In []:	<pre>import numpy as np import matplotlib.pyplot as plt from mpl_toolkits.mplot3d import Axes3D</pre>
	<pre>from scipy import linalg import pandas as pd from scipy.linalg import solve_banded # Set plot params plt.rcParams['figure.figsize'] = [5,5]</pre>
	<pre>plt.rcParams['font.size'] = 10 plt.rcParams['font.family'] = 'Times New Roman' plt.rcParams['axes.linewidth'] = 2 plt.rcParams['lines.linewidth'] = 2 plt.rcParams['xtick.direction'] = 'out' plt.rcParams['ytick.direction'] = 'out'</pre>
	<pre>plt.rcParams['xtick.minor.visible'] = True plt.rcParams['ytick.minor.visible'] = True plt.rcParams['xtick.major.size'] = 7 plt.rcParams['ytick.major.size'] = 7 plt.rcParams['xtick.minor.size'] = 3.5 plt.rcParams['ytick.minor.size'] = 3.5 plt.rcParams['xtick.major.width'] = 1.5</pre>
	<pre>plt.rcParams['ytick.major.width'] = 1.5 plt.rcParams['xtick.minor.width'] = 1.5 plt.rcParams['ytick.minor.width'] = 1.5 plt.rcParams['xtick.top'] = True plt.rcParams['ytick.right'] = True</pre>
	Consider an infinitely extended flat wall carrying out harmonic oscillation in its own plane (see Figure 1). Due to the no-slip condition, the flow velocity at the wall (where $y=0$) is $u(0,t)=U_0cos(nt)$. Check the Exact solution
	<pre>def exact_solution(n, nu, y, t): etas = np.sqrt(n/(2*nu)) * y return np.cos(n*t - etas) * np.exp(-etas) import numpy as np import matplotlib.pyplot as plt</pre>
	<pre># Given parameters nu = 1 n = 2 U0 = 1 L = 10 T = 10 * np.pi dt = 0.001</pre>
	<pre>dy = 0.01 # Discretize the spatial domain y = np.arange(0, L+dy, dy) # Calculate eta</pre>
	<pre>eta = np.sqrt(n / (2 * nu)) * y # Time values for nt and nt - T nts = [0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi] nts_T = [value + T for value in nts] # Calculate the exact solution at nt values</pre>
	<pre># Catcutate the exact solution at ht values u_exact_nt = [U0 * np.exp(-eta) * np.cos(t - eta) for t in nts] u_exact_nt_T = [U0 * np.exp(-eta) * np.cos(n*t - eta) for t in nts_T] # Plot the velocity profiles at nt values for i in range(len(nts)): plt.plot(y, u_exact_nt[i], label=f'nt={nts[i]:.2f}')</pre>
	<pre>plt.legend() plt.xlabel('Position y') plt.ylabel('Velocity v') plt.title('Velocity profiles : exact solution') plt.grid(True) plt.show()</pre>
	<pre># Plot the velocity profiles at nt - T values exacts = [] for i in range(len(nts_T)): data = u_exact_nt_T[i] exacts.append(data) plt.plot(y, data, label=f'(nt-T)={nts[i]:.2f}')</pre>
	<pre>plt.legend() plt.xlabel('Position y') plt.ylabel('Velocity v') plt.title('Velocity profiles : exact solution') plt.grid(True) plt.show()</pre>
	Velocity profiles : exact solution nt=0.00 nt=1.57 nt=3.14
	0.50 nt=4.71 nt=6.28
	-0.25 -0.50
	-0.75 -1.00 0 2 4 6 8 10
	Velocity profiles : exact solution (nt-T)=0.00
	0.75 (nt-T)=1.57 (nt-T)=3.14 (nt-T)=4.71 (nt-T)=6.28
	0.25 -0.25
	-0.50 -0.75 -1.00
	0 2 4 6 8 10 Position y 2. (Numerical analysis)
	 Consider two infinitely long plates placed at y = 0 and y = L. The bottom plate (y = 0) is oscillating with u(0,t) = cos(nt), while the top plate (y = L) is stationary. We aim to obtain velocity profiles u(y, t) between two plates by solving Eq. 1 under the assumptions v = 1, n = 2, U0 = 1 and L = 10. (1) Solve Eq.(1) numerically using the first-order forward difference in time and second-order central difference in space (FTCS scheme). Plot the velocity profiles at nt = 0, π/2,
	π , $3\pi/2$, 2π . Also plot the quasi-steady state velocity profiles at (nt – T) = 0, $\pi/2$, π , $3\pi/2$, 2π . Note that T represents the transient period required to reach the quasi-steady state solutions. You may use T = 10π .
	$rac{u(i+1,j)-u(i,j)}{\Delta t} = rac{ u*(u(i,j+1)-2u(i,j)+u(i,j-1))}{(\Delta y)^2} \ u(i+1,j) = u(i,j) + rac{ u*\Delta t*(u(i,j+1)-2u(i,j)+u(i,j-1))}{\Delta y^2}$
In []:	<pre>def FTCS(T, L, n, nu, dt): """" Forward-Time Central-Space (FTCS) scheme for solving a 1D heat equation. Parameters: - T: Total simulation time - L: Length of the domain</pre>
	 n: Oscillation frequency nu: Diffusion coefficient dt: Time step size Returns: u: Solution array
	<pre>N = 100 M = int(2 * T / dt) y = np.linspace(0, L, N) t = np.linspace(0, 2 * T, M) dy = y[1] - y[0]</pre>
	<pre>dy = y[1] - y[0] # Initialize u array with boundary conditions u = np.zeros((M, N)) u[:, 0] = np.cos(n * t) u[:, -1] = 0 # FTCS scheme</pre>
	<pre>for i in range(M - 1): for j in range(1, N - 1): u[i + 1, j] = u[i, j] + nu * (dt / dy**2) * (u[i, j + 1] - 2 * u[i, j] + u[i, j - 1]) return u</pre>
In []:	# Given parameters nu = 1 n = 2 U0 = 1 L = 10 T = 10*np.pi nts = np.pri/2
In 「¹	<pre>T = 10*np.pi nts = np.array([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi]) dt = 0.001 N = 100 y = np.linspace(0, L, N)</pre> u = FTCS(T, L, n, nu, dt)
[]:	<pre># Plotting for initial for nt in nts: # t = nt/n nt = nt/n # for dataframe</pre>
	<pre>data = u[int(nt/dt), :] plt.plot(y, data, label=f'nt = {nt:.2f}, initial') plt.legend(loc = 'upper right') plt.xlabel('Position y') plt.ylabel('Velocity v') plt.title('Velocity profiles') plt.grid(True)</pre>
	<pre>plt.show() # Plotting for quasi-steady for nt in nts: idx = int((nt+T) / dt) data = u[idx, :] plt.plot(y, data, label=f'(nt-T) = {nt:.2f}, quasi-steady')</pre>
	Velocity profiles nt = 0.00, initial nt = 0.79, initial nt = 1.57, initial
	0.50
	-0.25 -0.50
	-0.75 -1.00 0 2 4 6 8 10
	Position y Velocity profiles (nt-T) = 0.00, quasi-steady (nt-T) = 1.57, quasi-steady
	(nt-T) = 3.14, quasi-steady (nt-T) = 4.71, quasi-steady (nt-T) = 6.28, quasi-steady
	0.00 -0.25
	-0.50 -0.75 -1.00
	0 2 4 6 8 10 Position y Crank-Nicolson scheme
	(2) Repeat (1) using the Crank-Nicolson (C-N) scheme in time. A 행렬 생성 $lpha=1/2$ upper diagonal: $rac{- u*\Delta t}{(4*\Delta y^2)}$ diagonal : $rac{1+ u*\Delta t}{(2*\Delta y^2)}$
In []:	lower diagonal : $\frac{-\nu*\Delta t}{(4*\Delta y^2)}$ b 벡터 생성: $b=u[i,1:-1]+\nu*\Delta t/(4*\Delta y^2)*(u[i,:-2]-2*u[i,1:-1]+u[i,2:])$ def CN(T, L, n, nu, dt):
	Crank-Nicolson scheme for solving a 1D heat equation. Parameters: - T: Total simulation time - L: Length of the domain - n: Oscillation frequency - nu: Diffusion coefficient
	- dt: Time step size Returns: - u_cn: Solution array
	<pre>N = 100 M = int(2 * T / dt) y = np.linspace(0, L, N) t = np.linspace(0, 2 * T, M) dy = y[1] - y[0] # Initialize u_cn array with boundary conditions</pre>
	<pre>u_cn = np.zeros((M, N)) u_cn[:, 0] = np.cos(n * t) u_cn[:, -1] = 0 # Crank-Nicolson scheme for i in range(M - 1):</pre>
	<pre># Create A matrix for Crank-Nicolson A_upper = -nu * dt / (4 * dy**2) * np.ones(N - 2) A_mid = np.ones(N - 2) + nu * dt / (2 * dy**2) A_lower = -nu * dt / (4 * dy**2) * np.ones(N - 2) A = np.vstack((A_upper, A_mid, A_lower)) # Create b vector for Crank-Nicolson</pre>
	<pre>b = u_cn[i, 1:-1] + nu * dt / (4 * dy**2) * (u_cn[i, :-2] - 2 * u_cn[i, 1:-1] + u_cn[i, 2:]) # Solve Ax = b u_cn[i + 1, 1:-1] = solve_banded((1, 1), A, b) return u_cn</pre>
In []:	<pre>u_cn = CN(T, L, n, nu, dt) # Plotting for nt in nts: #t = nt/n nt = nt/n</pre>
	<pre>plt.plot(y, u_cn[int(nt/dt), :], label=f'nt = {nt:.2f}, initial') plt.legend() plt.xlabel('Position y') plt.ylabel('Velocity v') plt.title('Velocity profiles') plt.grid(True)</pre>
	$r_{1}+r_{2}-r_{3}$
	<pre>plt.show() # Plotting for nt in nts: idx = int((nt+T) / dt) data = u_cn[idx, :] plt.plot(y, data, label=f'(nt-T) = {nt:.2f}, quasi-steady')</pre>
	<pre># Plotting for nt in nts: idx = int((nt+T) / dt) data = u_cn[idx, :]</pre>
	<pre># Plotting for nt in nts: idx = int((nt+T) / dt) data = u_cn[idx, :] plt.plot(y, data, label=f'(nt-T) = {nt:.2f}, quasi-steady') plt.legend() plt.xlabel('Position y') plt.ylabel('Velocity v') plt.title('Velocity profiles') plt.grid(True) plt.show()</pre> Velocity profiles 1.00
	<pre># Plotting for nt in nts: idx = int((nt+T) / dt) data = u_cn[idx, :] plt.plot(y, data, label=f'(nt-T) = {nt:.2f}, quasi-steady') plt.legend() plt.xlabel('Position y') plt.ylabel('Velocity v') plt.title('Velocity profiles') plt.grid(True) plt.show()</pre> Velocity profiles 1.00 nt = 0.00, initial nt = 0.79, initial nt = 1.57, initial nt = 2.36, initial nt = 2.36, initial nt = 3.14, init
	<pre># Plotting for nt in nts: idx = int((nt+T) / dt) data = u_cn[idx, :] plt.plot(y, data, label=f'(nt-T) = {nt:.2f}, quasi-steady') plt.legend() plt.xlabel('Position y') plt.ylabel('Velocity v') plt.grid(True) plt.show()</pre> Velocity profiles 1.00 nt = 0.00, initial nt = 1.57, initial nt = 1.57, initial nt = 2.36, initial nt = 3.14, initial nt = 3.14, initial
	# Plotting for nt in nts: idx = int((nt+T) / dt) data = u_cn[idx, :] plt.plot(y, data, label=f'(nt-T) = {nt:.2f}, quasi-steady') plt.legend() plt.vlabel('velocity v') plt.title('velocity profiles') plt.grid(True) plt.show() Velocity profiles 1.00
	# Plotting for nt in nts:
	## Plotfing for nt is nts:
	### Plotting for nt is mati(state) / sty
	# Plotting for at ints:
	### Plating for a Intilicati / data ### Location (%) data = Location (%) ### Structure (%) ### Structur
In []:	### (###
	### Final Control
	### ret is 100 cm 1
	Section Sect
	### ### #### #########################
	(d) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (d) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (e) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (f) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap distance between the two plates on the velocity profiles. (g) Sepacit (2) with L = 2. Discuss the effect of the gap
	### Company of the Co
	Final Control of the
	### (A) Propose (2) with 1 = 2 Decease the effect of the gap distance between the two paths on the wicedly profiles. #### (A) Propose (2) with 1 = 2 Decease the effect of the gap distance between the two paths on the wicedly profiles. ###################################
	(c) Proposed (2) colon 1 = 2. Declares the effect of the gas distance between the two plates on the verosity profiles. (d) Proposed (2) colon 1 = 2. Declares the effect of the gas distance between the two plates on the verosity profiles. (e) Colon (2) co
	### 12 10 10 10 10 10 10 10
	### A Company of the
	Formula is the second of the s
	And seemed at the control of the con
	The property of the control of the c
	Supplementary and the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance answers the recipion or the or only problem. Supplementary is the state of the gap distance and the recipion of the or only problem. Supplementary is the state of the gap distance and the recipion of the or only problem. Supplementary is the state of the gap distance and the recipion of the or only problem. Supplementary is the state of the gap distance and the recipion of the or only problem. Supplementary is the state of the gap distance and the recipion of the or only problem. Supplementary is the state of the gap distance and the recipion of the or only problem. Supplementary is the state of the gap distance and the recipion of the or only problem. Supplementary is the state of the gap distance and the recipion of the or only problementary is the gap distance and the gap distance and the recipion of the or only problementary is the gap distance and the gap d
	The second of th
	Compared
	Control of the contro
	### Company of the Co
	And the cost of th
	He have a protect of the protect of
	Polygonia and the control of the con
	The state of the s
	Polyment of the control of the contr
	A SAME CANADA SAME
	The state of the s
	All Street (Day to 1) and the street of the
	ADMINISTRATION OF THE PROPERTY
	The state of the s
	Selection of the select

HW6. Stokes second problem