1. Analytical solution

(1)

3차원 x방향 나비에 스토크스 방정식은 다음과 같다.

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}=-\frac{1}{\rho}\frac{\partial p}{\partial x}+v(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2})$$

y방향과 z방향 속도는 무시할 수 있고, x방향으로 압력과 속도의 변화가 없다고 가정하면 다음과 같이 정리된다.

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

(2)

경계 조건과 초기 조건은 다음과 같다.

$$u(y,0)=0$$

$$u(0,t)=U_0\cos(nt)$$

$$u(\infty,t)=0$$

속도 $u=U_0 Re[e^{int}f(y)]$로 나타낼 수 있으므로 이를 방정식에 대입하면,

$$f''-\frac{in}{v}f=0$$

경계 조건을 활용하면,

$$f(y)=\exp\left[-\frac{1+i}{\sqrt{2}}\sqrt{\frac{n}{v}}y\right]$$

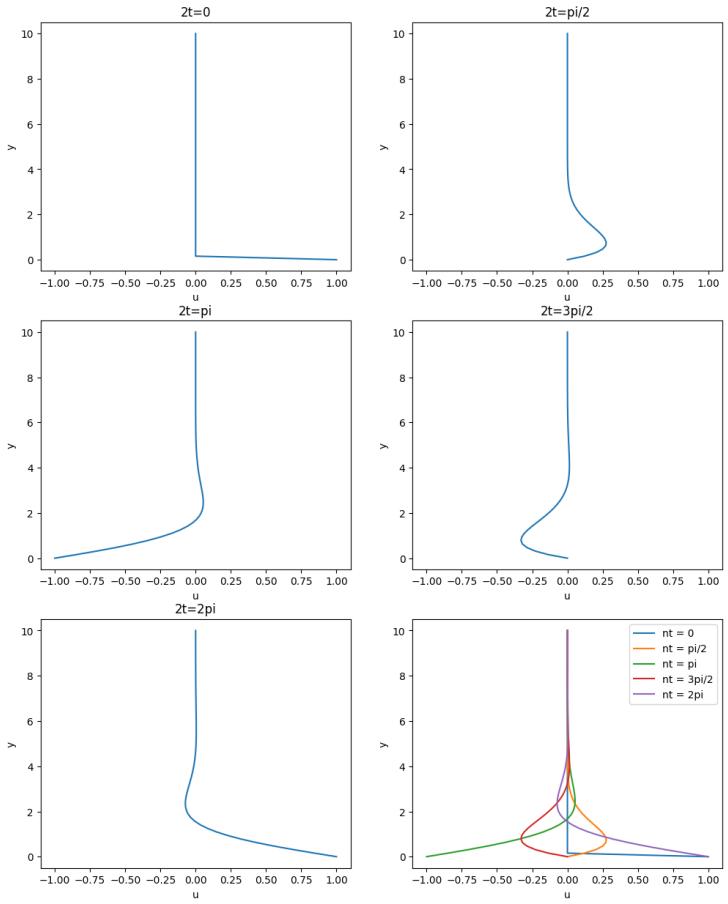
$$\therefore u(y,t)=U_0 e^{-\eta_s}\cos(nt-\eta_s)\ \ where\ \eta_s=\sqrt{\frac{n}{2v}}y$$

2. Numerical analysis

(1) 다음과 같이 FTCS scheme을 구현했다.

```
#FTCS method
for i in range(1,N):
  u_y = np.zeros(Ny)
  u_y[0] = 0
  u_y[-1] = U_0*np.cos(n*t[i])
  for j in range(1,Ny-1):
    u_y[j] = u[j,-1] + v*h*(u[j+1, -1] -2*u[j, -1] +u[j-1,-1])/hy**2
  u_y_T = u_y.reshape(-1,1)
  u = np.hstack((u, u_y_T))
```

u는 N x Ny matrix로, 행은 y좌표 열은 시간을 나타낸다. T=0일 때 열 벡터를 초기조건에 맞게 생성한 후 FTCS 알고리즘에 따라 시간 t에 대한 열벡터를 추가한다. 결과는 다음과 같다.



또한 코드 구조는 똑같이 하고 시뮬레이션 시간을 길게 하여 t=T일 때부터 결과를 확인했다.

```
#time nodes
t_max = (2*np.pi)/n
t_n = 512+1
t = np.linspace(0, t_max, t_n)
```

t=[0, 2pi]

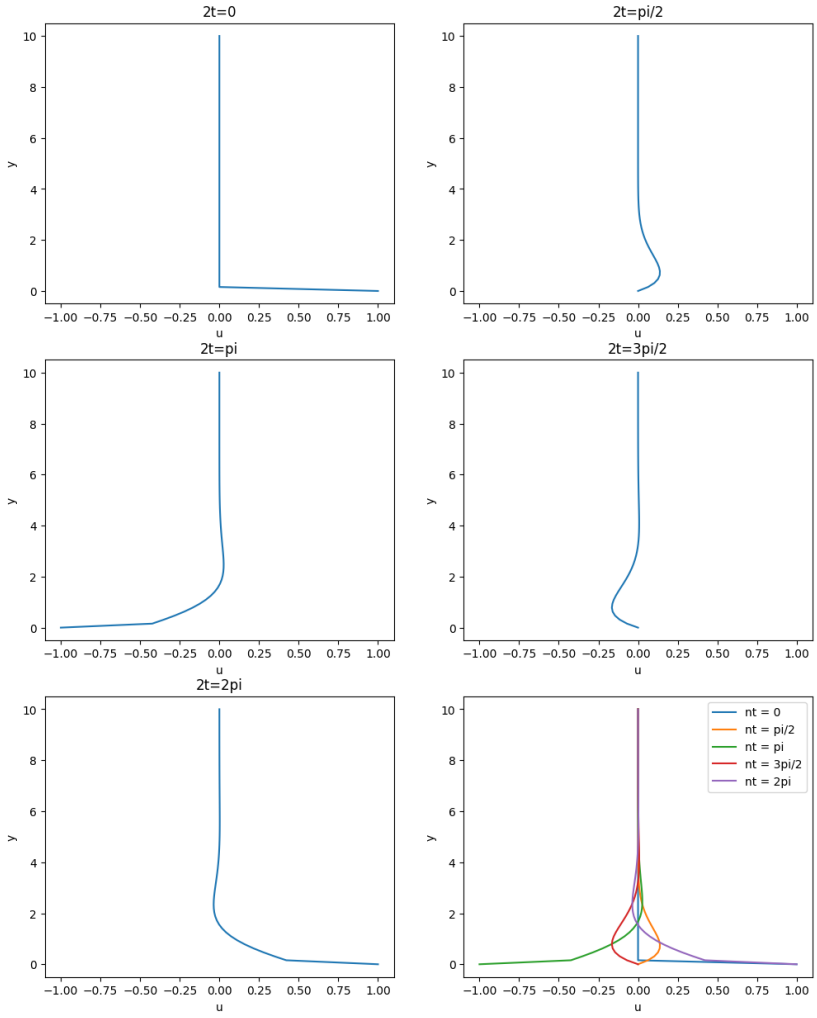
```
#time nodes
T = 10*np.pi
t_max = (2*np.pi+T)/n
t_n = 6*512+1
t = np.linspace(0, t_max, t_n)
```

t=[10pi,12pi]

(2) 다음과 같이 Crank-Nicolson scheme을 구현했다.

```
# coefficient matrix
A = np.zeros((Ny-2,Ny-2))
B = np.zeros((Ny-2,Ny-2))

for i in range(Ny-2):
  A[i,i] = 2*a + 1
  B[i,i] = -2*a +1
  if i > 0:
    A[i, i-1] = -a
    B[i, i-1] = a
  if i < Ny-3:
    A[i, i+1] = -a
    B[i, i+1] = a

#Crank-Nicolson scheme
for i in range(N-1):
  u_col = u[i, 1:-1]
  b = B @ u_col
  b[0] += a * u[i, 0]
  b[-1] += a * u[i, -1]

  u_new = np.linalg.solve(A, b)
  u[i+1, 1:-1] = u_new
  u[i+1,-1] = U_0*np.cos(n*t[i+1])
```

이번에는 matrix u는 Ny x N이고, coefficient matrix가 tridiagonal matrix이므로 위와 같이 코드를 작성해 알고리즘을 구현했다. 결과는 다음과 같다.


2t=0
2t=pi/2
2t=pi
2t=3pi/2
2t=2pi
nt = 0
nt = pi/2
nt = pi
nt = 3pi/2
nt = 2pi
y
u

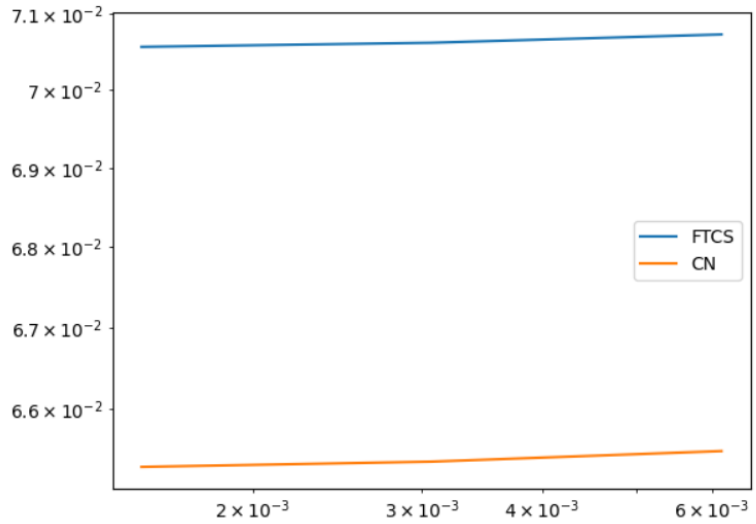
(3) time node의 개수를 513, 1025, 2049로 바꾸어 가며 t=pi/2일 때 exact solution과 numerical solution의 L2 error를 비교했다.

```
t_n = [512+1, 1024+1, 2048+1]
```

```
error_FTCS= []
error_CN = []
delta_t = []

for i in t_n:
  t = np.linspace(0, t_max, i)
  h = t[1] - t[0]
  N = len(t)

  t_index = np.argmin(np.abs(n*t - (np.pi)))

  result_FTCS = FTCS(t,h,N,t_index)
  result_CN = CN(t,h,N,t_index)
  exact = exact_solution(np.flip(y),t[t_index])

  delta_t.append(h)
  error_FTCS.append(error(exact,result_FTCS))
  error_CN.append(error(exact,result_CN))

plt.loglog(delta_t, error_FTCS, label='FTCS')
plt.loglog(delta_t, error_CN, label='CN')
plt.legend()

slope_ftcs = np.polyfit(np.log(delta_t), np.log(error_FTCS), 1)[0]
slope_cn = np.polyfit(np.log(delta_t), np.log(error_CN), 1)[0]

print(f"FTCS 수렴 차수 (기울기): {slope_ftcs:.3f}")
print(f"Crank-Nicolson 수렴 차수 (기울기): {slope_cn:.3f}")
```

결과는 다음과 같다.

```
FTCS 수렴 차수 (기울기): 0.002
Crank-Nicolson 수렴 차수 (기울기): 0.002
```



FTCS의 수렴 차수는 약 1, Crank-Nicolson의 수렴 차수는 약 2가 되어야 하지만, 코드를 실행한 결과 둘 다 수렴 차수가 0에 근접했다. 이는 잘못된 결과이므로 디버깅하려 노력했지만 올바른 답을 찾지 못했다.

(4) 파라미터 중 L을 10 대신 2로 바꾼 후 코드를 실행한 결과는 다음과 같다.

```
#parameters
v = 1
n = 2
U_0 = 1
L = 2
```