MATH 3340 - Scientific Computing Final Project

Libao Jin

July 17, 2021

Honesty Clause: **I hereby testify that all the work on this project is solely my own. I have complied with the academic honesty guidelines as stated in the University regulations.**

Name: ______________________ Date: ____________

Instruction

1. Go to `https://www.overleaf.com` and sign in (required).
2. Open template, click *Menu* (up left corner), then *Copy Project*.
3. Go to `LaTeX/meta.tex` (the file `meta.tex` under the folder `LaTeX`) to change the section and your name, e.g.,
   - change author to `\author{Carl F. Gauss}`
4. You need to write function/script files, store results to output/plot files. Here are suggested names for function files, script files, output files, and plot files:

Problem	Function File	Script File	Output File	Plot File
1	`cubic_spline.m`	`final_p1.m`	`final_p1.txt`	`final_p1a.pdf`
1				& `final_p1b.pdf`
2	`backward_euler.m`	`final_p2.m`		`final_p2.pdf`
3	`golden_section.m`	`final_p3.m`	`final_p3.txt`	`final_p3.pdf`
3	& `successive_parabolic.m`			

   Once finished, you need to upload these files to the folder `src` on Overleaf. If you have different filenames, please update the filenames in `\lstinputlisting{../src/your_script_name.m}` accordingly. You can code in the provided files in final.zip, and use the MATLAB script `save_results.m` to generate the output files and store the graphs to `.pdf` files automatically (the script filenames should be exactly same as listed above).
5. Recompile, and download the generated `.pdf` file.
6. **Important**: Enter your name and the date in the above boxes *before* you submit it on WyoCourses.

Problem 1. Consider the data in the following table:

k	0	1	2	3
x_k	0.0	1.761062	3.522123	5.283185
y_k	1.0	-0.1891196	-0.9284676	0.5403023

The values of y in this table have been obtained as $y_k = \cos(x_k)$. Your goal will be to create two different spline interpolants using these data points, and compare them with the original function.

(a) Compute the usual spline interpolant ${}^1S(x)$ that uses the natural boundary conditions. Plot this interpolant versus the original function $\cos(x)$ using 100 data points equally spaced between x_0 and x_3; make sure to also indicate the four data points on the plot using a special marker (you may use $*$ or $\circ$ for example).

(b) You will probably agree that the comparison at point (a) above doesn't look very good. This is due to the fact that the natural boundary conditions do not match the behavior of the $\cos(x)$ function. Your task for this point is to modify your code to account for the correct second derivatives at the end points. In other words, create a new interpolant ${}^2S(x)$ that satisfies the following conditions:

$$ {}^2S''(x_0) = -\cos(x_0); \quad {}^2S''(x_3) = -\cos(x_3). $$

You can do this by modifying the system of equations for the spline coefficients accordingly. Plot again the newly-obtained interpolant versus the original function in the same manner as above. Turn in your codes together with the two plots.

Solution.

- Output file `final_p1.txt`:

```
(REPLACE THIS WITH YOUR OWN OUTPUT)
Problem 1(a)
i        d_i        c_i        b_i        a_i
0  -0.190813  -0.000000   0.439014   0.717356
1   0.171674  -0.629681  -0.253635   0.946300
2  -0.168437   0.606372  -0.309577  -0.916166

Problem 1(b)
i        d_i        c_i        b_i        a_i
0  -0.055942  -0.358678   0.670366   0.717356
1   0.147509  -0.543286  -0.321794   0.946300
2  -0.046114   0.518779  -0.380610  -0.916166
```

- Plot files `final_p1a.pdf` and `final_p1b.pdf`:

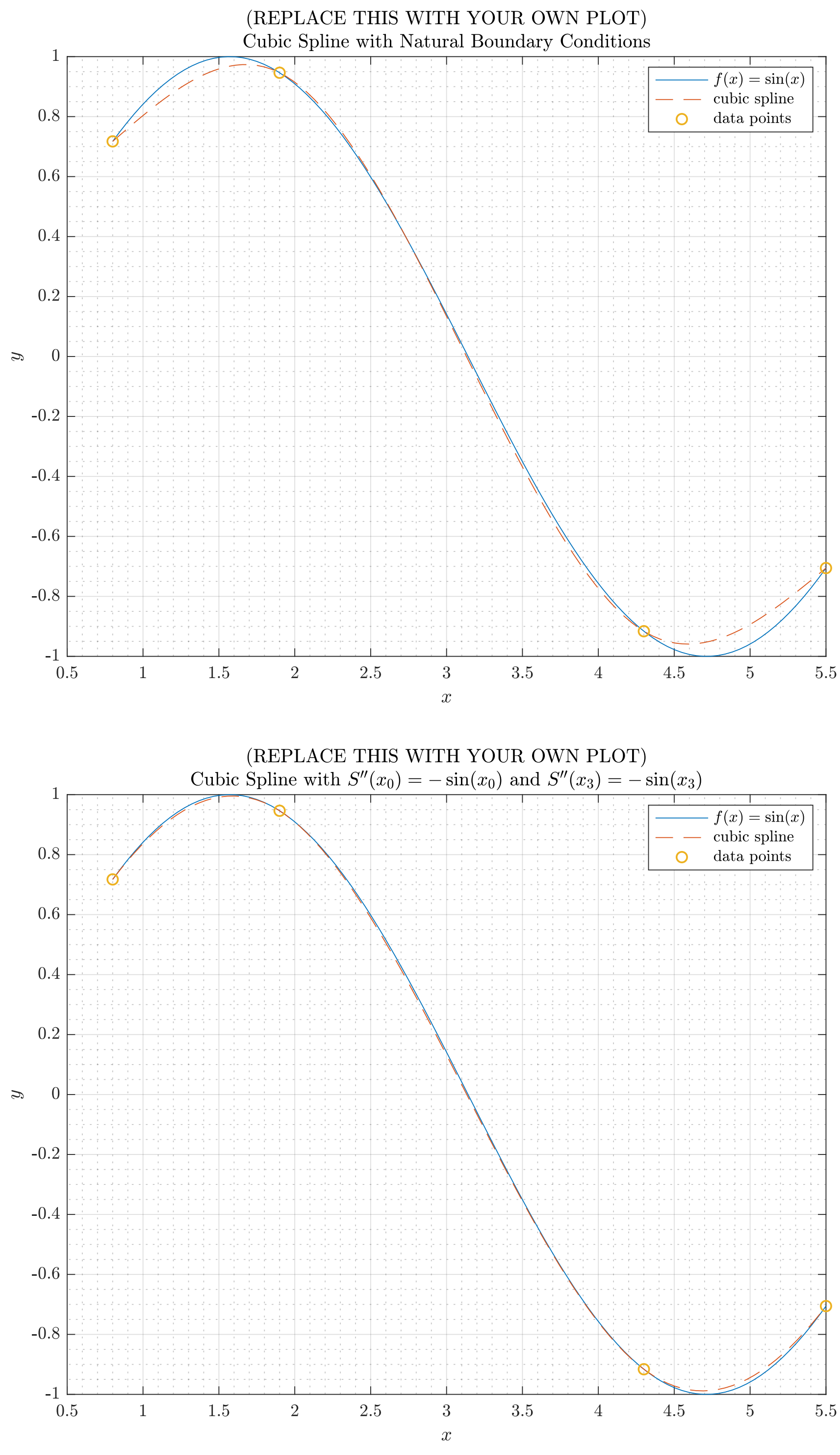

Figure 1: Cubic Spline With Different Boundary Conditions

- Function file `cubic_spline.m`:

```
function [y, coefs] = cubic_spline(xdata, ydata, x, boundary_condiction_option)
%CUBIC_SPLINE: Cubic Spline Interpolants
%
% Syntax: [y, coefs] = cubic_spline(xdata, ydata, x, bc)
% Inputs:
%   xdata = a vector, the set of nodes x_k
%   ydata = a vector, the values of f at x_k, i.e., y_k = f(x_k)
%   x     = a vector, the set of points (fine grid) used to evaluate p(x)
%   boundary_condition_option = a scalar:
%              0 for natural boundary condition
%              1 for the boundary condition stated in the problem statement
% Outputs:
%   y     = a vector, the values of y at the points x
%   coefs = a matrix, coefficient matrix of which the columns are d, c, b, a.
%
% Author: first_name last_name
% Date: mm/dd/yyyy


% PUT YOUR CODE HERE

end
```

- Script file `final_p1.m`:

```
% MATH 3340, Spring 2021
% Final Project Problem 1
% Author: firstname lastname
% Date: mm/dd/yyyy

clear; close all; clc; format short;
% Change default text interpreter to LaTeX
set(groot,'defaultTextInterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex')

% PUT YOUR CODE HERE
```

□

Problem 2. The implicit, backward Euler method for the general first order initial value problem

$$\frac{du}{dt} = f(u, t), \quad u(t_0) = u_0$$

is given by

$$u_{n+1} = u_n + (t_{n+1} - t_n) f(u_{n+1}, t_{n+1}), \quad n = 0, 1, \ldots$$

Use this method to solve numerically the differential equation

$$\frac{du}{dt} = 2 + \sqrt{u - 2t + 3}$$

subject to the initial condition $u(0) = 1$. Use a constant time step $\Delta t = t_{n+1} - t_n = 0.05$ and advance the solution to $t = 2$. Turn in the code and the plot of $u(t)$ versus t. On the plot, compare your numerical solution with the exact solution $u(t) = 1 + 4t + t^2/4$. Use markers to highlight the points produced by the numerical solution.

NOTE: You will need to solve a nonlinear equation at each time step. Solutions that do not perform this task as required will not receive credit.

Solution.

- Plot file `final_p2.pdf`:

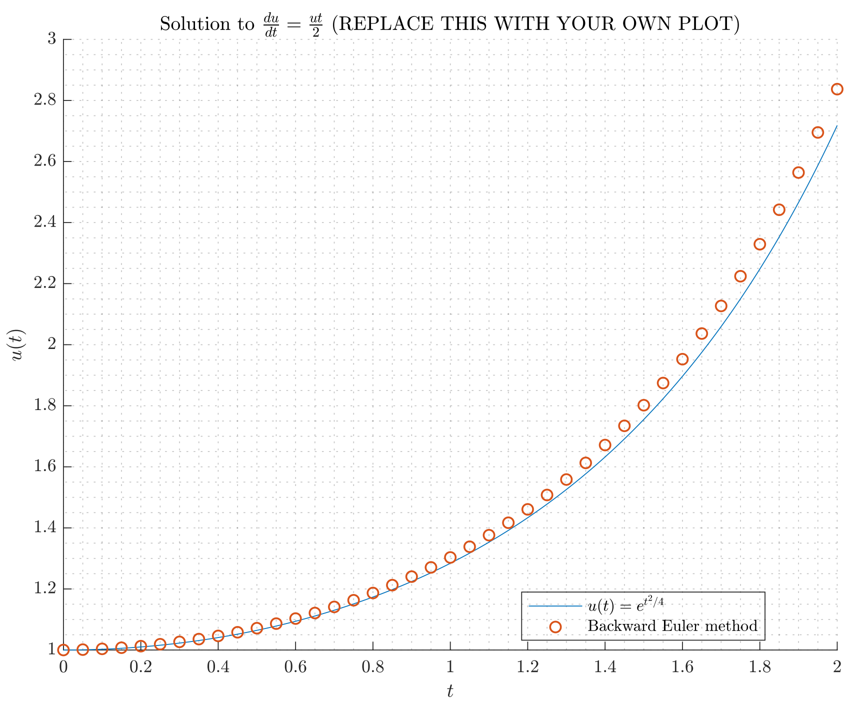

Figure 2: Solution to $du/dt = 2 + \sqrt{u - 2t + 3}$ using Implicit Backward Euler Method

- Function file `backward_euler.m`:

```
function U = backward_euler(f, t, u0)
%BACKWARD_EULER: Solve du/dt = f(t, u) with u(t0) = u0 using Backward Euler Method
%
% Syntax: U = backward_euler(f, t, u0)
% Inputs:
%   f     = function handle, the right-hand side of the ODE du/dt = f(t, u)
%   t     = a vector, the time points at which solution to be found. Note: t(1) = a, t(end) = b
%   u0    = a scalar, the initial value of the solution to the ODE
% Outputs:
%   U     = a vector, the solution of ODE corresponds to t, i.e., U(1) = U(t(1)) = u0, and etc.
%
% Author: first_name last_name
% Date: mm/dd/yyyy

% PUT YOUR CODE HERE

end
```

- Script file `final_p2.m`:

```
% MATH 3340, Spring 2021
% Final Project Problem 2
% Author: firstname lastname
% Date: mm/dd/yyyy

clc; clear; figure(3); hold on;
% Change default text interpreter to LaTeX
set(groot,'defaultTextInterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex')

% PUT YOUR CODE HERE
```

□

Problem 3. Use both successive parabolic interpolation and the golden section search method to find the minimum of the function $f(x) = |x^2 - 2| + |2x + 3|$ on the interval $[-4, 0]$ with a tolerance of 10^{-8} and an identical accuracy. You are again requested to do this with MATLAB codes which you must show, together with the output; recall that golden search method code was discussed in detail in class. For parabolic interpolation, remember that some approximation steps may produce points that are not feasible and must be replaced. Write your code such that it reverts to the golden section search in that case.

Solution.

- Output file `final_p3.txt`:

```
(REPLACE THIS WITH YOUR OWN OUTPUT)
f(x) = |3 * x^2 - 5| + |4 * x - 3| on [-2, 2]
                 method        x_min     f(x_min)
Successive Parabolic  1.29099445  2.16397780
       Golden Section  1.29099444  2.16397781
```

- Plot file `final_p3.pdf`:

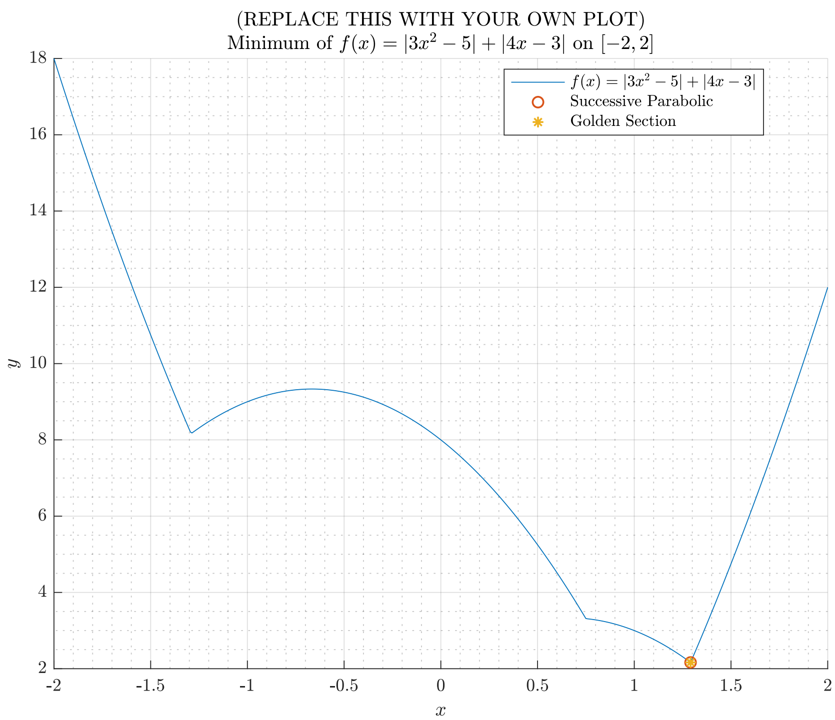

Figure 3: Minimum Obtained by Successive Parabolic Interpolation and Golden Section Search Method

- Function file `successive_parabolic.m`:

```
function [m, fm] = successive_parabolic(f, a, b, tol)

%SUCCESSIVE_PARABOLIC: Find the minimum of function f on [a, b] using
% Successive Parabolic Interpolation.
```

```
%
% Syntax: [m, fm] = successive_parabolic(f, a, b, tol)
% Inputs:
%   f   = function of which the minimum is desired, function handle
%   a   = left endpoint of the interval
%   b   = right endpoint of the interval
%   tol = tolerance
% Outputs:
%   m   = the minimizer at which the minimum of f(x) can be obtained
%   fm  = the minimum of the function
% Author: firstname lastname
% Date: mm/dd/yyyy

% PUT YOUR CODE HERE

end
```

- Function file golden_section.m:

```
function [m, fm] = golden_section(f, a, b, tol)

%GOLDEN_SECTION: Find the minimum of function f on [a, b] using
% Golden Section Search Method.
%
% Syntax: [m, fm] = golden_section(f, a, b, tol)
% Inputs:
%   f   = function of which the minimum is desired, function handle
%   a   = left endpoint of the interval
%   b   = right endpoint of the interval
%   tol = tolerance
% Outputs:
%   m   = the minimizer at which the minimum of f(x) can be obtained
%   fm  = the minimum of the function
% Author: firstname lastname
% Date: mm/dd/yyyy

% PUT YOUR CODE HERE

end
```

- Script file final_p3.m:

```
% MATH 3340, Spring 2021
% Final Project Problem 3
% Author: firstname lastname
% Date: mm/dd/yyyy

clc; clear; figure(4); hold on;
% Change default text interpreter to LaTeX
set(groot,'defaultTextInterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex')

% PUT YOUR CODE HERE
```

□