2주차 과제1 #3

2022145079 임혜린

본 과제는 Python, VS Code를 사용하였음을 밝힙니다.

공통 조건 (Stokes second problem)

Stokes second problem

무한하게 펼쳐진 평평한 벽이 주기적인 진동을 한다고 가정한다(See Figure 1). 이 때, 점착조건(No-slip condition)으로 벽에서의 속도 $u(0,t) = U_0 cos(nt)$ 를 만족한다.

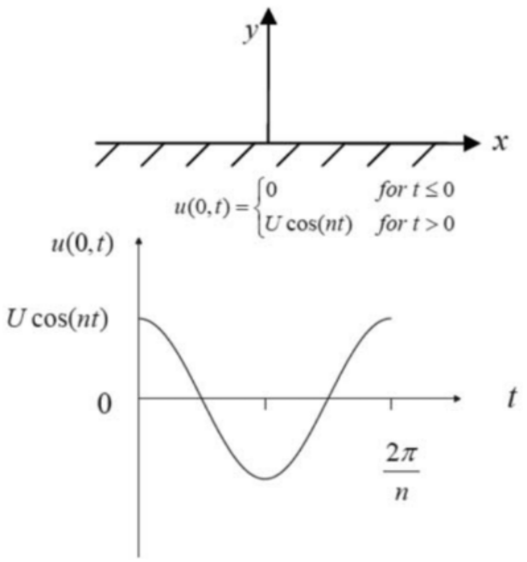


Figure 1 Schematic diagram of Stokes second problem

1-1

문제

1. (Analytic solution)

(1) 3차원 나비에-스톡스 방정식에서, Stokes second problem을 풀기 위한 간소화 된 지배방정식을 유도하시오. 유도 과정에서 사용되는 가정들 또한 서술하시오.

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial y^2}$$

풀이

운동 방정식 (x축 방향) $\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$

조건 ① $\frac{\partial u}{\partial x} = \frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 u}{\partial z^2} = 0$ (주어진 식에서 u는 y,t에 대한 함수) ② $v = w = 0$ ③ $\frac{\partial p}{\partial x} = 0$

$$\frac{\partial u}{\partial t} + \overset{①}{u\frac{\partial u}{\partial x}} + \overset{②}{v\frac{\partial u}{\partial y}} + \overset{②}{w\frac{\partial u}{\partial z}} = -\overset{③}{\frac{1}{\rho}\frac{\partial p}{\partial x}} + \nu\left(\overset{①}{\frac{\partial^2 u}{\partial x^2}} + \frac{\partial^2 u}{\partial y^2} + \overset{①}{\frac{\partial^2 u}{\partial z^2}}\right)$$ 의 이유로 소거

$\Rightarrow \frac{\partial u}{\partial t} = \nu\frac{\partial^2 u}{\partial y^2}$

1-2

문제

(2) 위에 주어진 Stokes second problem의 해가 다음과 같음을 보이시오.

$$u(y,t) = U_0 e^{-\eta_s}\cos(nt - \eta_s)\,, where\ \eta_s = \sqrt{\frac{n}{2\nu}}y.$$

풀이

$u = U_0 f(y) g(t)$, $g(t) = e^{int}$ $f(0) = 1$

$\frac{\partial u}{\partial t} = \nu\frac{\partial^2 u}{\partial y^2} \Rightarrow \cancel{U_0}\, in \times f(y)\cancel{e^{int}} = \nu \times \cancel{U_0} \times f''(y)\cancel{e^{int}}$

$f''(y) - \frac{in}{\nu}f(y) = 0.$ $f(y) = Ae^{\sqrt{\frac{in}{\nu}}y} + Be^{-\sqrt{\frac{in}{\nu}}y} = Ae^{\sqrt{\frac{n}{2\nu}}(1+i)y} + Be^{-\sqrt{\frac{n}{2\nu}}(1+i)y}$ $\left(\because \sqrt{\frac{in}{\nu}} = \sqrt{\frac{n}{\nu}}\left(\frac{1}{\sqrt{2}} + \frac{i}{\sqrt{2}}\right)\right)$

$y \to \infty$일 때 $u \to 0$ 이어야하므로 $A = 0$, $y = 0$일 때 $f(0) = 1$ 이므로 $B = 1$

$\therefore f(y) = e^{-\sqrt{\frac{n}{2\nu}}(1+i)y}$, $u = U_0 e^{-\sqrt{\frac{n}{2\nu}}(1+i)y + int} = U_0 e^{-\sqrt{\frac{n}{2\nu}}y} \times e^{i(nt - \sqrt{\frac{n}{2\nu}}y)} \Rightarrow U_0 e^{-\eta_s}\cos(nt - \eta_s)$

2-1

2. (Numerical analysis)

무한하게 펼쳐진 두 개의 평판이 각 각 y=0과 y=L에 위치한다고 가정한다. 바닥에 있는 평판(y=0)은 $u(0,t) = \cos(nt)$ 로 진동하는 반면에, 위에 있는 평판(y=L)은 고정되어있다. 주어진 조건 하에서 Stokes second problem의 지배방정식을 사용하여 다음 조건 하에 속도profile u(y,t)를 구하시오. $\nu = 1, n = 2, U_0 = 1$, 그리고 $L = 10$.

(1) 주어진 방정식을 first-order forward difference in time 그리고 second-order central difference in space(FTCS scheme)으로 계산하시오. 속도 profile은 $nt = 0, \frac{\pi}{2}, \pi, 3\pi/2, 2\pi$ 일 때에 대하여 그리시오. 또한 quasi-steady state velocity profile을 $(nt - T) = 0, \frac{\pi}{2}, \pi, 3\pi/2, 2\pi$ 에 대하여 구하시오. T는 일종의 quasi-steady state 해를 얻기 위한 transient period로서 $T = 10\pi$ 로 주어진다.

FTCS는 시간 방향 전진(Forward Time), 공간 방향 중앙차분(Central Space) 방식으로 진행하는 근사 방법을 뜻한다. 1-1의 Stokes second problem의 지배 방정식에 FTCS를 적용하면

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \nu \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta y^2}$$

이와 같으며 안정성 조건은 $r = \frac{\nu \Delta t}{\Delta y^2} \leq \frac{1}{2}$ 이다.

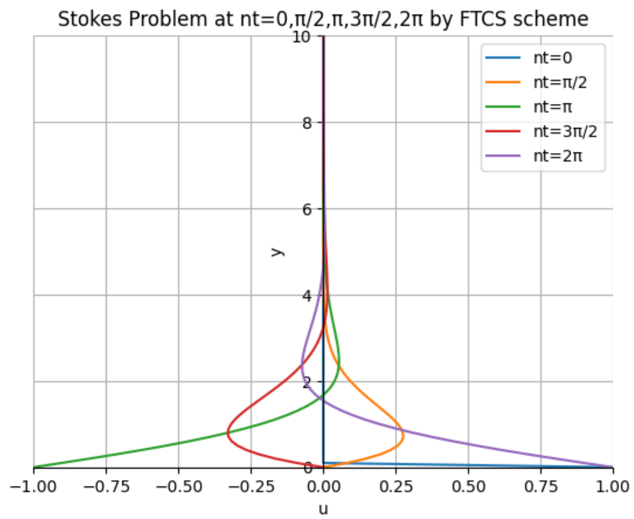
```
import numpy as np
import math
import matplotlib.pyplot as plt

v=1
n=2
U0=1
L=10
T=2*np.pi
t=0

dy=0.1
n_y=101
dt=T/2000
y=np.linspace(0,L,n_y)

u_n=np.zeros(n_y)
u_n[0]=U0*np.cos(n*t)
u_profile=[u_n.copy()]


for i in range(8000):
    t=i*dt
    u_n1=np.zeros(n_y)
    u_n1[0]=U0*np.cos(n*t)
    u_n1[-1]=0
    for j in range(1, n_y-1):
        u_n1[j]=u_n[j]+(v*dt/dy**2)*(u_n[j+1]-2*u_n[j]+u_n[j-1])
    u_n=u_n1
    for k in [0.5*np.pi, np.pi, 1.5*np.pi, 2*np.pi, 10*np.pi, 10.5*np.pi, 11*np.pi, 11.5*np.pi, 12*np.pi]:
        if np.abs(t-k/n)<dt/2:
            u_profile.append(u_n1.copy())
```

주어진 조건에 따라 v=1이며 dy=0.1로 놓았으므로 dt≤0.005이다. 조건을 만족하는 dt=2π/2000만큼 t를 증가해가며 t에서의 속도 리스트 u_n을 통해 u_n1[j]를 도출하는 식을 만든 후 해당 t가 그래프를 그리고자 하는 nt에 해당하는 경우 u_profile 리스트에 추가하는 형식을 사용하였다.

u_profile[0]~ u_profile[4]는 FTCS scheme at nt=0, π/2, π, 3π/2, 2π 그래프, u_profile[5]~ u_profile[9]는 준정상 상태에서의 속도, 즉 FTCS scheme at quasti-steady state (at nt=10π, 10.5π, 11π, 11.5π, 12π) 그래프를 나타내는 데 사용되었다.

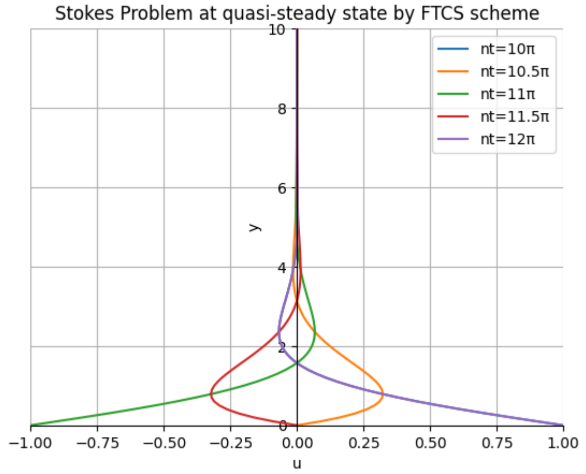
FTCS scheme at nt=0, π/2, π, 3π/2, 2π의 그래프 코드와 결과

```
fig, ax = plt.subplots()
ax.set_xlim(-1, 1)
ax.set_ylim(0, 10)
ax.spines['left'].set_position('center')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['bottom'].set_position(('data', 0))

ax.plot(u_profile[0], y, label='nt=0')
ax.plot(u_profile[1], y, label='nt=π/2')
ax.plot(u_profile[2], y, label='nt=π')
ax.plot(u_profile[3], y, label='nt=3π/2')
ax.plot(u_profile[4], y, label='nt=2π')

ax.set_xlabel('u')
ax.set_ylabel('y')
ax.legend()
plt.grid()
ax.set_title('Stokes Problem at nt=0,π/2,π,3π/2,2π by FTCS scheme')
plt.show()
```



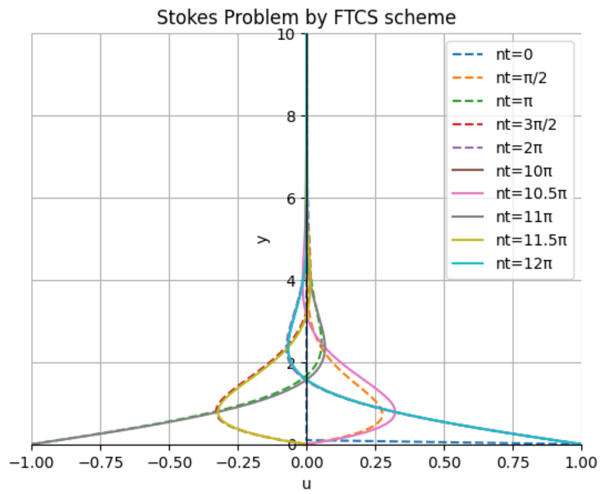
FTCS scheme at quasi-steady state (at nt=10π, 10.5π, 11π, 11.5π, 12π)의 그래프 코드와 결과

```
fig, ax = plt.subplots()
ax.set_xlim(-1, 1)
ax.set_ylim(0, 10)
ax.spines['left'].set_position('center')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['bottom'].set_position(('data', 0))

ax.plot(u_profile[5], y, label='nt=10π')
ax.plot(u_profile[6], y, label='nt=10.5π')
ax.plot(u_profile[7], y, label='nt=11π')
ax.plot(u_profile[8], y, label='nt=11.5π')
ax.plot(u_profile[9], y, label='nt=12π')

ax.set_xlabel('u')
ax.set_ylabel('y')
ax.legend()
plt.grid()
ax.set_title('Stokes Problem at quasi-steady state by FTCS scheme')
plt.show()
```



FTCS scheme at nt=0, π/2, π, 3π/2, 2π, 10π, 10.5π, 11π, 11.5π, 12π의 그래프 코드와 결과

```
fig, ax = plt.subplots()
ax.set_xlim(-1, 1)
ax.set_ylim(0, 10)
ax.spines['left'].set_position('center')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['bottom'].set_position(('data', 0))

ax.plot(u_profile[0], y, linestyle='--', label='nt=0')
ax.plot(u_profile[1], y, linestyle='--', label='nt=π/2')
ax.plot(u_profile[2], y, linestyle='--', label='nt=π')
ax.plot(u_profile[3], y, linestyle='--', label='nt=3π/2')
ax.plot(u_profile[4], y, linestyle='--', label='nt=2π')
ax.plot(u_profile[5], y, label='nt=10π')
ax.plot(u_profile[6], y, label='nt=10.5π')
ax.plot(u_profile[7], y, label='nt=11π')
ax.plot(u_profile[8], y, label='nt=11.5π')
ax.plot(u_profile[9], y, label='nt=12π')

ax.set_xlabel('u')
ax.set_ylabel('y')
ax.legend()
plt.grid()
ax.set_title('Stokes Problem by FTCS scheme')
plt.show()
```



(두 그래프 함께 나타내기)

실선이 준정상 상태에서의 그래프이다.

2-2

(2) 위 문제를 시간에 대하여 Crank-Nicolson scheme을 사용하여 계산하시오

Crank-Nicolson scheme은 시간 미분을 전 시간 단계와 다음 시간 단계의 중앙 평균으로 근사한다. 시간 방향 전진이었던 FTCS와 달리 n번째와 n+1번째 시점의 함수값을 모두 포함하여 미분값을 구하기 때문에 n_y*n_y 벡터와 np.linalg.solve를 이용하여 코드를 구성하였다.

Stokes second condition의 지배방정식에 적용하면 아래와 같다.

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}=\nu\frac{1}{2}\left(\frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{\Delta y^2}+\frac{u_{i+1}^{n+1}-2u_i^{n+1}+u_{i-1}^{n+1}}{\Delta y^2}\right)$$

$\alpha=\frac{\nu\Delta t}{2(\Delta y)^2}$ 로 잡았을 때, $-\alpha u_{i-1}^{n+1}+(1+2\alpha)u_i^{n+1}-\alpha u_{i+1}^{n+1}=\alpha u_{i-1}^n+(1-2\alpha)u_i^n+\alpha u_{i+1}^n$ 이므로 이를 만족시키기 위해 n_y-2*n_y-2 벡터 M을 만든 후, 위 식의 우변 값을 y에 따라 저장한 리스트 u_x와 M을 np.linalg.solve하여 이것을 len이 n_y인 리스트 u_n1의 양 끝 값을 제외한 부분에 저장하였다. u_n1의 양 끝 값은 boundary condition을 만족하기 위해 따로 지정하였다.

$$M=\begin{bmatrix}1+2\alpha & -\alpha & 0 & \dots & 0\\ -\alpha & 1+2\alpha & -\alpha & \dots & 0\\ 0 & -\alpha & 1+2\alpha & \dots & 0\\ \vdots & \vdots & \vdots & \ddots & -\alpha\\ 0 & 0 & 0 & -\alpha & 1+2\alpha\end{bmatrix}$$

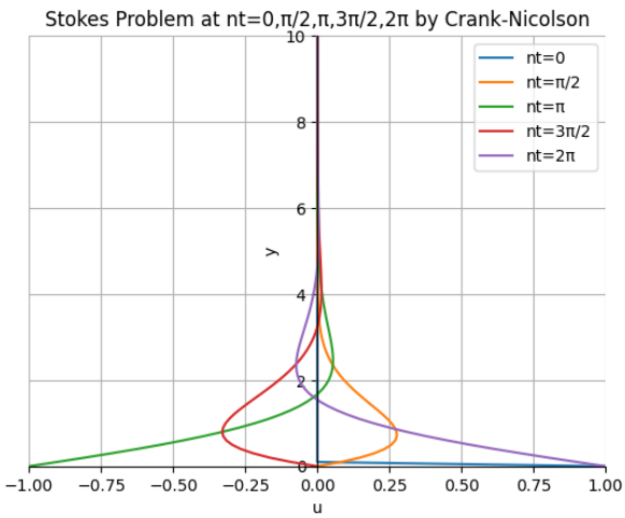
이렇게 저장된 u_n1은 t가 변함에 따라 리셋되며, 해당 t가 그래프를 그리고자 하는 nt에 해당하는 경우 u_profile 리스트에 추가되도록 하였다.

```
import numpy as np
import matplotlib.pyplot as plt

v=1
n=2
U0=1
L=10
T=2*np.pi
dy=0.1
n_y=101
dt=T/2000
y=np.linspace(0,L,n_y)

a=v*dt/(2*dy**2)
M=np.zeros((n_y-2, n_y-2))
for i in range(n_y-3):
    M[i][i]=1+2*a
    M[i+1][i]=-a
    M[i][i+1]=-a
M[n_y-3][n_y-3]=1+2*a

t=0
u_n=np.zeros(n_y)
u_n1=np.zeros(n_y)
```

```
u_profile=np.zeros(n_y)
u_profile[0]=U0*np.cos(n*t)
u_profile=[u_profile]


for k in range(2000):
    t+=dt
    u_n1[0]=U0*np.cos(n*t+dt)
    u_n1[-1]=0
    u_ex=np.zeros(n_y-2)
    for i in range(n_y-2):
        u_ex[i]=a*u_n[i]+(1-2*a)*u_n[i+1]+a*u_n[i+2]
    u_ex[0]+=a*U0*np.cos(n*t)
    u_n1[1:-1]=np.linalg.solve(M, u_ex)
    for l in [0.5*np.pi, np.pi, 1.5*np.pi, 2*np.pi,
              10*np.pi, 10.5*np.pi, 11*np.pi, 11.5*np.pi, 12*np.pi]:
        if np.abs(t-l/n)<dt/2:
            u_n1_bc=u_n1.copy()
            u_profile.append(u_n1_bc.copy())
    u_n[...]=u_n1[...]
```

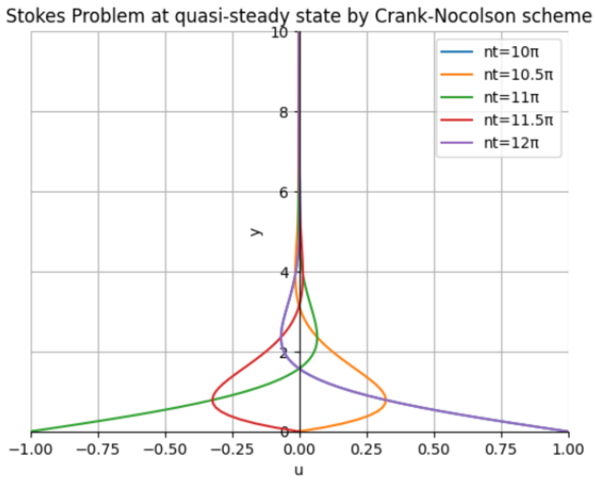
Crank-Nicolson scheme at nt=0, π/2, π, 3π/2, 2π의 그래프 코드와 결과

```
fig, ax = plt.subplots()
ax.set_xlim(-1, 1)
ax.set_ylim(0, 10)
ax.spines['left'].set_position('center')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['bottom'].set_position(('data', 0))

ax.plot(u_profile[0], y, label='nt=0')
ax.plot(u_profile[1], y, label='nt=π/2')
ax.plot(u_profile[2], y, label='nt=π')
ax.plot(u_profile[3], y, label='nt=3π/2')
ax.plot(u_profile[4], y, label='nt=2π')

ax.set_xlabel('u')
ax.set_ylabel('y')
ax.legend()
plt.grid()
ax.set_title('Stokes Problem at nt=0,π/2,π,3π/2,2π by Crank-Nicolson'
plt.show()
```



Crank-Nicolson scheme at quasi-steady state (nt=10π,10.5π,11π,11.5π,12π)의 그래프 코드와 결과

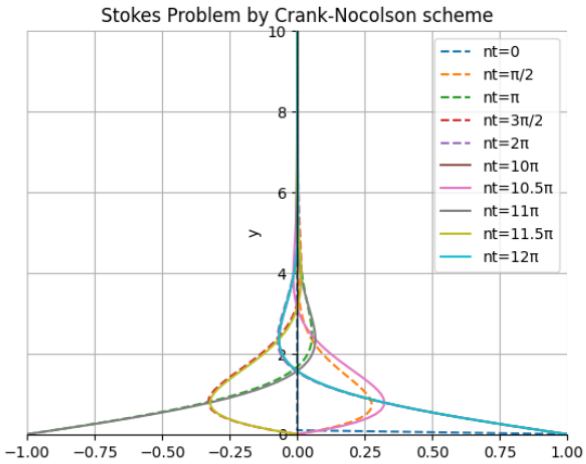
```
fig, ax = plt.subplots()
ax.set_xlim(-1, 1)
ax.set_ylim(0, 10)
ax.spines['left'].set_position('center')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['bottom'].set_position(('data', 0))

ax.plot(u_profile[5], y, label='nt=10π')
ax.plot(u_profile[6], y, label='nt=10.5π')
ax.plot(u_profile[7], y, label='nt=11π')
ax.plot(u_profile[8], y, label='nt=11.5π')
ax.plot(u_profile[9], y, label='nt=12π')

ax.set_xlabel('u')
ax.set_ylabel('y')
ax.legend()
plt.grid()
ax.set_title('Stokes Problem at quasi-steady state by Crank-Nocolson sche
plt.show()
```



Crank-Nicolson scheme at nt=0, π/2, π, 3π/2, 2π, 10π, 10.5π, 11π, 11.5π, 12π 그래프 코드와 결과

```
fig, ax = plt.subplots()
ax.set_xlim(-1, 1)
ax.set_ylim(0, 10)
ax.spines['left'].set_position('center')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['bottom'].set_position(('data', 0))

ax.plot(u_profile[0], y, linestyle='--', label='nt=0')
ax.plot(u_profile[1], y, linestyle='--', label='nt=π/2')
ax.plot(u_profile[2], y, linestyle='--', label='nt=π')
ax.plot(u_profile[3], y, linestyle='--', label='nt=3π/2')
ax.plot(u_profile[4], y, linestyle='--', label='nt=2π')
ax.plot(u_profile[5], y, label='nt=10π')
ax.plot(u_profile[6], y, label='nt=10.5π')
ax.plot(u_profile[7], y, label='nt=11π')
ax.plot(u_profile[8], y, label='nt=11.5π')
ax.plot(u_profile[9], y, label='nt=12π')
```

```
ax.set_xlabel('u')
ax.set_ylabel('y')
ax.legend()
plt.grid()
ax.set_title('Stokes Problem by Crank-Nocolson scheme')
plt.show()
```



(두 그래프 함께 나타내기)

실선이 준정상 상태에서의 그래프이다.

2-3

(3) 각각 다른 두개의 scheme에 대하여 시간의 변화에 따른 해의 수렴율을 구하시오 (log(Δt) vs log (l2norm)). FTCS는 시간에 대한 1차, Crank-Nicolson은 2차의 수렴율을 보여야 하며, 오차계산에 사용되는 exact solution은

$$u(y,t) = U_0 e^{-\eta_s} \cos(nt - \eta_s)\,, where\ \eta_s = \sqrt{\frac{n}{2\nu}}y.$$

을 사용하여 구하시오.

기본 코드는 2-1, 2-2와 같으나 for문을 이용하여 dt를 변화시키며 진행하였다. 동일한 dt번째의 시점을(ex: t=5*k*dt) 잡아 그때의 각 y에서의 u를 scheme으로 도출한 값과 exact solution 값의 L2 norm을 구하여 error_FTCS 또는 error_CN에 추가하였다. 이렇게 구한 각 dt에서의 error을 모아 log10 그래프를 그린 결과 FTCS는 약 1.56, Crank-Nicolson은 약 1.2의 order of accuracy를 보였다.

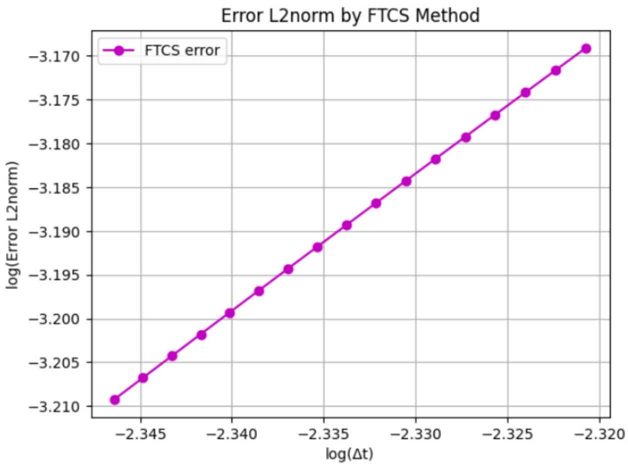
- FTCS Method의 경우

```
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.stats import linregress

def u_exact(y, t):
    n_s=math.sqrt(n/(2*v))*y
    return U0*np.exp(-n_s)*np.cos(n*t-n_s)
```

```python
v=1
n=2
U0=1
L=10
T=2*np.pi
t=0

dy=0.1
n_y=101
y=np.linspace(0,L,n_y)
error_FTCS=[]

u_n=np.zeros(n_y)
dt_list=[]

for k in range(1300, 1400, 5):
    dt=T/k
    dt_list.append(dt)
    u_profile=np.zeros(n_y)
    u_profile[0]=U0*np.cos(n*t)
    u_profile=[u_profile]
    u_n[0]=U0*np.cos(n*t)
    u_ex=[u_exact(y, 0)]
    for i in range(5*k):
        t=i*dt
        u_n1=np.zeros(n_y)
        u_n1[0]=U0*np.cos(n*t)
        u_n1[-1]=0
        for j in range(1, n_y-1):
            u_n1[j]=u_n[j]+(v*dt/dy**2)*(u_n[j+1]-2*u_n[j]+u_n[j-1])
        u_n=u_n1
        u_profile.append(u_n1.copy())
        u_ex_t=u_exact(y, t)
        u_ex.append(u_ex_t)
    error_FTCS.append(np.linalg.norm(u_profile[-1]-u_ex[-1], 2)*np.sqrt(dy))
```

Error L2 norm by FTCS Method의 그래프 코드와 결과

```python
# print(len(dt_list), len(error_FTCS))
# print(dt_list)
# print(len(u_profile), len(u_ex))

dt_list=dt_list[3:]
error_FTCS=error_FTCS[3:]

plt.plot(np.log10(dt_list), np.log10(error_FTCS), marker='o', color='m', label='FTCS error')
plt.xlabel('log(Δt)')
plt.ylabel('log(Error L2norm)')
plt.title('Error L2norm by FTCS Method')
plt.grid()
plt.legend()
plt.show()

print(f'order of accuracy: {linregress(np.log10(dt_list), np.log10(error_FTCS)).slope}')
```



order of accuracy: 1.564693767112058

order of accuracy: 1.564693767112058

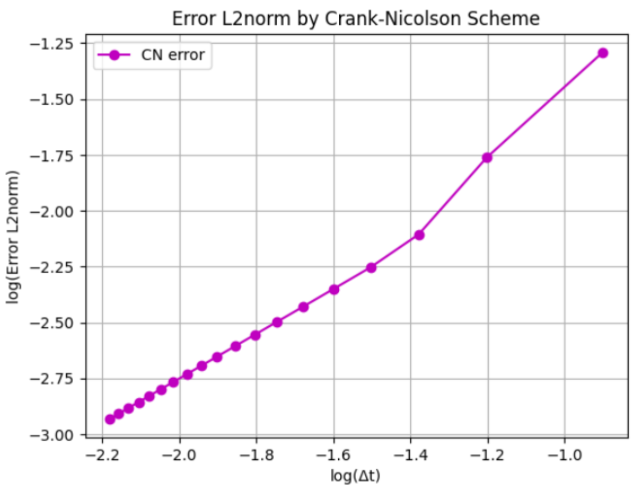
- Crank-Nicolson scheme의 경우

```
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.stats import linregress

def u_exact(y, t):
    n_s=math.sqrt(n/(2*v))*y
    return U0*np.exp(-n_s)*np.cos(n*t-n_s)

v=1
n=2
U0=1
L=10
T=2*np.pi
dy=0.01
n_y=1001
dt=T/2000
y=np.linspace(0,L,n_y)
dt_list=[]
t=0
u_n=np.zeros(n_y)
u_n1=np.zeros(n_y)
error_CN=[]


for k in range(50, 1000, 50):
    dt=T/k
    dt_list.append(dt)
    u_profile=np.zeros(n_y)
    u_profile[0]=U0*np.cos(n*t)
    u_profile=[u_profile]
    u_n[0]=U0*np.cos(n*t)
    u_ex=[u_exact(y, 0)]
    a=v*dt/(2*dy**2)
    M=np.zeros((n_y-2, n_y-2))
    for i in range(n_y-3):
        M[i][i]=1+2*a
        M[i+1][i]=-a
        M[i][i+1]=-a
    M[n_y-3][n_y-3]=1+2*a
    for l in range(3*k):
```

```
        t+=dt
        u_n1[0]=U0*np.cos(n*t+dt)
        u_n1[-1]=0
        u_example=np.zeros(n_y-2)
        for i in range(n_y-2):
            u_example[i]=a*u_n[i]+(1-2*a)*u_n[i+1]+a*u_n[i+2]
        u_example[0]+=a*U0*np.cos(n*t)
        u_n1[1:-1]=np.linalg.solve(M, u_example)
        u_n1_bc=u_n1.copy()
        u_profile.append(u_n1_bc.copy())
        u_n[...]=u_n1[...]
        u_ex_t=u_exact(y, t)
        u_ex.append(u_ex_t)
    error_CN.append(np.linalg.norm(u_profile[3*k]-u_ex[3*k], 2)*np.sqrt(dy))
```

Error L2 norm by Crank-Nicolson scheme의 그래프 코드와 결과

```
plt.plot(np.log10(dt_list), np.log10(error_CN), marker='o', color='m', label='CN error')
plt.xlabel('log(Δt)')
plt.ylabel('log(Error L2norm)')
plt.title('Error L2norm by Crank-Nicolson Scheme')
plt.grid()
plt.legend()
plt.show()

print(f'order of accuracy: {linregress(np.log10(dt_list), np.log10(error_CN)).slope}')
```



order of accuracy: 1.2029220799363685

order of accuracy: 1.2029220799363685

2-4

(4) (2)번 문제를 L=2의 조건에 대하여 다시 진행하고, 두 평판의 거리가 속도 profile에 미치는 영향을 서술하시오.

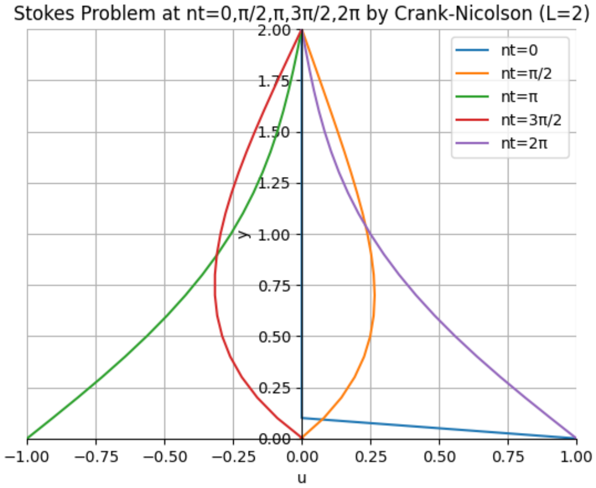
코드는 L=2, n_y=21으로 변경된 것 외에는 2-2와 동일하다.

```python
import numpy as np
import matplotlib.pyplot as plt

v=1
n=2
U0=1
L=2
T=2*np.pi
dy=0.1
n_y=21
dt=T/2000
y=np.linspace(0,L,n_y)

a=v*dt/(2*dy**2)
M=np.zeros((n_y-2, n_y-2))
for i in range(n_y-3):
    M[i][i]=1+2*a
    M[i+1][i]=-a
    M[i][i+1]=-a
M[n_y-3][n_y-3]=1+2*a

t=0
u_n=np.zeros(n_y)
u_n1=np.zeros(n_y)
u_profile=np.zeros(n_y)
u_profile[0]=U0*np.cos(n*t)
u_profile=[u_profile]


for k in range(2000):
    t+=dt
    u_n1[0]=U0*np.cos(n*t+dt)
    u_n1[-1]=0
    u_ex=np.zeros(n_y-2)
    for i in range(n_y-2):
        u_ex[i]=a*u_n[i]+(1-2*a)*u_n[i+1]+a*u_n[i+2]
    u_ex[0]+=a*U0*np.cos(n*t)
    u_n1[1:-1]=np.linalg.solve(M, u_ex)
    for l in [0.5*np.pi, np.pi, 1.5*np.pi, 2*np.pi,
              10*np.pi, 10.5*np.pi, 11*np.pi, 11.5*np.pi, 12*np.pi]:
        if np.abs(t-l/n)<dt/2:
            u_n1_bc=u_n1.copy()
            u_profile.append(u_n1_bc.copy())
    u_n[...]=u_n1[...]
```

Crank-Nicolson scheme at nt=0, π/2, π, 3π/2, 2π의 그래프 코드와 결과

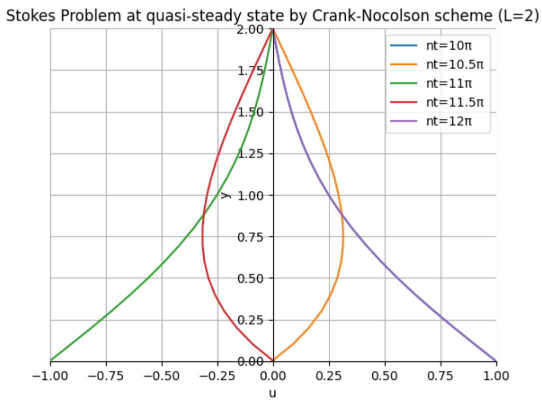
```python
fig, ax = plt.subplots()
ax.set_xlim(-1, 1)
ax.set_ylim(0, 10)
ax.spines['left'].set_position('center')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['bottom'].set_position(('data', 0))

ax.plot(u_profile[0], y, label='nt=0')
ax.plot(u_profile[1], y, label='nt=π/2')
ax.plot(u_profile[2], y, label='nt=π')
ax.plot(u_profile[3], y, label='nt=3π/2')
ax.plot(u_profile[4], y, label='nt=2π')
```

```python
ax.set_xlabel('u')
ax.set_ylabel('y')
ax.legend()
plt.grid()
ax.set_title('Stokes Problem at nt=0,π/2,π,3π/2,2π  by Crank-Nicolson (L=2)')
plt.show()
```



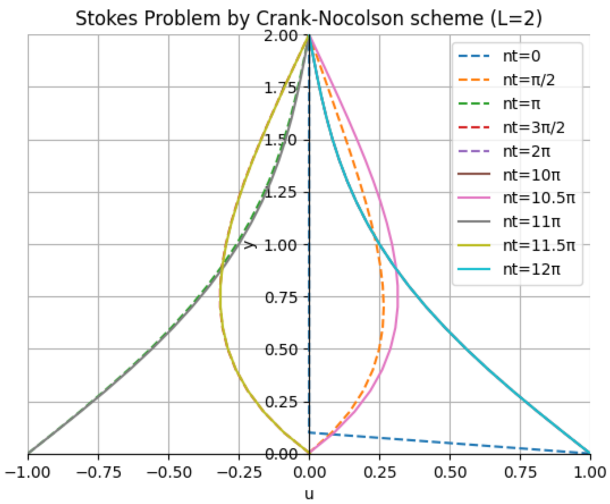
Crank-Nicolson scheme at quasi-steady state (nt=10π,10.5π,11π,11.5π,12π)의 그래프 코드와 결과

```python
fig, ax = plt.subplots()
ax.set_xlim(-1, 1)
ax.set_ylim(0, 10)
ax.spines['left'].set_position('center')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['bottom'].set_position(('data', 0))

ax.plot(u_profile[5], y, label='nt=10π')
ax.plot(u_profile[6], y, label='nt=10.5π')
ax.plot(u_profile[7], y, label='nt=11π')
ax.plot(u_profile[8], y, label='nt=11.5π')
ax.plot(u_profile[9], y, label='nt=12π')

ax.set_xlabel('u')
ax.set_ylabel('y')
ax.legend()
plt.grid()
ax.set_title('Stokes Problem at quasi-steady state by Crank-Nocolson scheme (L=2)')
plt.show()
```



Crank-Nicolson scheme at nt=0, π/2, π, 3π/2, 2π, 10π, 10.5π, 11π, 11.5π, 12π 그래프 코드와 결과

```
fig, ax = plt.subplots()
ax.set_xlim(-1, 1)
ax.set_ylim(0, 10)
ax.spines['left'].set_position('center')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['bottom'].set_position(('data', 0))

ax.plot(u_profile[0], y, linestyle='--', label='nt=0')
ax.plot(u_profile[1], y, linestyle='--', label='nt=π/2')
ax.plot(u_profile[2], y, linestyle='--', label='nt=π')
ax.plot(u_profile[3], y, linestyle='--', label='nt=3π/2')
ax.plot(u_profile[4], y, linestyle='--', label='nt=2π')
ax.plot(u_profile[5], y, label='nt=10π')
ax.plot(u_profile[6], y, label='nt=10.5π')
ax.plot(u_profile[7], y, label='nt=11π')
ax.plot(u_profile[8], y, label='nt=11.5π')
ax.plot(u_profile[9], y, label='nt=12π')

ax.set_xlabel('u')
ax.set_ylabel('y')
ax.legend()
plt.grid()
ax.set_title('Stokes Problem by Crank-Nocolson scheme (L=2)')
plt.show()
```



코드에는 y를 0부터 10까지 보이는 것으로 써 있으나 더 편하게 보기 위해 0, 2으로 바꾸어 진행하였다.

결과적으로 L=10일 때보다 진동의 영향이 L까지 더욱 효과적으로 전달되는 것을 발견할 수 있다.