

Problem 1

Solve the one-way wave equation (hyperbolic PDE):

$$u_t + u_x = 0$$

where

$$u(x, 0) = u_0(x) = \begin{cases} 1 - |x| & \text{if } |x| < 1 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Use **Lax-Friedrichs** with space and time domains of $x \in [-2, 10]$ and $t \in [0, 8]$ respectively. Use left boundary condition $u(-2, t) = u_0(-2 - t)$ and right boundary condition $u(10, t) = u_0(10 - t)$.

Problem 1: Set Up

Lax-Friedrichs defines the equation $u_t + u_x = 0$ as...

$$\frac{u_i^{j+1} - \frac{1}{2}(u_{i+1}^j + u_{i-1}^j)}{\Delta t} + \frac{(u_{i+1}^j + u_{i-1}^j)}{2\Delta t} = 0$$

Factoring out u_j^{j+1} ...

$$u_i^{j+1} = \frac{1}{2}(u_{i+1}^j + u_{i-1}^j) - \frac{\Delta t}{2\Delta x}(u_{i+1}^j + u_{i-1}^j)$$

Defining $r = \frac{\Delta t}{\Delta x}$ and further simplifying, creates the needed finite difference scheme...

$$u_i^{j+1} = \frac{1-r}{2}(u_{i+1}^j) + \frac{1+r}{2}(u_{i-1}^j)$$

Creating a system of matrices, excluding $u_0^j = u(-2, t)$ and $u_N^j = u(10, t)$ from the first and final rows, respectively, leaves...

$$\vec{u}_i^{j+1} - \begin{bmatrix} \frac{1+r}{2}u_0^j \\ 0 \\ \vdots \\ 0 \\ \frac{1-r}{2}u_N^j \end{bmatrix} = \begin{bmatrix} 0 & \frac{1-r}{2} & 0 & & \\ \frac{1+r}{2} & 0 & \frac{1-r}{2} & & \\ 0 & \frac{1+r}{2} & 0 & \ddots & \\ & & \ddots & \ddots & \frac{1-r}{2} \\ & & & \frac{1+r}{2} & 0 \end{bmatrix} \begin{bmatrix} u_1^j \\ u_2^j \\ \vdots \\ u_{N-2}^j \\ u_{N-1}^j \end{bmatrix}$$

For the boundary conditions $u(-2, t) = u_0(-2 - t)$ and $u(10, t) = u_0(10 - t)$, given that $t \in [0, 8]$, $u_0(-2 - t) = 0$ because $|-2 - t| > 1$ and $u_0(10 - t) = 0$ because $|10 - t| > 1$ for any $t \in [0, 8]$, which provides...

$$\vec{u}_i^{j+1} = \begin{bmatrix} 0 & \frac{1-r}{2} & 0 & & \\ \frac{1+r}{2} & 0 & \frac{1-r}{2} & & \\ 0 & \frac{1+r}{2} & 0 & \ddots & \\ & & \ddots & \ddots & \frac{1-r}{2} \\ & & & \frac{1+r}{2} & 0 \end{bmatrix} \begin{bmatrix} u_1^j \\ u_2^j \\ \vdots \\ u_{N-2}^j \\ u_{N-1}^j \end{bmatrix}$$

Problem 1: Implementation

For this problem, I chose my Δx as...

$$\Delta x = \left[\left(\frac{1}{2}\right)^7, \left(\frac{1}{2}\right)^8, \left(\frac{1}{2}\right)^9, \left(\frac{1}{2}\right)^{10} \right]$$

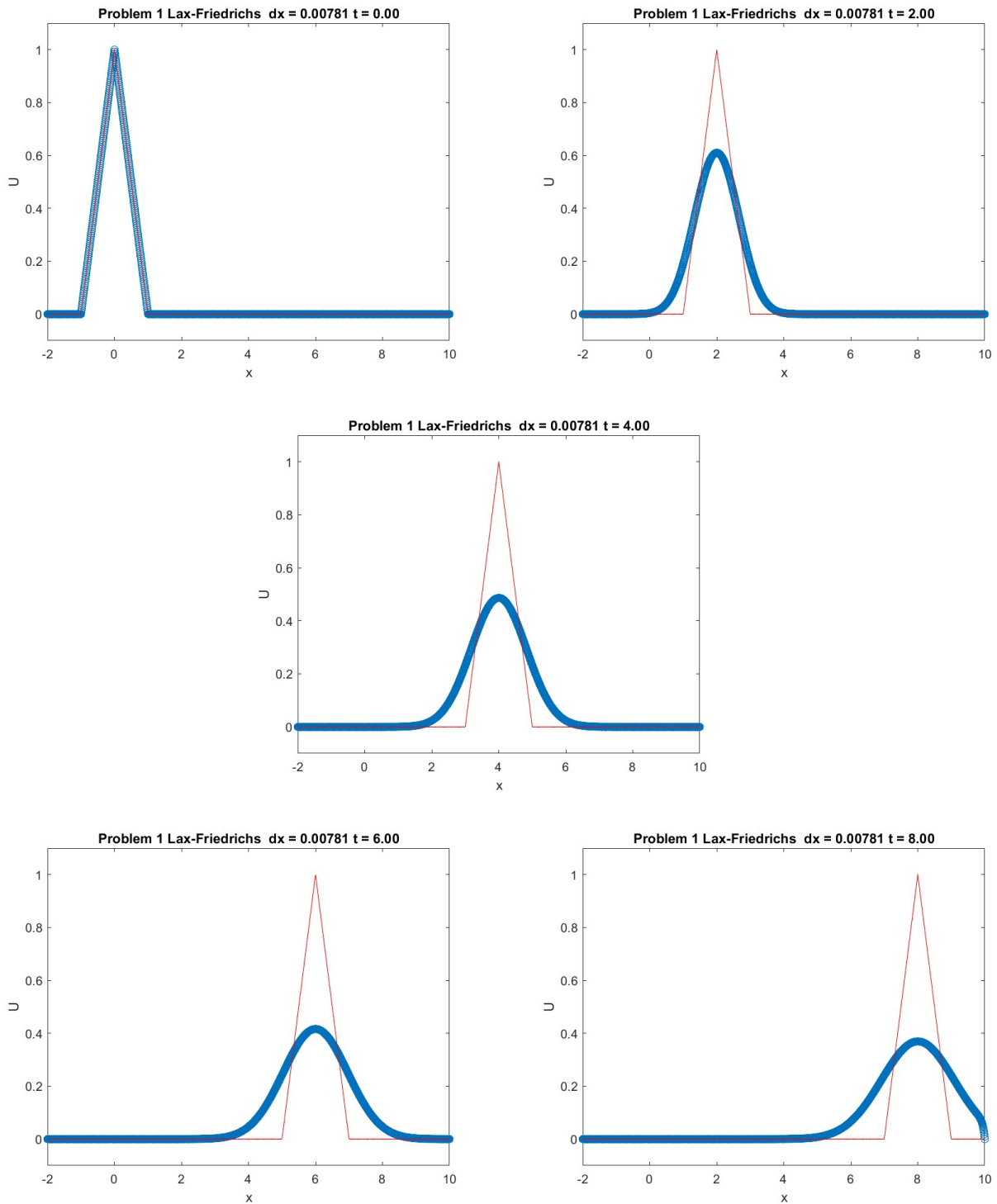
in order to have a more refined error calculation.

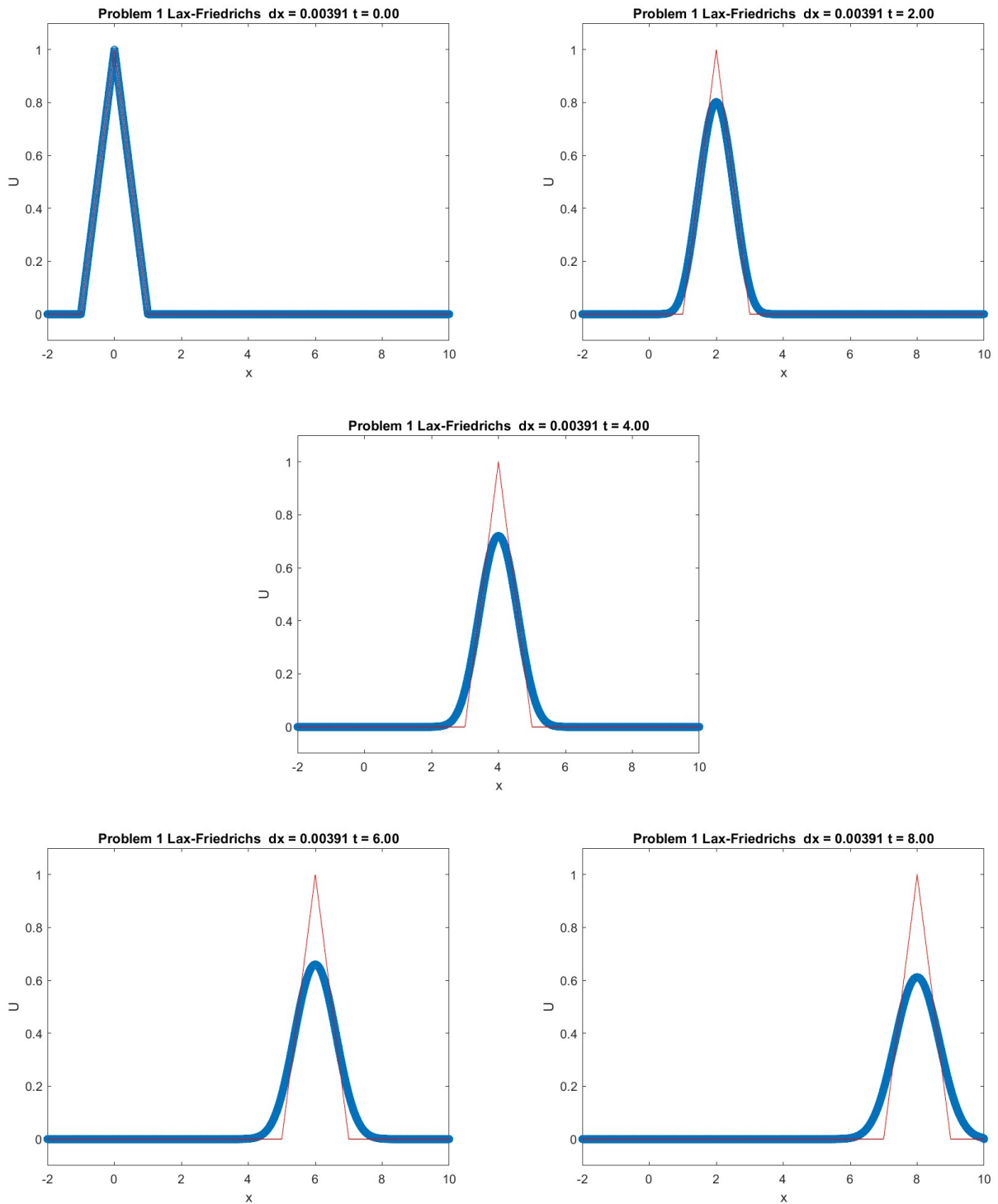
I also chose my Δt , following the Lax-Friedrich CFL, as...

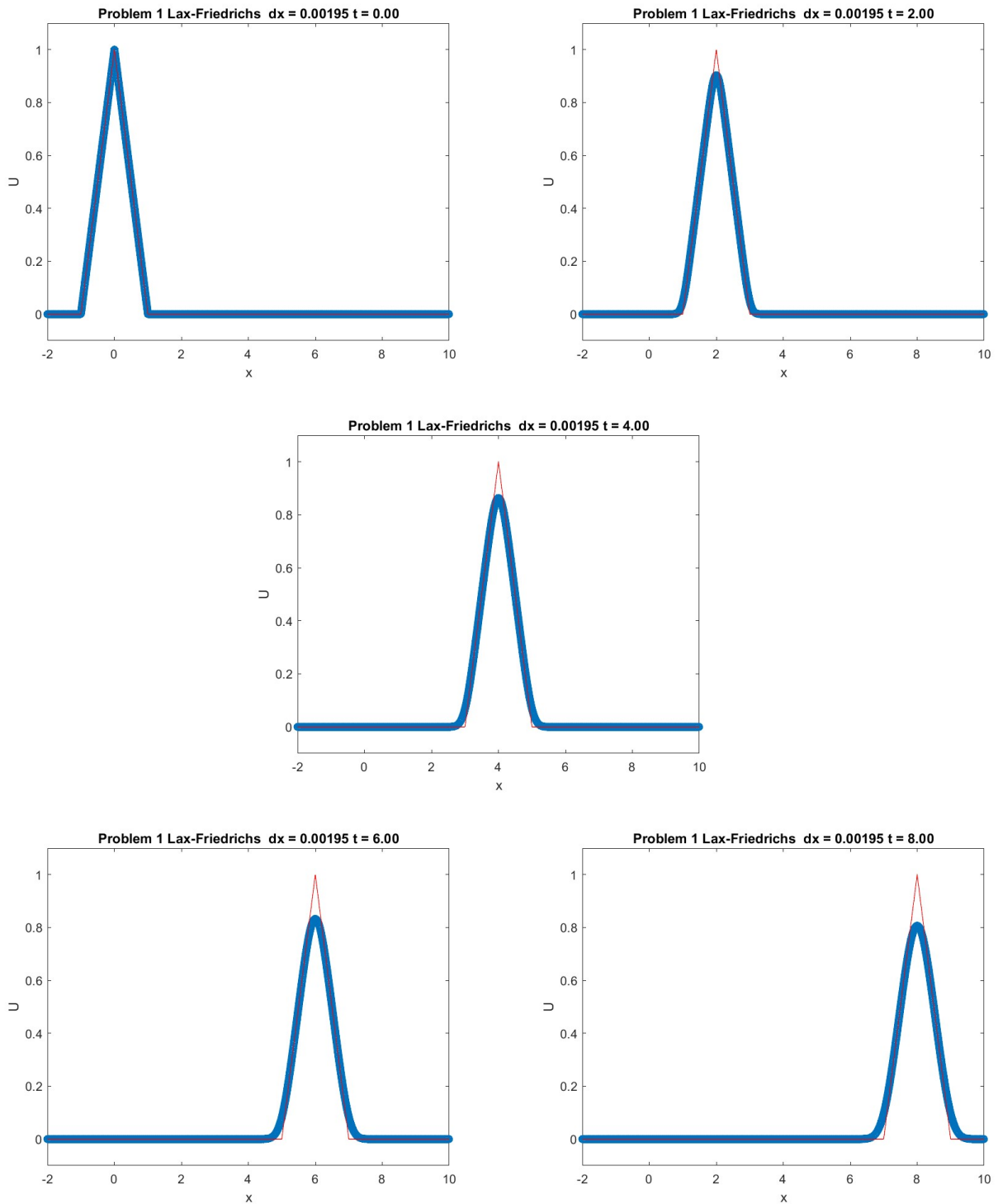
$$\Delta t = \frac{\left(\frac{1}{2}\right)^{10}}{2} \text{ such that } |r| = \left| \frac{\Delta t}{\Delta x} \right| \leq 1$$

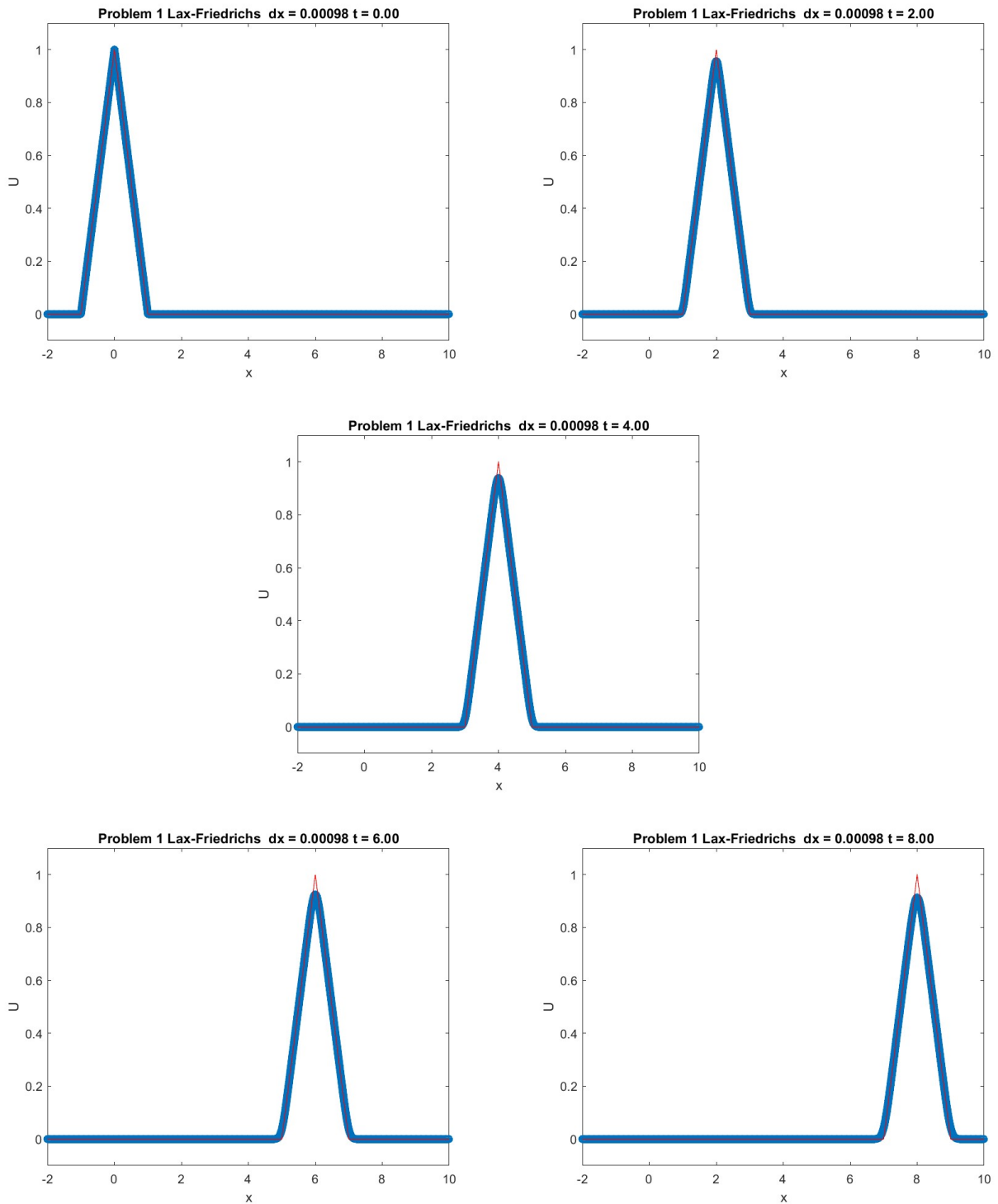
For plotting the numerical vs analytical solution, I chose to plot the Comparison when $t = [0, 2, 4, 6, 8]$

The resulting plots starting on the next page show that numerical result, in blue, follows the analytical result, in red, very well in the beginning. However as the time increases, the numerical and analytical answer begin to separate, especially for the higher Δx values.

Figure 1: $\Delta x = \left(\frac{1}{2}\right)^7$

Figure 2: $\Delta x = \left(\frac{1}{2}\right)^8$

Figure 3: $\Delta x = \left(\frac{1}{2}\right)^9$

Figure 4: $\Delta x = \left(\frac{1}{2}\right)^{10}$

Problem 1: Error Analysis

Per instructions, the error is calculated at the final time step $t = 8$. I am using the same list of Δx values as well.

One thing to note is that I decided that the best place calculate the error would be at when $x = 8$. This is because the peak of the wave is at $x = 8$ at $t = 8$, thus being a good place to calculate the accuracy of the scheme.

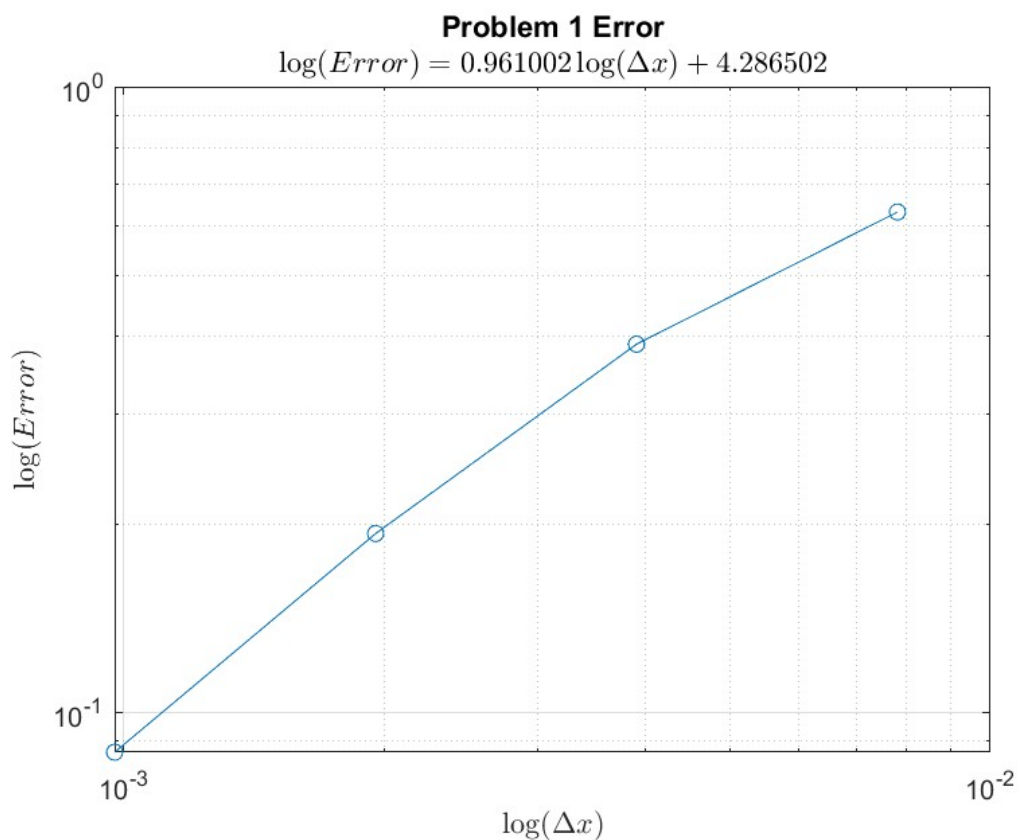


Figure 5: Problem 1 LogLog Plot of Error

The plot shows that the spatial error of this scheme is one with the coefficient of the loglog plot being one.

```

1 %% HW 7
2 % Zachary Humphries
3 % COMP 521
4 % Fall 2022
5
6 clc
7 clear
8 close all
9
10 %% Problem 1
11
12 a = -2;           % Left Boundary (x)
13 b = 10;          % Right Boundary (x)
14
15 T = 8;           % Final Time
16 plot_list = [T/4, T/2, 3*T/4, T];
17
18 dx_list = [0.5^7, 0.5^8, 0.5^9, 0.5^10];
19 dt = dx_list(end)/2; % To meet condition of |r| <= 1
20 error_list = zeros(length(dx_list), 1);
21
22 for i=1:length(dx_list)
23     dx = dx_list(i);
24     xgrid_short = [a+dx:dx:b-dx];
25     xgrid_long = [a:dx:b];
26
27     ICV = xgrid_short';
28     ICV = problem1_u0(ICV);
29
30     left_boundary = 0;
31     right_boundary = 0;
32
33     r=dt/dx;
34
35     U = ICV;
36
37     actual = problem1_u0(0-xgrid_long);
38
39     figure(1+(5*(i-1)))
40     plot([a xgrid_short b], [left_boundary U' right_boundary] , 'o-')
41     hold on
42     plot([xgrid_long], [actual] , 'r-')
43     axis([a b -0.1 1.1])
44     str = sprintf('Problem 1 Lax-Friedrichs \t dx = %.5f t = %.2f', dx, 0.0);
45     title(str)
46     xlabel('x')
47     ylabel('U')
48     hold off
49
50     A = problem1_matrix(a,b,dx,dt);
51
52     for dt_j = dt: dt : T
53         left_boundary = problem1_u0(-2-dt_j);
54         right_boundary = problem1_u0(10-dt_j);
55

```



```

56     U(1) = U(1);
57     U(end) = U(end);
58
59     U = A*U;
60
61
62     actual = problem1_u0(dt_j-xgrid_long);
63     for k=1:length(plot_list)
64         if (dt_j == plot_list(k))
65             figure(1+k+(5*(i-1)))
66             plot([a xgrid_short b], [left_boundary U' right_boundary] , 'o
-')
67
68             hold on
69             plot([xgrid_long], [actual] , 'r-')
70             axis([a b -0.1 1.1])
71             str = sprintf('Problem 1 Lax-Friedrichs \t dx = %.5f t = %.2f'
, dx, dt_j);
72             title(str)
73             xlabel('x')
74             ylabel('U')
75             hold off
76             pause(0.1)
77         end
78     end
79     index_actual = find(xgrid_long(1, :) == 8);
80     index_U = find(xgrid_short(1, :) == 8);
81     error_list(i,1) = abs(U(index_U,1) - actual(1,index_actual));
82 end
83
84 figure
85 forward_poly = polyfit(log(dx_list), log(error_list), 1);
86 loglog(dx_list, error_list, "o-"); grid on;
87 title("Problem 1 Error")
88 subtitle_name_forward = strcat("\log(Error) = ", sprintf("%2.6f",
forward_poly(1)), "\log(\Delta x) + ", sprintf("%2.6f", forward_poly(2)), "
$");
89 subtitle(subtitle_name_forward, 'interpreter', 'latex')
90 xlabel("\log(\Delta x)", 'interpreter', 'latex')
91 ylabel("\log(Error)", 'interpreter', 'latex')
92
93
94
95 function A = problem1_matrix(a,b,dx,dt)
96     r = dt/dx;
97     m = (b-a)/dx;
98     one = ones(m-1,1);
99     diag1 = (1+r)/2 * one;
100    diag2 = (1-r)/2 * one;
101
102    A = spdiags([diag1 zeros(m-1,1) diag2], -1:1,m-1,m-1);
103    A = sparse(A);
104 end
105
106 function u0x = problem1_u0(x)

```

```
107     u0x = zeros(size(x));  
108     for i=1:length(x)  
109         if abs(x(i)) < 1  
110             u0x(i) = 1-abs(x(i));  
111         else  
112             u0x(i) = 0;  
113         end  
114     end  
115 end
```

Problem 1 Matlab Code

Problem 2

Find the numerical solution for the following heat equation:

$$u_t + u_{xx} = 0 \quad \text{for } 0 < x < 1 \quad \text{and} \quad 0 \leq t \leq 0.1$$

with the initial condition $u(x, 0) = f(x) = \sin(\pi x) + \sin(3\pi x) \quad \forall x \in [0, 1]$ and boundary conditions:

$$\begin{aligned} u(0, t) = c_1 = 0 & \quad \text{for } x = 0 \quad \text{and} \quad 0 \leq t \leq 0.1 \\ u(1, t) = c_2 = 0 & \quad \text{for } x = 1 \quad \text{and} \quad 0 \leq t \leq 0.1 \end{aligned}$$

The exact solution is

$$u(x, t) = \sin(\pi x)e^{-\pi^2 t} + \sin(3\pi x)e^{-9\pi^2 t}$$

Problem 2: Set Up

The explicit scheme used in class for the heat equation $u_t = c^2 u_{tt}$ as...

$$\frac{u_i^{j+1} - u_i^j}{\Delta t} = c^2 \frac{u_{i-1}^j - 2u_i^j + u_{i+1}^j}{\Delta x^2}$$

Since $c = 1$, solving for u_i^{j+1} the equation is rewritten as...

$$u_i^{j+1} = u_i^j + r(u_{i-1}^j - 2u_i^j + u_{i+1}^j) \quad \text{with} \quad r = \frac{\Delta t}{\Delta x^2}$$

Creating a system of matrices, excluding $u_0^j = u(0, t) = 0$ and $u_N^j = u(1, t) = 0$ from the first and final rows, respectively, leaves...

$$\vec{u}_i^{j+1} - \begin{bmatrix} ru_0^j \\ 0 \\ \vdots \\ 0 \\ ru_N^j \end{bmatrix} = \begin{bmatrix} 1-2r & r & 0 & & \\ r & 1-2r & r & & \\ 0 & r & 1-2r & \ddots & \\ & & \ddots & \ddots & r \\ & & & r & 1-2r \end{bmatrix} \begin{bmatrix} u_1^j \\ u_2^j \\ \vdots \\ u_{N-2}^j \\ u_{N-1}^j \end{bmatrix}$$

Given that the boundary conditions $u_0^j = u(0, t) = 0$ and $u_N^j = u(1, t) = 0$ for any $t \in [0, 0.1]$, ru_0^j and ru_N^j can be eliminated providing...

$$\vec{u}_i^{j+1} = \begin{bmatrix} 1-2r & r & 0 & & \\ r & 1-2r & r & & \\ 0 & r & 1-2r & \ddots & \\ & & \ddots & \ddots & r \\ & & & r & 1-2r \end{bmatrix} \begin{bmatrix} u_1^j \\ u_2^j \\ \vdots \\ u_{N-2}^j \\ u_{N-1}^j \end{bmatrix}$$

Problem 2: Implementation with $\Delta x = 0.2$ and $\Delta t = 0.02$

3D mesh plots showing the numerical and exact results through time, as well as the error between the two are below

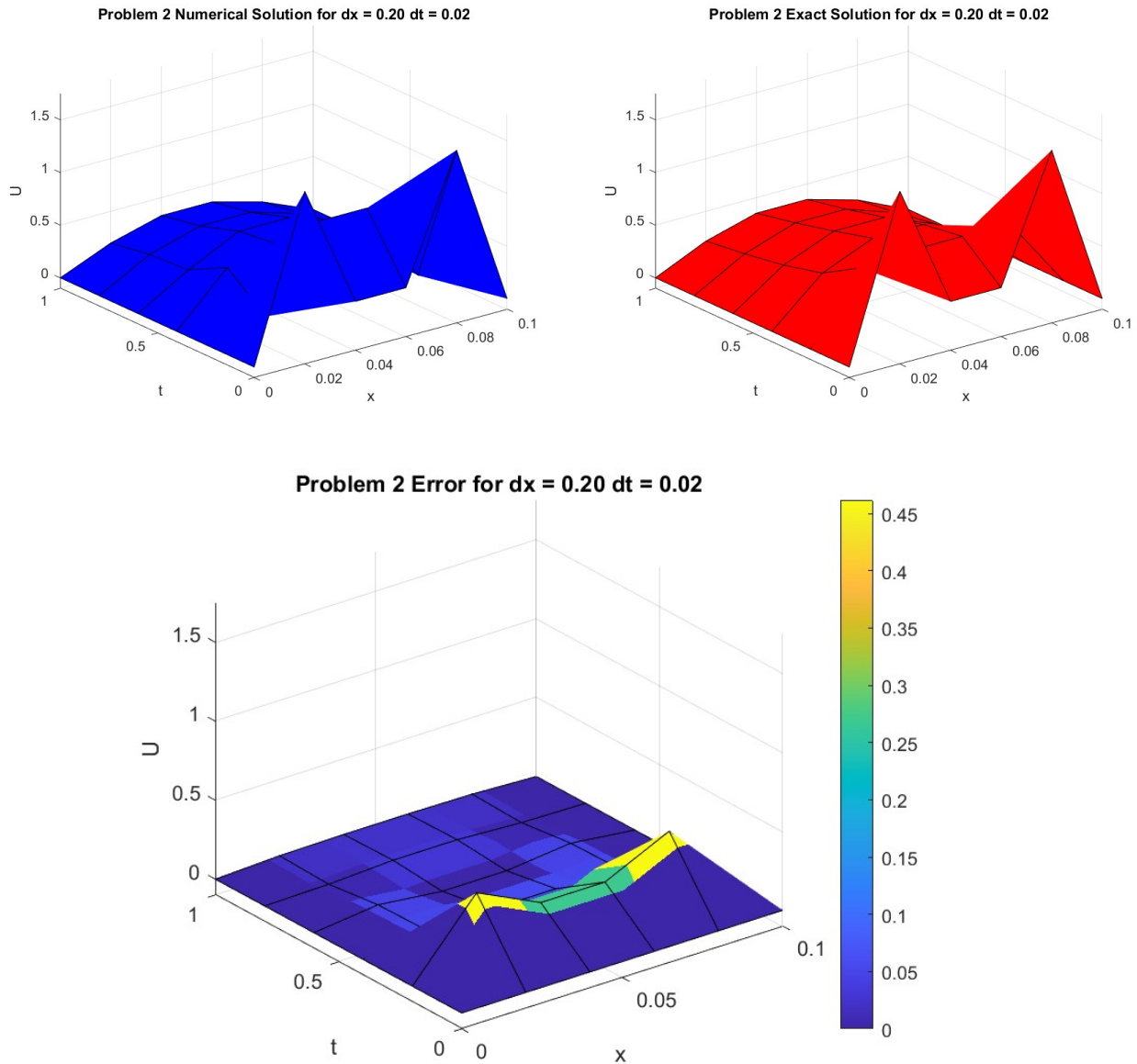


Figure 6: Problem 2 with $\Delta x = 0.2$ and $\Delta t = 0.02$

With a rather large Δx and Δt , there is a greater error between the numerical and exact results towards the beginning, However, as the heat dissipates to $u = 0$ through time, the two results begin to align with a lower error.

Problem 2: Error Analysis

For this problem, I chose my Δx as...

$$\Delta x = \left[\left(\frac{1}{2}\right)^3, \left(\frac{1}{2}\right)^4, \left(\frac{1}{2}\right)^5, \left(\frac{1}{2}\right)^6, \left(\frac{1}{2}\right)^7 \right]$$

in order to have a more refined error calculation.

I also chose my Δt , following the Von Neumann stability criterion, as...

$$\Delta t = \frac{\left(\left(\frac{1}{2}\right)^7\right)^2}{4} \quad \text{such that} \quad |r| = \left| \frac{\Delta t}{\Delta x^2} \right| \leq \frac{1}{2}$$

One last thing to note is that I chose the midpoint at the final iteration, $t = 0.1$ and $x = 0.5$, as my value to do the error analysis.

A log-log plot showing the error of the scheme with differing Δx is shown below...

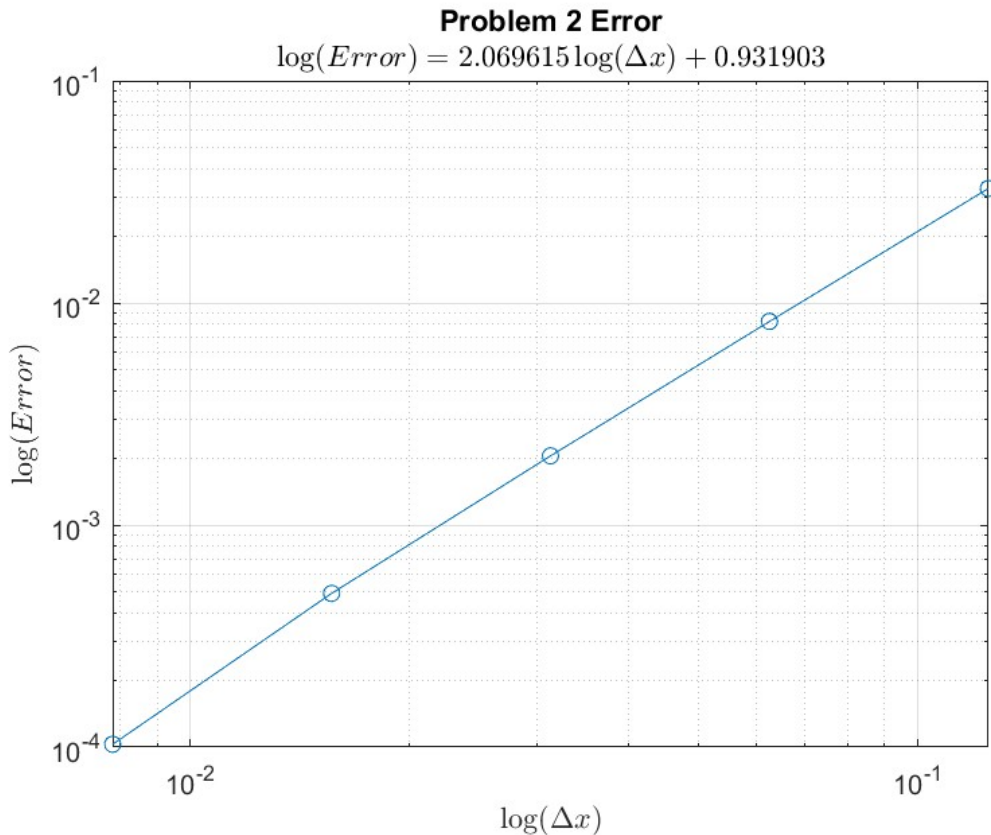


Figure 7: Problem 2 LogLog Plot of Error

The plot shows that the spatial accuracy of this scheme is two with the coefficient of the loglog plot being two.

```
1 %% HW 7
2 % Zachary Humphries
3 % COMP 521
4 % Fall 2022
5
6 clc
7 clear
8 close all
9
10 %% Problem 2
11
12 a = 0;          % Left Boundary (x)
13 b = 1;          % Right Boundary (x)
14
15 T = 0.1;        % Final Time
16
17 %% a)
18
19 dx_list = 0.2;
20 dt = 0.02;
21
22 estimated_U = zeros(length([0:dt:T]), length([a:dx_list:b]));
23 actual_U = zeros(length([0:dt:T]), length([a:dx_list:b]));
24
25
26 for i=1:length(dx_list)
27     dx = dx_list(i);
28     xgrid_short = [a+dx:dx:b-dx];
29     xgrid_long = [a:dx:b];
30
31     ICV = xgrid_short';
32     ICV = problem2_u0(ICV,0);
33
34     left_boundary = 0;
35     right_boundary = 0;
36
37     r=dt/(dx^2);
38
39     U = ICV;
40     estimated_U(1, :) = [left_boundary U' right_boundary];
41
42     actual = problem2_u0(xgrid_long, 0);
43     actual_U(1, :) = actual;
44
45     A = problem2_matrix(a,b,dx,dt);
46
47     count = 2;
48
49     for dt_j = dt: dt : T
50
51         U(1) = U(1);
52         U(end) = U(end);
53
54         U = A*U;
55
```

```

56     actual = problem2_u0(xgrid_long, dt_j);
57     actual_U(count, :) = actual;
58
59     estimated_U(count, :) = [left_boundary U right_boundary];
60
61     count = count + 1;
62 end
63 end
64
65 [X_mesh, T_mesh] = meshgrid([a:dx:b] , [0:dt:T]);
66
67
68 mesh(X_mesh, T_mesh, estimated_U, 'FaceColor','b', 'EdgeColor','k')
69 title([sprintf('Problem 2 Numerical Solution for dx = %.2f dt = %.2f', dx_list
70     , dt)])
71 xlabel('x')
72 ylabel('t')
73 zlabel('U')
74 axis([a b, 0, T, -0.1 1.75])
75 drawnow
76
77 figure
78 mesh(X_mesh, T_mesh, (actual_U), 'FaceColor','r', 'EdgeColor','k')
79 title([sprintf('Problem 2 Exact Solution for dx = %.2f dt = %.2f', dx_list, dt
80     )])
81 xlabel('x')
82 ylabel('t')
83 zlabel('U')
84 axis([a b, 0, T, -0.1 1.75])
85 drawnow
86
87 figure
88 mesh(X_mesh, T_mesh, (abs(actual_U-estimated_U)), 'FaceColor','texturemap', '
89     EdgeColor','k')
90 title([sprintf('Problem 2 Error for dx = %.2f dt = %.2f', dx_list, dt)])
91 xlabel('x')
92 ylabel('t')
93 zlabel('U')
94 axis([a b, 0, T, -0.1 1.75])
95 colorbar
96 caxis([0, max(abs(actual_U-estimated_U), []), 'all'])
97 drawnow
98
99 %% b)
100 % If you want plots similar to (a), uncomment the code below
101
102 dx_list = [0.5^3, 0.5^4, 0.5^5, 0.5^6, 0.5^7];
103 dt = dx_list(end)^2/4;
104
105 error_list = zeros(length(dx_list), 1);
106
107 for i=1:length(dx_list)
108     dx = dx_list(i);
109     xgrid_short = [a+dx:dx:b-dx];

```

```

108     xgrid_long = [a:dx:b];
109
110 %     estimated_U = zeros(length([0:dt:T]), length([a:dx:b]));
111 %     actual_U = zeros(length([0:dt:T]), length([a:dx:b]));
112
113     ICV = xgrid_short';
114     ICV = problem2_u0(ICV,0);
115
116     left_boundary = 0;
117     right_boundary = 0;
118
119     r=dt/(dx^2);
120
121     U = ICV;
122
123     actual = problem2_u0(xgrid_long, 0);
124
125 %     actual_U(1, :) = actual;
126 %
127 %     estimated_U(1, :) = [left_boundary U' right_boundary];
128
129     A = problem2_matrix(a,b,dx,dt);
130
131     count = 2;
132
133     for dt_j = dt: dt : T
134
135         U(1) = U(1);
136         U(end) = U(end);
137
138         U = A*U;
139
140         actual = problem2_u0(xgrid_long, dt_j);
141
142 %         actual_U(count, :) = actual;
143 %         estimated_U(count, :) = [left_boundary U' right_boundary];
144
145         count = count+1;
146     end
147
148     error_list(i,1) = abs(U(((length(U)+1)/2),1)-actual(1,((length(U)+1)/2)));
149
150
151
152 %     [X_mesh, T_mesh] = meshgrid([a:dx:b] , [0:dt:T]);
153 %
154 %     figure
155 %     mesh(X_mesh, T_mesh, estimated_U, 'FaceColor','b')
156 %     title([sprintf('Problem 2 Numerical Solution for dx = %.5f dt = %.5f',
157 % dx, dt)])
157 %     xlabel('x')
158 %     ylabel('t')
159 %     zlabel('U')
160 %     axis([a b, 0, T, -0.1 1.75])
161 %     drawnow

```



```

162 %
163 %     figure
164 %     mesh(X_mesh, T_mesh, (actual_U), 'FaceColor','r')
165 %     title([sprintf('Problem 2 Exact Solution for dx = %.5f dt = %.5f', dx,
    dt)])
166 %     xlabel('x')
167 %     ylabel('t')
168 %     zlabel('U')
169 %     axis([a b, 0, T, -0.1 1.75])
170 %     drawnow
171 %
172 %     figure
173 %     mesh(X_mesh, T_mesh, (abs(actual_U-estimated_U)), 'FaceColor','
    texturemap', 'EdgeColor', "none")
174 %     title([sprintf('Problem 2 Error for dx = %.5f dt = %.5f', dx, dt)])
175 %     xlabel('x')
176 %     ylabel('t')
177 %     zlabel('U')
178 %     axis([a b, 0, T, 0, max(abs(actual_U-estimated_U), [], 'all')])
179 %     colorbar
180 %     caxis([0, max(abs(actual_U-estimated_U), [], 'all')])
181 %     drawnow
182
183 end
184
185 figure
186 forward_poly = polyfit(log(dx_list), log(error_list), 1);
187 loglog(dx_list, error_list, "o-"); grid on;
188 title("Problem 2 Error")
189 subtitle_name_forward = strcat("$\log(\text{Error}) = ", sprintf("%2.6f",
    forward_poly(1)), "\log(\Delta x) + ", sprintf("%2.6f", forward_poly(2)), "
    $");
190 subtitle(subtitle_name_forward, 'interpreter','latex')
191 xlabel("$\log(\Delta x)$", 'interpreter','latex')
192 ylabel("$\log(\text{Error})$", 'interpreter','latex')
193
194 function A = problem2_matrix(a,b,dx,dt)
195     r = dt/(dx^2);
196     m = (b-a)/dx;
197     one = ones(m-1,1);
198     diag1 = r * one;
199     diag2 = (1-2*r) * one;
200
201     A = spdiags([diag1 diag2 diag1], -1:1, m-1, m-1);
202     A = sparse(A);
203 end
204
205 function u0x = problem2_u0(x, time)
206     u0x = (sin(pi*x)*exp(-pi^2 *time))+(sin(3*pi*x)*exp(-9*pi^2 *time));
207 end

```

Problem 2 Matlab Code