Ch 3. Bounded Grids

Solving homework using Python

Q. Reproduction of Matlab Code to Python

2020.10.14(전산천문학)

202050135 홍지민

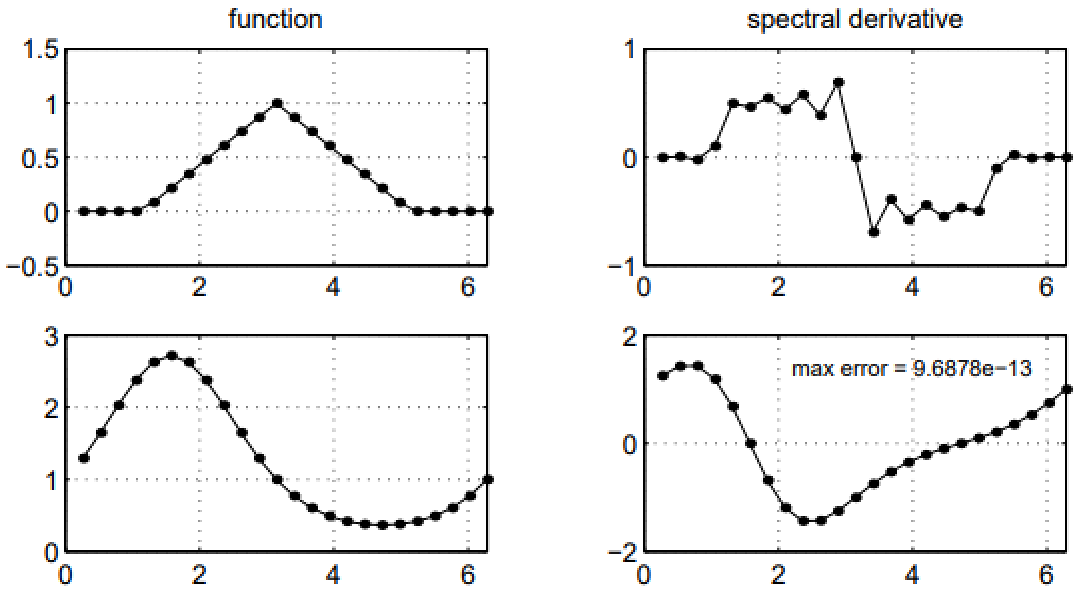
Program1. Matlab Code

1. Implement Program 4 and produce a plot similar to Output 4.

Program 4

```
% p4.m - periodic spectral differentiation

% Set up grid and differentiation matrix:
  N = 24; h = 2*pi/N; x = h*(1:N)';
  column = [0 .5*(-1).^(1:N-1).*cot((1:N-1)*h/2)]';
  D = toeplitz(column,column([1 N:-1:2]));

% Differentiation of a hat function:
  v = max(0,1-abs(x-pi)/2); clf
  subplot(3,2,1), plot(x,v,'.-','markersize',13)
  axis([0 2*pi -.5 1.5]), grid on, title('function')
  subplot(3,2,2), plot(x,D*v,'.-','markersize',13)
  axis([0 2*pi -1 1]), grid on, title('spectral derivative')

% Differentiation of exp(sin(x)):
  v = exp(sin(x)); vprime = cos(x).*v;
  subplot(3,2,3), plot(x,v,'.-','markersize',13)
  axis([0 2*pi 0 3]), grid on
  subplot(3,2,4), plot(x,D*v,'.-','markersize',13)
  axis([0 2*pi -2 2]), grid on
  error = norm(D*v-vprime,inf);
  text(2.2,1.4,['max error = ' num2str(error)])
```

Output 4



Program1. Python Code

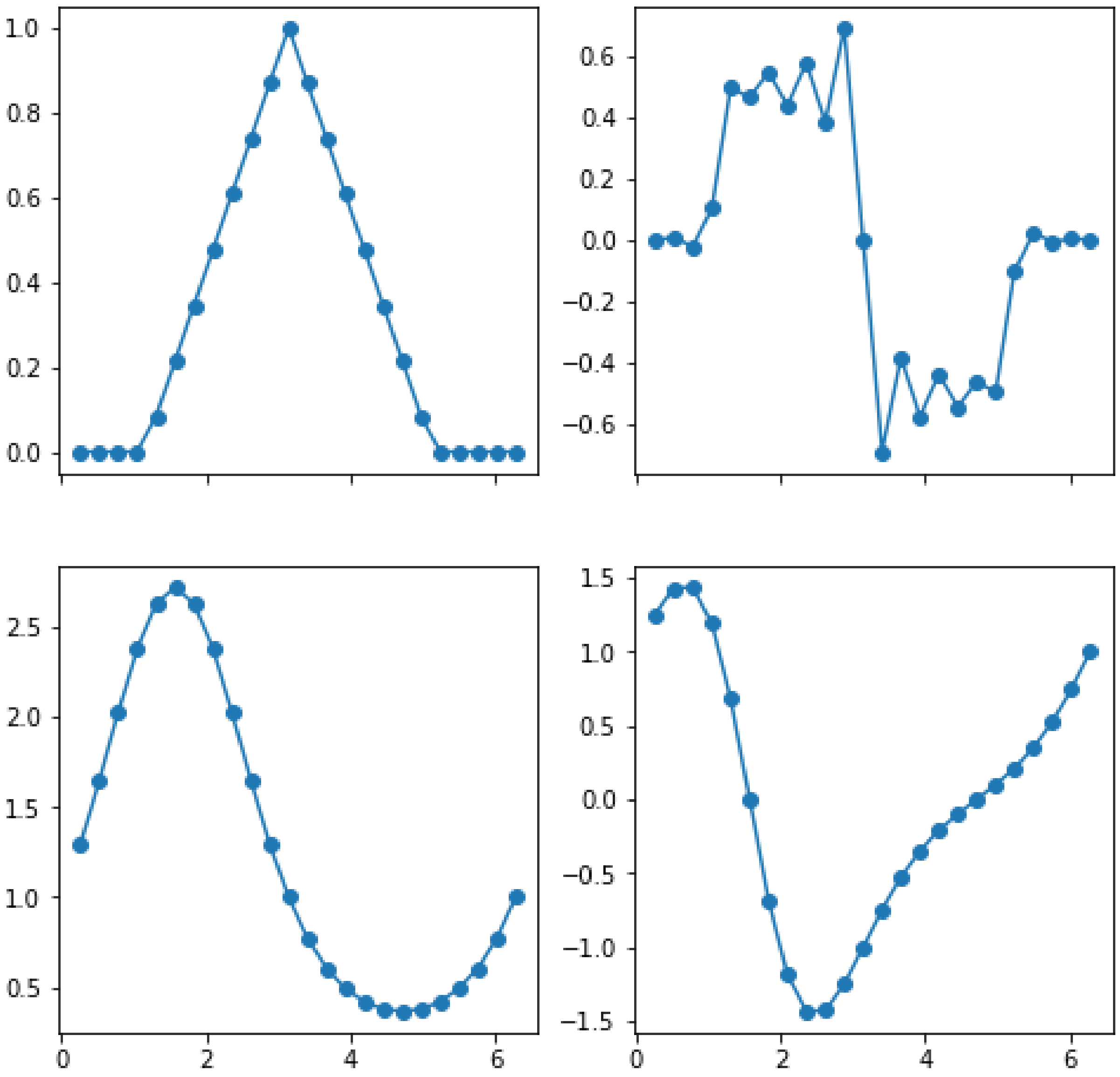
```python
import numpy as np
from matplotlib import pyplot as plt
from scipy.linalg import toeplitz as tp

N = 24
h = 2*np.pi / N

fig, ax = plt.subplots(2,2,sharex=True,sharey=False,figsize=(8,8))

x = np.matrix(np.linspace(h,h*N,N,endpoint=True)).T
column = 0.5*(-1)**np.arange(1,N,1)*(1/np.tan(np.arange(1,N,1)*h/2))
column = np.insert(column,0,0)
column = np.matrix(column).T

column_1 = [0]

for i in np.arange(N-1,1-1,-1):
  column_1.append(column[i])

D = tp(column,column_1)

v = 1 - abs(x - np.pi)/2
v[v < 0] = 0
ax[0,0].plot(x,v,marker='o')
ax[0,1].plot(x,np.dot(D,v),marker='o')

v = np.power(np.e,np.sin(x))
vprime = np.cos(x)*np.matrix(v).T
ax[1,0].plot(x,v,marker='o')
ax[1,1].plot(x,np.dot(D,v),marker='o')
```

Add zero to the beginning of an array

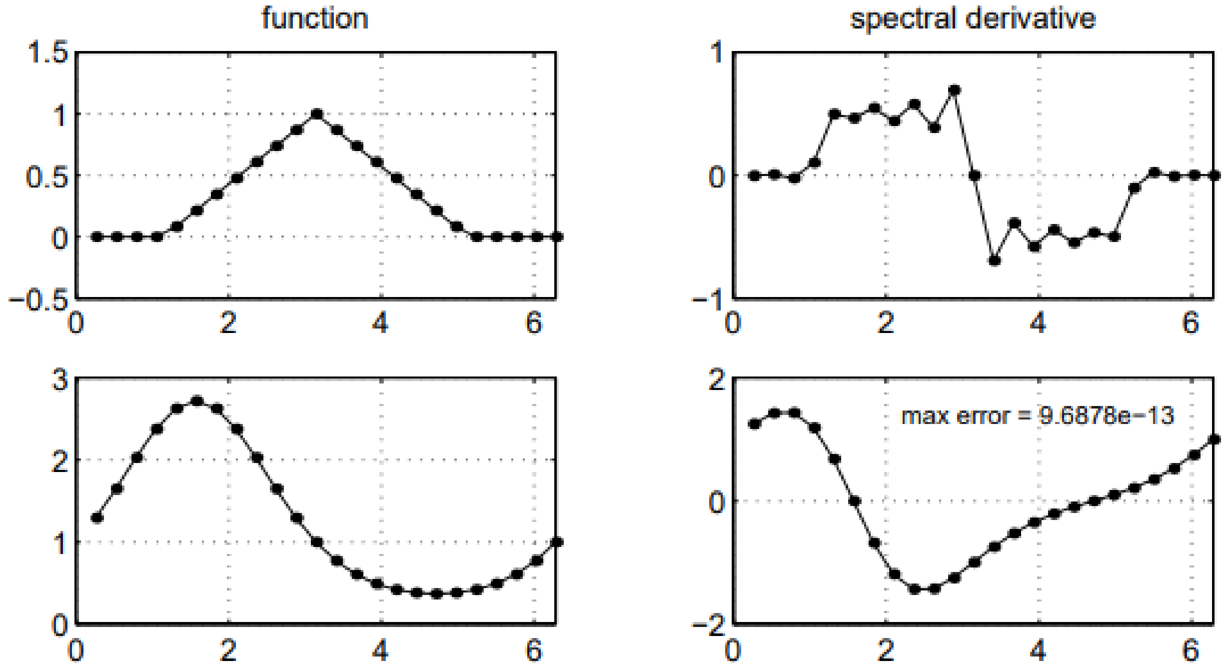
Toeplitz matrix

$$D_N = \begin{pmatrix} 0 & & & -\frac{1}{2}\cot\frac{1h}{2} \\ -\frac{1}{2}\cot\frac{1h}{2} & \ddots & \ddots & \frac{1}{2}\cot\frac{2h}{2} \\ \frac{1}{2}\cot\frac{2h}{2} & \ddots & & -\frac{1}{2}\cot\frac{3h}{2} \\ -\frac{1}{2}\cot\frac{3h}{2} & & \ddots & \vdots \\ \vdots & \ddots & \ddots & \frac{1}{2}\cot\frac{1h}{2} \\ \frac{1}{2}\cot\frac{1h}{2} & & & 0 \end{pmatrix}.$$

In Python



Output 4



In Matlab

Program2. Python Code

2. Implement Program 5 and produce a plot similar to Output 5.

```
fig, ax = plt.subplots(2,2,sharex=True,sharey=False,figsize=(8,8))

from scipy.fft import fft, ifft

x = np.matrix(h * np.arange(1,N+1,1)).T
v = 1 - abs(x - np.pi)/2
v[v < 0] = 0
v = np.matrix(v).T
v_hat = fft(v)

w_hat_1 = np.arange(0,N/2)
w_hat_2 = np.arange(-N/2+1,0)
w_hat_A = []
w_hat_A = np.append(w_hat_A,w_hat_1)
w_hat_A = np.append(w_hat_A,0)
w_hat_A = np.append(w_hat_A,w_hat_2)
w_hat = (1j * w_hat_A * v_hat)

w = np.real(ifft(w_hat))

ax[0,0].plot(x,v.T,marker='o')
ax[0,1].plot(x,w.T,marker='o')

v = np.power(np.e,np.sin(x))
v = np.matrix(v).T
vprime = np.cos(x) * v
v_hat = fft(v)
w_hat = (1j * w_hat_A * v_hat)
w = np.real(ifft(w_hat))

ax[1,0].plot(x,v.T,marker='o')
ax[1,1].plot(x,w.T,marker='o')
```

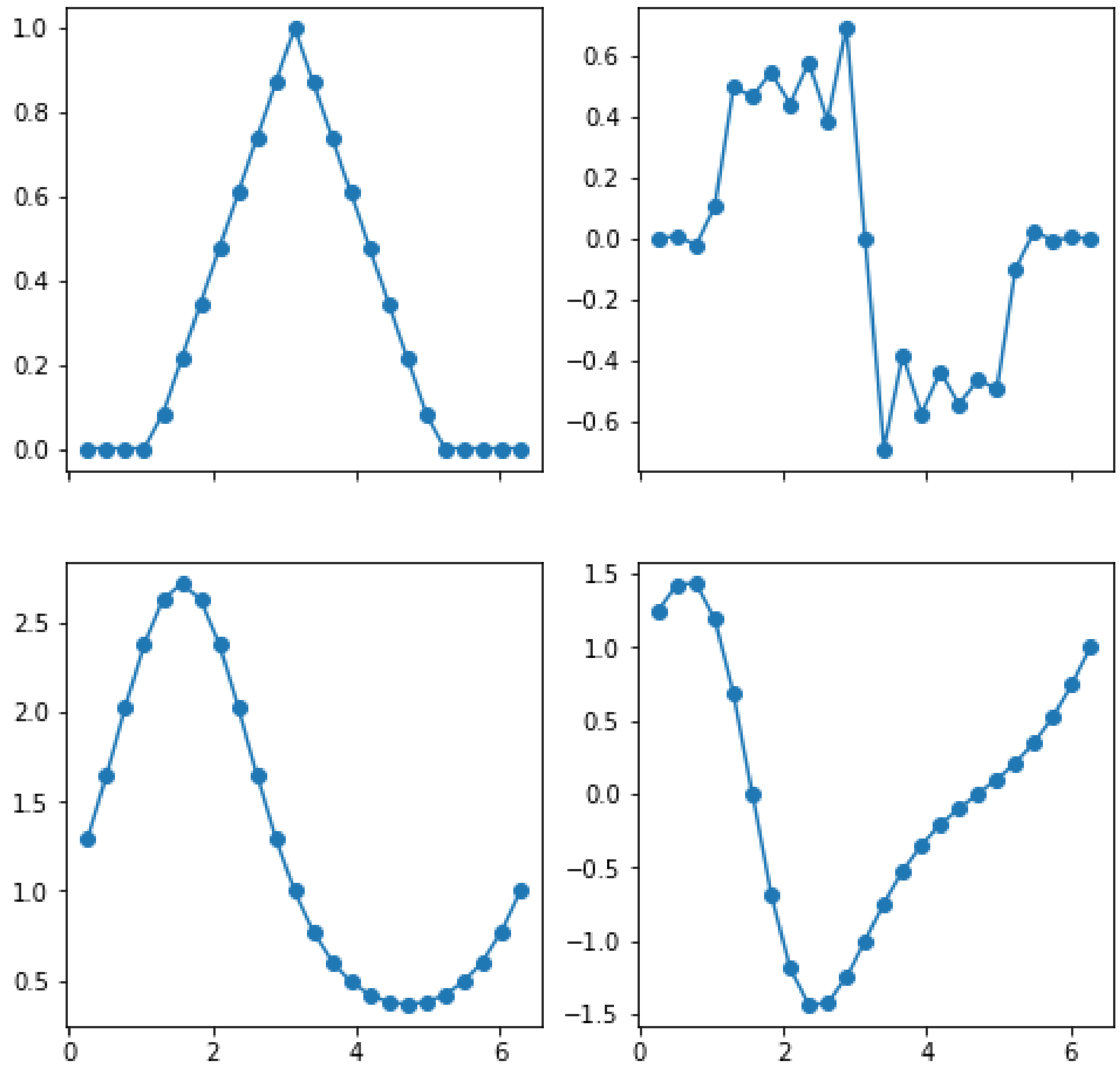
Fourier transform

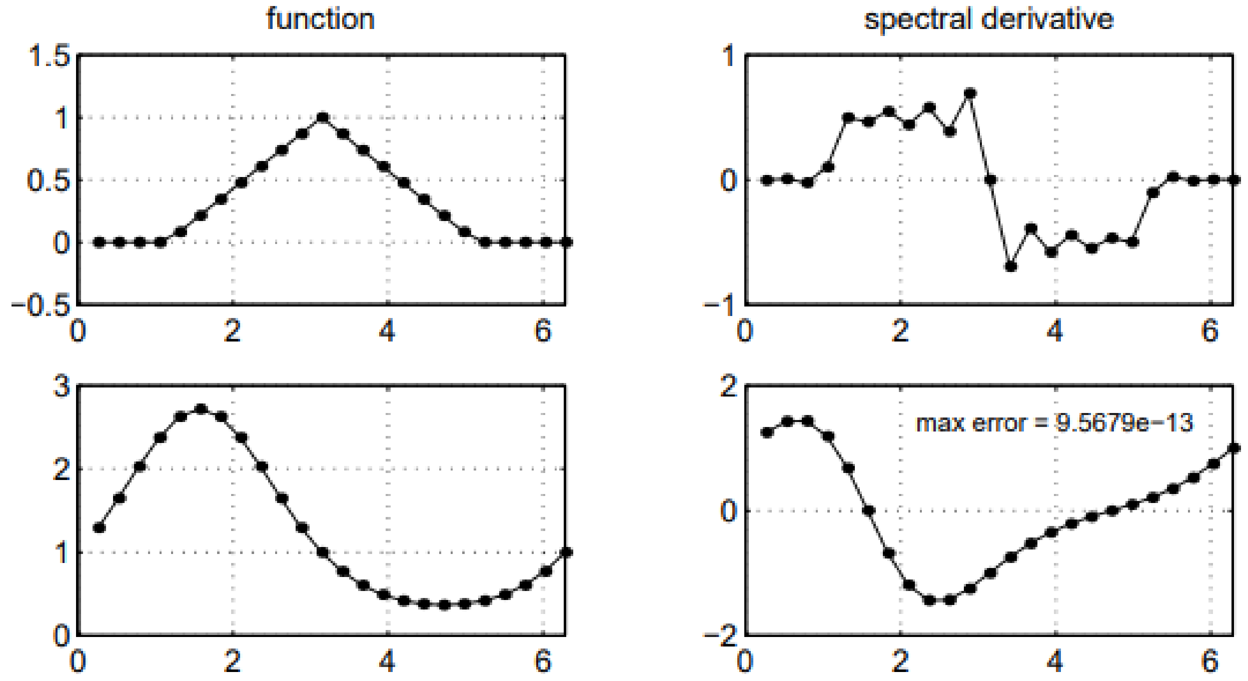
I couldn't think of a simple method, so I proceeded one by one.

```
w_hat = 1i*[0:N/2-1 0 -N/2+1:-1] .* v_hat;
```

Inverse Fourier transform

In Python





In Matlab

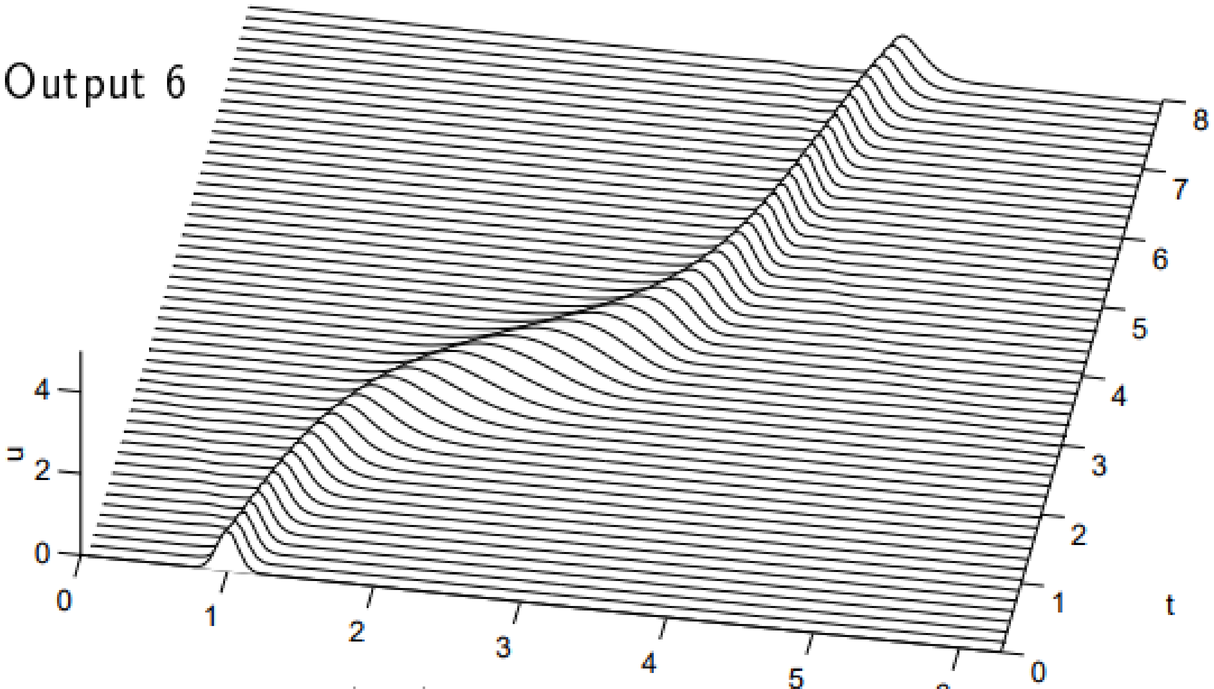
Program3. Matlab Code

3. Implement Program 6 and produce a plot similar to Output 6.

Program 6

```
% p6.m - variable coefficient wave equation

% Grid, variable coefficient, and initial data:
  N = 128; h = 2*pi/N; x = h*(1:N); t = 0; dt = h/4;
  c = .2 + sin(x-1).^2;
  v = exp(-100*(x-1).^2); vold = exp(-100*(x-.2*dt-1).^2);

% Time-stepping by leap frog formula:
  tmax = 8; tplot = .15; clf, drawnow
  plotgap = round(tplot/dt); dt = tplot/plotgap;
  nplots = round(tmax/tplot);
  data = [v; zeros(nplots,N)]; tdata = t;
  for i = 1:nplots
    for n = 1:plotgap
      t = t+dt;
      v_hat = fft(v);
      w_hat = 1i*[0:N/2-1 0 -N/2+1:-1] .* v_hat;
      w = real(ifft(w_hat));
      vnew = vold - 2*dt*c.*w; vold = v; v = vnew;
    end
    data(i+1,:) = v; tdata = [tdata; t];
  end
  waterfall(x,tdata,data), view(10,70), colormap([0 0 0])
  axis([0 2*pi 0 tmax 0 5]), ylabel t, zlabel u, grid off
```

Output 6



Output 6: Solution of (3.13).

Program3. Python Code

```
N = 2**8 # 128=2**7 , 256=2**8
h = 2*np.pi/N
x = h*np.arange(1,N+1)
t = 0
dt = h/4

c = 0.2 + np.sin(x-1)**2
v = np.power(np.e,-100 * (x-1) ** 2)
vold = np.power(np.e,-100 * (x-0.2*dt-1) ** 2)

tmax = 8
tplot = 0.15
plotgap = round(tplot/dt)
dt = tplot/plotgap
nplots = round(tmax/tplot)
data = np.matrix(v)
tdata = [t]

w_hat_1 = np.arange(0,N/2)
w_hat_2 = np.arange(-N/2+1,0)
w_hat_A = []
w_hat_A = np.append(w_hat_A,w_hat_1)
w_hat_A = np.append(w_hat_A,0)
w_hat_A = np.append(w_hat_A,w_hat_2)
```

This function is represented by rounding the decimal point.

```
plotgap = round(tplot/dt); dt = tplot/plotgap;
nplots = round(tmax/tplot);
```

I couldn't think of a simple method, so I proceeded one by one.

```
w_hat = 1i*[0:N/2-1 0 -N/2+1:-1] .* v_hat;
```

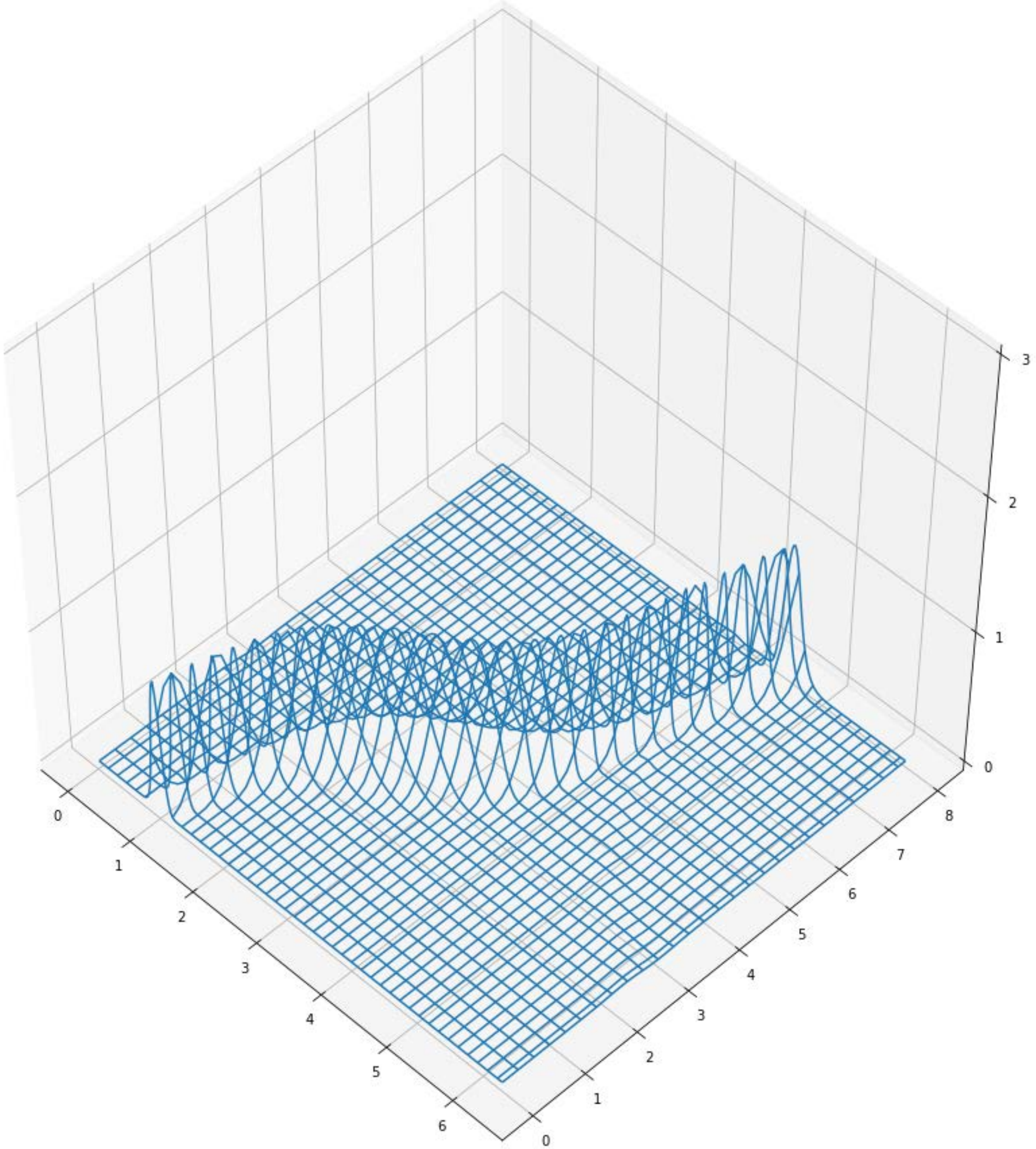
Program3. Python Code

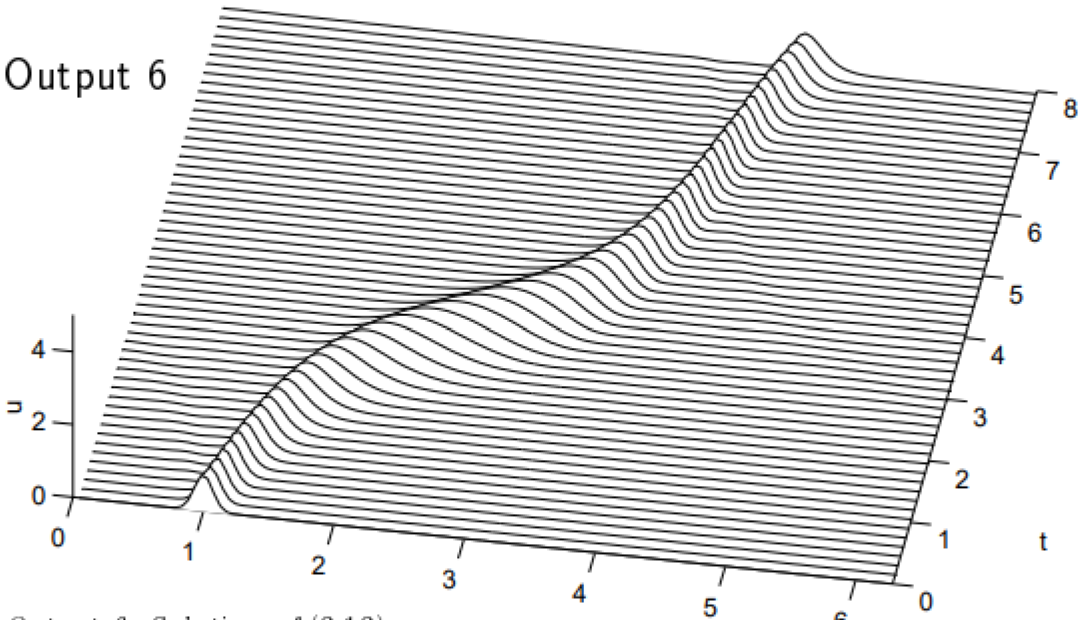
```
for i in range(1,nplots+1):
  for n in range(1,plotgap+1):
    t = t + dt
    v_hat = fft(v)
    w_hat = (1j * w_hat_A * v_hat)
    w = np.real(ifft(w_hat))
    vnew = vold - 2 * dt * c * w
    vold = v
    v = vnew
  data = np.vstack([data,v])
  tdata = np.append(tdata,t)

import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(16,16))
ax = fig.gca(projection='3d')
```

Add the portion of a matrix that corresponds to a row, circling the repeating statement.

Set 3D Graph

In Python





Output 6: Solution of (3.13).

In Matlab

Program4. Python Code

```
v1 = np.power(np.e,-100 * (x1-1) ** 2)
vold1 = np.power(np.e,-100 * (x1-0.2*dt-1) ** 2)

column2 = 0.5*(-1)**np.arange(1,N,1)*(1/np.tan(np.arange(1,N,1)*h/2))
#column2 = (-1)*(-1)**np.arange(1,N,1) / (2*np.sin(np.arange(1,N,1)*h/2))
value = 0
#value = ((-1)*(np.pi**2)/(3*h**2)) - 1/6
column2 = np.insert(column2,0,value)

column_3 = []

for i in np.arange(N-1,1-1,-1):
  column_3.append(column2[i])

column_3 = np.insert(column_3,0,value)
D1 = tp(column2,column_3)

data1 = np.matrix(v1)
tdata1 = [t1]

for j in range(1,nplots+1):
  for m in range(1,plotgap+1):
    t1 = t1 + dt
    vnew1 = vold1 - 2 * dt * c * np.dot(D1,v1)
    vold1 = v1
    v1 = vnew1
  data1 = np.vstack([data1,v1])
  tdata1 = np.append(tdata1,t1)
```

4. (Exercise 3.7) Recompute Output 6 by a modified program based on matrices rather than the FFT. Which program is slower and which one is faster? How does the answer change if N is increased from 128 to 156?

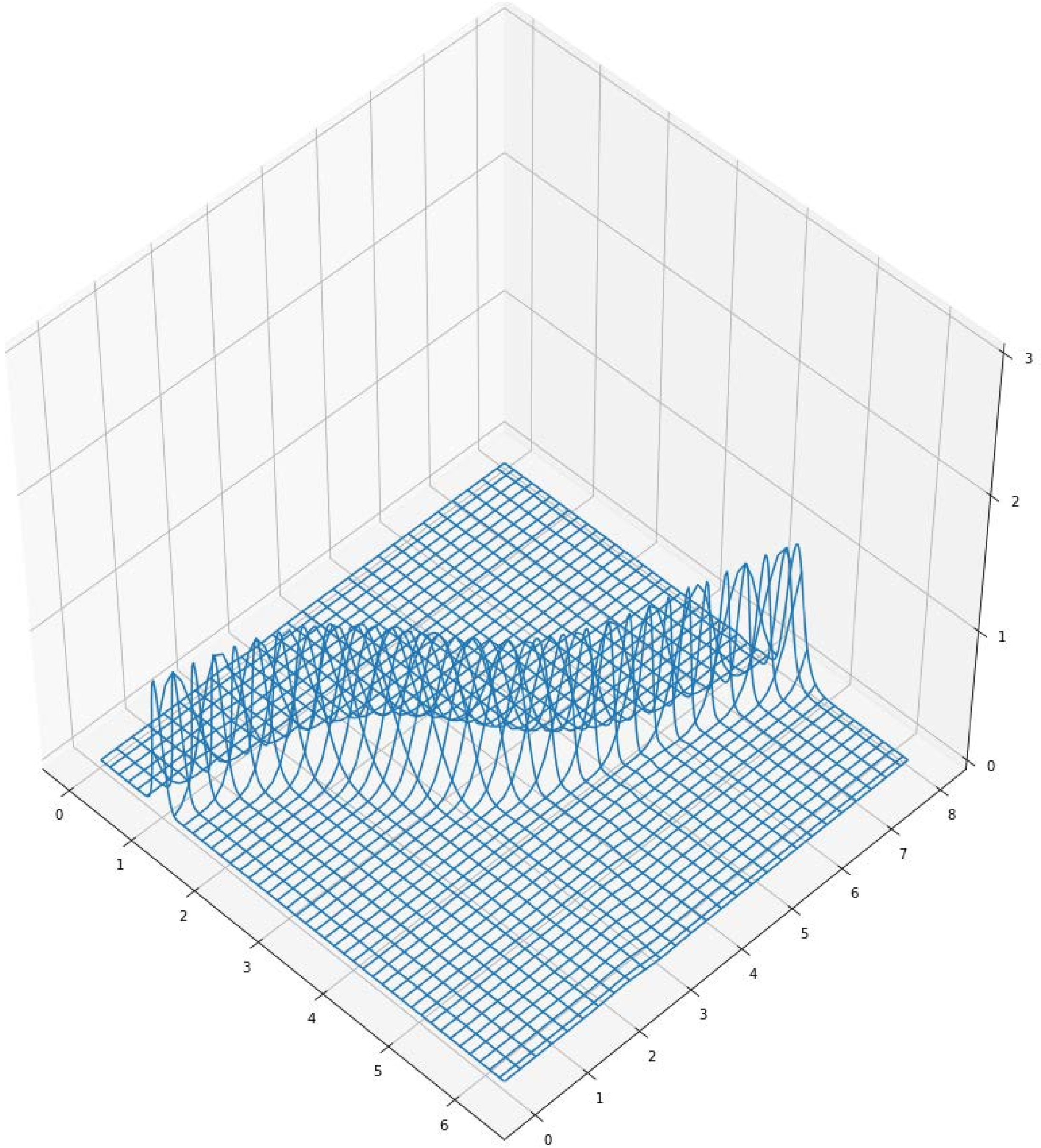
$$S_N'(x_j) = \begin{cases} 0 & j \equiv 0 \pmod N, \\ \frac{1}{2}(-1)^j \cot(jh/2) & j \not\equiv 0 \pmod N. \end{cases}$$

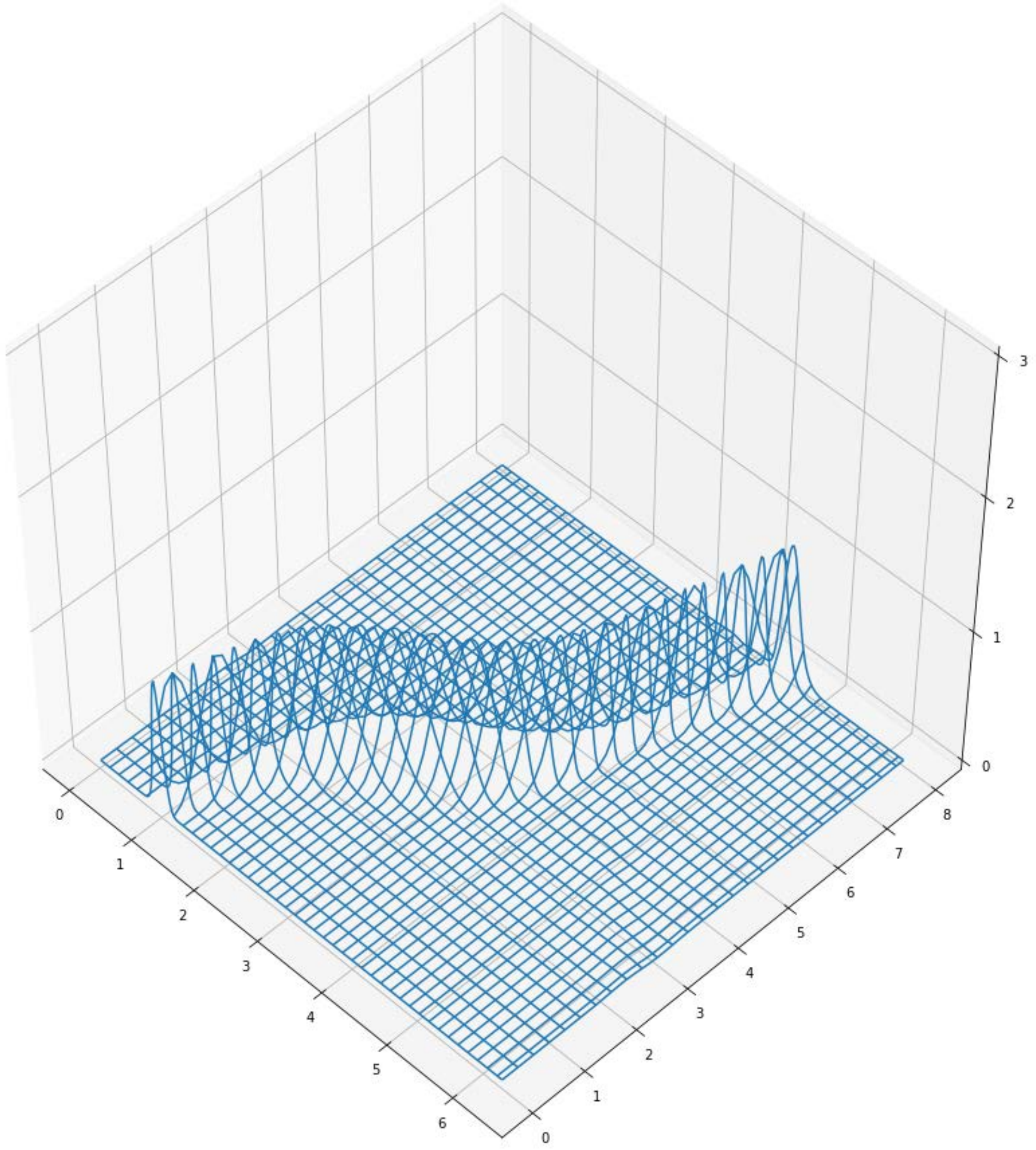
$$D_N = \begin{pmatrix} 0 & & & -\frac{1}{2}\cot\frac{1h}{2} \\ -\frac{1}{2}\cot\frac{1h}{2} & \ddots & \ddots & \frac{1}{2}\cot\frac{2h}{2} \\ \frac{1}{2}\cot\frac{2h}{2} & \ddots & & -\frac{1}{2}\cot\frac{3h}{2} \\ -\frac{1}{2}\cot\frac{3h}{2} & & \ddots & \vdots \\ \vdots & \ddots & \ddots & \frac{1}{2}\cot\frac{1h}{2} \\ \frac{1}{2}\cot\frac{1h}{2} & & & 0 \end{pmatrix}.$$

Set this matrix

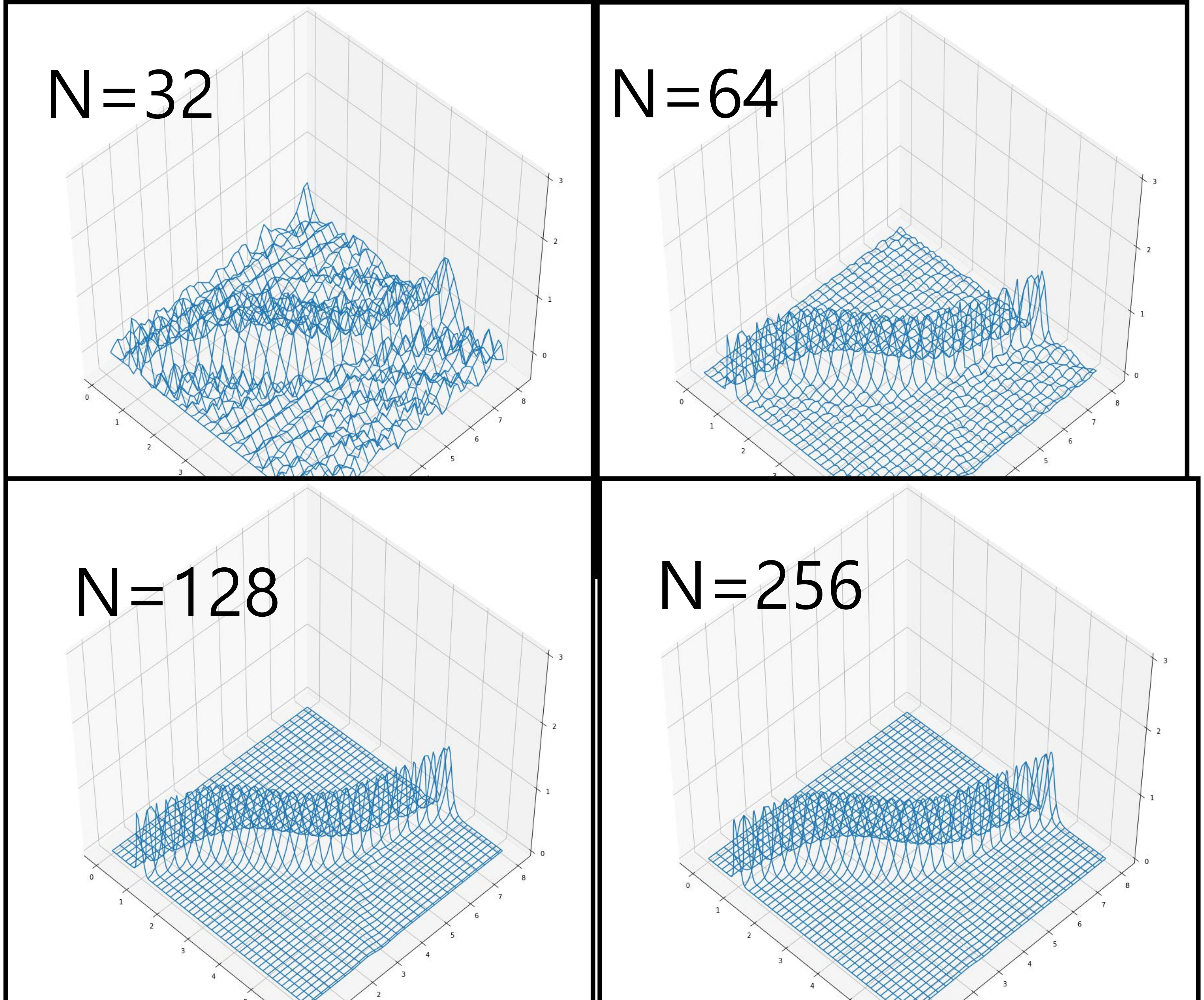
calculating the multiplication of a matrix

Using Matrix





Using FFT, IFFT


N=32
N=64
N=128
N=256

Program5. Python Code

```python
column_sign = 0.5 * np.power((-1),range(1,N+1))
column_cot = (-1)*column_sign

np.insert(column_sign,0,0)
np.insert(column_cot,0,0)

D2 = tp(column_sign,column_cot)
D2 = D2
```

```python
t2 = 0

v2 = np.power(np.e,-100 * (x2-1) ** 2)
vold2 = np.power(np.e,-100 * (x2-0.2*dt-1) ** 2)

data2 = np.matrix(v2)
tdata2 = [t2]

for k in range(1,nplots+1):
  for l in range(1,plotgap+1):
    t2 = t2 + dt
    vnew2 = vold2 - 2 * dt * c * np.dot(D2,v2)
    vold2 = v2
    v2 = vnew2
  data2 = np.vstack([data2,v2])
  tdata2 = np.append(tdata2,t2)


fig = plt.figure(figsize=(16,16))
ax = fig.gca(projection='3d')
```

calculating the multiplication of a matrix

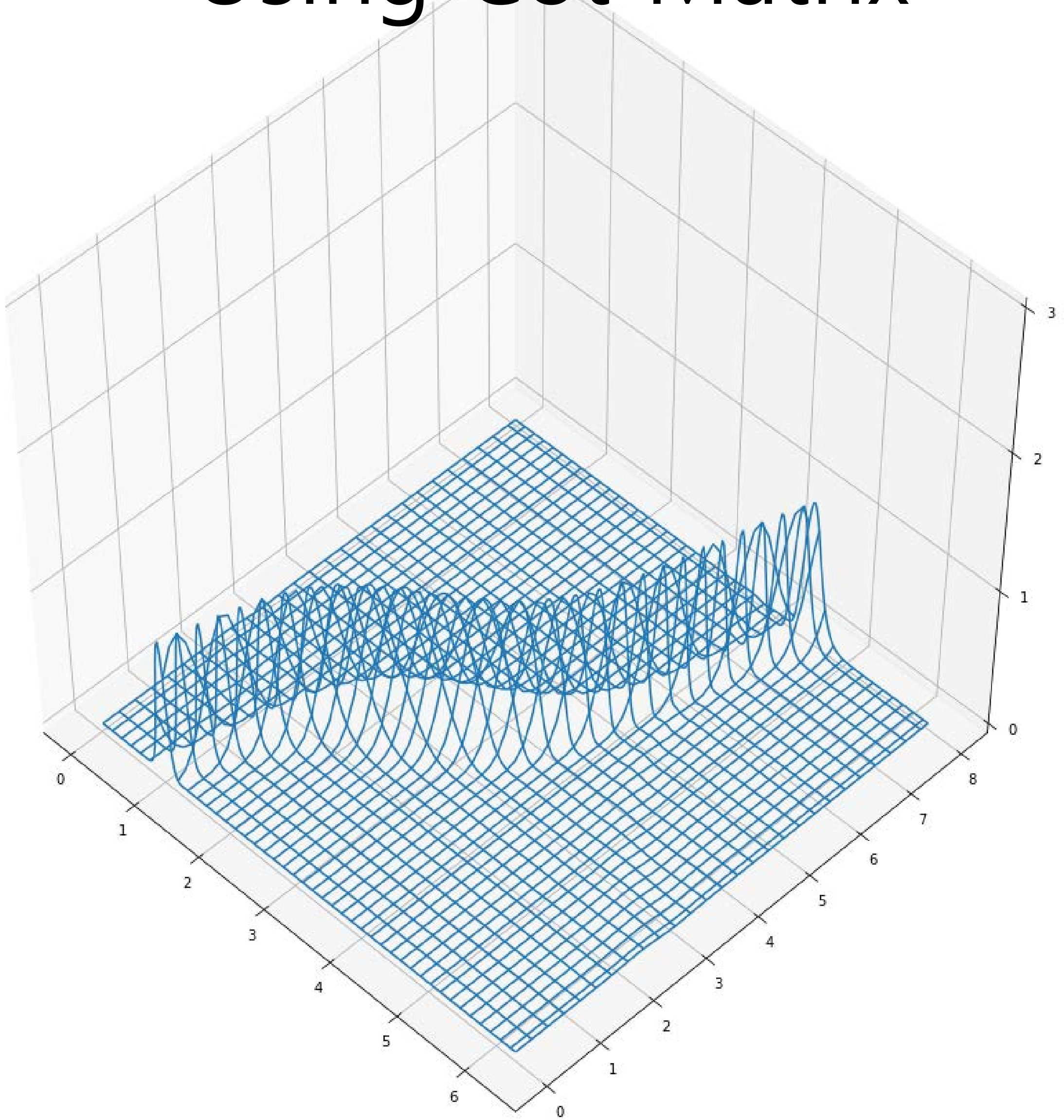
5. (Exercise 3.8) Recompute Output 6 by a modified program based on the finite difference leap frog formula

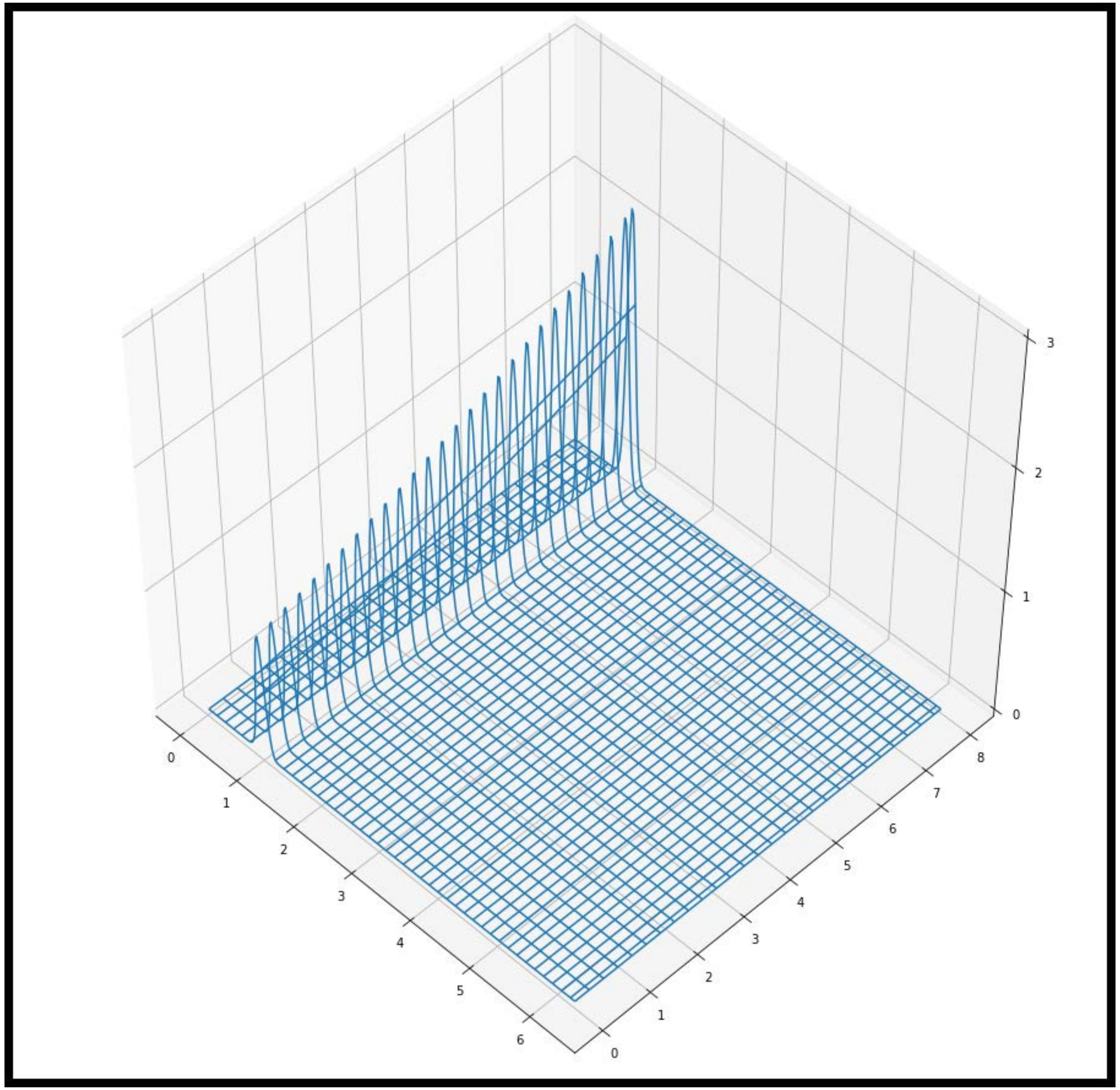
$$\frac{v_j^{(n+1)} - v_j^{(n-1)}}{2\Delta t} = -c(x_j)\left(Dv^{(n)}\right)_j, \quad j = 1, \ldots, N$$

rather than a spectral method. Here, D is the 2nd-order finite differentiation matrix of Eq. (1.2). Produce plots for $N = 128$ and 256. Comment.

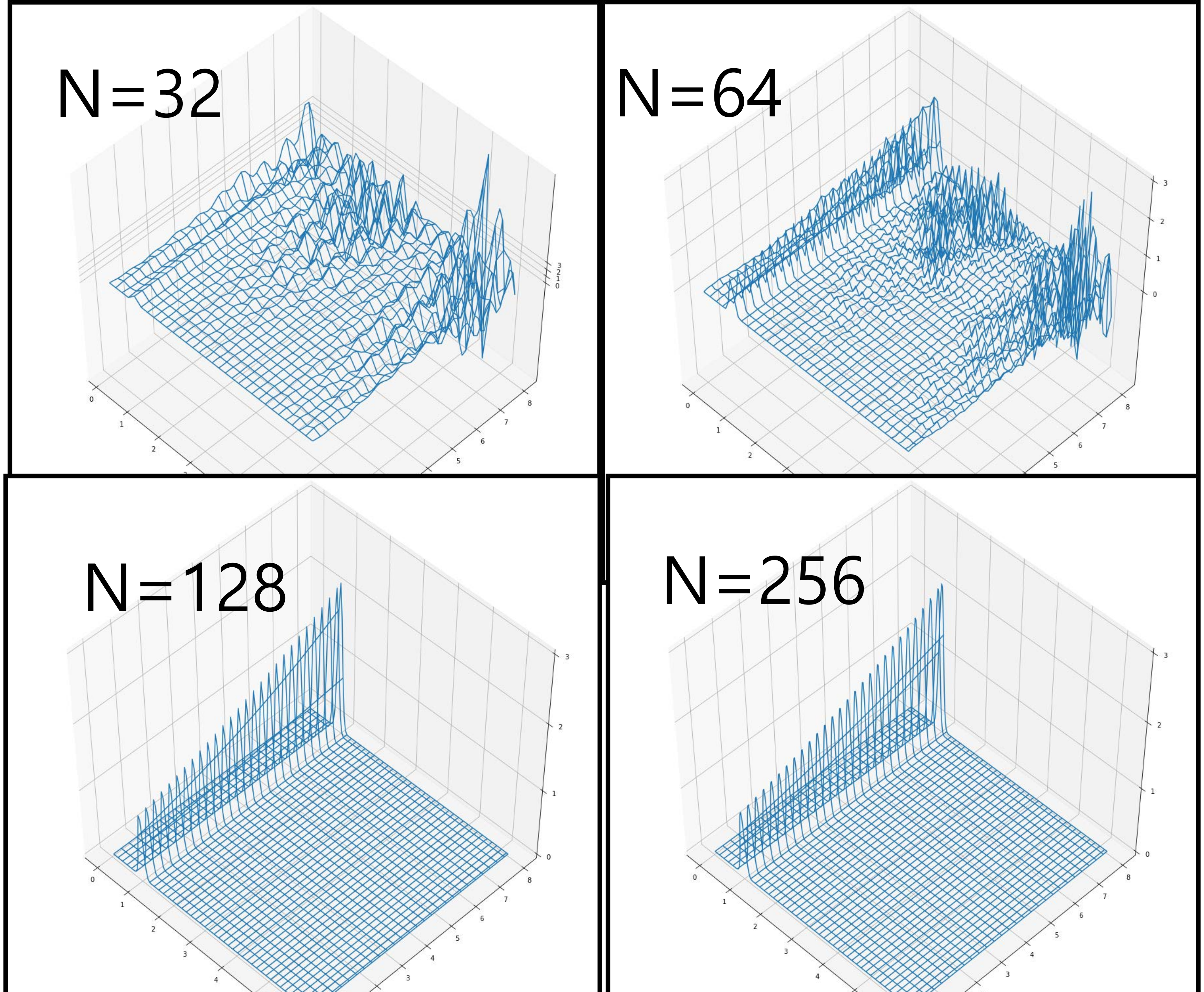
$$\begin{pmatrix} w_1 \\ \\ \vdots \\ \\ w_N \end{pmatrix} = h^{-1} \begin{pmatrix} 0 & \frac{1}{2} & & & -\frac{1}{2} \\ -\frac{1}{2} & 0 & \ddots & & \\ & & \ddots & & \\ & & \ddots & 0 & \frac{1}{2} \\ \frac{1}{2} & & & -\frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ \\ \vdots \\ \\ u_N \end{pmatrix}. \tag{1.2}$$

Using Cot Matrix





Using Eq(1.2)


N=32
N=64
N=128
N=256