

1 INTRODUCTION

Consider the linear convection-diffusion equation

$$\frac{\partial T}{\partial t} + v \frac{\partial T}{\partial y} = \alpha \frac{\partial^2 T}{\partial y^2}, \quad v = \sin(\pi y), \quad \alpha = \frac{1}{\text{Pr Re}}, \quad (1)$$

subject to the initial conditions

$$\begin{aligned} \text{(a)} \quad T(y, t = 0) &= \cos(2\pi y) \sin(\pi y) \\ \text{(b)} \quad T(y, t = 0) &= \cos(2\pi y), \end{aligned} \quad (2)$$

and parameters

$$\begin{aligned} \text{Re} &= 1 && \text{(Reynolds number, molten glass)} \\ \text{Pr} &= 25 && \text{(Prandtl number, molten glass)} \\ \Delta t &= 0.001 && \text{(Time step)} \\ L_y &= 2 && \text{(Domain } y\text{-length)} \\ N &= 2^n + 1 && \text{(Number of } y\text{-points, where } n = 5, 6). \end{aligned} \quad (3)$$

1.1 PROBLEM 1

Use the Fourier pseudo-spectral (FPS) method to numerically integrate (1) with the given parameters. Use the Euler explicit method for time advancement. Higher resolution with $n = 6$ will improve the accuracy of the method for the initial condition (a). Plot T as a function of time t at $t = \{0.2, 2, 5, 10\}$.

1.2 PROBLEM 2

Use the FTCS Euler explicit method with second-order finite differences for the same computation, and compare results to the FPS method using the same mesh resolution.

2 METHODOLOGY

2.1 PROBLEM 1

We re-write (1) as

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial y^2} - v \frac{\partial T}{\partial y}. \quad (4)$$

The first and second spatial derivatives of T are calculated by taking the Fourier transform of T , multiplying the Fourier coefficients by ik_n and $-k_n^2$, respectively to take the first and second derivatives in Fourier space, and then taking the inverse Fourier transform to recover the first and second derivatives in physical space. With values of $\partial T / \partial t$ known at all grid points, values of T at the next time step are calculated using the explicit Euler method,

$$T^{n+1} = T^n + \Delta t \frac{\partial T}{\partial t}. \quad (5)$$

2.2 PROBLEM 2

To implement the FTCS explicit method, we approximate (1) using forward-differences in time and central-differences in space as

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} + v \frac{T_{i+1}^n - T_{i-1}^n}{2\Delta y} = \alpha \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{\Delta y^2}, \quad (6)$$

which can be solved for T_i^{n+1} as

$$T_i^{n+1} = T_i^n + \Delta t \left(\alpha \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{\Delta y^2} - v \frac{T_{i+1}^n - T_{i-1}^n}{2\Delta y} \right). \quad (7)$$

This constitutes an explicit equation for T^{n+1} ; we can advance in time without the need to solve any matrix equations. We enforce periodic boundary conditions by wrapping T_{i-1} to T_N when $i = 1$ and vice versa.

3 RESULTS

Since most interesting behavior occurs for $t < 4$, we neglect to display solutions for the later requested times of $t = \{5, 10\}$. Based on computations performed but not presented here, the trends observed during $t < 4$ can be extrapolated in these cases.

3.1 PROBLEM 1

Solutions to the linear convection-diffusion equation using the FPS method are shown in Figure 1.

3.2 PROBLEM 2

Solutions using the FTCS method are compared to those from the FPS method in Figure 2. The difference between the two methods, $T_{\text{FPS}} - T_{\text{FTCS}}$, at various time steps is shown in Figure 3.

4 DISCUSSION

Both methods qualitatively predict the convection and diffusion behavior of both initial conditions at both mesh resolutions fairly well. In agreement with the given expression for convective velocity v , T is convected toward $y = 1$ at a rate sinusoidally dependent upon the distance to the center of the spatial domain. For initial condition (a), the initial condition sets $T = 0$ at all points where $v = 0$. Thus as positive and negative values of T convect equally toward the domain center, diffusion smooths the equilibrium T field to zero at all spatial locations. For initial condition (b), locations where $v = 0$ correspond to locations where $T = 1$. The negative values of T convect toward the center rapidly and diffuse, resulting in an equilibrium value at all spatial points of $T \sim 0.765$.

Though the FPS and FTCS methods appear qualitatively similar in Figure 2, the error plots in Figure 3 reveal some shortcomings of the latter. First, as the grid is refined in Figure 3, it is no surprise that the point-wise discrepancies between methods decrease in magnitude. Furthermore, the FTCS method for initial condition (b) has difficulty capturing diffusion accurately. The equilibrium FPS value is $T \sim 0.765$ for both $n = \{5, 6\}$ (likely grid-independent), whereas the FTCS method's limiting values are $T \sim \{0.806, 0.788\}$, respectively, and trend toward the equilibrium value of the FPS method with roughly 5% and 3% error. If we take the FPS solution as 'truth,' which is not unreasonable for these periodic problems, the FTCS method suffers during the convection-dominated stage of the solution as well. Here, absolute point-wise errors can be as high as 0.04 for $n = 5$, which constitutes a 4% error relative to the maximum initial T .

We conclude that the FPS method is superior to the FTCS method for period problems such as the linear convection-diffusion equation studied here, especially for low- n grids.

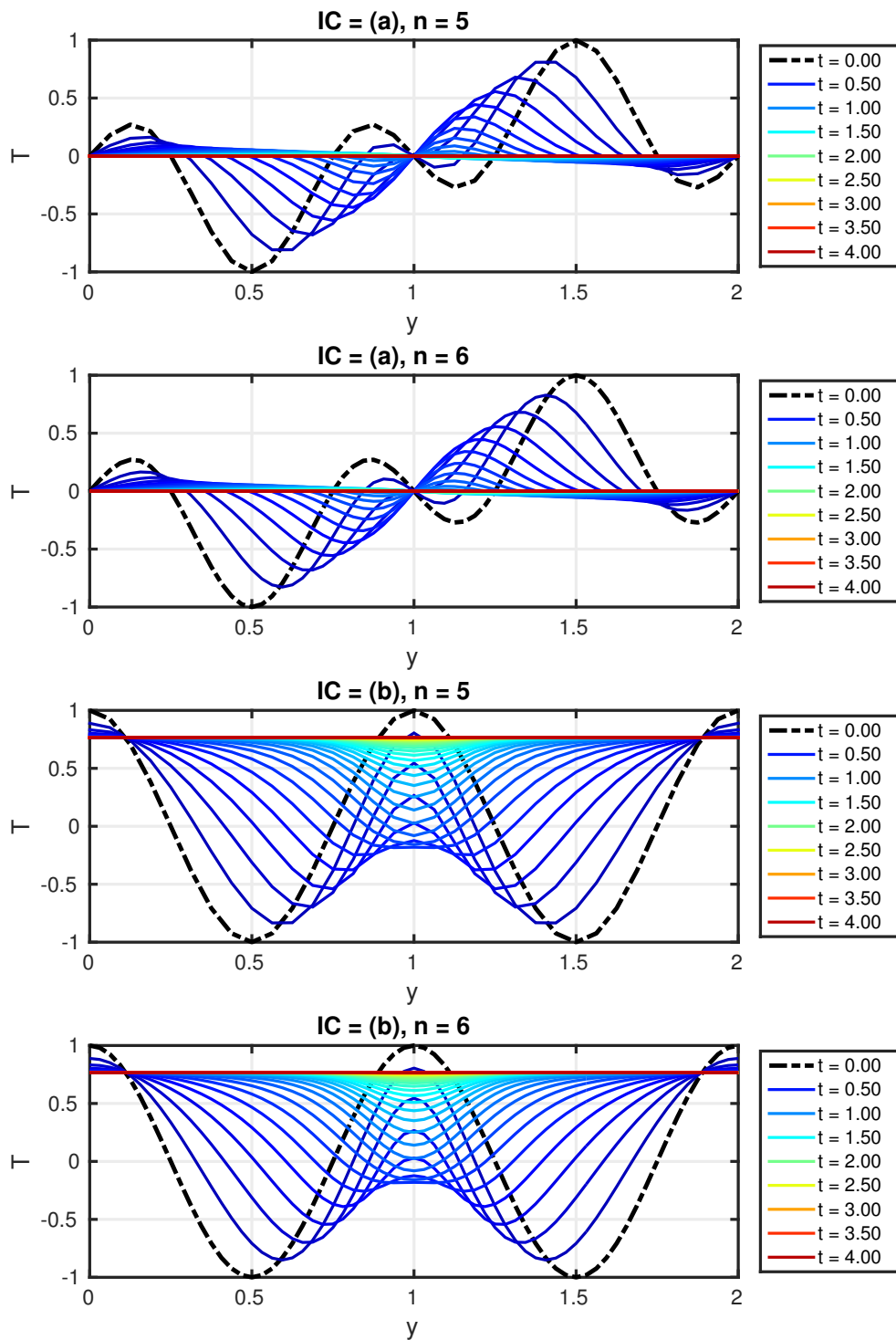


Figure 1: Solutions to the linear convection-diffusion equation using the FPS method for different initial conditions and mesh resolution parameters. Most interesting behavior occurs when $t < 4$. Trends can be extrapolated to future times $t = 5, 10$, which are not shown. Initial condition displayed as black dot-dashed line ($\cdot - \cdot -$).

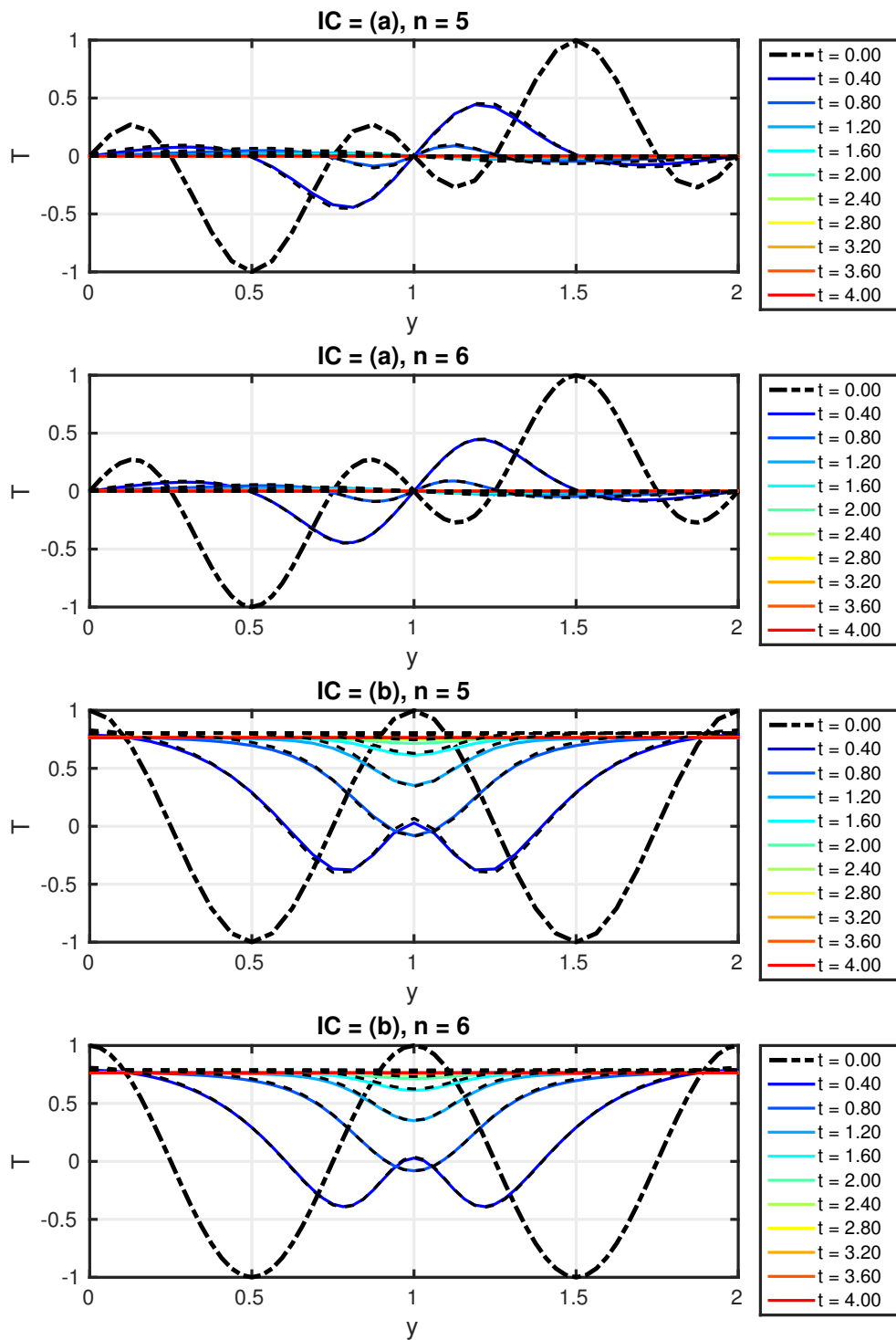


Figure 2: Solution comparison between the FPS method (colored lines) and the FTCS method (---) for $t < 4$. Initial condition displayed as black dot-dashed line ($\cdot-\cdot-$). Agreement between the FPS and FTCS methods is decent. The limiting FPS value is $T \sim 0.765$ for both $n = \{5, 6\}$, whereas the FTCS method's limiting values are $T \sim \{0.806, 0.788\}$ respectively. Further analysis is deferred to Figure 3.

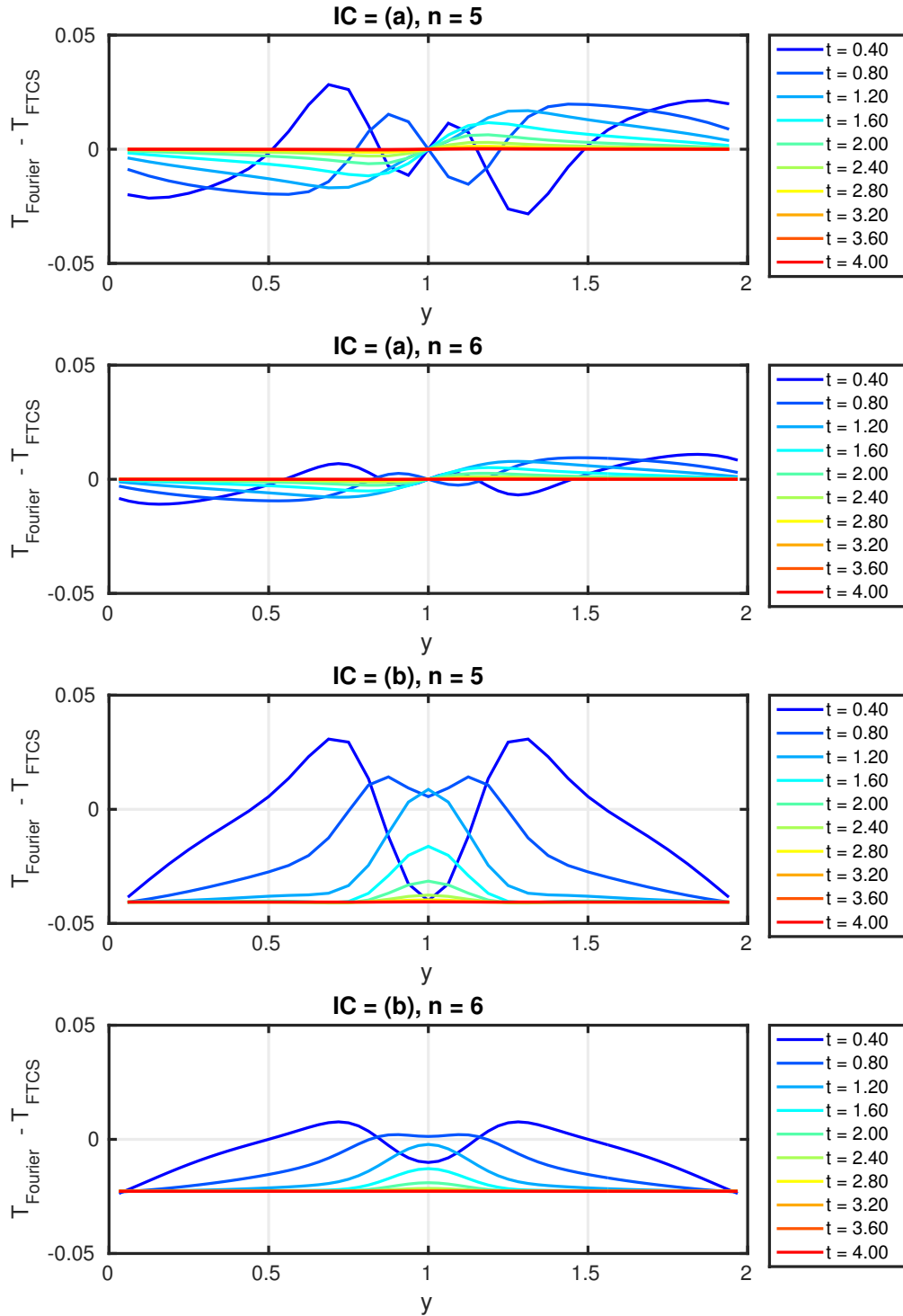


Figure 3: Difference between the FPS and FTCS solutions at evenly-spaced discrete times $t < 4$, omitting the initial condition. Different initial conditions and spatial resolutions produce different behavior.

5 REFERENCES

No external references were used other than the course notes for this assignment.

APPENDIX: MATLAB CODE

The following code listings generate all figures presented in this homework assignment. The Fourier codes `find_dfdn.m` and `find_d2fdn2.m` are provided by Prof. Biringen.

Listing 1: Problem_1.m

```
1 function [T_history] = Problem_1(varargin)
2
3     %%%%%%
4     % Solves the linear convection-diffusion equation using the Fourier pseudo-spectral
5     % method.
6     %
7     % Ryan Skinner, November 2015
8     %%%
9
10    Set_Default_Plot_Properties();
11
12    switch length(varargin)
13        case 0
14            n_plot = 41;
15        case 1
16            n_plot = varargin{1};
17        otherwise
18            error('Too many arguments passed to Problem_1');
19    end
20
21    cases = {'a',5},{'a',6},{'b',5},{'b',6}};
22    T_history = cell(length(cases),1);
23    for case_i = 1:length(cases)
24
25        IC_str = cases{case_i}{1};
26        n = cases{case_i}{2};
27
28        % Spatial domain.
29        nn = 2^n + 1;
30        Ly = 2.0;
31        y = linspace(0, Ly, nn)';
32
33        % Temporal domain.
34        dt = 0.001;
35        if strcmp(IC_str,'a')
36            t_final = 4;
37        else
38            t_final = 4;
39        end
40        t = [0:dt:t_final];
41
42        % Physical parameters.
43        Re = 1;
44        Pr = 25;
45        alpha = 1 / (Re * Pr);
46        v = sin(pi*y);
47
48        % Initialize solution.
49        T = nan(nn,length(t));
50        if strcmp(IC_str,'a')
51            T(:,1) = cos(2*pi*y) .* sin(pi*y);
52        else
53            T(:,1) = cos(2*pi*y);
54        end
55    end
```

```

56     %%%
57     % Solve problem numerically.
58     %%%
59
60     for t_n = 1:(length(t)-1)
61
62         Tn = T(:,t_n);
63
64         dTdy = find_dfdn( Tn',nn,Ly)';
65         d2Tdy2 = find_d2fdn2(Tn',nn,Ly)';
66
67         % Update solution.
68         T(:,t_n+1) = Tn + dt * (alpha * d2Tdy2 - v .* dTdy);
69
70     end
71
72     %%%
73     % Process results.
74     %%%
75
76     cmap = jet(n_plot);
77     step_numbers = round(linspace(1,length(t),n_plot));
78     hf = figure(case_i);
79     set(hf,'Position',[100,500,900,300]);
80     hold on;
81     plot_handles = [];
82     for t_n = 1:length(step_numbers)
83         tmp = sprintf('t = %.2f', t(step_numbers(t_n)));
84         if t_n == 1
85             hp = plot(y, T(:,step_numbers(t_n)), 'k-', 'LineWidth', 3, 'DisplayName', tmp);
86         else
87             hp = plot(y, T(:,step_numbers(t_n)), 'DisplayName', tmp, 'Color', cmap(t_n,:));
88         end
89         if(mod(t_n-1,5) == 0 || n_plot <= 11)
90             plot_handles(end+1) = hp;
91         end
92     end
93     title(sprintf('IC = (%s), n = %.0f',IC_str,n));
94     xlabel('y');
95     ylabel('T');
96     ylim([-1,1]);
97     xlim([0,Ly]);
98     hleg = legend(plot_handles);
99     set(hleg,'Location','eastoutside');
100
101     T_history{case_i} = T;
102
103     end
104
105     disp('Done.');
```

```

106
107 end
```

Listing 2: find_dfdn.m

```

1 function dfdn = find_dfdn(f,nn,L)
2 % ASEN 5327 Class Notes Fall 2010
3 N=nn-1;
4 n=[0:(N/2)-1 -(N/2):-1];
5 %L=1; %show fuction is periodic on wavelength of L
6 kn=2*pi*n./L;
7 kn((N/2)+1)=0;
8
9 %find Fourier components
10 f_tilde=fft(f,N);
11 dfdn=ifft(i*kn.*f_tilde,N);
12 dfdn(nn)=dfdn(1);
13
14 end
```

Listing 3: find_d2fdn2.m

```
1 function d2fdn2 = find_d2fdn2(f,nn,L)
2 %ASEN5327 Class notes Fall 2010
3 N=nn-1;
4 n=[0:(N/2)-1 -(N/2):-1];
5 kn=2*pi*n./L;
6 kn((N/2)+1)=0;
7 f_tilde=fft(f,N);
8 d2fdn2=ifft(-(kn.^2).*f_tilde,N);
9 d2fdn2(nn)=d2fdn2(1);
10 end
```

Listing 4: Problem_2.m

```
1 function [] = Problem_2()
2
3 %%%%%
4 % Solves the linear convection-diffusion equation using the FTCS explicit method.
5 %
6 % Ryan Skinner, November 2015
7 %%%
8
9 Set_Default_Plot_Properties();
10
11 % Display results from Problem 1 at only a few time steps.
12 n_plot = 11;
13 T_prob1 = Problem_1(n_plot);
14
15 cases = {'a',5},{'a',6},{'b',5},{'b',6}};
16 for case_i = 1:4
17
18     IC_str = cases{case_i}{1};
19     n = cases{case_i}{2};
20
21     % Spatial domain.
22     nn = 2^n + 1;
23     Ly = 2.0;
24     y = linspace(0, Ly, nn)';
25     dy = y(2) - y(1);
26
27     % Temporal domain.
28     dt = 0.001;
29     if strcmp(IC_str,'a')
30         t_final = 4;
31     else
32         t_final = 4;
33     end
34     t = [0:dt:t_final];
35
36     % Physical parameters.
37     Re = 1;
38     Pr = 25;
39     alpha = 1 / (Re * Pr);
40     v = sin(pi*y);
41
42     % Initialize solution.
43     T = nan(nn,length(t));
44     if strcmp(IC_str,'a')
45         T(:,1) = cos(2*pi*y) .* sin(pi*y);
46     else
47         T(:,1) = cos(2*pi*y);
48     end
49
50 %%%
51 % Solve problem numerically.
```



```

52     %%%
53
54     for t_n = 1:(length(t)-1)
55
56         Tn = T(:,t_n);
57
58         for i = 1:nn
59             if i == 1 ; Tn1=Tn(nn); else Tn1=Tn(i-1); end % Periodic BCs
60             if i == nn; Tn1=Tn( 1); else Tn1=Tn(i+1); end % Periodic BCs
61             T(i,t_n+1) = Tn(i) + dt * ( alpha * (Tn1 - 2*Tn(i) + Tn1) / (dy^2) ...
62                                     - v(i) * (Tn1          - Tn1) / (2*dy) );
63         end
64     end
65
66     %%%
67     % Process results.
68     %%%
69
70
71     step_numbers = round(linspace(1,length(t),n_plot));
72     figure(case_i);
73     hold on;
74     for t_n = 2:length(step_numbers)
75         plot(y, T(:,step_numbers(t_n)), 'k--');
76     end
77
78     cmap = jet(n_plot);
79     hf = figure(length(cases) + case_i);
80     set(hf,'Position',[100,500,900,300]);
81     hold on;
82     plot_handles = [];
83     for t_n = 2:length(step_numbers)
84         tmp = sprintf('t = %.2f', t(step_numbers(t_n)));
85         method_err = (T_prob1{case_i}(:,step_numbers(t_n)) - T(:,step_numbers(t_n)));
86         hp = plot(y(2:end-1), method_err(2:end-1), 'DisplayName', tmp, 'Color', cmap(t_n,:));
87         if(mod(t_n-1,5) == 0 || n_plot <= 11)
88             plot_handles(end+1) = hp;
89         end
90     end
91     title(sprintf('IC = (%s), n = %.0f',IC_str,n));
92     xlabel('y');
93     ylabel('T_{Fourier} - T_{FTCS}');
94     ylim([-0.05,0.05]);
95     xlim([0,Ly]);
96     hleg = legend(plot_handles);
97     set(hleg,'Location','eastoutside');
98
99     end
100
101     disp('Done. ');
102
103 end

```