.t_max = t_max
        self.m = m  #added condition input

    def grid_generate(self):
        self.i_max = int(self.x_max/self.dx)
        self.n_max = int(self.t_max/self.dt)

        self.u = np.zeros(( self.i_max  ))


    def IC(self,m):
        for i in range(0, self.i_max):
            self.u[i] = math.sin(m*i*self.dx)

    def Iteration_Formula(self, u_W, u):
        u_new = u + (self.dt/(self.dx))*(u_W-u)
        return u_new
```

```
    def Iterative(self):
        u_next = copy.deepcopy(self.u)

        for n in range(1, self.n_max+1):
            for i in range(0,self.i_max):
                u_next[i] = self.Iteration_Formula(self.u[i-1], self.u[i])
            self.u[:] = u_next[:]
        self.u_Full = self.getTHElastBACK(self.u)
        return self.u_Full


    def getTHElastBACK(self,u_lost):
        u_Fullget = np.copy(u_lost)
        u_Fullget = np.append(u_Fullget, [u_lost[0]])
        return u_Fullget

    def plot_result(self):
        x = np.arange(0,self.i_max+1)
        print(self.u_Full) #TESTING ##############DELETE AFT TEST
        plt.plot(x*(2*math.pi/20),self.u_Full)
        plt.show()


    def exactsoln(self):
        u_e = np.zeros(self.i_max+1)
        for i in range(self.i_max+1):
            x = i * self.dx
            u_e[i] = math.sin(self.m * (x - self.t_max))
        return u_e

    def plot_result_compare(self):
        x = np.arange(0,self.i_max+1)
        u_e =  self.exactsoln()

        plt.plot(x*(2*math.pi/20),self.u_Full)
        plt.plot(x*(2*math.pi/20),u_e)
        print(u_e - self.u_Full)
        plt.show()


class Fn1(UPwind1st_Solver):

    def IC(self):
        for i in range(0, self.i_max):
            self.u[i] = math.sin(i*self.dx) +0.5*math.sin(4*i*self.dx)

    def Iteration_Formula(self, u):
        u_new = u - self.dt*(u**2)
        return u_new

    def Iterative(self):
        u_next = copy.deepcopy(self.u)

        for n in range(1, self.n_max+1):
            for i in range(0,self.i_max):
                u_next[i] = self.Iteration_Formula(self.u[i])
            self.u[:] = u_next[:]
        self.u_Full = self.getTHElastBACK(self.u)
        return self.u_Full



class Fn2(UPwind1st_Solver):

    def IC(self):
        for i in range(0, self.i_max):
            self.u[i] = math.sin(i*self.dx) +0.5*math.sin(4*i*self.dx)

    def Iteration_Formula(self, u,i):
        u_new = u - self.dt * math.sin(3*i*self.dx)*u
        return u_new
```

```
    def Iterative(self):
        u_next = copy.deepcopy(self.u)

        for n in range(1, self.n_max+1):
            for i in range(0,self.i_max):
                u_next[i] = self.Iteration_Formula(self.u[i],i)
            self.u[:] = u_next[:]
        self.u_Full = self.getTHElastBACK(self.u)
        return self.u_Full



class Post_op():
    def __init__(self, Function):
        self.Function = Function
    def append(self):
        return 0


    def FFT(self, k, u_analysis, delta_x):
        dx = copy.deepcopy(delta_x)
        u = copy.deepcopy(u_analysis)
        N = self.Function.i_max
        yf = fft(u[:-1])
        kx = fftfreq(N, dx)[:N // 2]

        y_plot = 2.0 / N * np.abs(yf[:N // 2])

        return kx, y_plot




def Aliasing_grid_rearrange(dx, dt):
    t = copy.deepcopy(dt)
    x = copy.deepcopy(dx)
    t = t
    x= x/100
    return x, t



def resultcompare(k, x_max, dx, dxa, y1, y1a):
    i_max = int(x_max/dx)
    i_maxa = int(x_max/dxa)
    x = np.arange(0,i_max+1)
    xa = np.arange(0,i_maxa+1)
    plt.plot(x*dx, y1, label = 'aliasied')
    plt.plot(xa*dxa, y1a, label = 'unaliasied')
    plt.grid()
    plt.xlabel('Result on k=%d'%k)
    plt.ylabel('Amplitude')
    plt.legend()
    plt.show()


def main():
    x_max = 2*math.pi
    dx = 2*math.pi/20
    dt = 0.1

    k = 100                         ## Contorl Iteration counts

    t_max = dt*k
    m = 2

    Function1 = copy.deepcopy(Fn2(dx, dt, x_max, t_max, m))
    Function1.grid_generate()
    Function1.IC()


    u1 = Function1.Iterative()
```

```
    Post1 = Post_op(Function1)
    xf, y1  = Post1.FFT(k, u1, dx)

    dxa, dta = Aliasing_grid_rearrange(dx, dt)

    Function1a = copy.deepcopy(Fn2(dxa, dta, x_max, t_max, m))
    Function1a.grid_generate()
    Function1a.IC()
    u1a = Function1a.Iterative()
    Post1a = Post_op(Function1a)
    xfa, y1a  = Post1a.FFT(k, u1a, dxa)


    #resultcompare(k, x_max, dx, dxa, u1, u1a) # Plot compare in real space

###################### Plot compare in frequence space #############
    plt.plot(xf, y1, label = 'aliasied')
    plt.plot(xfa, y1a, label = 'unaliasied')

    plt.grid()
    plt.xlim(0,2)

    plt.xlabel('wavenumber')
    plt.ylabel('Amplitude')
    plt.legend()
    plt.title('k = %d'%k)
    plt.show()

######################  ###################### ######################

if __name__ == '__main__':
    main()
```