1 Introduction

We solve the following boundary value problems (BVPs) using primarily central differences on uniform and non-uniform meshes, the Thomas algorithm, the Euler method, and LU-decomposition.

1.1 Problem 1

Recall the boundary value problem for free convection along a vertical plate, as used in Problem 2 from Homework 3:

$$\begin{aligned} F''' + 3FF'' - 2F'^2 + \theta = 0\,, \\ \theta'' + 3\mathrm{Pr}F\theta' = 0\,, \end{aligned} \tag{1}$$

$$\begin{aligned} \eta = 0: \quad F = F' = 0, \quad \theta = 1\,, \\ \eta \to \infty: \quad F' \to 0, \quad \theta \to 0\,. \end{aligned} \tag{2}$$

Convert the 3rd-order F-equation into a system of one 1st- and one 2nd-order equations, and solve this system coupled with the θ-equation. With this approach, the BCs can be directly applied at $\eta = \{0, 10\}$. Solve the 2nd-order equation with 2nd-order central differences, and solve the 1st-order equation with with the explicit Euler method. Use $N = 101$ equally-spaced grid points, and solve the three equations iteratively, subject to your own convergence criterion. Solve the resulting finite difference system for the 2nd-order equations using the Thomas algorithm.

1. Compare results for F' and θ to those obtained in Homework 3 for $\mathrm{Pr} = \{1, 10\}$.
2. Note that this time, we are solving a coupled BVP; compare the gradients $F''(0)$ and $\theta'(0)$ for each Prandtl number with those from Homework 3.

1.2 Problem 2

Discretize the boundary value problem

$$\frac{d^2y}{d\theta^2} + \frac{y}{4} = 0\,, \qquad y(-1) = 0\,, \qquad y(1) = 2\,, \qquad -1 \le \theta \le 1 \tag{3}$$

using second-order central differences with $N = 51$ unequally-spaced grid points obtained from

$$\theta_i = \cos\left[\pi\,(i-1)/(N-1)\right]\,, \qquad i = 1, \ldots, N\,. \tag{4}$$

Solve the resulting system using LU-decomposition, and compare results to the exact solution.

2 Methodology

2.1 Problem 1

We first re-write the original boundary value problem as

$$F' = g \tag{5}$$

$$g'' = -3Fg' + 2g^2 - \theta \tag{6}$$

$$\theta'' = -3\mathrm{Pr}F\theta'\,, \tag{7}$$

$$\begin{aligned} \eta = 0: &\qquad F = g = 0, &\qquad \theta = 1\,, \\ \eta \to \infty: &\qquad g \to 0, &\qquad \theta \to 0\,. \end{aligned} \tag{8}$$

We proceed to discretize these equations on a uniform grid, indexing the solution variables as F_i, g_i, and θ_i, where $i = 1, \ldots, N$. Equations (6) and (7) are solved in a coupled manner using second-order central differences. Then the first equation (5) is solved independently using the Euler method,

$$\begin{aligned} F_1 &= F(0) \\ F_i &= F_{i-1} + h g_i\,, \qquad i = 2, \ldots, N\,, \end{aligned} \tag{9}$$

where h is the grid spacing. This procedure is repeated iteratively until the sum of relative errors per grid point between the current and previous iteration over all three variables decreases to $\epsilon < 10^{-5}$. Thus during each iteration, (6) and (7) are solved simultaneously, and then the value of F is updated according to (9).

The second-order central difference method approximates the first- and second-derivatives of a function $f(x)$ at location x_i as

$$\begin{aligned} \left.\frac{df}{dx}\right|_{x_i} &\approx \frac{1}{2h}\left(f_{i+1} - f_{i-1}\right) \\ \left.\frac{d^2 f}{dx^2}\right|_{x_i} &\approx \frac{1}{h^2}\left(f_{i+1} - 2f_i + f_{i-1}\right)\,, \end{aligned} \tag{10}$$

where $f_i \equiv f(x_i)$. These approximations are substituted into the governing equations, and, for the current BVP, yield a g-equation and a θ-equation:

$$\overbrace{\left(\frac{1}{h^2} - \frac{3\hat{F}_i}{2h}\right)}^{b_i} g_{i-1} + \overbrace{\left(\frac{-2}{h^2} - 2\hat{g}_i\right)}^{a_i} g_i + \overbrace{\left(\frac{1}{h^2} + \frac{3\hat{F}_i}{2h}\right)}^{c_i} g_{i+1} = \overbrace{-\hat{\theta}_i}^{d_i}\,, \tag{11}$$

$$\overbrace{\left(\frac{1}{h^2} - \frac{3\mathrm{Pr}\hat{F}_i}{2h}\right)}^{b_i} g_{i-1} + \overbrace{\left(\frac{-2}{h^2}\right)}^{a_i} g_i + \overbrace{\left(\frac{1}{h^2} + \frac{3\mathrm{Pr}\hat{F}_i}{2h}\right)}^{c_i} g_{i+1} = \overbrace{0}^{d_i}\,, \tag{12}$$

where the hat operator $\hat{\cdot}_i$ indicates use of $\cdot_i$ from the previous iteration. This method allows us to circumvent the non-linearity in the g-equation, (6), and further reminds us that the equations for g and θ are to be solved simultaneously.

Each of these equations defines a system within the interior of the domain, which can be written as $\mathbf{Af} = \mathbf{y}$, where $\mathbf{A}$ is tridiagonal due to the form of the central difference equations. In more explicit form,

$$\begin{bmatrix} a_2 & c_2 & & & & & \\ b_3 & a_3 & c_3 & & & & \\ & b_4 & a_4 & c_4 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & b_{N-3} & a_{N-3} & c_{N-3} & \\ & & & & b_{N-2} & a_{N-2} & c_{N-2} \\ & & & & & b_{N-1} & a_{N-1} \end{bmatrix} \begin{bmatrix} f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_{N-3} \\ f_{N-2} \\ f_{N-1} \end{bmatrix} = \begin{bmatrix} d_2 - b_2 f_1 \\ d_3 \\ d_4 \\ \vdots \\ d_{N-3} \\ d_{N-2} \\ d_{N-1} - c_{N-1} f_N \end{bmatrix}\,, \tag{13}$$

where a_i, b_i, c_i, and d_i are obtained by applying central differences to the governing equation for f_i, as in (11) and (12). The subtracted terms in the first and last elements of $\mathbf{y}$ account for Dirichlet boundary conditions imposed as known values of f_1 and f_N.

Here, the Thomas algorithm is used to solve each central difference matrix equation. To discuss this algorithm, we index elements in the right-hand-side vector (y), and the sub- (b), super- (c), and diagonal

(a) terms starting with $i = 1$. Furthermore, we let the number of diagonal elements be n. The first step of the Thomas algorithm is to eliminate the sub-diagonal with

$$\begin{aligned} r_i &= b_i/a_i \\ a_{i+1} &= a_{i+1} - r_i c_i \\ y_{i+1} &= y_{i+1} - r_i y_i \,, \qquad i = 1, \ldots, n-1 \,. \end{aligned} \tag{14}$$

Next, back-substitution is performed,

$$\begin{aligned} f_n &= y_n/a_n \\ f_i &= (y_i - c_i f_{i+1})/a_i \,, \qquad i = n-1, \ldots, 1 \,, \end{aligned} \tag{15}$$

which yields the solution vector $\mathbf{f}$.

Because this method is iterative, we must establish initial guesses $F^{(0)}$, $g^{(0)}$, and $\theta^{(0)}$, which span the BVP's domain $\eta \in [0, 10]$. The exact values are not terribly important, but it is desirable that they satisfy the requisite boundary conditions and have reasonable non-zero values within the domain. These considerations, coupled with some insight into the behavior of $F(\eta)$ from Homework 3, result in our choice of initial conditions:

- $F^{(0)}$ linearly interpolates 0 to $\{\frac{1}{2}, \frac{1}{4}\}$ over η for $\mathrm{Pr} = \{1, 10\}$.
- $g^{(0)}$ maps a sinusoid on the interval $[0, \pi]$ with amplitude $\frac{1}{2}$ to the full domain of η.
- $\theta^{(0)}$ linearly interpolates 1 to 0 over η.

This concludes our discussion of the solution method for Problem 1. However, two minor points remain:

1. The Runge-Kutta method used in Homework 3 uses 501 grid points. In order to compare our solution more closely to that of Homework 3, we use $N = 501$ grid points instead of the requested 101.
2. Lastly, for part (b), the values of $F''(0)$ and $\theta'(0)$ are calculated using the second-order-accurate forward difference formula
$$\frac{df_i}{dx} \approx \frac{1}{2h}\left(-3f_i + 4f_{i+1} - f_{i+2}\right) \,. \tag{16}$$

2.2 Problem 2

To discretize (3) using the unequally-spaced grid points given in (4), we use the central difference formula for a second-order-accurate second-derivative,

$$\begin{aligned} \frac{d^2y}{d\theta^2} \approx &\left[\frac{-2}{(\theta_{i+1} - \theta_{i-1})}\left(\frac{1}{\theta_{i+1} - \theta_i} + \frac{1}{\theta_i - \theta_{i-1}}\right)\right] y_i + \left[\frac{2}{(\theta_{i+1} - \theta_{i-1})(\theta_i - \theta_{i-1})}\right] y_{i-1} \\ &+ \left[\frac{2}{(\theta_{i+1} - \theta_{i-1})(\theta_{i+1} - \theta_i)}\right] y_{i+1} \\ = &\,[A_i]\, y_i + [B_i]\, y_{i-1} + [C_i]\, y_{i+1} \,, \end{aligned} \tag{17}$$

to rewrite the governing equation as

$$y'' + \frac{y}{4} = 0 \quad \rightarrow \quad [B_i]\, y_{i-1} + [\tfrac{1}{4} + A_i]\, y_i + [C_i]\, y_{i+1} = 0 \,. \tag{18}$$

Formation of the matrix equation proceeds in the standard manner used in Problem 1, resulting in a tridiagonal matrix equation $\mathbf{Ax} = \mathbf{f}$.

To solve our system for $\mathbf{x}$, we employ LU decomposition. The goal is to perform the decomposition $\mathbf{A} = \mathbf{LU}$, where $\mathbf{L}$ is lower-triangular, and $\mathbf{U}$ is upper-triangular. Consider the tridiagonal matrix decomposition $\mathbf{A} = \mathbf{LU}$ in more detail,

$$\begin{bmatrix} a_1 & c_1 & & & \\ b_1 & a_2 & c_2 & & \\ & b_2 & \ddots & \ddots & \\ & & \ddots & \ddots & c_{N-1} \\ & & & b_{N-1} & a_N \end{bmatrix} = \begin{bmatrix} l_1 & & & & \\ b_1 & l_2 & & & \\ & b_2 & \ddots & & \\ & & \ddots & \ddots & \\ & & & b_{N-1} & l_N \end{bmatrix} \begin{bmatrix} 1 & u_1 & & & \\ & 1 & u_2 & & \\ & & \ddots & \ddots & \\ & & & \ddots & u_{N-1} \\ & & & & 1 \end{bmatrix}, \tag{19}$$

where all blank elements are 0. The values of l_i and u_i are determined iteratively using

$$\begin{aligned} l_1 &= a_1 \\ u_{i-1} &= c_{i-1} \,/\, l_{i-1} \\ l_i &= a_i - b_{i-1} u_{i-1} \,, \qquad i = 2, \ldots, N \,. \end{aligned} \tag{20}$$

Next, back-substitution is used to find the solution vector $\mathbf{x}$. First, the system $\mathbf{Lz} = \mathbf{f}$ is solved using

$$\begin{aligned} z_1 &= f_1 \,/\, l_1 \\ z_i &= (f_i - b_{i-1} z_{i-1}) \,/\, l_i \,, \qquad i = 2, \ldots, N \,, \end{aligned} \tag{21}$$

and then we solve the system $\mathbf{Ux} = \mathbf{z}$ for $\mathbf{x}$ with

$$\begin{aligned} x_N &= z_N \\ x_i &= z_i - u_i x_{i+1} \,, \qquad i = N-1, N-2, \ldots, 1 \,. \end{aligned} \tag{22}$$

LU decomposition is very efficient in scenarios where the governing equations remain the same, but solutions are desired for many instantiations of $\mathbf{f}$. Since $\mathbf{L}$ and $\mathbf{U}$ must be computed only once, repeated solution procedures are rapid once these matrices are known.

Finally, to assess numerical accuracy, we note that the analytical solution to (3) is

$$y(\theta) = \frac{2}{\sin(1)} \sin\left(\frac{\theta + 1}{2}\right) . \tag{23}$$

3 Results

3.1 Problem 1

Plots comparing the central difference and shooting method results for F, F', and θ are presented in Figure 1. Note that the shooting method used for Homework 3 was coupled to a fourth-order Runge-Kutta integration scheme, so the governing equations are satisfied to higher-order accuracy than the second-order central difference scheme employed here.

Table 1 shows the differences in F'' and θ' boundary conditions at $\eta = 0$ for different Prandtl numbers and methods. Note the listed values for Homework 3 are hard-coded into its solver, whereas those for Homework 4 are found iteratively.

3.2 Problem 2

Analytical and central difference solutions to (3) are presented in Figure 2. The global relative error, essentially summing over all points in the relative error plot, is 1.61×10^{-3}.

BC	Pr	Homework 3	Homework 4
$F''(0)$	1	0.6000	0.6425
$F''(0)$	10	0.4100	0.4206
$\theta'(0)$	1	-0.5671	-0.5644
$\theta'(0)$	10	-1.1700	-1.1582

Table 1: Comparison of the boundary conditions $F''(0)$ and $\theta'(0)$ between Prandtl numbers and the methods employed in Homework 3 and Homework 4.

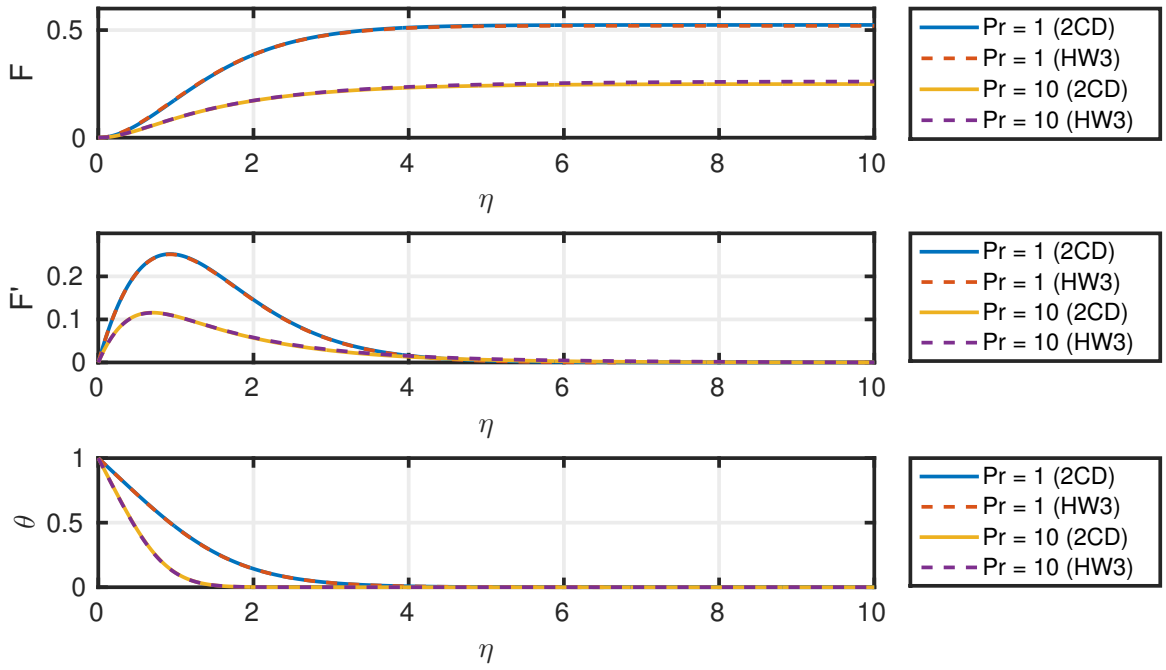

Figure 1: F, F', and θ as functions of η for Prandtl numbers $\text{Pr} = \{1, 10\}$. The second-order central difference (2CD) solutions are compared to the results of Homework 3 (HW3) using the fourth-order Runge-Kutta method.

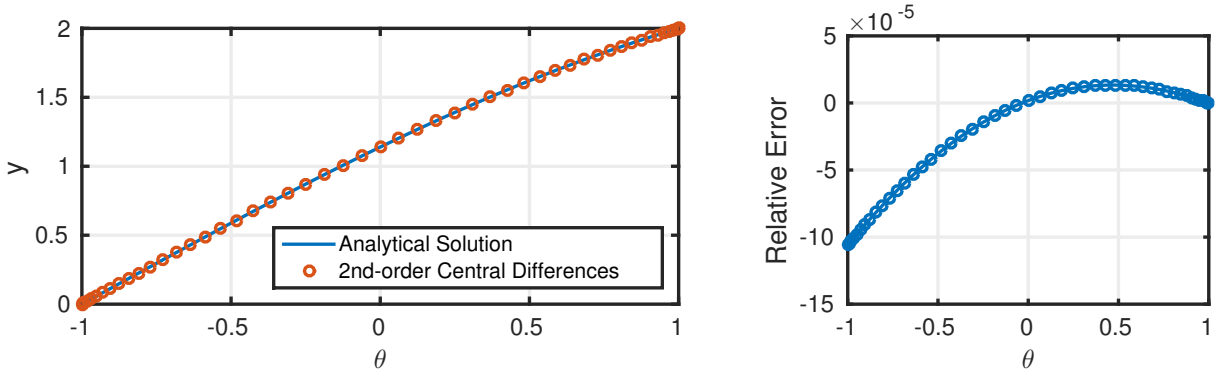

Figure 2: Analytical and central difference solutions to (3) using non-uniform mesh spacing, and the point-wise relative error.

4 Discussion

4.1 Problem 1

Visually, the solutions for F, F', and θ obtained using our second-order central difference method are identical to those from Homework 3, for both Prandtl numbers considered. Differences in $F''(0)$ and $\theta'(0)$ from Table 1 are more pronounced, however, with disagreement as high as 7% in the case of $F''(0)$ for $\mathrm{Pr} = 1$. This could be due to any number of reasons.

First, our current solution uses a combination of second-order central differences and the first-order Euler method. Though the functions we consider have relatively low curvature, the fourth-order Runge-Kutta method used in Homework 3 will undoubtedly do a better job of capturing solution behavior. I am inclined to believe that the solution method in Homework 3 is more 'trustworthy' for this reason. However, we were given $F''(0)$ and $\theta'(0)$ as initial conditions in Homework 3, which means that any departure from their true values—however small—could not be corrected for.

The more 'correct' way to do this problem is clearly the current approach, wherein coupled central difference equations are solved iteratively, and no boundary conditions other than those with a physical basis are required. To obtain a more accurate solution, we would simply use central difference and integration methods of higher-order accuracy.

4.2 Problem 2

Results for the non-uniform grid solution match the exact analytical solution very well, as demonstrated by the low global and point-wise relative error. LU decomposition proves to be an effective method for solution of the matrix equation, but we are unable to take advantage of its superior efficiency in this problem, due to the fact that we are solving our matrix system once for a single RHS vector.

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. However, though `Problem_1` calls a modified version of Homework 3's code through the function `Problem_1_Shooting`, the latter function will not be listed here in the interest of brevity.

Listing 1: `Problem_1.m`

```
function [] = Problem_1()

    %%%%%%
    % Solves the differential equations governing free convection along a vertical plate
    % using central differences for the two coupled equations and the RK4 method for the
    % uncoupled equation.
    %
    % Ryan Skinner, October 2015
    %%%

    Set_Default_Plot_Properties();

    %%%
    % Calculate the solutions from Homework 3, for Pr = {1, 10}.
    %%%

    [T1, Y1, T10, Y10] = Problem_1_Shooting(false);
```

```
18
19      %%%
20      % Define variables specific to the boundary-value problem.
21      %%%
22
23      % Solution domain.
24      eta0 = 0;
25      etaf = 10;
26      N = 501;
27      eta = linspace(eta0, etaf, N)';
28      h = eta(2) - eta(1);
29
30      % Prandtl number.
31      Pr = [1, 10];
32      nPr = length(Pr);
33
34      % Boundary conditions (0 = initial, f = final).
35      BC.F0  = 0;
36      BC.g0  = 0;
37      BC.gf  = 0;
38      BC.th0 = 1;
39      BC.thf = 0;
40
41      % Solution variables (indexed by Prandtl number and then time step).
42       F = nan(nPr, N);
43       g = nan(nPr, N);
44      th = nan(nPr, N);
45
46      % Initial guesses for solution variables, based on linear approximation to Homework 3.
47       F(1,:) = linspace(0, 0.50, N);
48       F(2,:) = linspace(0, 0.25, N);
49       g(1,:) = sin(linspace(0, 1, N)*pi) / 2;
50       g(2,:) = sin(linspace(0, 1, N)*pi) / 2;
51      th(1,:) = linspace(1, 0, N);
52      th(2,:) = linspace(1, 0, N);
53
54      % Convergence criterion.
55      epsilon = 1e-5;
56
57      %%%
58      % Solve our three equations (for F, g, and theta) iteratively.
59      %%%
60
61      % Loop through Prandtl numbers.
62      for iPr = 1:nPr
63
64          fprintf('Working on Pr = %i\n', Pr(iPr));
65
66          % Loop until convergence criterion is met.
67          norm = inf;
68          iteration = 1;
69          while norm > epsilon
70
71              % Containers for previous iterations' values to determine convergence.
72               F_prev =  F(iPr,:);
73               g_prev =  g(iPr,:);
74              th_prev = th(iPr,:);
75
76              %%%
77              % STEP 1: Solve the g-equation.
78              %%%
79
80              [diag, sub, sup, rhs] = Assemble_g( h, BC, F_prev, g_prev, th_prev );
81              sol = Thomas(diag, sub, sup, rhs);
82              g(iPr,:) = [BC.g0; sol; BC.gf];
83
84              %%%
85              % STEP 2: Solve the theta-equation.
86              %%%
```

```
87
88              [diag, sub, sup, rhs] = Assemble_th( h, BC, Pr(iPr), F_prev );
89              sol = Thomas(diag, sub, sup, rhs);
90              th(iPr,:) = [BC.th0; sol; BC.thf];
91
92              %%%
93              % STEP 3: Integrate g to obtain F using the Euler method.
94              %%%
95
96              F(iPr,:) = Euler(g_prev, BC.g0, h);
97
98              %%%
99              % STEP 4: Assess convergence.
100             %%%
101
102              F_norm = sum(abs( F_prev -  F(iPr,:))) / N;
103              g_norm = sum(abs( g_prev -  g(iPr,:))) / N;
104             th_norm = sum(abs(th_prev - th(iPr,:))) / N;
105
106             norm = F_norm + g_norm + th_norm;
107
108             fprintf('Iteration: %02i,  norm: %8.2e, F: %8.2e, g: %8.2e, th: %8.2e\n', ...
109                             iteration,         norm,   F_norm,   g_norm,   th_norm);
110
111             iteration = iteration + 1;
112
113         end
114
115     end
116
117     %%%
118     % Process results.
119     %%%
120
121     hF  = figure();
122     hg  = figure();
123     hth = figure();
124
125     for iPr = 1:length(Pr)
126
127         if Pr(iPr) == 1
128             T = T1;
129             Y = Y1;
130         elseif Pr(iPr) == 10
131             T = T10;
132             Y = Y10;
133         else
134             error('Prandtl number %.2f not supported.',Pr(iPr));
135         end
136
137         % Plot F
138
139         figure(hF);
140         hold on;
141         plot(eta, F(iPr,:),     'DisplayName',sprintf('Pr = %i (2CD)',Pr(iPr)));
142         plot(  T,   Y(:,1),'--','DisplayName',sprintf('Pr = %i (HW3)',Pr(iPr)));
143         xlabel('\eta');
144         ylabel('F');
145
146         % Plot F'
147
148         figure(hg);
149         hold on;
150         plot(eta, g(iPr,:),     'DisplayName',sprintf('Pr = %i (2CD)',Pr(iPr)));
151         plot(  T,   Y(:,2),'--','DisplayName',sprintf('Pr = %i (HW3)',Pr(iPr)));
152         xlabel('\eta');
153         ylabel('F''');
154
155         % Plot theta
```

```
156
157         figure(hth);
158         hold on;
159         plot(eta,th(iPr,:),     'DisplayName',sprintf('Pr = %i (2CD) ',Pr(iPr)));
160         plot(  T,   Y(:,4),'--','DisplayName',sprintf('Pr = %i (HW3)',Pr(iPr)));
161         xlabel('\eta');
162         ylabel('\theta');
163
164         % Determine F''(0) and theta'(0) using second-order forward differences.
165         Fpp = (-3* g(iPr,1) + 4* g(iPr,2) -    g(iPr,3)) / (2*h);
166         thp = (-3*th(iPr,1) + 4*th(iPr,2) -   th(iPr,3)) / (2*h);
167
168         fprintf('Pr = %2i: F''''(0) = %7.4f, th''(0) = %7.4f\n', Pr(iPr), Fpp, thp);
169
170     end
171
172     for h = [hF, hg, hth]
173         figure(h);
174         hleg = legend('show');
175         set(hleg, 'Location', 'eastoutside');
176     end
177
178 end
```

Listing 2: `Assemble_g.m`

```
1  function [diag, sub, sup, rhs] = Assemble_g( h, BC, F, g, th )
2
3      %%%%%%
4      % Assembles the LHS matrix and the RHS vector for the g-system.
5      %   diag -- diagonal
6      %    sub -- sub-diagonal
7      %    sup -- super-diagonal
8      %    rhs -- right-hand side vector
9      %
10     % Ryan Skinner, October 2015
11     %%%
12
13     N = length(F);
14
15     diag_range = 2:N-1;
16      sub_range = 3:N-1;
17      sup_range = 2:N-2;
18
19     diag = (-2/h^2) - 2 *  g(diag_range);
20      sub = ( 1/h^2) - 3 *  F(sub_range) / (2*h);
21      sup = ( 1/h^2) + 3 *  F(sup_range) / (2*h);
22      rhs =          - 1 * th(diag_range);
23
24     % Account for boundary conditions, even though g0 = gf = 0.
25     rhs(1)   = rhs(1)   - ((1/h^2) - 3 * F(diag_range(1))   / (2*h)) * BC.g0;
26     rhs(end) = rhs(end) - ((1/h^2) + 3 * F(diag_range(end)) / (2*h)) * BC.gf;
27
28 end
```

Listing 3: `Assemble_th.m`

```
1  function [diag, sub, sup, rhs] = Assemble_th( h, BC, Pr, F )
2
3      %%%%%%
4      % Assembles the LHS matrix and the RHS vector for the theta-system.
5      %   diag -- diagonal
6      %    sub -- sub-diagonal
7      %    sup -- super-diagonal
8      %    rhs -- right-hand side vector
9      %
10     % Ryan Skinner, October 2015
11     %%%
```

```
12
13      N = length(F);
14
15      diag_range = 2:N-1;
16       sub_range = 3:N-1;
17       sup_range = 2:N-2;
18
19      diag = (-2/h^2) * ones(length(diag_range), 1);
20       sub = ( 1/h^2) - 3 * Pr * F(sub_range) / (2*h);
21       sup = ( 1/h^2) + 3 * Pr * F(sup_range) / (2*h);
22       rhs = zeros(length(diag_range),1);
23
24      % Account for boundary conditions, even though thf = 0.
25      rhs(1)   = rhs(1)   - ((1/h^2) - 3 * Pr * F(diag_range(1))   / (2*h)) * BC.th0;
26      rhs(end) = rhs(end) - ((1/h^2) + 3 * Pr * F(diag_range(end)) / (2*h)) * BC.thf;
27
28  end
```

Listing 4: `Thomas.m`

```
1   function [ x ] = Thomas( diag, sub, sup, rhs )
2
3       %%%%%%
4       % Solves a tri-diagonal matrix system using the Thomas algorithm.
5       %   diag -- diagonal
6       %    sub -- sub-diagonal
7       %    sup -- super-diagonal
8       %    rhs -- right-hand side vector
9       %    sol -- solution vector
10      %
11      % Ryan Skinner, October 2015
12      %%%
13
14      % Eliminate the sub-diagonal.
15      for i = 1:length(sup)
16          r = sub(i) / diag(i);
17          diag(i+1) = diag(i+1) - r * sup(i);
18           rhs(i+1) =  rhs(i+1) - r * rhs(i);
19      end
20
21      % Back-substitute and calculate the solution vector.
22      x = zeros(length(diag),1);
23      x(end) = rhs(end) / diag(end);
24      for i = length(diag)-1:-1:1
25          x(i) = (rhs(i) - sup(i) * x(i+1)) / diag(i);
26      end
27
28  end
```

Listing 5: `Euler.m`

```
1   function [ F ] = Euler( g, F0, delta )
2
3       %%%%%%
4       % Integrates the data set g using the Euler method, given the initial value F0 and the
5       % uniform spacing delta between grid points.
6       %
7       % Ryan Skinner, October 2015
8       %%%
9
10      F = nan(length(g),1);
11      F(1) = F0;
12      for i = 1:(length(g)-1)
13          F(i+1) = F(i) + delta * g(i);
14      end
15
16  end
```

Listing 6: `Problem_2.m`

```
function [] = Problem_2()

    %%%%%%
    % Solves the differential equation (y'' + y/4 = 0) using second-order central
    % differences with unequally-spaced grid points and LU-decomposition.
    %
    % Ryan Skinner, October 2015
    %%%

    Set_Default_Plot_Properties();

    %%%
    % Define variables specific to the boundary-value problem.
    %%%

    % Solution domain: the closed interval [-1, 1].
    N = 51;
    th = flip(cos(pi*(0:N-1)/(N-1)));

    % Boundary conditions (0 = initial, f = final).
    BC.y0 = 0;
    BC.yf = 2;

    % Assemble central difference matrix equation for y.
    [diag, sub, sup, rhs] = Assemble_y(th, BC);

    % Solve matrix equation using LU-decomposition.
    [l, u] = LU_Decompose(diag,sub,sup);
    y = LU_Solve(sub, l, u, rhs);
    y = [BC.y0; y; BC.yf];

    % Compute analytical solution.
    y_exact = (2/sin(1)) * sin((th+1)/2)';

    % Compute relative error (pointwise).
    relerr = (y(2:end-1)-y_exact(2:end-1))./y_exact(2:end-1);

    figure();
    hold on;
    plot(th, y_exact, 'LineStyle', '-', 'DisplayName', 'Analytical Solution');
    plot(th,       y, 'o', 'DisplayName', '2nd-order Central Differences');
    xlabel('\theta');
    ylabel('y');
    hleg = legend('show');
    set(hleg, 'Location', 'southeast');

    figure();
    hold on;
    plot(th(2:end-1), relerr, '-o');
    xlabel('\theta');
    ylabel('Relative Error');

    error = sum(abs(relerr));
    fprintf('Total error norm: %.2e\n',error);

end
```

Listing 7: `Assemble_y.m`

```
function [diag, sub, sup, rhs] = Assemble_y( th, BC )

    %%%%%%
    % Assembles the LHS matrix and the RHS vector for the y-system.
    %   diag -- diagonal
    %    sub -- sub-diagonal
    %    sup -- super-diagonal
    %    rhs -- right-hand side vector
```

```
    %
    % Ryan Skinner, October 2015
    %%%

    N = length(th);

    B = nan(N-2,1);
    C = nan(N-2,1);
    D = nan(N-2,1);
    for i = 2:N-1
        B(i-1) =   2 / ((th(i+1) - th(i-1)) * (th(i+1) - th(i)));
        C(i-1) =   2 / ((th(i+1) - th(i-1)) * (th(i)   - th(i-1)));
        D(i-1) = (-2 / ( th(i+1) - th(i-1))) * (1/(th(i+1)-th(i)) + 1/(th(i)-th(i-1)));
    end

    diag = 1/4 + D;
    sub = C(2:end);
    sup = B(1:end-1);
    rhs = zeros(N-2,1);

    % Account for boundary conditions.
    rhs(1)   = rhs(1)   - C(1)   * BC.y0;
    rhs(end) = rhs(end) - B(end) * BC.yf;

end
```

Listing 8: `LU_Decompose.m`

```
function [ l, u ] = LU_Decompose( a, b, c )

    %%%%%%
    % Decomposes a tri-diagonal matrix equation into L and U matrices.
    %     a -- diagonal
    %     b -- sub-diagonal
    %     c -- super-diagonal
    %     l -- diagonal of L matrix
    %     u -- super-diagonal of U matrix
    %
    % Ryan Skinner, October 2015
    %%%

    N = length(a);

    l = nan(N,  1);
    u = nan(N-1,1);

    l(1) = a(1);
    for i = 2:N
        u(i-1) = c(i-1) / l(i-1);
        l(i) = a(i) - b(i-1) * u(i-1);
    end

end
```

Listing 9: `LU_Solve.m`

```
function [ x ] = LU_Solve( b, l, u, rhs )

    %%%%%%
    % Solves an LU matrix system for the solution vector x.
    %     b -- sub-diagonal of L matrix
    %     l -- diagonal of L matrix
    %     u -- super-diagonal of U matrix
    %   rhs -- right-hand side vector of matrix equation
    %     x -- solution vector
    %
    % Ryan Skinner, October 2015
    %%%
```

```
13
14      N = length(l);
15
16      z = nan(N,1);
17      x = nan(N,1);
18
19      z(1) = rhs(1) / l(1);
20      for i = 2:N
21          z(i) = (rhs(i) - b(i-1) * z(i-1)) / l(i);
22      end
23
24      x(N) = z(N);
25      for i = N-1:-1:1
26          x(i) = z(i) - u(i) * x(i+1);
27      end
28
29  end
```