AE703: Computational Methods

Term Paper

Name: Girish Varma C Roll Number: 241010023

Abstract

This study examines the stability and behaviour of the second-order Runge-Kutta (RK2) method for solving convection-diffusion equations, with a focus on the stability function, order of accuracy, and numerical implementation using Discrete Fourier Transform (DFT). The stability criterion $|\xi| \leq 1$ is analyzed to determine the maximum allowable time step (Δt max) for various wavenumbers (k). Three MATLAB implementations are presented: (1) determining Δt max by plotting the stability function for a range of wavenumbers, (2) evaluating the stability and behaviour of solutions for stable ($\Delta t = 0.9 \cdot \Delta t$ max), critical ($\Delta t = \Delta t$ max), and unstable ($\Delta t = 1.1 \cdot \Delta t$ max) conditions, and (3) comparing overlapping solutions for Δt max and $0.9 \cdot \Delta t$ max. The results demonstrate the transition from smooth, stable solutions to instability with increasing Δt, emphasizing the role of DFT in capturing numerical behaviour and extending the findings to higher-order methods.

Question Number 1:

Part (a):

To find a stable Δt, from the fact |ξ| ≤ 1 must be satisfied for RK2, plot |ξ| against Δt for each k to obtain the Δtmax possible

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0.05\,\frac{\partial^2 u}{\partial x^2}$$

$$u(x,0) = \begin{cases} 1 - 25(x-0.2)^2 & \text{if } 0 \leq x < 0.4 \\ 0 & \text{otherwise} \end{cases}$$

Role of DFT and IDFT

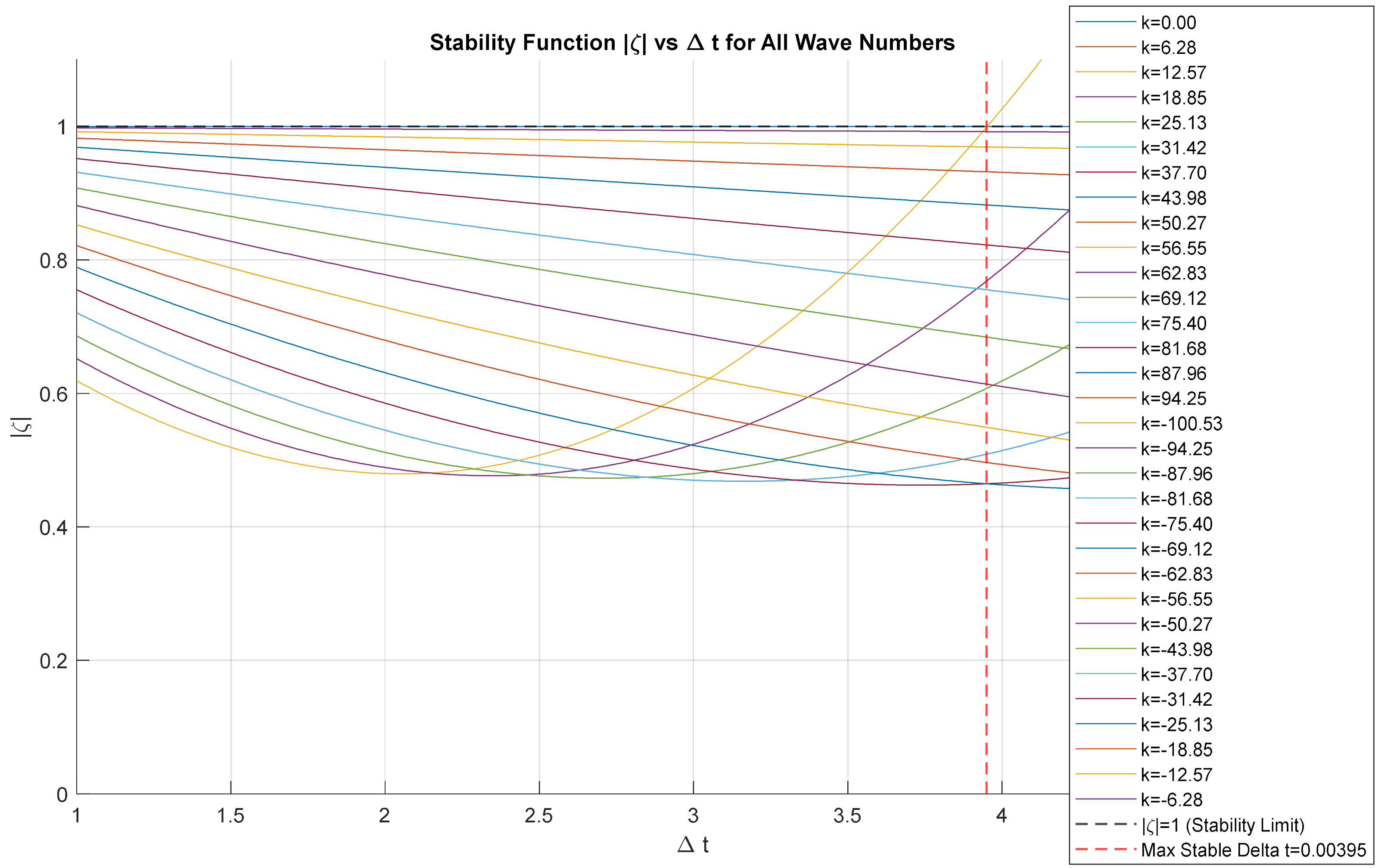
All three codes implement DFT and IDFT for frequency-domain analysis:

- **Manual Implementation**: DFT and IDFT are manually coded, allowing for direct control and analysis of frequency components without relying on MATLAB's built-in FFT/IFFT functions.
- **Frequency Domain Insights**: By transforming the solution to the frequency domain, the impact of numerical parameters on individual wavenumbers is directly observable, aiding in stability analysis and detecting instability.
- **Reconstruction Accuracy**: The IDFT ensures accurate reconstruction of the solution in the spatial domain, validating the physical consistency of numerical results.

```matlab
clc; clear all; close all;
L = 1.0;          % Domain length
N = 32;           % Number of grid points
dx = L / N;
k = (2 * pi / L) * [0:(N/2 - 1), -(N/2):-1]; % Wavenumbers
D = 0.05;         % Diffusion coefficient
dt_values = linspace(0.001, 0.005, 500); % Range of Delta t
zeta_values = zeros(length(dt_values), length(k));
for i = 1:length(dt_values)
    dt = dt_values(i);
    for j = 1:length(k)
        zeta_values(i, j) = abs(1 + dt * (1i * k(j) - D * k(j)^2) + ...
                               0.5 * (dt * (1i * k(j) - D * k(j)^2))^2);
    end
end
zeta_max_per_dt = max(zeta_values, [], 2); % Max |zeta| for each dt
stable_indices = find(zeta_max_per_dt <= 1);
if ~isempty(stable_indices)
    dt_max = dt_values(stable_indices(end));
else
    error('No stable Delta t found in the given range.');
end
fprintf('Maximum stable Delta t: %.5f\n', dt_max);
figure;
hold on;
for j = 1:length(k)
    plot(dt_values, zeta_values(:, j), 'DisplayName', sprintf('k=%.2f', k(j)));
end
yline(1, 'k--', 'LineWidth', 1, 'DisplayName', '|\zeta|=1 (Stability Limit)','Interpreter', 'tex');
xline(dt_max, 'r--', 'LineWidth', 1, ...
    'DisplayName', sprintf('Max Stable Delta t=%.5f',dt_max));
ylim([0, 1.1]);
xlim([0.001, 0.005]);
xlabel('\Delta t', 'Interpreter', 'tex');
ylabel('|\zeta|', 'Interpreter', 'tex');
title('Stability Function |\zeta| vs \Delta t for All Wave Numbers', 'Interpreter', 'tex');
legend('show', 'Interpreter', 'tex');
grid on;
hold off;
```

Output:

```
Maximum stable Delta t: 0.00395
```


Stability Function |ζ| vs Δ t for All Wave Numbers
|ζ|
0
0.2
0.4
0.6
0.8
1
1
1.5
2
2.5
3
3.5
4
Δ t
k=0.00
k=6.28
k=12.57
k=18.85
k=25.13
k=31.42
k=37.70
k=43.98
k=50.27
k=56.55
k=62.83
k=69.12
k=75.40
k=81.68
k=87.96
k=94.25
k=-100.53
k=-94.25
k=-87.96
k=-81.68
k=-75.40
k=-69.12
k=-62.83
k=-56.55
k=-50.27
k=-43.98
k=-37.70
k=-31.42
k=-25.13
k=-18.85
k=-12.57
k=-6.28
|ζ|=1 (Stability Limit)
Max Stable Delta t=0.00395

Part (b):

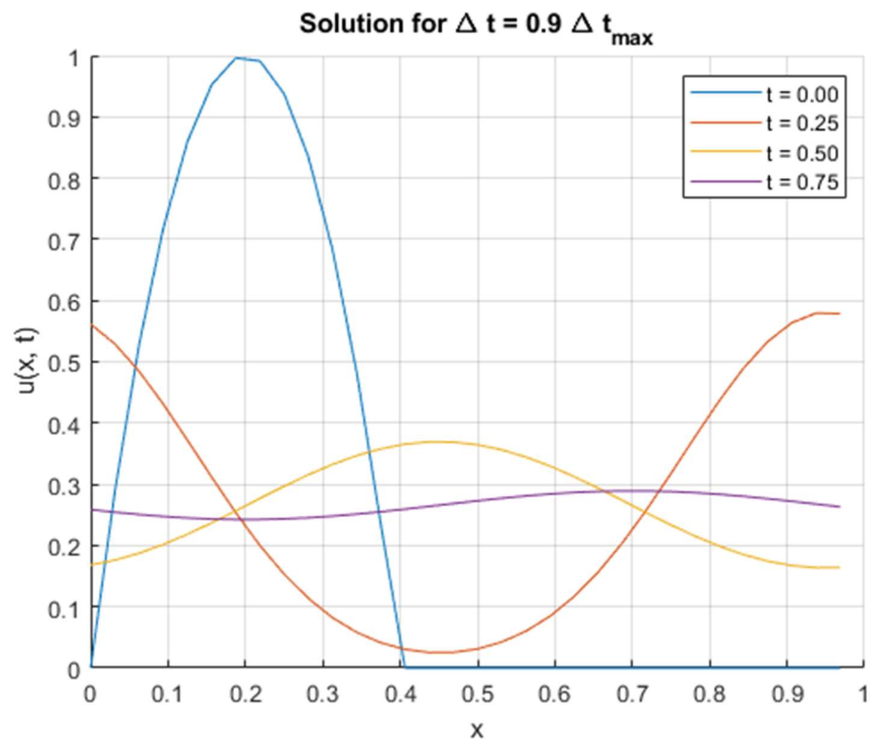
Use a Δt slightly less than Δtmax and plot the solutions u(x, t) against x for t = 0, 0.25, 0.5 and 0.75. The solution should convect and diffuse in time.

Considering Delta t = 0.9* Delta t max

```
clc; clear all; close all;
L = 1.0;          % Domain length
N = 32;           % Number of grid points
dx = L / N;
x = linspace(0, L - dx, N);
D = 0.05;         % Diffusion coefficient
t_steps = [0, 0.25, 0.5, 0.75]; % Time steps to plot

dt_max = 0.00395; % Known maximum stable Delta t
dt = 0.9 * dt_max; % Use Delta t = 0.9 * Delta t_max
u_initial = zeros(size(x));
u_initial(x >= 0.0 & x < 0.4) = 1 - 25 * (x(x >= 0.0 & x < 0.4) - 0.2).^2;
k = (2 * pi / L) * [0:(N/2 - 1), -(N/2):-1];
dft = @(u) arrayfun(@(n) sum(u .* exp(-1i * k(n) * x)), 1:N);
idft = @(uk) real(arrayfun(@(j) sum(uk .* exp(1i * k * x(j))) / N, 1:N));
fft_u = dft(u_initial);
solutions = u_initial;
% RK2
for t_idx = 2:length(t_steps)
    steps = round(t_steps(t_idx) / dt);
    for step = 1:steps
        du = dt * (1i * k .* fft_u - D * k.^2 .* fft_u);
        fft_u_half = fft_u + 0.5 * du;
        du_half = dt * (1i * k .* fft_u_half - D * k.^2 .* fft_u_half);
        fft_u = fft_u + du_half;
    end
    u_new = idft(fft_u);
    solutions = [solutions; u_new];
end
figure;
hold on;
for t_idx = 1:length(t_steps)
    plot(x, solutions(t_idx, :), 'DisplayName', sprintf('t = %.2f', t_steps(t_idx)));
end
xlabel('x', 'Interpreter', 'tex');
ylabel('u(x, t)', 'Interpreter', 'tex');
title('Solution for \Delta t = 0.9 \Delta t_{max}', 'Interpreter', 'tex');
legend('show', 'Interpreter', 'tex');
grid on;
hold off;
```

Output:



Part (c):

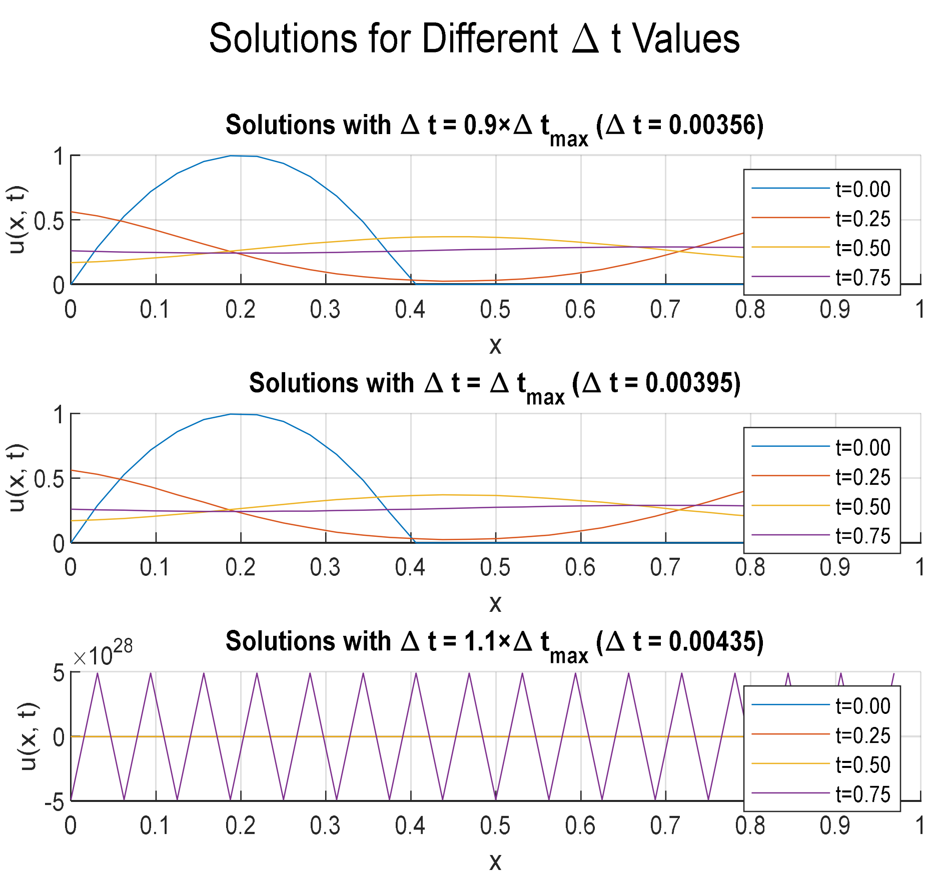
Use a Δt slightly larger than Δtmax and observe the differences over part (b)

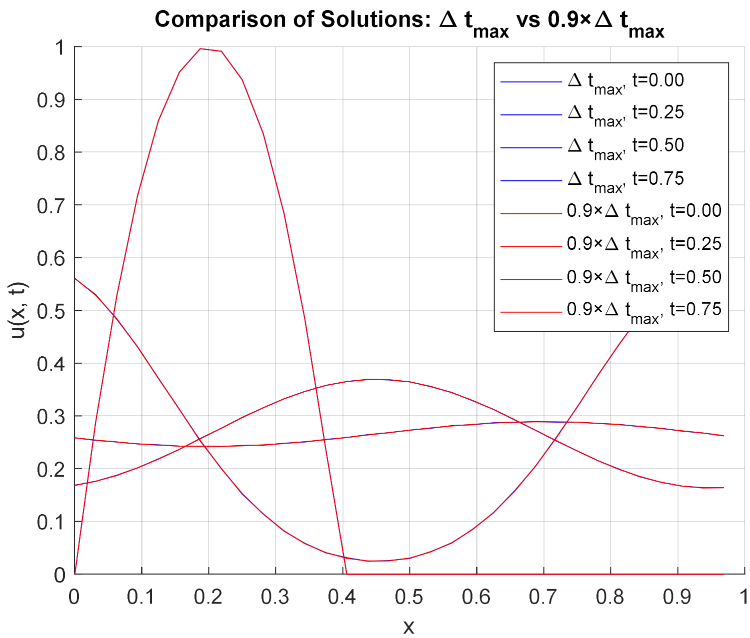
Considering Delta t = 1.1* Delta t max and comparing with (b)

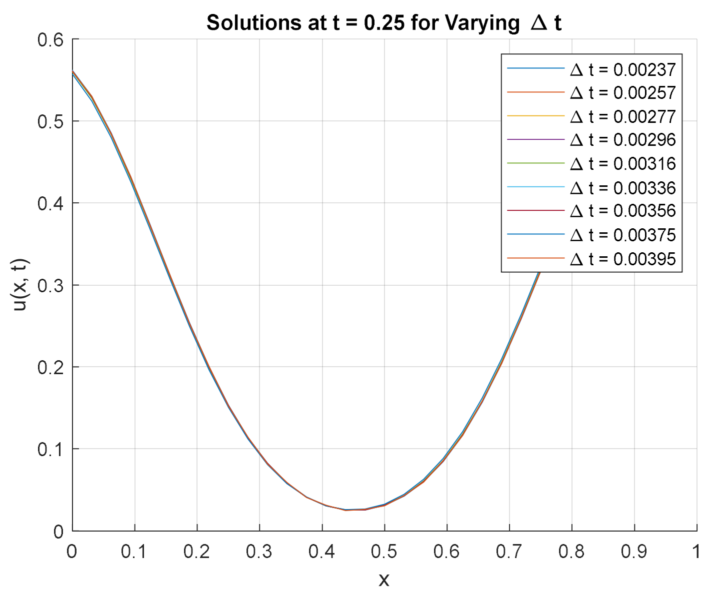
```
clc; clear all; close all;
L = 1.0;          % Domain length
N = 32;           % Number of grid points
dx = L / N;
x = linspace(0, L - dx, N);
D = 0.05;
t_steps = [0, 0.25, 0.5, 0.75];
dt_max = 0.00395; % Known maximum stable Delta t
dt_values = [0.9 * dt_max, dt_max, 1.1 * dt_max];
dt_labels = {'0.9×\Delta t_{max}', '\Delta t_{max}', '1.1×\Delta t_{max}'};
num_cases = length(dt_values);
u_initial = zeros(size(x));
u_initial(x >= 0.0 & x < 0.4) = 1 - 25 * (x(x >= 0.0 & x < 0.4) - 0.2).^2;
k = (2 * pi / L) * [0:(N/2 - 1), -(N/2):-1];
dft = @(u) arrayfun(@(n) sum(u .* exp(-1i * k(n) * x)), 1:N);
idft = @(uk) real(arrayfun(@(j) sum(uk .* exp(1i * k * x(j))) / N, 1:N));
all_solutions = cell(num_cases, 1);
for case_idx = 1:num_cases
    dt = dt_values(case_idx);
    fft_u = dft(u_initial); % Manual DFT
    solutions = u_initial;
```

```
    for t = t_steps(2:end)
        steps = round(t / dt);
        for step = 1:steps
            du = dt * (1i * k .* fft_u - D * k.^2 .* fft_u);
            fft_u_half = fft_u + 0.5 * du;
            du_half = dt * (1i * k .* fft_u_half - D * k.^2 .* fft_u_half);
            fft_u = fft_u + du_half;
        end
        u_new = idft(fft_u);
        solutions = [solutions; u_new];
    end
    all_solutions{case_idx} = solutions;
end
figure;
for case_idx = 1:num_cases
    subplot(num_cases, 1, case_idx);
    hold on;
    solutions = all_solutions{case_idx};
    for i = 1:length(t_steps)
        plot(x, solutions(i, :), 'DisplayName', sprintf('t=%.2f', t_steps(i)));
    end
    title(['Solutions with \Delta t = ', dt_labels{case_idx}, ' (\Delta t = ', num2str(dt_values(case_idx), '%.5f'), ')'], 'Interpreter', 'tex')
    xlabel('x', 'Interpreter', 'tex');
    ylabel('u(x, t)', 'Interpreter', 'tex');
    legend('show', 'Interpreter', 'tex');
    grid on;
    hold off;
end
sgtitle('Solutions for Different \Delta t Values', 'Interpreter', 'tex');
```

Output:







Discussion

The three codes illustrate different aspects of stability analysis and numerical simulation using RK2 for convection-diffusion equations:

Part(a): Analyzes the stability function $|\xi|$ to determine Δt max for a range of wavenumbers (k) and plots the stability function for all k against Δt.

Part(b): Simulates the solution behaviour for three Δt values ($0.9 \cdot \Delta t$ max, Δt max, $1.1 \cdot \Delta t$ max) using manual DFT/IDFT, evaluating the transition from stable to unstable conditions.

Part(c): Compares overlapping solutions for Δt max and $0.9 \cdot \Delta t$ max 0.9·Δt max, highlighting differences in solution behaviour due to time step variations.

Order of Accuracy: RK2 achieves second-order accuracy, ensuring a global error of $O((\Delta t)^2)$). This makes it suitable for capturing smooth convection-diffusion dynamics while balancing computational costs.

Stability Function Analysis: The first code demonstrates the sensitivity of $|\xi|$ to Δt and k. As k increases, Δt max decreases, emphasizing the need to consider high-frequency components when choosing Δt.

The derivation of the Fourier Coefficients and the stability function for the RK2 Method have been attached at the end.

Analysis of Results:

Part (a): The plots of $|\xi|$ vs Δt for various k k show that stability is maintained ($|\xi| \leq 1$) for low k even at larger Δt, but the stability limit shrinks with increasing k. The maximum stable Δt max is identified as the largest Δt for which all $|\xi|$ values remain below 1.

Part (b): Solutions for $\Delta t = 0.9 \cdot \Delta t$ max are smooth and stable, while solutions for $\Delta t = \Delta t$ max show signs of marginal instability. At $\Delta t = 1.1 \cdot \Delta t$ max, the solutions exhibit oscillations and divergence, reflecting numerical instability.

Part (c): The overlapping plots highlight subtle differences in solution behaviour between Δt max and $0.9 \cdot \Delta t$ max, with the latter producing slightly smoother results due to enhanced stability margins.

Comparison with Higher-Order Methods:

Relaxed Stability Constraints: RK3 and RK4 allow larger Δt for stability compared to RK2, with RK4 often permitting time steps up to twice as large.

Enhanced Accuracy: The fourth-order accuracy of RK4 significantly reduces truncation error, making it ideal for high-fidelity simulations of convection-diffusion problems.

Computational Trade-Offs: The added cost of higher-order methods is offset by their improved efficiency for larger time steps.

Conclusion

This study comprehensively examines the stability and behaviour of the RK2 method for solving convection-diffusion equations, emphasizing stability function analysis and the role of DFT. The key findings are:

Stability and Accuracy: RK2 achieves second-order accuracy but requires careful adherence to stability constraints ($|\xi| \leq 1$) to avoid numerical instability.

Transition from Stability to Instability: Solutions are smooth and accurate for $\Delta t < \Delta t$ max, but instability arises rapidly beyond Δt max, as demonstrated by the second and third codes.

Higher-Order Methods: RK3 and RK4 offer enhanced accuracy and relaxed stability constraints, making them attractive alternatives for more complex simulations.