ME 4565: Introduction to Computational Fluid Dynamics

Spring 2024

Problem Set #8

Due March 29th, 5:00 PM (Submit to Canvas)

Problem 1 - Relaxation Method:
For the nonlinear second order ODE

$$\frac{d^2\phi}{dx^2} + \left(\frac{d\phi}{dx}\right)^2 = 1 - \phi - \phi^2$$

a. Derive the relaxation equation by plugging in $\phi(x) = \phi_g(x) + \zeta(x)$. Eliminate the higher order ζ terms and rearrange the equation so that only terms that depend on ζ are on the left hand side.

(Hand derivation)

$$\frac{d^2\phi}{dx^2} + \frac{d\phi}{dx}\left(\frac{d\phi}{dx}\right) = 1 - \phi - \phi^2 \qquad \phi = \phi_g + \zeta$$

$$\frac{d^2\phi_g}{dx^2} + \frac{d^2\zeta}{dx^2} + \left(\frac{d\phi_g}{dx}\right)^2 + 2\frac{d\phi_g}{dx}\frac{d\zeta}{dx} + \left(\frac{d\zeta}{dx}\right)^2 = 1 - (\phi_g + \zeta) - (\phi_g + \zeta)^2$$

$$\frac{d^2\phi_g}{dx^2} + \frac{d^2\zeta}{dx^2} + \left(\frac{d\phi_g}{dx}\right)^2 + 2\frac{d\phi_g}{dx}\frac{d\zeta}{dx} + \left(\frac{d\zeta}{dx}\right)^2 = 1 - \phi_g - \zeta - (\phi_g^2 + 2\phi_g\zeta + \zeta^2)$$

$$\frac{d^2\phi_g}{dx^2} + \frac{d^2\zeta}{dx^2} + \left(\frac{d\phi_g}{dx}\right)^2 + 2\frac{d\phi_g}{dx}\frac{d\zeta}{dx} + \cancel{\left(\frac{d\zeta}{dx}\right)^2} = 1 - \phi_g - \zeta - \phi_g^2 - 2\phi_g\zeta - \cancel{\zeta^2}$$

$$\boxed{\frac{d^2\zeta}{dx^2} + \zeta(1 + 2\phi_g) + 2\frac{d\phi_g}{dx}\frac{d\zeta}{dx} = 1 - \frac{d^2\phi_g}{dx^2} - \left(\frac{d\phi_g}{dx}\right)^2 - \phi_g - \phi_g^2}$$

b. Discretize ζ and ϕ_g and write the matrix equation corresponding to the FDE for a uniform grid. Assume the boundary conditions are $\phi(x = 0) = 1$ and $\phi(x = \pi) = -1$.

(Hand derivation)

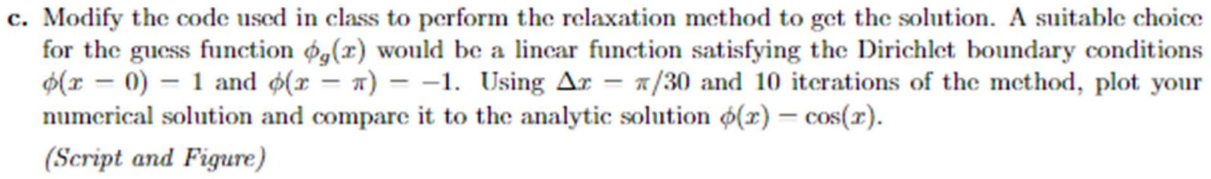
$$\frac{d^2\zeta}{dx^2} + \zeta(1 + 2\phi_g) + 2\frac{d\phi_g}{dx}\frac{d\zeta}{dx} = 1 - \frac{d^2\phi_g}{dx^2} - \left(\frac{d\phi_g}{dx}\right)^2 - \phi_g - \phi_g^2$$

$$\underline{\underline{D_2}}\,\zeta + \underline{\zeta}\left(\underline{\underline{\text{ones}}} + \text{diag}(2\underline{\phi_g})\right) + \text{diag}\left(2\underline{\underline{D_1}}\,\underline{\phi_g}\right)\underline{\underline{D_1}}\,\underline{\zeta} = \text{ones} - \underline{\underline{D_2}}\,\underline{\phi_g} - \text{diag}(\underline{\underline{D_1}}\,\phi_g)(\underline{\underline{D_1}}\,\phi_g) - \underline{\phi_g} - \text{diag}(\underline{\phi_g})(\phi_g)$$

$$\underbrace{\left[\underline{\underline{D_2}} + \left(\left(\underline{\underline{\text{ones}}} + \text{diag}(2\underline{\phi_g})\right) + \text{diag}(2\underline{\underline{D_1}}\,\underline{\phi_g})\underline{\underline{D_1}}\right]}_{\vec{A}\ \text{interior pts}}\ \zeta = \underbrace{\text{ones} - \underline{\underline{D_2}}\,\underline{\phi_g} - \text{diag}(\underline{\underline{D_1}}\,\phi_g)(\underline{\underline{D_1}}\,\phi_g) - \underline{\phi_g} - \text{diag}(\underline{\phi_g})(\phi_g)}_{\vec{b}\ \text{interior pts}}$$

$$\begin{array}{l} \text{First row} \\ \\ \text{Last row} \end{array} \begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots \\ & & & & \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \zeta_1 \\ \\ \zeta_n @\pi \end{bmatrix} = \begin{bmatrix} 0 \\ \\ 0 \end{bmatrix} \Bigg]$$

using the boundery Conditions



c. Modify the code used in class to perform the relaxation method to get the solution. A suitable choice for the guess function $\phi_g(x)$ would be a linear function satisfying the Dirichlet boundary conditions $\phi(x = 0) = 1$ and $\phi(x = \pi) = -1$. Using $\Delta x = \pi/30$ and 10 iterations of the method, plot your numerical solution and compare it to the analytic solution $\phi(x) = \cos(x)$.

(Script and Figure)

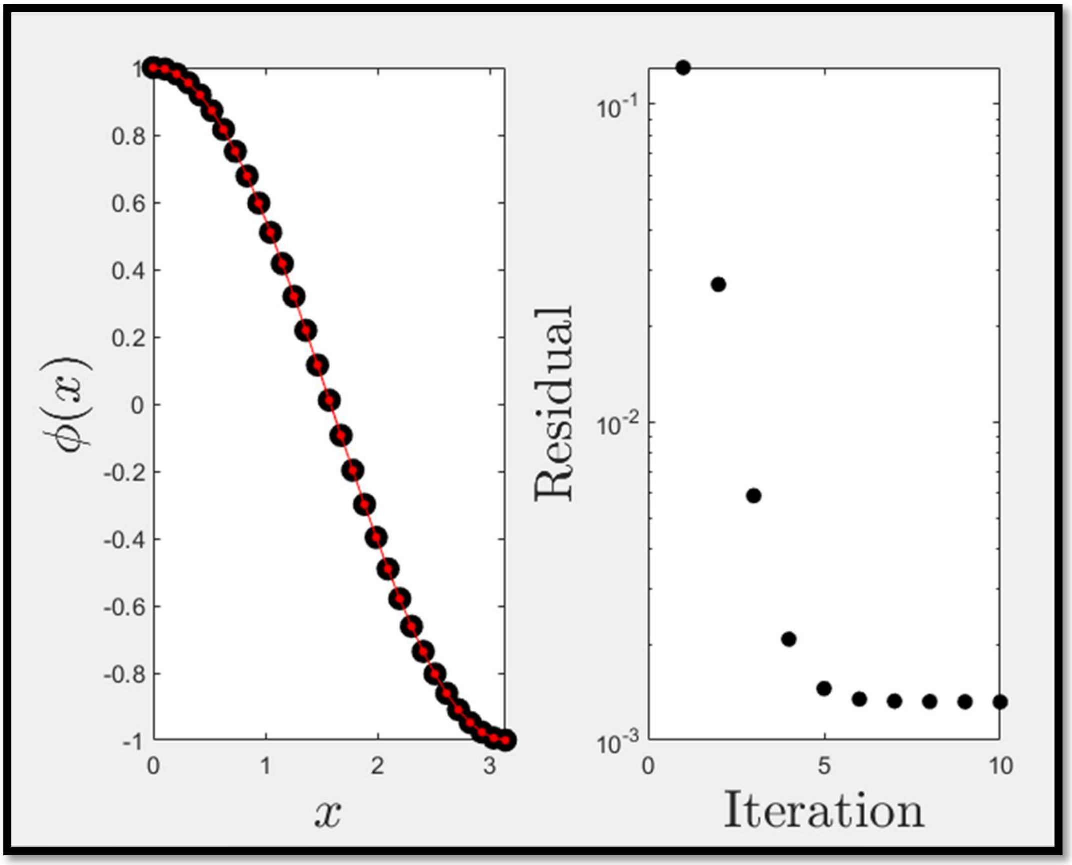


Figure 1: $\phi(x)$ vs x plot with residual vs iteration subplot for Problem 1c. This figure was produced based on derivations from 1a and 1b.

Problem 1c Script: HW8P1C.m

```
% Problem 1 Part C
clear; close all; clc;

% Sets the number of iterations for the relaxation method to 10.
nIter =  10;

% Creates the spatial domain for the problem with a specified step size, and makes
'x' a column vector.
dx =  pi/30;
x  = (0:dx:pi)';
nX = length(x);
```

```
% Initializes the guess for the solution ('phi_g') that satisfies the given boundary
conditions at x=0 and x=pi.
phi_g = -2/pi*x + ones(nX,1);

% Initializes the first and second derivative matrices, D1 and D2, without including
boundary conditions yet.
D1 = zeros(nX,nX); % First derivative matrix with central difference, except at
boundaries.
D2 = zeros(nX,nX); % Second derivative matrix with central difference, except at
boundaries.

% Defines derivative approximations at the left boundary using forward difference.
D1(1, 1:3) = [-3 4 -1]/dx/2;
D2(1, 1:4) = [2 -5 4 -1]/dx^2;

% Sets up central difference approximations for interior points.
for i = 2:nX-1
    D1(i, i-1:i+1) = [-1 0 1]/dx/2;
    D2(i, i-1:i+1) = [1 -2 1]/dx^2;
end

% Defines derivative approximations at the right boundary using backward difference.
D1(nX, nX-2:nX) = [1 -4 3]/dx/2;
D2(nX, nX-3:nX) = [-1 4 -5 2]/dx^2;

% Iterates through the relaxation process to solve the BVP.
for i = 1:nIter

    % Constructs the 'A' matrix from the differential operators and the current
guess, according to the specific problem.
    A = D2 + diag(2*D1*phi_g)*D1 + (ones(nX, 1) + diag(2*phi_g));

    % Constructs the 'b' vector, representing the right-hand side of the differential
equation.
    b = ones(nX, 1) - D2*phi_g - (D1*phi_g).^2 - phi_g - phi_g.^2;

    % Modifies the 'A' matrix and 'b' vector to incorporate boundary conditions for
the correction.
    A(1,:) = 0; A(1, 1) = 1; b(1) = 0; % Boundary condition at x=0.
    A(nX, :) = 0; A(nX, nX) = 1; b(nX) = 0; % Boundary condition at x=pi.

    % Solves the linear system A*zeta = b to find the correction 'zeta'.
    zeta = A\b;

    % Updates the guess by adding the correction 'zeta'.
    phi_g = phi_g + zeta;

    % Calculates the residual of the ODE to monitor convergence.
    R = D2*phi_g + (D1*phi_g).^2 + phi_g + phi_g.^2 - 1;
    res(i) = mean(abs(R(2:end-1))); % Ignores boundary points.

    % Plots the current guess and the updated guess for visualization, as well as the
convergence of the residual.
    subplot(121);
    hold off
```

Problem 2 - Transient Advection Equation with a Source:
For this problem, we are going to develop a model for the transient advection equation with a source

$$\frac{\partial \phi}{\partial t} + c\frac{\partial \phi}{\partial x} = \underset{\hookrightarrow \Delta t q^{i}}{q(x,t)} \qquad \text{EEBS} \qquad \underset{\hookrightarrow q_i}{q(x,t)}$$

a. Derive the Explicit Euler, $O(\Delta t)$, backward-space (left-sided, $O(\Delta x)$) discretization for this PDE. *(Hand derivation)*

$$\hookrightarrow \frac{\phi_i - \phi_{i-1}}{\Delta x}$$

$$\frac{\partial \phi}{\partial t} = -c\frac{\partial \phi}{\partial x} + q(x,t)$$

$$\phi_i^{n+1} = \phi_i^n + \Delta t\left(\frac{\partial \phi}{\partial t}\right)_i^n \rightsquigarrow \phi_i^{n+1} = \phi_i^n + \Delta t\left(-c\frac{\partial \phi}{\partial x}{}_i^n + q(x,t)_i^n\right)$$

$$\phi_i^{n+1} = \phi_i^n + \Delta t\left(-c\,\frac{\phi_i^n - \phi_{i-1}^n}{\Delta x} + q\left(x^n, t_i\right)\right)$$

$$\boxed{\phi_i^{n+1} = \phi_i^n - \underbrace{\frac{c\Delta t}{\Delta x}}_{\alpha}\left(\phi_i^n - \phi_{i-1}^n\right) + \Delta t q_i^n}$$

$$\phi_i^{n+1} = \phi_i^n - \alpha\phi_i^n + \alpha\phi_{i-1}^n + q\Delta t(x_i^n, t_i^n)$$
$$= (1-\alpha)\phi_i^n + \alpha(\phi_{i-1}^n) + \ldots$$

b. Given the spatial discretization $x = 0 : 0.2 : 1$ (6-point discretization), the boundary condition $\phi(x = 0, t) = 1$, and the initial condition $\phi(x = 0, t = 0) = 1$ and $\phi(x \neq 0, t = 0) = 0$, write the system of equations that you will use to determine ϕ_i^{n+1} for each node i. *(Hand derivation)*

$0, 0.2, 0.4, 0.6, 0.8, 1$
$\hookrightarrow$ 6 pts

$i=2$
$$\phi_i^{n+1} = (1-\alpha)\phi_2^n - \alpha(\phi_1)^n$$

Boundary Condition: $\phi(x=0, t) = 1$

Initial Conditions: $\phi(x=0, t=0) = 1$ & $\phi(x \neq 0, t=0) = 0$] whenever $t=0$ & any x value → 0.

$$i=2 \rightarrow \phi_2^{n+1} = \phi_2^n(1-\alpha) + \Delta t q_2^n + \alpha\phi_1$$

$$\begin{array}{cccccccccc}
\underline{\phi_{n+1}} & & & & \underline{\underline{B}} & & & \underline{\phi_n} & & \underline{b} \\
 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & & & \\
\phi_1^{n+1} & 0 & 0 & 0 & 0 & 0 & 0 & \phi_1^n & & 1 \\
\phi_2^{n+1} & +\alpha & 1-\alpha & 0 & 0 & 0 & 0 & \phi_2^n & & q_2^n\Delta t \\
\phi_3^{n+1} = & 0 & +\alpha & 1-\alpha & 0 & 0 & 0 & \phi_3^n & + & q_3^n\Delta t \\
\phi_4^{n+1} & 0 & 0 & +\alpha & 1-\alpha & 0 & 0 & \phi_4^n & & q_4^n\Delta t \\
\phi_5^{n+1} & 0 & 0 & 0 & +\alpha & 1-\alpha & 0 & \phi_5^n & & q_5^n\Delta t \\
\phi_6^{n+1} & 0 & 0 & 0 & 0 & +\alpha & 1-\alpha & \phi_6^n & & q_6^n\Delta t
\end{array}$$

] b/c no boundary condition given

c. For the parameters $c = 0.5$, $\Delta t = 0.1$ and $q(x,t) \equiv 0$ (no source), perform three time steps by hand to approximate $\phi(x, t = 0.3)$.

(Hand derivation)

$\Delta x = 0.2$

$\Delta t = 0.1$

$$\phi_i^{n+1} = \phi_i^n - \underbrace{\frac{C\Delta t}{\Delta x}}_{\alpha}(\phi_i^n - \phi_{i-1}^n) + \overset{0}{\cancel{\Delta t q_i^n}}$$

$$\phi_i^{n+1} = \phi_i^n - \alpha(\phi_i^n) + \alpha(\phi_{i-1}^n)$$

$$\phi_i^{n+1} = (1-\alpha)\phi_i^n + \alpha(\phi_{i-1}^n)$$

$$= 0.75\underbrace{\phi_i^n}_{} + 0.25\underbrace{\phi_{i-1}^n}_{}$$

2 not3 we care (no i+1)

$$\underset{1}{\alpha} = \frac{C\Delta t}{\Delta x} = \frac{0.5(0.1)}{0.2} = 0.25$$

$\phi(x, t = 0.1)$

$\phi(x, t = 0.2)$

$\phi(x, t = 0.3)$

$t_0 = 0$		$t_1 = 0.1$		$t_2 = 0.2$		$t_3 = 0.3$	
ϕ_1^0	$1 \rightarrow$	ϕ_1^1	1	ϕ_1^2	1	ϕ_1^3	1
ϕ_2^0	$0 \nearrow$	ϕ_2^1	0.25	ϕ_2^2	0.4375	ϕ_2^3	0.578125
ϕ_3^0	$0 \nearrow$	ϕ_3^1	0	ϕ_3^2	0.0625	ϕ_3^3	0.15625
ϕ_4^0	0	ϕ_4^1	0	ϕ_4^2	0	ϕ_4^3	0.015625
ϕ_5^0	0	ϕ_5^1	0	ϕ_5^2	0	ϕ_5^3	0
ϕ_6^0	0	ϕ_6^1	0	ϕ_6^2	0	ϕ_6^3	0

$$\phi_2^{\overset{\overset{0+1}{\downarrow}}{n+1}} = 0.75\phi_2^{\overset{\overset{0}{\downarrow}}{n}} + 0.25\phi_1^{\overset{\overset{0}{\downarrow}}{n}}$$

$$0.75\phi_2^0 + 0.25\phi_1^0$$

$$= 0 + 1 = 1$$

at $t_2 = 0.2$

$$\phi_2^2 = 0.75\phi_2^1 + 0.25\phi_1^1$$

$$= 0.75(0.25) + 0.25(1)$$

$$= 0.4375$$

$$\phi_3^2 = 0.75\phi_3^1 + 0.25\phi_2^1$$

$$= 0.75(0) + 0.25(0.25)$$

$$= 0.0625$$

$t_3 = 0.3$

$$\phi_2^3 = 0.75\phi_2^2 + 0.25\phi_1^2$$

$$= 0.75(0.4375) + 0.25(1)$$

$$= 0.578125$$

$$\phi_3^3 = 0.75\phi_3^2 + 0.25\phi_2^2$$

$$= 0.75(0.0625) + 0.25(0.4375)$$

$$= 0.15625$$

$$\phi_4^3 = 0.75\phi_4^2 + 0.25\phi_3^2$$

$$= 0.75(0) + 0.25(0.0625)$$

$$= 0.015625$$

$$\phi_5^3 = 0.75\phi_5^2 + 0.25\phi_4^2 = 0$$

d. For your system, transform your system in a matrix equation, which is of the form

$$\underline{\phi}^{n+1} = \underline{\underline{B}}\,\underline{\phi}^{n} + \underline{b}.$$

Repeat the three steps using your matrix equation (feel free to code up the matrix multiplication) and compare to your result from **c**.

(Hand derivation)

$$\phi_i^{n+1} = \phi_i^{n} - \underbrace{\frac{C\Delta t}{\Delta x}}_{\alpha}\left(\phi_i^{n} + \phi_{i-1}^{n}\right) + \underbrace{q\,\Delta t\,(x,t)_i^{n}}_{0}$$

$\Delta t = 0.1$

$$\phi_i^{n+1} = \phi_i^{n} - \alpha\left(\phi_i^{n}\right) + \alpha\left(\phi_{i-1}^{n}\right)$$

$\phi(x, t=0.1)$

$\phi(x, t=0.2)$

$$\phi_i^{n+1} = (1-\alpha)\phi_i^{n} + \alpha\left(\phi_{i-1}^{n}\right)$$

$\phi(x, t=0.3)$

$$= 0.75\,\phi_i^{n} + 0.25\,\phi_{i-1}^{n}$$

$$\alpha = \frac{C\Delta t}{\Delta x} = \frac{0.5(0.1)}{0.2} = 0.25$$

$$\phi^{n+1} = B \cdot \phi^{n}$$

$\underline{\underline{A}}$ $t = 0.1$ $\underline{\Phi_{n+1}}$ $\underline{b}$

$$\begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ \end{array}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0.25 & 0.75 & 0 & 0 & 0 & 0 \\ 0 & 0.25 & 0.75 & 0 & 0 & 0 \\ 0 & 0 & 0.25 & 0.75 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0.75 & 0 \\ 0 & 0 & 0 & 0 & 0.25 & 0.75 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0. \end{bmatrix} = \begin{bmatrix} 1 \\ 0.25 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\phi^{n+2} = B \cdot \phi^{n+1}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0.25 & 0.75 & 0 & 0 & 0 & 0 \\ 0 & 0.25 & 0.75 & 0 & 0 & 0 \\ 0 & 0 & 0.25 & 0.75 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0.75 & 0 \\ 0 & 0 & 0 & 0 & 0.25 & 0.75 \end{bmatrix} \begin{bmatrix} 1 \\ 0.25 \\ 0 \\ 0 \\ 0 \\ 0. \end{bmatrix} = \begin{bmatrix} 1 \\ 0.4375 \\ 0.0625 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

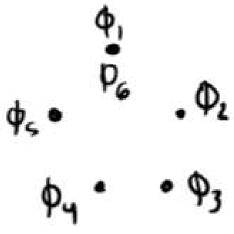
$$\phi^{n+3} = B \cdot \phi^{n+2}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0.25 & 0.75 & 0 & 0 & 0 & 0 \\ 0 & 0.25 & 0.75 & 0 & 0 & 0 \\ 0 & 0 & 0.25 & 0.75 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0.75 & 0 \\ 0 & 0 & 0 & 0 & 0.25 & 0.75 \end{bmatrix} \begin{bmatrix} 1 \\ 0.4375 \\ 0.0625 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.578125 \\ 0.15625 \\ 0.015625 \\ 0 \\ 0 \end{bmatrix}$$

e. We are going to change the Dirichlet boundary condition at $x = 0$ to periodic boundary conditions between $x = 0$ and $x = 1$. Which equations in your system need to be changed? What is the general form of the modified matrix equation that accounts for periodicity?

(Hand derivation)

Periodic boundary conditions



$$\frac{\partial \phi}{\partial t} + c\frac{\partial \phi}{\partial x} = q(x,t)$$

$$\downarrow$$

$$\phi_i^{n+1} = \phi_i^n - \frac{C\Delta t}{\Delta x}\left(\phi_i^n - \phi_{i-1}^n\right) + q\Delta t\,(x,t)$$

$i=1$

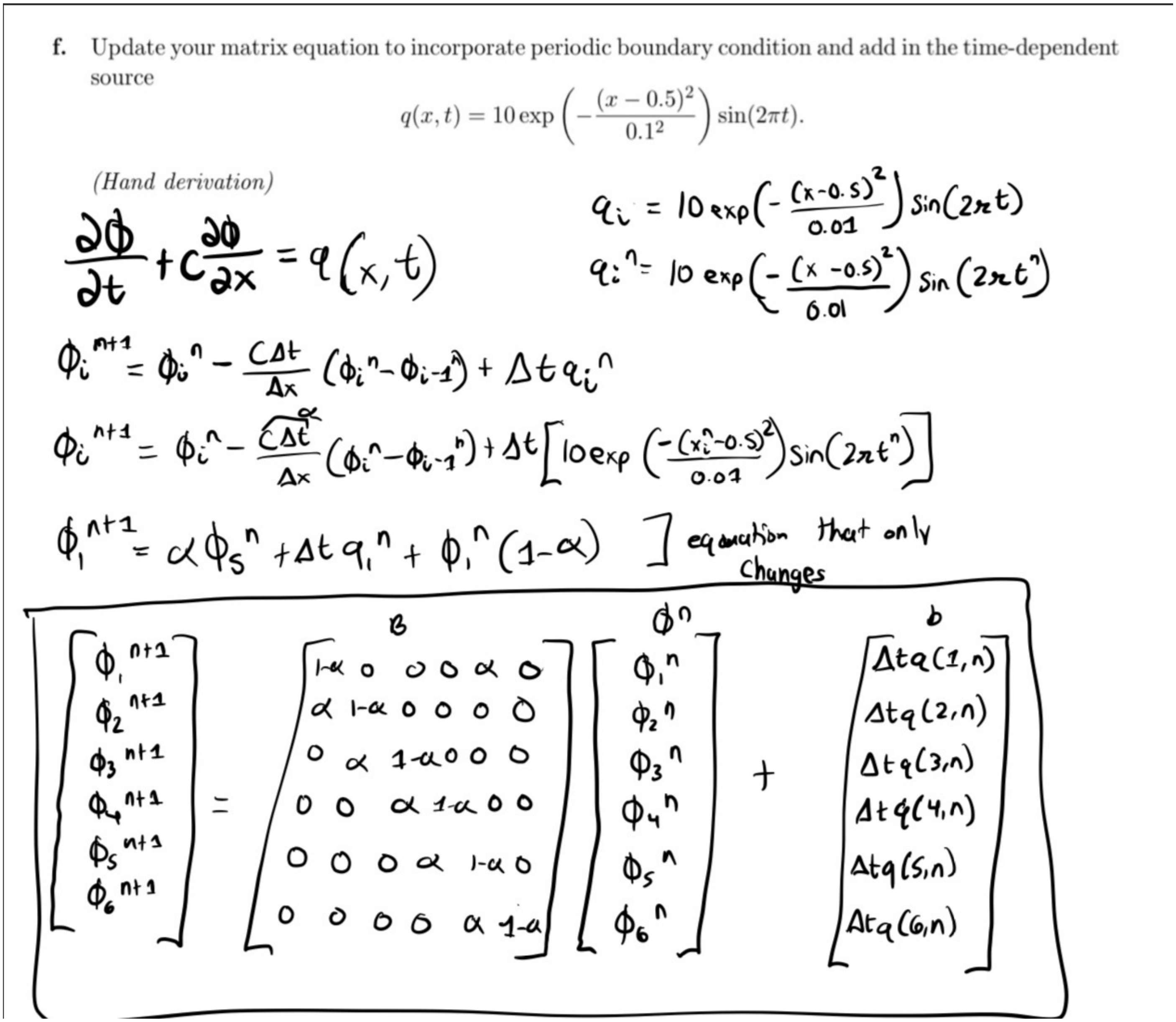
$$\phi_1^{n+1} = \phi_1^n - \frac{C\Delta t}{\Delta x}\left(\phi_1^n - \phi_5^n\right) + \Delta t\, q_1^n \qquad (1) \rightsquigarrow \text{only new equation}$$

$$\hookrightarrow \phi_1^{n+1} = \phi_1^n - \alpha\phi_1^n + \alpha\phi_5^n + \Delta t\, q_1^n = \phi_1^n(1-\alpha) + \alpha\phi_5^n + \Delta t\, q_1^n$$

$$\phi_6^{n+1} = \phi_6^n - \underline{\frac{C\Delta t}{\Delta x}}\left(\phi_6^n - \phi_5^n\right) + \Delta t\, q_6^n$$

$$\underline{\phi_{n+1}} \qquad \underline{\underline{B}} \qquad \underline{\phi_n} \qquad \underline{b}$$

$$\begin{array}{c} \\ \phi_1^{n+1} \\ \phi_2^{n+1} \\ \phi_3^{n+1} \\ \phi_4^{n+1} \\ \phi_5^{n+1} \\ \phi_6^{n+1} \end{array} = \begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 1-\alpha & 0 & 0 & 0 & \alpha & 0 \\ +\alpha & 1-\alpha & 0 & 0 & 0 & 0 \\ 0 & +\alpha & 1-\alpha & 0 & 0 & 0 \\ 0 & 0 & +\alpha & 1-\alpha & 0 & 0 \\ 0 & 0 & 0 & +\alpha & 1-\alpha & 0 \\ 0 & 0 & 0 & 0 & +\alpha & 1-\alpha \end{array} \begin{array}{c} \\ \phi_1^n \\ \phi_2^n \\ \phi_3^n \\ \phi_4^n \\ \phi_5^n \\ \phi_6^n \end{array} + \begin{array}{c} \\ q_1^n\Delta t \\ q_2^n\Delta t \\ q_3^n\Delta t \\ q_4^n\Delta t \\ q_5^n\Delta t \\ q_6^n\Delta t \end{array}$$



f. Update your matrix equation to incorporate periodic boundary condition and add in the time-dependent source

$$q(x,t) = 10\exp\left(-\frac{(x-0.5)^2}{0.1^2}\right)\sin(2\pi t).$$

(Hand derivation)

$$q_i = 10\exp\left(-\frac{(x-0.5)^2}{0.01}\right)\sin(2\pi t)$$

$$q_i^n = 10\exp\left(-\frac{(x-0.5)^2}{0.01}\right)\sin(2\pi t^n)$$

$$\frac{\partial\phi}{\partial t} + c\frac{\partial\phi}{\partial x} = q(x,t)$$

$$\phi_i^{n+1} = \phi_i^n - \frac{C\Delta t}{\Delta x}(\phi_i^n - \phi_{i-1}^n) + \Delta t\, q_i^n$$

$$\phi_i^{n+1} = \phi_i^n - \frac{\overset{\alpha}{\overbrace{C\Delta t}}}{\Delta x}(\phi_i^n - \phi_{i-1}^n) + \Delta t\left[10\exp\left(\frac{-(x_i^n-0.5)^2}{0.01}\right)\sin(2\pi t^n)\right]$$

$$\phi_1^{n+1} = \alpha\phi_5^n + \Delta t\, q_1^n + \phi_1^n(1-\alpha)$$] equation that only changes

$$\begin{bmatrix}\phi_1^{n+1}\\ \phi_2^{n+1}\\ \phi_3^{n+1}\\ \phi_4^{n+1}\\ \phi_5^{n+1}\\ \phi_6^{n+1}\end{bmatrix} = \overset{B}{\begin{bmatrix}1-\alpha & 0 & 0 & 0 & \alpha & 0\\ \alpha & 1-\alpha & 0 & 0 & 0 & 0\\ 0 & \alpha & 1-\alpha & 0 & 0 & 0\\ 0 & 0 & \alpha & 1-\alpha & 0 & 0\\ 0 & 0 & 0 & \alpha & 1-\alpha & 0\\ 0 & 0 & 0 & 0 & \alpha & 1-\alpha\end{bmatrix}} \overset{\phi^n}{\begin{bmatrix}\phi_1^n\\ \phi_2^n\\ \phi_3^n\\ \phi_4^n\\ \phi_5^n\\ \phi_6^n\end{bmatrix}} + \overset{b}{\begin{bmatrix}\Delta t q(1,n)\\ \Delta t q(2,n)\\ \Delta t q(3,n)\\ \Delta t q(4,n)\\ \Delta t q(5,n)\\ \Delta t q(6,n)\end{bmatrix}}$$

g. Given the new initial condition $\phi(x, t = 0) \equiv 0$ everywhere, and the same c, Δx, and Δt as used above, simulate the temperature profile and plot the profile at $t = 1$.

(Script and Figure)

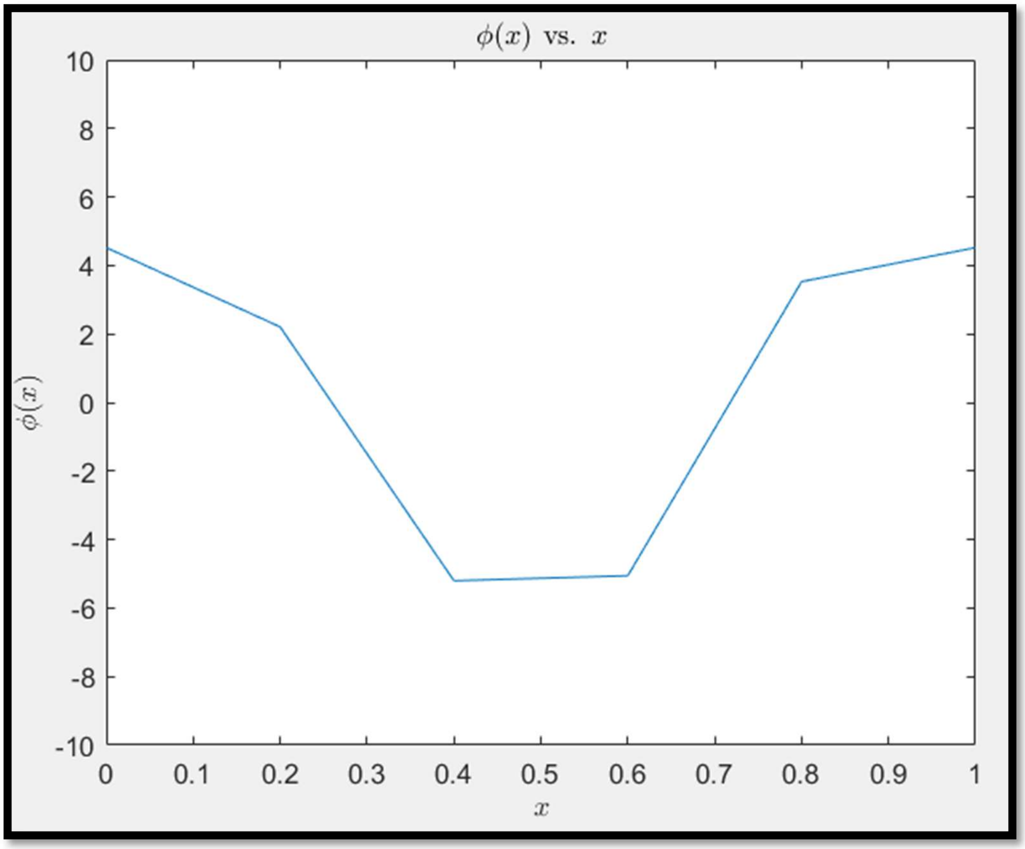
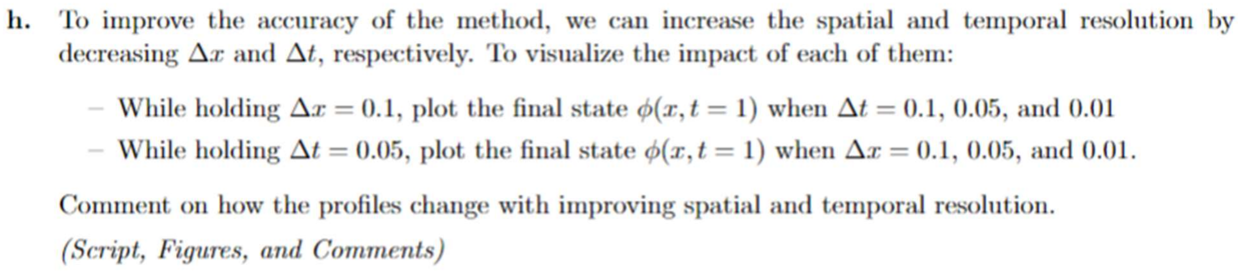


Figure 2: $\phi(x)$ vs x plot from Problem 2g. Resolution of dx = 0.2 and dt = 0.1 makes plot look pretty bad.

Problem 2g Script: HW8P2G.m

```
clear; close all; clc;

%% Spatial Discretization for Problem
dx = .2; % Define the spatial step size.
x = (0:dx:1)'; % Create a column vector of spatial points from 0 to 1.
nX = length(x); % Number of spatial points.

%% Time Discretization
dt = .1; % Define the time step size.
t = (0:dt:1)'; % Create a column vector of temporal points from 0 to 1.
nT = length(t); % Number of temporal points.

%% Other Parameters
c = .5; % Define the wave speed.
alpha = c*dt/dx; % Calculate the CFL number.
phi_n = zeros(nX, 1); % Initialize the solution vector for time step n.

%% B matrix Backwards Approximation
B = zeros(nX); % Initialize the B matrix.
B(1, 1) = 1-alpha; % Set the first row of B matrix.
B(1, nX-1) = alpha; % Periodic boundary condition.
```

```
for i = 2:nX
    B(i, i-1:i) = [alpha 1-alpha]; % Set the rest of B matrix using backward
difference.
end

%% Time marching using Explicit Euler
phi_np1 = zeros(nX, 1); % Initialize the solution vector for time step n+1.
for n = 1:nT-1
    % b vector (does change with time)
    b = 10*exp(-(x-.5).^2/.1^2)*sin(2*pi*t(n)); % Define the source term.

    % Explicit Euler
    phi_np1 = B*phi_n + b; % Compute phi at time step n+1 using explicit Euler.
    phi_n = phi_np1; % Update the solution for the next time step.

    % Plot point
    plot(x, phi_n); % Plot the current solution.
    axis([0 1 -10 10]); % Set the axes limits.
    title('$$\phi(x)$$ vs. $$x$$', 'Interpreter', 'latex'); % Set the title with
Greek letter using LaTeX.
    xlabel('$$x$$', 'Interpreter', 'latex'); % Set the x-axis label using LaTeX.
    ylabel('$$\phi(x)$$', 'Interpreter', 'latex'); % Set the y-axis label using
LaTeX.
    drawnow; % Update the plot.
end
```



h. To improve the accuracy of the method, we can increase the spatial and temporal resolution by decreasing Δx and Δt, respectively. To visualize the impact of each of them:

- While holding $\Delta x = 0.1$, plot the final state $\phi(x, t = 1)$ when $\Delta t = 0.1$, 0.05, and 0.01
- While holding $\Delta t = 0.05$, plot the final state $\phi(x, t = 1)$ when $\Delta x = 0.1$, 0.05, and 0.01.

Comment on how the profiles change with improving spatial and temporal resolution.

(Script, Figures, and Comments)

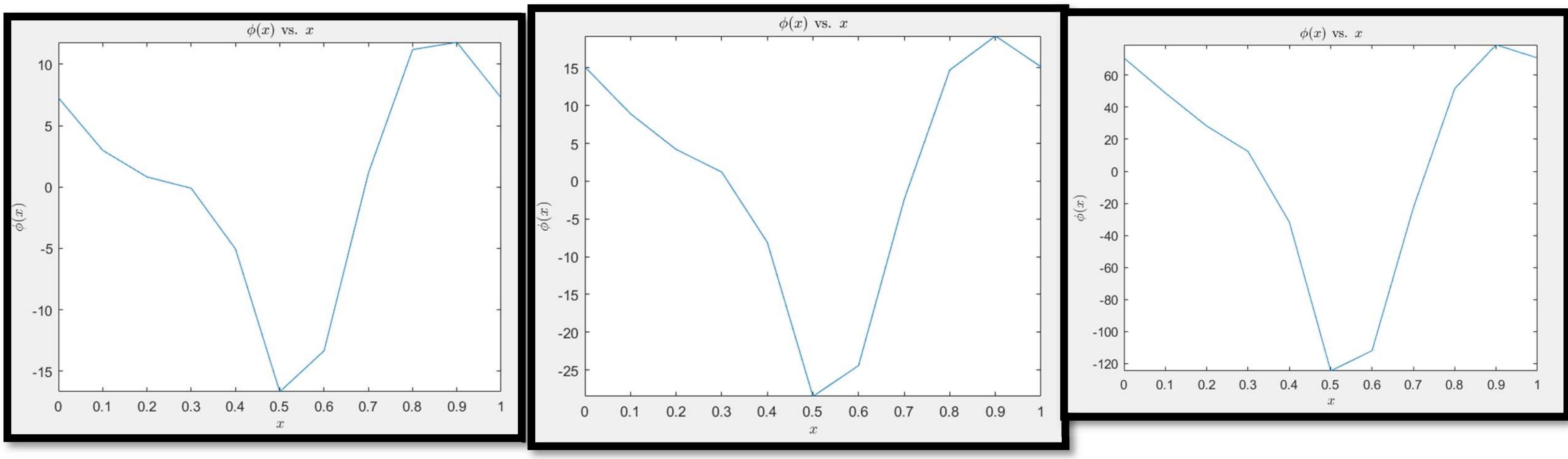


Figure 3: $\phi(x)$ vs x plot from Problem 2h. We kept delta x at 0.1 but the delta t = 0.1, 0.05 and 0.01 respectively

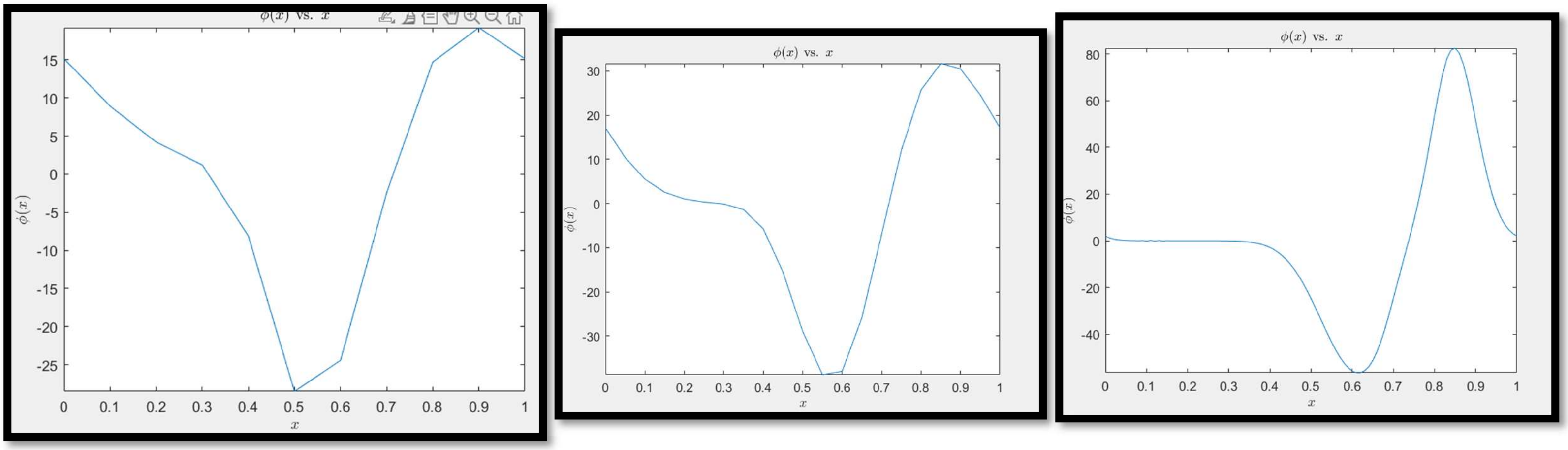


Figure 4: $\phi(x)$ vs x plot from Problem 2h. We kept delta t = 0.1 but the delta x = 0.1, 0.05, 0.01 respectively

Comments:

The profiles in the figures illustrate how finer temporal resolution sharpens the peaks and troughs in the plot of ϕ(x) versus x. When Δt decreases, the oscillations in ϕ(x) become more pronounced, indicating greater sensitivity to changes over smaller time steps. Smaller Δt values also lead to a more defined structure, possibly capturing more complex behaviors or instabilities in the modeled system. Additionally, the amplitude of ϕ(x) increases with finer temporal resolutions, suggesting that the system's response is more extreme or variable at these scales. Lastly, the evolution of ϕ(x) over x at smaller Δt is less smooth, hinting at more rapid temporal dynamics being captured.

Problem 2h Script: HW8P2G.m

```
% Problem 2 Part H
clear; close all; clc;

%% Spatial Discretization for Problem
dx = .01; % spatial step size.   -> (holding dx=0.1 when dt = 0.1, 0.05, 0.01)
x = (0:dx:1)'; % Create a column vector of spatial points from 0 to 1.
nX = length(x); % Number of spatial points.

%% Time Discretization
dt = .05; % time step size.     -> (holding dt=0.05 when dx=0.1, 0.05, 0.01)
t = (0:dt:1)'; % Create a column vector of temporal points from 0 to 1.
nT = length(t); % Number of temporal points.

%% Other Parameters
c = .5; % wave speed.
alpha = c*dt/dx; % CFL #
phi_n = zeros(nX, 1); % Initialize the solution vector for time step n.

%% B matrix Backwards Approximation
B = zeros(nX); % Initialize the B matrix.
B(1, 1) = 1-alpha; % Set the first row of B matrix.
B(1, nX-1) = alpha; % Periodic boundary condition.
for i = 2:nX
```

```
    B(i, i-1:i) = [alpha 1-alpha]; % Set the rest of B matrix using backward
difference.
end

%% Time marching using Explicit Euler
phi_np1 = zeros(nX, 1); % Initialize the solution vector for time step n+1.
for n = 1:nT-1
    % b vector (does change with time)
    b = 10*exp(-(x-.5).^2/.1^2)*sin(2*pi*t(n)); % Define the source term.

    % Explicit Euler
    phi_np1 = B*phi_n + b; % Compute phi at time step n+1 using explicit Euler.
    phi_n = phi_np1; % Update the solution for the next time step.

    % Plot point
    plot(x, phi_n); % Plot the current solution.
    axis([0 1 -10 10]); % Set the axes limits.
    title('$$\phi(x)$$ vs. $$x$$', 'Interpreter', 'latex'); % Set the title with
Greek letter using LaTeX.
    xlabel('$$x$$', 'Interpreter', 'latex'); % Set the x-axis label using LaTeX.
    ylabel('$$\phi(x)$$', 'Interpreter', 'latex'); % Set the y-axis label using
LaTeX.
    drawnow; % Update the plot.
end
```

Problem 3 - Steady-State Spin Up

A common step in experimental systems that feature rotating fluids is to spin up a tank of fluid originally at rest to solid body rotation. For the first three problems in this exam, we will be studying this spin up process for different conditions. We will derive the steady-state spin up *angular velocity* Ω profiles for two different assumptions. The momentum equation for the *azimuthal* or *tangential velocity* u_θ is

$$\frac{\partial u_\theta}{\partial t} + u_r\frac{\partial u_\theta}{\partial r} + \frac{u_\theta}{r}\frac{\partial u_\theta}{\partial \theta} + \frac{u_r u_\theta}{r} + u_z\frac{\partial u_\theta}{\partial z} = -\frac{1}{\rho r}\frac{\partial p}{\partial \theta} + \nu\left(\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial}{\partial r}(r u_\theta)\right) + \frac{1}{r^2}\frac{\partial^2 u_\theta}{\partial \theta^2} + \frac{\partial^2 u_\theta}{\partial z^2} + \frac{2}{r^2}\frac{\partial u_\theta}{\partial \theta}\right).$$

a. We are going to derive the steady-state spin up angular velocity profiles under two different sets of assumptions. Note that $u_\theta(r, z, t) = r\Omega(r, z, t)$. Assume for both cases that there is no variation in the azimuthal (θ) direction and no velocity in the radial (r) or axial (z) directions.

↳ angular velocity the same at every angle θ

i. Assuming that there is no variation in the axial direction, write out the simplified steady-state PDE for the *angular velocity* $\Omega(r)$. Solve for the angular velocity $\Omega(r)$ given the boundary conditions $\Omega(r = R) = \Omega_w$ and the symmetry condition at the center $d\Omega/dr = 0$ at $r = 0$.

no pressure due to symmetry

$$\cancel{\frac{\partial u_\theta}{\partial t}} + u_r\cancel{\frac{\partial u_\theta}{\partial r}} + \cancel{\frac{u_\theta}{r}\frac{\partial u_\theta}{\partial \theta}} + \cancel{\frac{u_r u_\theta}{r}} + u_z\cancel{\frac{\partial u_\theta}{\partial z}} = \cancel{\frac{-1}{r}\frac{\partial p}{\partial \theta}} + v\left(\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial}{\partial r}(r u_\theta)\right) + \cancel{\frac{1}{r^2}\frac{\partial^2 u_\theta}{\partial \theta^2}} + \cancel{\frac{\partial^2 u_\theta}{\partial z^2}} + \cancel{\frac{2}{r^2}\frac{\partial u_\theta}{\partial \theta}}\right)$$

Steady state

no radial u_r or axial velocity u_z

no radial or axial velocity

$$v\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial}{\partial r}(r u_\theta)\right) = \frac{v}{r}\left(u_\theta + r\frac{\partial u_\theta}{\partial r}\right) \rightarrow \frac{\partial}{\partial r}\left(\frac{u_\theta}{r} + \frac{\partial u_\theta}{\partial r}\right)$$

$$\hookrightarrow v\left(-\frac{u_\theta}{r^2} + \frac{1}{r}\frac{du_\theta}{dr} + \frac{\partial^2 u_\theta}{\partial r^2}\right)$$

$$0 = v\left(-\frac{u_\theta}{r^2} + \frac{1}{r}\frac{du_\theta}{dr} + \frac{\partial^2 u_\theta}{\partial r^2}\right) \qquad u_\theta(r) = r\underbrace{\Omega(r)}_{\text{angular velocity}}$$

$$0 = v\left(\frac{-r\Omega}{r^2} + \frac{1}{r}\frac{\partial}{\partial r}(r\Omega) + \frac{\partial^2}{\partial r^2}(r\Omega)\right)$$

Boundary Conditions

1) $\Omega(R) = \Omega_w$

2) $\left.\frac{\partial \Omega}{\partial r}\right|_{r=0} = 0$ b/c symmetry

$$0 = v\left(\frac{-\Omega}{r} + \frac{1}{r}\left[r\frac{\partial \Omega}{\partial r} + \Omega\right] + \frac{\partial}{\partial r}\left(r\frac{\partial \Omega}{\partial r} + \Omega\right)\right)$$

$$0 = v\left(\cancel{\frac{-\Omega}{r}} + \frac{\partial \Omega}{\partial r} + \cancel{\frac{\Omega}{r}} + r\frac{\partial^2 \Omega}{\partial r^2} + 2\frac{\partial \Omega}{\partial r}\right)$$

make $y = \frac{\partial \Omega}{\partial r}$

$$0 = \cancel{v}\left(r\frac{\partial^2 \Omega}{\partial r^2} + 3\frac{\partial \Omega}{\partial r}\right) \rightarrow r\frac{\partial^2 \Omega}{\partial r^2} + 3\frac{\partial \Omega}{\partial r} = 0 \rightarrow r\frac{\partial^2 \Omega}{\partial r^2} = -3\frac{\partial \Omega}{\partial r}$$

$$r\frac{dy}{dr} = -3y \rightarrow \frac{dy}{dr} = \frac{-3}{r}y \rightarrow \int \frac{1}{y}dy = \int \frac{-3}{r}dr \rightarrow \underset{e}{\ln|y|} = \underset{e}{-3\ln|r|} + C_1$$

$$y = e^{C_1}\cdot r^{-3} \rightarrow \frac{\partial \Omega}{\partial r} = e^{C_1}\cdot r^{-3} \rightarrow \int \partial\Omega = \int e^{C_1}\cdot r^{-3}\,\partial r$$

$$\frac{r^{-3+1}}{-3+1} = \frac{r^{-2}}{-2}$$

$$\Omega(r) = -\frac{C_1}{2r^2} + C_2 \qquad \Omega(r) = C_2 = \Omega_w$$

$$\Omega'(r) = -\frac{1}{2}C_1\left(\frac{-2}{r^3}\right)$$

$$\Omega'(r) = \frac{C_1}{r^3} \rightarrow r=0$$

$$= \frac{1}{\infty} = 0 \quad \text{thus } C_1 = 0$$

$$\boxed{\Omega(r) = \Omega_w}$$

ii. Assuming that there is no variation in the radial direction, write out the steady-state PDE for the *angular velocity* $\Omega(z)$. Solve for the angular velocity $\Omega(z)$ given the no-slip boundary conditions $\Omega(z=0)=\Omega_w$ and $\Omega(z=H)=\Omega_w$.

no pressure due to symmetry

no variation radial

$$\cancel{\frac{\partial u_\theta}{\partial t}} + \cancel{u_r \frac{\partial u_\theta}{\partial r}} + \cancel{\frac{u_\theta}{r}\frac{\partial u_\theta}{\partial \theta}} + \cancel{\frac{u_r u_\theta}{r}} + \cancel{u_z \frac{\partial u_\theta}{\partial z}} = \cancel{\frac{-1}{r}\frac{\partial p}{\partial \theta}} + \nu\left(\cancel{\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial}{\partial r}(r u_\theta)\right)} + \cancel{\frac{1}{r^2}\frac{\partial^2 u_\theta}{\partial \theta^2}} + \frac{\partial^2 u_\theta}{\partial z^2} + \cancel{\frac{2}{r^2}\frac{\partial u_\theta}{\partial \theta}}\right)$$

Steady state

no radial u_r or axial velocity u_z

no radial or axial velocity

$$0 = \nu\frac{\partial^2 u_\theta}{\partial z^2} \rightarrow 0 = \nu\frac{\partial^2}{\partial z^2}(\Omega(z)) = \nu\frac{\partial^2 \Omega}{\partial z^2}$$

$$0 = \cancel{\nu}\frac{\partial^2 \Omega}{\partial z^2} \rightarrow \int 0 = \int \frac{\partial^2 \Omega}{\partial z^2} \rightarrow \int C_1 = \int \frac{\partial \Omega}{\partial z} \rightarrow C_1 z + C_2 = \Omega(z)$$

Boundary Conditions

1) $\Omega(z=0) = \Omega_w$

2) $\Omega(z=H) = \Omega_w$

$$\Omega(z=0) = C_1\cancel{(0)} + C_2 \rightarrow \Omega(z=H) = C_1(H) + \Omega_w$$

$$\Omega_w = C_2 \qquad \Omega_w = C_1(H) + \Omega_w$$

$$0 = C_1(H) \rightarrow C_1 = 0$$

$$\boxed{\Omega(z) = \Omega_w}$$

On the code should be straight line

b. Write a script that solves the boundary value problem in **i**. Use a centered space approximation, uniform mesh, and ghost points at Neuman boundary conditions to solve the ODE. Use the properties: number of nodes including ghost point $nR = 30$, tank radius $R = 1$, and wall angular velocity $\Omega_w = 1$. Plot the result to confirm that it matches your analytic model.

Script and Figure

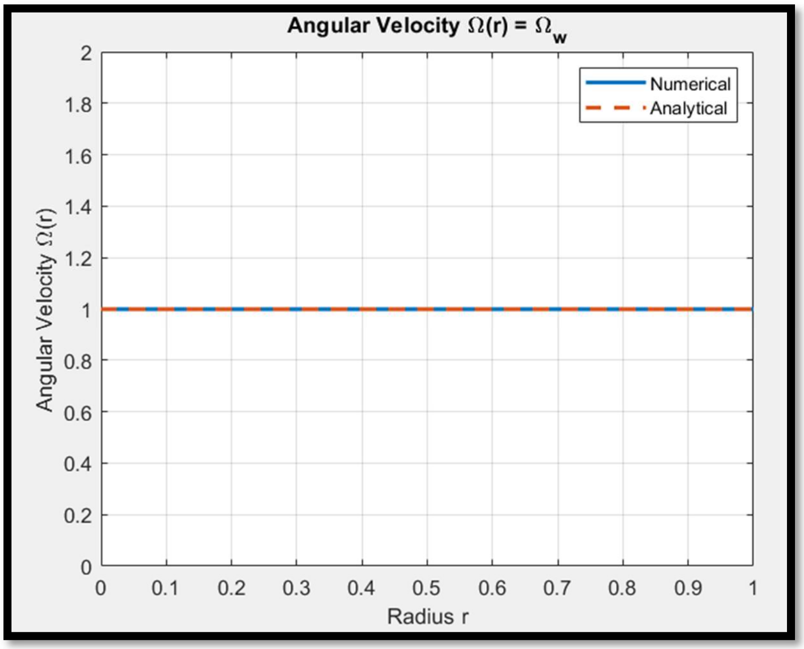
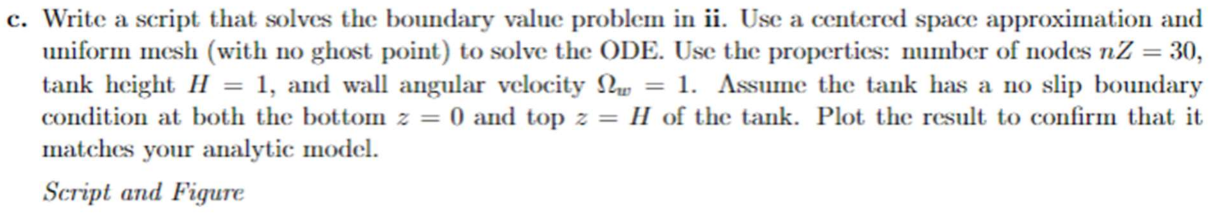


Figure 2: Plot of Angular Velocity vs Radius r (numerical and analytical (part a, i derivation for analytical).

Problem 3B Script: HW8P3B.m

```
% Problem 3 Part B
clear; close all; clc;
% Given properties
nR = 30;               % Number of nodes including ghost point
R = 1;                 % Tank radius
Omega_w = 1;           % Wall angular velocity

% Discretization
dr = R / (nR - 1);     % Step size
r = linspace(0, R, nR)'; % Radial positions array

% Initialize the matrix A and vector b for the system A*omega = b
A = zeros(nR);         % Matrix A will hold the finite difference coefficients
b = zeros(nR, 1);      % Vector b is the right-hand side of the equation

% Fill in the finite difference coefficients into matrix A
for i = 2:nR-1
    A(i, i-1) = 1/dr^2 - 1/(2*dr*r(i));
    A(i, i) = -2/dr^2;
    A(i, i+1) = 1/dr^2 + 1/(2*dr*r(i));
end

% Neumann boundary condition at r=0 using ghost point
A(1, 1) = -3/(2*dr);
A(1, 2) = 2/dr;
A(1, 3) = -1/(2*dr);

% Dirichlet boundary condition at r=R
A(nR, nR) = 1;
b(nR) = Omega_w;

% Solve the system
omega = A\b;

% Plot the result
plot(r, omega, 'LineWidth', 2);
title('Angular Velocity \Omega(r) = \Omega_w');
xlabel('Radius r');
ylabel('Angular Velocity \Omega(r)');

% Set y-axis limits and ticks
ylim([0 2]); % Set the y-axis limits from 0 to 2
yticks(0:0.2:2); % Set the y-ticks from 0 to 2 with an interval of 0.2

grid on;
hold on;

% Analytical solution
omega_analytic = Omega_w * ones(nR, 1);
plot(r, omega_analytic, '--', 'LineWidth', 2);
legend('Numerical', 'Analytical');
```



c. Write a script that solves the boundary value problem in **ii**. Use a centered space approximation and uniform mesh (with no ghost point) to solve the ODE. Use the properties: number of nodes $nZ = 30$, tank height $H = 1$, and wall angular velocity $\Omega_w = 1$. Assume the tank has a no slip boundary condition at both the bottom $z = 0$ and top $z = H$ of the tank. Plot the result to confirm that it matches your analytic model.

Script and Figure

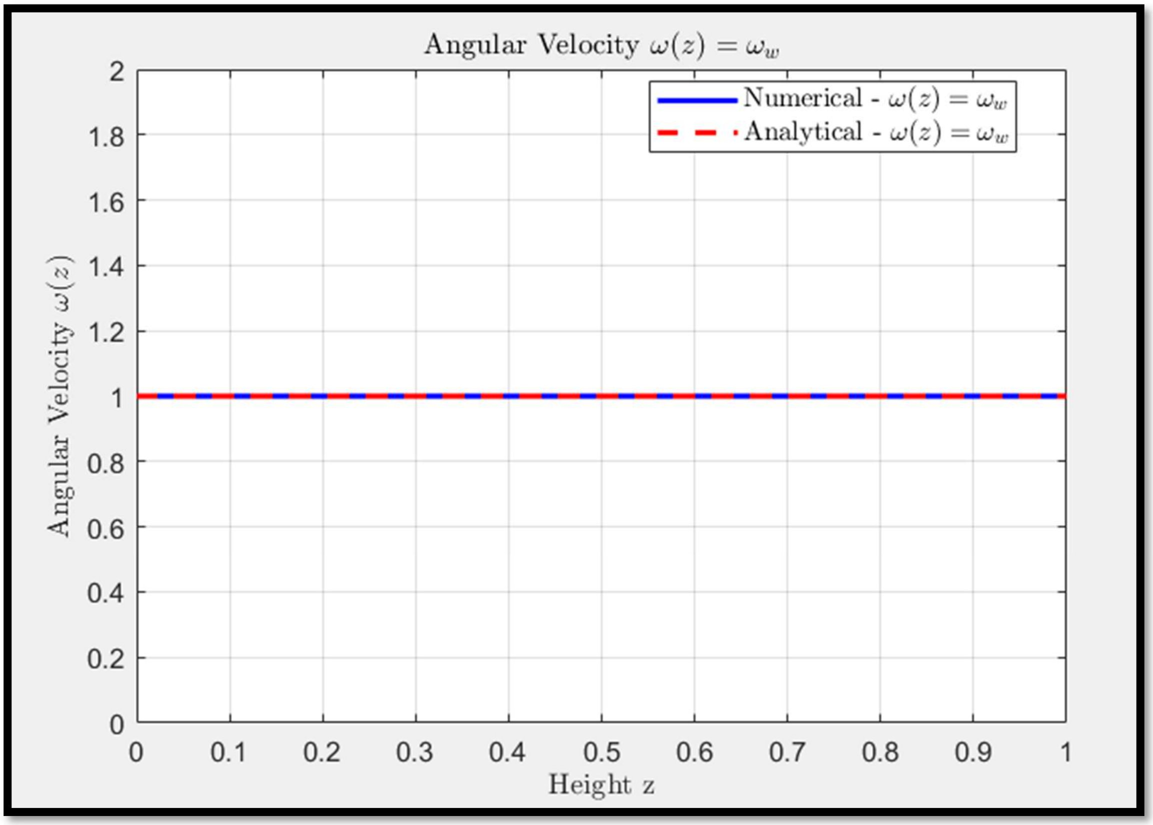
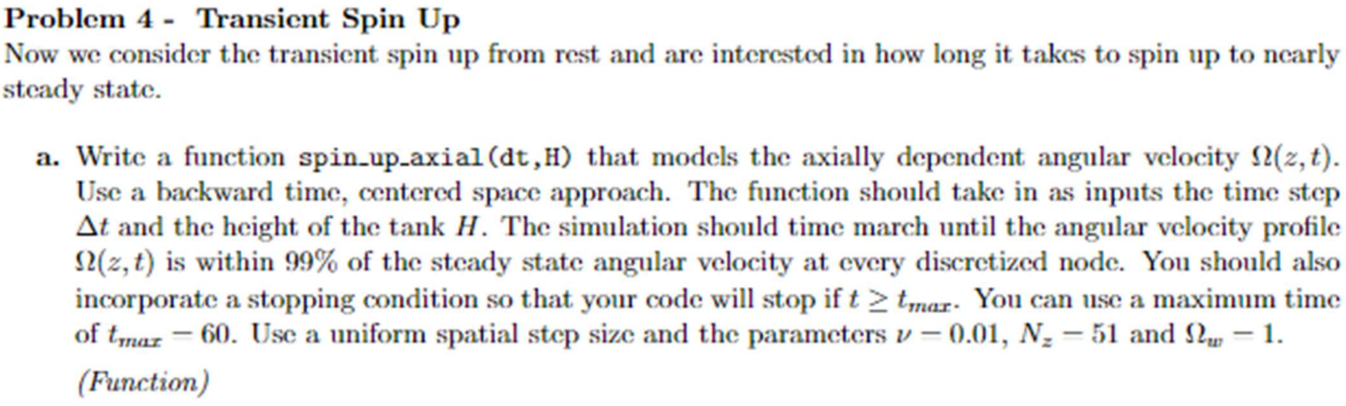


Figure 3: Plot of Angular Velocity vs z (numerical and analytical (part a, ii derivation for analytical).

Problem 3C Script: HW8P3C.m

```
% Problem 3 Part C
clear; close all; clc;

% Given properties
nZ = 30;                % Number of nodes in z-direction
H = 1;                  % Tank height
Omega_w = 1;            % Wall angular velocity

% Discretization
dz = H / (nZ - 1);      % Step size
z = linspace(0, H, nZ)'; % Height positions array

% Since Omega(z) is constant and equals Omega_w, the solution is a horizontal line
omega_numerical = Omega_w * ones(nZ, 1); % Numerical solution
```

```
% Plot the numerical solution
plot(z, omega_numerical, 'b-', 'LineWidth', 2);
hold on; % Hold on for multiple plots on the same figure

% Plot the analytical solution (will overlay with numerical)
omega_analytic = Omega_w * ones(nZ, 1); % Analytical solution
plot(z, omega_analytic, 'r--', 'LineWidth', 2);

% Improve the appearance of the plot
title('Angular Velocity \(\omega(z) = \omega_w\)', 'Interpreter', 'latex',
'FontSize', 14);
xlabel('Height z', 'Interpreter', 'latex', 'FontSize', 12);
ylabel('Angular Velocity \(\omega(z)\)', 'Interpreter', 'latex', 'FontSize', 12);
ylim([0 2]); % Set y-axis limits
yticks(0:0.2:2); % Set y-axis ticks
grid on; % Add grid
set(gca, 'FontSize', 10); % Set axes font size

% Create legend
legend({'Numerical - \(\omega(z) = \omega_w\)', 'Analytical - \(\omega(z) =
\omega_w\)'}, ...
       'Interpreter', 'latex', 'FontSize', 10, 'Location', 'best');

% Set figure size
set(gcf, 'Position', [100, 100, 600, 400]); % Set the figure size

% Finalize the plot
hold off; % Release the plot hold
```



Problem 4 - Transient Spin Up

Now we consider the transient spin up from rest and are interested in how long it takes to spin up to nearly steady state.

a. Write a function **spin_up_axial(dt,H)** that models the axially dependent angular velocity $\Omega(z,t)$. Use a backward time, centered space approach. The function should take in as inputs the time step Δt and the height of the tank H. The simulation should time march until the angular velocity profile $\Omega(z,t)$ is within 99% of the steady state angular velocity at every discretized node. You should also incorporate a stopping condition so that your code will stop if $t \geq t_{max}$. You can use a maximum time of $t_{max} = 60$. Use a uniform spatial step size and the parameters $\nu = 0.01$, $N_z = 51$ and $\Omega_w = 1$.

(Function)

Function Code: spin_up_axial.m

```
function t_s = spin_up_axial(dt, H)

% This is for 4A -> function named spin_up_axial with inputs dt (time step) and H
(height)

%% Spatial Discretization
nZ = 51; % Number of vertical grid points
dz = H/(nZ-1); % Vertical spacing between grid points
z = (0:dz:H)'; % Vertical grid, from 0 to H, spaced by dz
```

```
%% Temporal Discretization
t = 0; % Start time
t_max = 60; % Maximum time to run the simulation

%% Various Problem Parameters
nu = .01; % Kinematic viscosity
alpha = nu*dt; % Alpha parameter for numerical scheme, scaled by time step
omega_w = 1; % Angular velocity at the wall
omega_state = omega_w; % Steady state angular velocity, set equal to wall velocity

%% Current State Angular Velocity
omega_n = zeros(nZ, 1); % Initialize angular velocity at all grid points to zero

%% Initial Condition
omega_n(1) = omega_w; % Set angular velocity at the first grid point (wall) to
omega_w
omega_n(nZ) = omega_w; % Set angular velocity at the last grid point (wall) to
omega_w

%% Second Derivative Matrix
D2 = zeros(nZ); % Initialize D2 matrix for calculating second derivatives
for i = 2:nZ-1
    D2(i, i-1:i+1) = [1 -2 1]/dz^2; % Populate D2 matrix with coefficients for finite
difference approximation of second derivatives
end

%% A Matrix
A = zeros(nZ); % Initialize A matrix
A = -alpha*D2 + eye(nZ); % Define A matrix, incorporating diffusion term and identity
for implicit scheme

%% Boundary Condition
A(nZ, :) = 0; % Set last row of A matrix to zero for boundary condition
A(nZ, nZ) = 1; % Set last element of A matrix to 1 for Dirichlet boundary condition
at z=H
A(1, :) = 0; % Set first row of A matrix to zero for boundary condition
A(1, 1) = 1; % Set first element of A matrix to 1 for Dirichlet boundary condition at
z=0

%% Implicit Euler Time Marching
while any(omega_n < .99*omega_state) && t < t_max
    % Loop until the solution is within 99% of the steady state or the maximum time
is reached

    % b matrix
    b = zeros(nZ, 1); % Initialize b matrix
    b = omega_n; % Set b matrix equal to current state angular velocity

    % Next State Angular Velocity
    omega_np1 = A\b; % Calculate next state angular velocity using implicit Euler
method
    omega_n = omega_np1; % Update current state angular velocity

    % Update time
    t = t + dt; % Increase time by time step
```

```
end

%% output spin up time
t_s = t; % Return the spin-up time

end
```

b. Write a script that calls your function `spin_up_axial` for the inputs $\Delta t = 0.02$ and $H = 0.1 : 0.1 : 1.0$. Plot the spin up time t_s as a function of height. Estimate the analytical functional relationship $t_s(H)$ from your plot.

(Script, Figure, Comment)

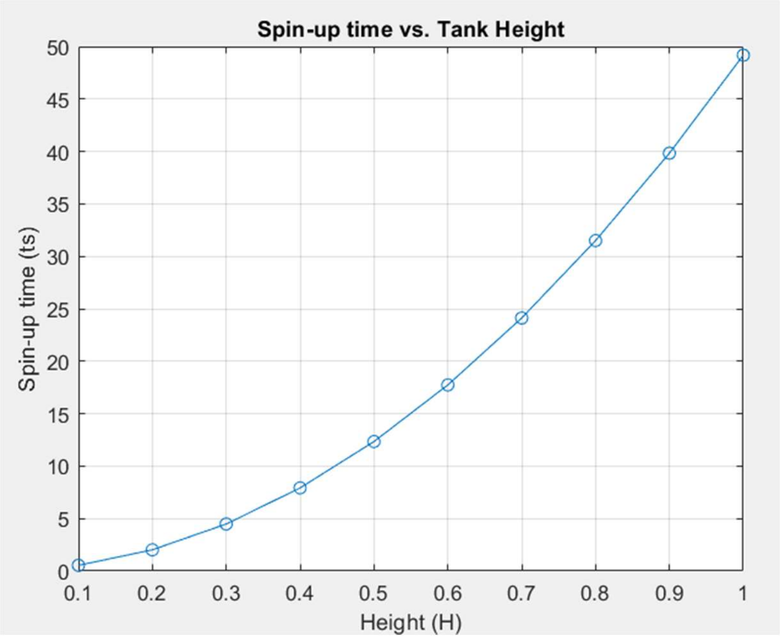


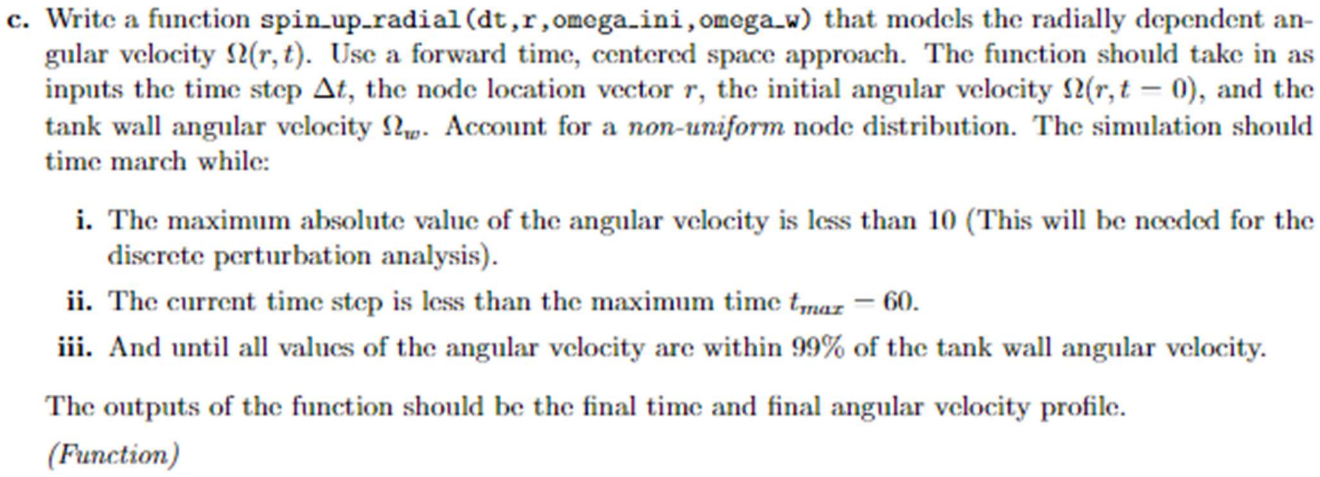
Figure 4: Spin up Time vs Tank Height using part 4a and 4b code. Graph appears to be quadratic.

The spin-up time (ts) vs height plot appears to be exponentially increasing. We can estimate the values by using points from the graph. The exponential equation I derived was about $t_s = 49.1502H^{1.9471}$. When H increases, the time it takes to spin up also increases.

Problem 4B Script: HW8P4B.m

```
% Problem 4 Part B
clear; close all; clc;

% Define the range of tank heights and the time step
H_values = 0.1:0.1:1.0;
dt = 0.02;
ts_values = zeros(length(H_values), 1); % Initialize an array to store spin-up times

% Loop over each height and call the spin_up_axial function
for i = 1:length(H_values)
    H = H_values(i);
    ts_values(i) = spin_up_axial(dt, H);
end
```

```
% Plot the spin-up time as a function of the tank height
figure; % Create a new figure
plot(H_values, ts_values, 'o-'); % Plot with circle markers and a line
xlabel('Height (H)'); % Label the x-axis
ylabel('Spin-up time (ts)'); % Label the y-axis
title('Spin-up time vs. Tank Height'); % Title for the plot
grid on; % Turn on the grid for easier reading
```



c. Write a function `spin_up_radial(dt,r,omega_ini,omega_w)` that models the radially dependent angular velocity $\Omega(r,t)$. Use a forward time, centered space approach. The function should take in as inputs the time step Δt, the node location vector r, the initial angular velocity $\Omega(r, t = 0)$, and the tank wall angular velocity Ω_w. Account for a *non-uniform* node distribution. The simulation should time march while:

- **i.** The maximum absolute value of the angular velocity is less than 10 (This will be needed for the discrete perturbation analysis).
- **ii.** The current time step is less than the maximum time $t_{max} = 60$.
- **iii.** And until all values of the angular velocity are within 99% of the tank wall angular velocity.

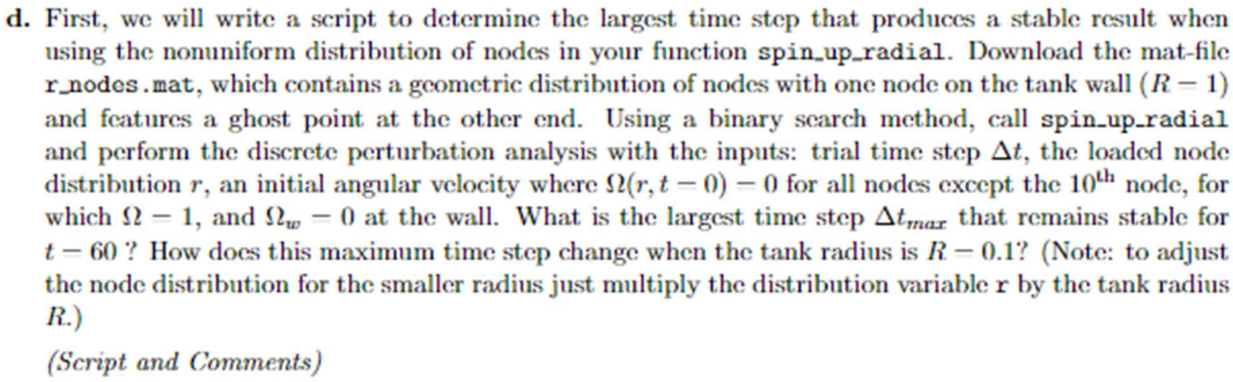
The outputs of the function should be the final time and final angular velocity profile.

(Function)

Problem 4C function: spin_up_radial.m

```
function [omega, t_s] = spin_up_radial(dt, r, omega_ini, omega_w)

%% Understanding our function values
% inputs: time step 'dt', radial positions 'r', angular velocity 'omega_ini', and
wall angular velocity 'omega_w'.
% The function returns the final angular velocity 'omega' and the spin-up time 't_s'.

%% Spatial Discretization
nR = length(r); % Calculates the number of radial positions.

%% Temporal Discretization
t = 0; % Initializes the time variable.
t_max = 60; % Sets the maximum simulation time.

%% Parameters
nu = .01; % Viscosity coefficient.
beta = nu*dt; % Pre-computed parameter for efficiency, incorporating the time step
and viscosity.
omega_ss = omega_w; % Sets the steady state angular velocity equal to the wall
angular velocity.

%% Current State Angular Velocity and initial condition
omega_n = omega_ini; % Initializes the angular velocity with the given initial
condition.

%% A Matrix and the boundaries
A = eye(nR); % Initializes matrix A as an identity matrix of size nR.
A(1, :) = 0; % Sets the first row of A to zeros for boundary conditions.
```

```
A(1, 1:2) = [-1 1]/(r(2) - r(1)); % Applies a forward difference scheme for the inner
boundary.
A(nR, :) = 0; % Sets the last row of A to zeros for boundary conditions.
A(nR, nR) = 1; % Ensures the outer boundary's angular velocity is unchanged
(Dirichlet condition).

%% First and Second Derivative Matrix
D1 = zeros(nR); % Initializes the first derivative matrix.
D2 = zeros(nR); % Initializes the second derivative matrix.
for i = 2:nR-1
    dr = r(i) - r(i-1); % Calculates the differential radial distance.
    alpha = (r(i+1) - r(i))/(r(i) - r(i-1)); % Computes the non-uniform grid scaling
factor.
    % Sets the coefficients for the first derivative central difference scheme.
    D1(i, i-1:i+1) = [-alpha/(1+alpha) (alpha-1)/alpha 1/(alpha+alpha^2)]/dr;
    % Sets the coefficients for the second derivative central difference scheme.
    D2(i, i-1:i+1) = [2/(1+alpha) -2/alpha 2/(alpha+alpha^2)]/dr^2;
end

%% B Matrix and its boundaries
B = zeros(nR); % Initializes matrix B.
B = beta*D2 + 3*beta*diag(1./r)*D1 + eye(nR); % Constructs B using the discretized
viscous term and identity.
B(1, :) = 0; % Sets the first row of B to zeros for boundary conditions.
B(nR, :) = 0; % Sets the last row of B to zeros for boundary conditions.

% Explicit Euler Scheme for Time Marching
while any(omega_n <= .99*omega_ss) && max(abs(omega_n)) < 10 && t < t_max
    % Continues the loop until the angular velocity is near steady state, below a max
value, or time exceeds max.

    b = zeros(nR, 1); % Initializes the source term vector for b matrix.
    b(1) = 0; % Sets the inner boundary condition.
    b(nR) = omega_w; % Sets the outer boundary angular velocity to the wall velocity.

    % Next State Angular Velocity
    omega_np1 = A\(B*omega_n + b); % Computes the next angular velocity state using
the explicit Euler method.
    omega_n = omega_np1; % Updates the current angular velocity state.

    % Update time
    t = t + dt; % Increments the time by the time step.
end

% Output Angular Velocity and Spin Up Time
omega = omega_n; % Outputs the final angular velocity.
t_s = t; % Outputs the total spin-up time.
end
```



d. First, we will write a script to determine the largest time step that produces a stable result when using the nonuniform distribution of nodes in your function `spin_up_radial`. Download the mat-file `r_nodes.mat`, which contains a geometric distribution of nodes with one node on the tank wall ($R = 1$) and features a ghost point at the other end. Using a binary search method, call `spin_up_radial` and perform the discrete perturbation analysis with the inputs: trial time step Δt, the loaded node distribution r, an initial angular velocity where $\Omega(r, t = 0) = 0$ for all nodes except the 10^{th} node, for which $\Omega = 1$, and $\Omega_w = 0$ at the wall. What is the largest time step Δt_{max} that remains stable for $t = 60$? How does this maximum time step change when the tank radius is $R = 0.1$? (Note: to adjust the node distribution for the smaller radius just multiply the distribution variable `r` by the tank radius R.)

(Script and Comments)

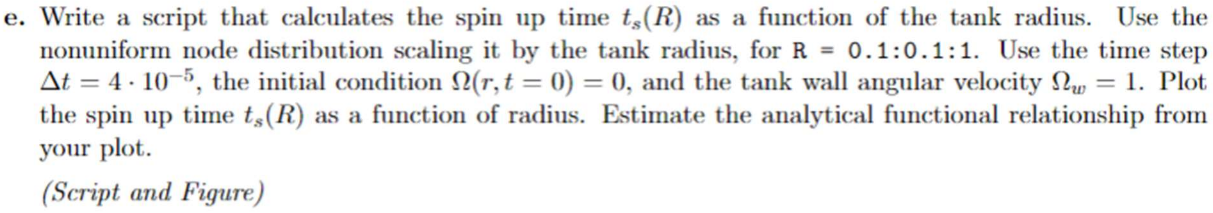
Comments:

When running .1 resolution for r= load('r_nodes-1.mat').r*.1 , the value is 4.5123*10^-5 for t=60. When no .1 resolution, the value is 0.0045. When the tank radius is reduced to R=0.1, the maximum stable time step for the simulation might decrease, depending on how the spatial discretization interacts with the numerical stability criteria (like the CFL condition). Smaller spatial domains typically require smaller time steps to maintain numerical stability and accuracy, as changes in the system's state can happen more rapidly over shorter distances. Therefore, if the spatial discretization in r_nodes-1.mat is adjusted for a smaller tank radius, you would likely need to use a smaller time step for the simulation to remain stable.

Problem 4D Script: HW8P4D.m

```
% Problem 4 Part D

clear; close all; clc;

%% Load Geometric Discretization in r spatial
r = load('r_nodes-1.mat').r; % Loads radial positions from a .mat file
nR = length(r); % Calculates the number of radial positions.

%% Guess dt For Stability
dt_lower = 0; % Initializes the lower bound for the time step guess.
dt_upper = 60; % Initializes the upper bound for the time step guess.

%% Wall Angular Velocity
omega_w = 0; % Sets the angular velocity of the wall to 0.

%% Initial Angular Velocity
omega_ini = zeros(nR, 1); % Initializes the angular velocity array with zeros.
omega_ini(10) = 1; % Sets the angular velocity at the 10th radial position to 1.

%% Current Guess and Guess History
old_dt_guess = -1; % Sets a placeholder value for the initial previous guess.
station_dt_guess = (dt_lower + dt_upper)/2; % Calculates the initial guess for dt.

while ~ismembertol(station_dt_guess, old_dt_guess, 10^-3)
```

```
    % Loops until the current guess for dt is within a tolerance of the previous
guess, indicating that a stable dt has been found.

    % Determine Angular Velocity
    [omega, t_s] = spin_up_radial(station_dt_guess, r, omega_ini, omega_w); % Calls
the function 'spin_up_radial' with the current dt guess to compute the angular
velocity profile.

    % Change Guess Bounds and Update Previous Guess
    if any(omega > 1)
        dt_upper = station_dt_guess; % If any angular velocity exceeds 1, decrease
the upper bound for dt.
    else
        dt_lower = station_dt_guess; % If all angular velocities are below 1,
increase the lower bound for dt.
    end

    % Update Guess and Guess History
    old_dt_guess = station_dt_guess; % Updates the previous dt guess.
    station_dt_guess = (dt_lower + dt_upper)/2; % Calculates a new dt guess as the
midpoint between the current bounds.
end

disp(station_dt_guess); % Displays the determined stable time step.
```



e. Write a script that calculates the spin up time $t_s(R)$ as a function of the tank radius. Use the nonuniform node distribution scaling it by the tank radius, for `R = 0.1:0.1:1`. Use the time step $\Delta t = 4 \cdot 10^{-5}$, the initial condition $\Omega(r, t = 0) = 0$, and the tank wall angular velocity $\Omega_w = 1$. Plot the spin up time $t_s(R)$ as a function of radius. Estimate the analytical functional relationship from your plot.

(Script and Figure)

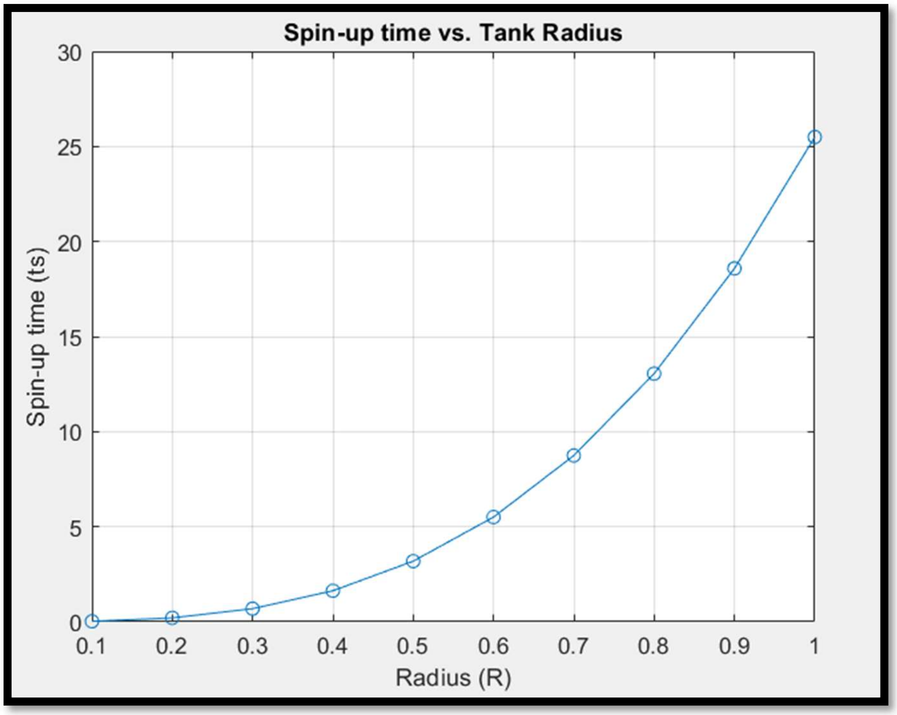
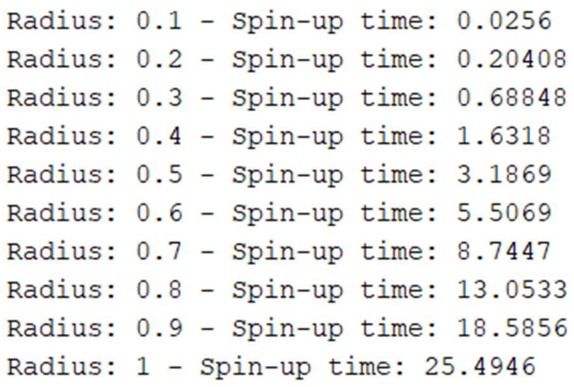


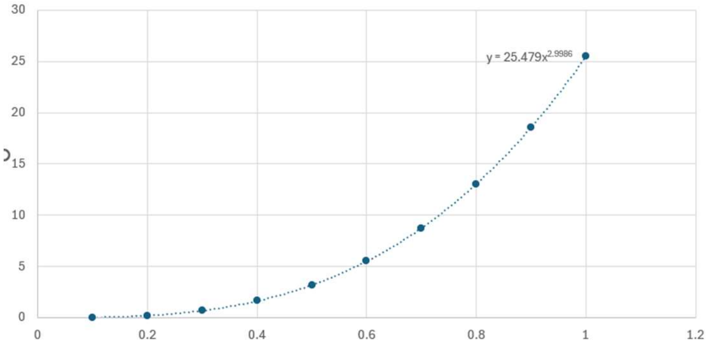
Figure 5: Spin up Time vs Tank Radius using part 4e code. It appears to exponential increasing toward R=1 with respect to time.



```
Radius: 0.1 - Spin-up time: 0.0256
Radius: 0.2 - Spin-up time: 0.20408
Radius: 0.3 - Spin-up time: 0.68848
Radius: 0.4 - Spin-up time: 1.6318
Radius: 0.5 - Spin-up time: 3.1869
Radius: 0.6 - Spin-up time: 5.5069
Radius: 0.7 - Spin-up time: 8.7447
Radius: 0.8 - Spin-up time: 13.0533
Radius: 0.9 - Spin-up time: 18.5856
Radius: 1 - Spin-up time: 25.4946
```

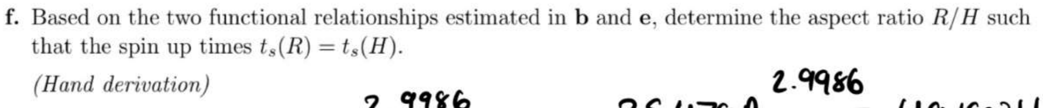
Comments:

The spin-up time (ts) vs height plot appears to be increasing. The shape is an upwards parabola. We can estimate the values by using points from the graph. The exponential equation I derived was about $t_s = 25.479R^{2.9986}$. When R increases, the time it takes to spin up also increases. Excel was also used to validify fitting of equation in graph.



Problem 4 Script: HW8P4E.m

```
% Problem 4 Part E
clear; close all; clc;

%% Define the parameters
R_values = 0.1:0.1:1.0; % Range of tank radii
dt = 4e-5; % Time step
omega_w = 1; % Angular velocity at the wall
nZ = 51; % Number of vertical grid points
H = 1; % Height of the tank (assuming constant height for all radii)
ts_values = zeros(length(R_values), 1); % Initialize an array to store spin-up times

%% Loop over each radius
for idx = 1:length(R_values)
    R = R_values(idx);
    dz = H / (nZ - 1); % Vertical spacing between grid points
    z = linspace(0, H, nZ)'; % Vertical grid, from 0 to H

    % Initialize angular velocity profile for current radius
    omega_n = zeros(nZ, 1); % Current state angular velocity
    omega_n(1) = omega_w; % Set angular velocity at the wall to omega_w
    omega_n(end) = omega_w; % Set angular velocity at the top to omega_w (if it's the
same as the wall)

    % Second Derivative Matrix for non-uniform grid scaled by radius
    D2 = zeros(nZ); % Initialize D2 matrix
    for i = 2:nZ-1
        r = R * (z(i) / H); % Scale the position by the radius
        D2(i, i-1:i+1) = [1 -2 1] / (dz^2 * r); % Scale coefficients by the non-
uniform grid
    end

    % Define the A matrix for the implicit scheme
    nu = .01; % Kinematic viscosity
    alpha = nu * dt / (R^2); % Alpha parameter for numerical scheme
    A = -alpha * D2 + eye(nZ); % Define A matrix

    % Set boundary conditions in A matrix for top and bottom of the tank
    A(nZ, :) = 0;
    A(nZ, nZ) = 1;
    A(1, :) = 0;
    A(1, 1) = 1;

    %% Time loop
    t = 0; % Start time
    omega_state = omega_w; % Steady state angular velocity, set equal to wall
velocity
    while any(abs(omega_n - omega_state) > 0.01 * omega_state) && t < 60
        % Implicit Euler Time Marching
        b = omega_n; % Set b equal to current state angular velocity
        omega_n = A\b; % Solve for next state
        t = t + dt; % Increase time by time step
    end

    % Record the spin-up time for the current radius
    ts_values(idx) = t;

    % Display the current radius and the computed spin-up time
    disp(['Radius: ', num2str(R), ' - Spin-up time: ', num2str(t)]);
end

% Plot spin-up time as a function of tank radius
figure;
plot(R_values, ts_values, 'o-');
xlabel('Radius (R)');
ylabel('Spin-up time (ts)');
title('Spin-up time vs. Tank Radius');
grid on;

% Ensure the figure stays open at the end of the script
disp('Plotting completed.');
```



f. Based on the two functional relationships estimated in **b** and **e**, determine the aspect ratio R/H such that the spin up times $t_s(R) = t_s(H)$.

(Hand derivation)

$$t_s(R) = 25.479R^{2.9986}$$

$$t_s(H) = 49.1502H^{1.9471}$$

$$\frac{25.479R^{2.9986}}{25.479} = \frac{49.1502H^{1.9471}}{25.479}$$

$$\left(R^{2.9986}\right)^{\frac{1}{2.9986}} = \left(1.929047H^{1.9471}\right)^{\frac{1}{2.9986}}$$

$$R = 1.2449694H^{0.64933}$$

$$\frac{R}{\underbrace{H^{0.64933}}_{\approx 1}} = 1.2449694$$

$$\boxed{\frac{R}{H} \approx 1.245}$$