MATH 3340 - Scientific Computing Assignment 6

Libao Jin

April 11, 2021

The deadline will be strictly enforced. If you do not submit in time there will be a 20% penalty for each day you're late. If you do not submit in time there will be a 20% pennalty upfront plus another 20% for each day you're late. Remember that you are allowed to work in teams of two on this assignment. You are encouraged to prepare your work in LaTeX; a template will be provided to help you put it all together. If you choose to submit a hard copy, you may submit only one copy for a team, indicating the names of both contributors. Online submission is encouraged, however, in that case both members of a team should submit the PDF file containing their work and showing both their names.

All plots generated in this homework should have a title, legend, and labeled x and y-axes.

Instruction

1. Go to `https://www.overleaf.com` and sign in (required).
2. 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{Albert Einstein \& Carl F. Gauss}`
4. For Problem 1 and 4, you are encouraged to type solutions in LaTeX. But if you want to write it on the printout, make sure your scanned work is *clear* enough, and compile all solutions *in order*, i.e., 1, 2, 3, 4, in a single PDF (failure to do so will lead to points deduction).
5. For Problem 2 and 3, you need to write function/script files, store results to output files, and save graphs to figure files. Here are suggested names for function files, script files, output files, and figure files:

Problem	Function File	Script File	Output File	Figure File
2(a)		`hw6_p2.m`		
2(b)	`cubic_spline.m`			
2(c)	`lagrange.m`			`hw6_p2.pdf`
3		`hw6_p3.m`		`hw6_p3_1.pdf` & `hw6_p3_2.pdf`

   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 hw6.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).
6. Recompile, download and upload the generated PDF to WyoCourses.
7. You may find LaTeX.Mathematical.Symbols.pdf and the second part of Lab 01 Slides and Lab 02 Slides helpful.

Problem 1. This problem must be done by hand. Consider the following data set (Table 1).

Table 1: Problem 1 Data Set

k	0	1	2	3
x_k	1.0	1.5	1.9	2.4
y_k	1.1	1.7	2.1	1.8

Using these data points, compute the cubic spline interpolant for the function $y = f(x)$. That is, computer the coefficients $\{a_j, b_j, c_j, d_j\}$ for $j = 0, 1, 2$. To do so, you will need to set up the system of four equations in four unknowns for the coefficients $c_j, j = 0, 1, 2, 3$. Use natural boundary conditions to reduce this system to only two equations for c_1 and c_2. Then determine the other coefficients using the relations for b_j and d_j. Finally, once you have all the coefficients, evaluate the interpolant $S(x)$ at the two points $x = 1.3$ and $x = 1.7$. Perform all computations rounding to three digits after the decimal point.

Solution. Results:

```

y =

    1.4513    1.9491


C =

    0.1818         0    1.1545    1.1000
   -2.5000    0.2727    1.2909    1.7000
    1.8182   -2.7273    0.3091    2.1000


f0 =

    1.2156


f1 =

    1.4513


f2 =

    1.9491


ans =

   -1.1111    1.3333    0.8111    1.1000
   -1.1111   -0.3333    1.3111    1.7000
   -1.1111   -1.6667    0.5111    2.1000


ans =

    1.8000


ans =

    1.8000
```

```matlab
% MATH 3340, Fall 2020
% Homework 6, Problem 1
% Author: Libao Jin
% Date: 10/28/2020

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

%% 1
xdata = [1.0; 1.5; 1.9; 2.4];
ydata = [1.1; 1.7; 2.1; 1.8];
x = [1.3, 1.7];
[y, C] = cubic_spline(xdata, ydata, x)
f0 = polyval(C(1, :), 1.1 - 1.0)
f1 = polyval(C(1, :), 1.3 - 1.0)
f2 = polyval(C(2, :), 1.7 - 1.5)

cs = spline(xdata, ydata);
cs.coefs


polyval(C(3,:), 0.5)
polyval(cs.coefs(3,:), 0.5)
```

□

Problem 2. Consider the test function

$$f(x) = \frac{1}{1 + 14x^2}.$$

The task here is to approximate this function using the cubic spline piecewise interpolant with natural boundary conditions, and compare this interpolant with a polynomial with natural boundary conditions, and compare this interpolant with a polynomial interpolant using the same data set. This problem builds on top of the interpolation problem on the last homework. Aside from reusing the code you have developed in the last homework, you *must* create a *minimum* of three code files for this problem: two function files and one script file, described below.

(a) Create a script file which defines the function $f(x)$, a set `xplot` of 100 equispaced points in $[-1, 1]$ that will be used for plotting, together with the values that the test function f produces at these points. Also generate a data set `xdata`, i.e., the nodes x_k in our math notation, to be used for interpolating this function by computing the values $y_k = f(x_k)$ where $\{x_k\}$ is an equispaced set in $[-1, 1]$ with $-1 = x_0 < x_1 < \cdots < x_n = 1$. Use again $n = 9$ (i.e., 10 nodes) like in the previous homework.

(b) Write your own spline interpolation routine. At a minimum, this should involve a function which calculates the coefficients of the piecewise cubic spline polynomial and a function which evaluates the piecewise spline interpolant. Some of you may want to break the computation of the spline coefficients in several separate functions instead of having only one function perform the whole task. Note that the spline interpolant is a piecewise function and there may be several ways in which to evaluate and plot this function. Some very efficient methods can be devised, or blunt force may be used to locate the evaluation point in a particular patch.

(c) In your script file, you should call the functions you wrote for part (b) together with those you created for polynomial interpolation in the previous homework, then plot the spline interpolant $S(x)$ together with the original function $f(x)$ and the polynomial interpolant $p_n(x)$ based on the same set of data points. On the same plot, show the data set with a marker. For the global polynomial interpolant, just reuse the functions you created for homework 5 on this new test function. To plot, compute the values of all three functions (that is, the original test function, the cubic spline and the global polynomial) on the set of points `xplot` you created in part (a). Your plot must have a title, legend, and labeled x and y axes.

Note: Results using the built-in MATLAB functions for spline interpolation will receive no credit. Nevertheless, you may find these functions helpful for verifying your results.

Solution.

- Function file `lagrange.m`

```
function [p] = lagrange(xdata, ydata, x)
%LAGRANGE: Lagrange Interpolants
% Syntax: [p] = lagrange(xdata, ydata, x)
% Inputs:
%   xdata = a column vector, the set of nodes x_k
%   ydata = a column vector, the values of f at x_k, i.e., y_k = f(x_k)
%   x     = a column vector, the set of points (fine grid) used to evaluate p(x)
% Outputs:
%   p     = a column vector, the values of p at the points x
%
% Author: Libao Jin
% Date: 10/28/2020

m = length(x);
n = length(xdata);
p = zeros(m, 1);
L = ones(m, n);
for k = 1:n
    for j = 1:n
        if j ~= k
            L(:, k) = L(:, k) .* (x - xdata(j)) / (xdata(k) - xdata(j));
        end
    end
end
for i = 1:m
    for k = 1:n
        p(i) = p(i) + ydata(k) * L(i, k);
    end
end

end
```

Function file `cubic_spline.m`

```
function [p, C] = cubic_spline(xdata, ydata, x)
%CUBIC_SPLINE: Cubic Spline Interpolants
% Syntax: [p] = cubic_spline(xdata, ydata, x)
% 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)
% Outputs:
%   p     = a vector, the values of p at the points x
%
% Author: first_name last_name
% Date: 10/28/2020

n = length(xdata);
h = diff(xdata);
A = zeros(n, n);
rhs = zeros(n, 1);
a = ydata;
for i = 1:n
    if i == 1 || i == n
        A(i, i) = 1;
        rhs(i) = 0;
    else
        A(i, i - 1) = h(i - 1);
        A(i, i) = 2 * (h(i - 1) + h(i));
        A(i, i + 1) = h(i);
        rhs(i) = 3 / h(i) * (a(i + 1) - a(i)) - 3 / h(i - 1) * (a(i) - a(i - 1));
    end
```

```
end

c = A \ rhs;

b = zeros(n - 1, 1);
for i = 1:n - 1
    b(i) = 1 / h(i) * (a(i + 1) - a(i)) - h(i) / 3 * (2 * c(i) + c(i + 1));
end

d = zeros(n - 1, 1);
for i = 1:n - 1
    d(i) = (c(i + 1) - c(i)) / (3 * h(i));
end

y = zeros(size(x));
for i = 1:length(x)
    for j = 1:length(xdata) - 1
        if x(i) >= xdata(j) && x(i) <= xdata(j + 1)
            y(i) = a(j) + b(j) * (x(i) - xdata(j)) + ...
                   c(j) * (x(i) - xdata(j)) ^ 2 + ...
                   d(j) * (x(i) - xdata(j)) ^ 3;
        end
    end
end

C = [a(1:(n - 1)) b(1:(n - 1)) c(1:(n - 1)) d(1:(n - 1))];
C = fliplr(C);
p = y;

end
```

- Script file `hw6_p2.m`

```
% MATH 3340, Fall 2020
% Homework 6, Problem 2
% Author: Libao Jin
% Date: 10/28/2020

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

test = 0;

%% 2(a)
% Put your code for 2(a) below
if test == 1
    f = @(x) 1 ./ (4 + 25 * x .^ 2);
    a = -2;
    b = 2;
    n = 8;
else
    f = @(x) 1 ./ (1 + 14 * x .^ 2);
```

```matlab
    a = -1;
    b = 1;
    n = 10;
end
xdata = linspace(a, b, n)';
ydata = f(xdata);
x = linspace(a, b, 100)';
fx = f(x);

%% 2(c)
figure(1); hold on;
% Put your code for 2(c) below
[y, C] = cubic_spline(xdata, ydata, x);
C
p = lagrange(xdata, ydata, x);

plot(x, fx, '-', 'LineWidth', 1)
plot(x, y, '--', 'LineWidth', 1)
plot(x, p, '-.', 'LineWidth', 1)
plot(xdata, ydata, 'o');
xlabel('$x$');
ylabel('$y$');
grid minor;
if test == 1
    legend({'$f(x) = \frac{1}{4 + 25 x^2}$', 'cubic spline', 'lagrange', 'nodes', }, '
        Location', 'best');
    title('Cubic Spline and Lagrange Interpolation (REPLACE THIS WITH YOUR OWN PLOT)');
else
    legend({'$f(x) = \frac{1}{1 + 16 x^2}$', 'cubic spline', 'lagrange', 'nodes', }, '
        Location', 'best');
    title('Cubic Spline and Lagrange Interpolation');
end
```

- Figure files: `hw6_p2.pdf` (Figure 1).

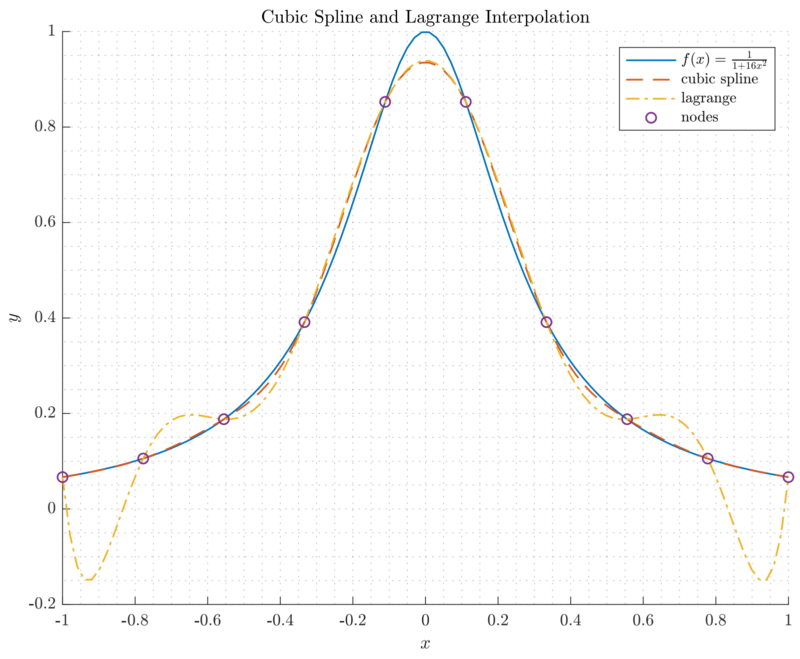


Figure 1: Data Points, Cubic Spline Interpolant, and Polynomial Interpolant

$\square$

Problem 3. Consider the function

$$f(x) = \frac{e^x}{1+x^2}.$$

Evaluate the error in its first derivative $f'(0)$, computed using the central first derivative formula, for a step ranging from $\Delta x = 0.1$ to $\Delta x = 10^{-7}$; decrease the value of the step Δx by a factor of 10 (one order of magnitude) for each successive evaluation. Plot the error versus the step Δx using a logarithmic scale on both axes. Repeat the same calculations for the second derivative $f''(0)$, also computed with the centered formula. You may find that for some values of Δx the error is zero; when using a logarithmic scale, this may lead to trouble when plotting, since the logarithm of zero is not defined (MATLAB does not complain, but Octave may produce an error message). To avoid this problem, you can just set the error to the machine accuracy `eps` whenever it is lower than `eps`.

Solution.

- Function file `central_difference.m`

```matlab
function [fp1, fpp1, fp2, fpp2] = central_difference(f, t, h)
fp1 = (-f(t + 2 * h) + 8 * f(t + h) - 8 * f(t - h) + f(t - 2 * h)) ./ (12 * h);
fpp1 = (-f(t + 2 * h) + 16 * f(t + h) - 30 * f(t) + 16 * f(t  - h) - f(t - 2 * h)) ./ (12 *
    h .^ 2);
fp2 = (f(t + h) - f(t - h)) ./ (2 * h);
fpp2 = (f(t + h) - 2 * f(t) + f(t - h)) ./ (h.^2);
end
```

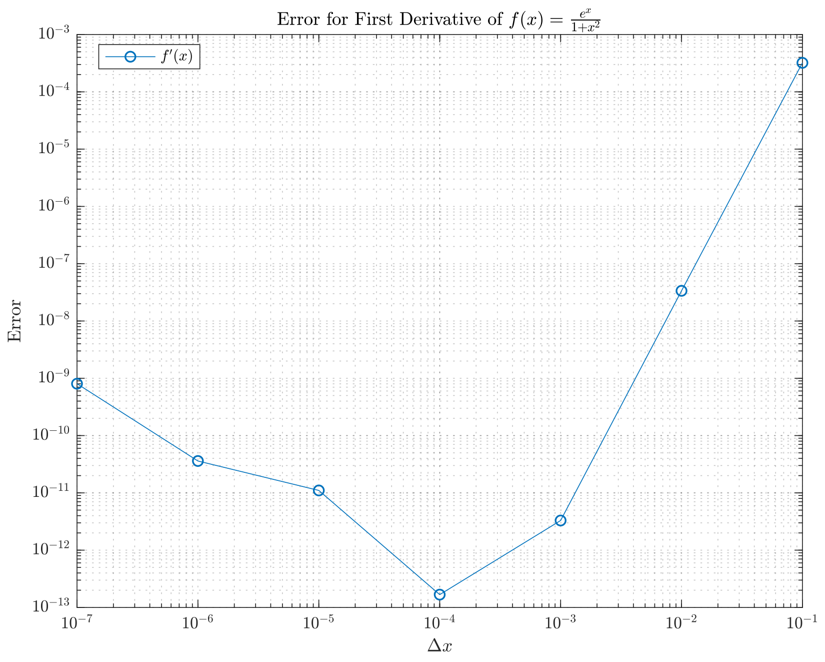
- Script file `hw6_p3.m`

```matlab
% MATH 3340, Fall 2020
% Homework 6, Problem 3
% Author: Libao Jin
% Date: 10/28/2020

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

test = 0;
if test == 1
    f = @(x) exp(x) ./ (1 + x + x.^2 / 2 + x.^3 / 3 + x.^4 / 4);
    fp_exact = 0;
    fpp_exact = 0;
else
    f = @(x) exp(x) ./ (1 + x.^2);
    fp_exact = 1;
    fpp_exact = -1;
end
% Put your code for 3 for calculating the error of first derivative using central difference
     below
n = [-1:-1:-7]';
dx = 10 .^ n;
[fp1, fpp1, fp2, fpp2] = central_difference(f, 0, dx)

```

```
figure(1);
fp_err = abs(fp1 - fp_exact);
fp_err(fp_err == 0) = eps;
loglog(dx, fp_err, '-o');
grid minor;
xlabel('$\Delta x$');
ylabel('Error');
if test == 1
    title('Error for First Derivative of $f(x) = \frac{e^{x}}{1 + x + x^2 / 2 + x^3 / 3 + x
        ^4 / 4}$ (REPLACE THIS WITH YOUR OWN PLOT)');
else
    title('Error for First Derivative of $f(x) = \frac{e^{x}}{1 + x^2}$');
end
legend({'$f''(x)$'}, 'Location', 'best');

figure(2);
% Put your code for 3 for calculating the error of second derivative using central
    difference
fpp_err = abs(fpp1 - fpp_exact);
fpp_err(fpp_err == 0) = eps;
loglog(dx, fpp_err, '-o');
grid minor;
xlabel('$\Delta x$');
ylabel('Error');
if test == 1
    title('Error for Second Derivative of $f(x) = \frac{e^{x}}{1 + x + x^2 / 2 + x^3 / 3 + x
        ^4 / 4}$ (REPLACE THIS WITH YOUR OWN PLOT)');
else
    title('Error for Second Derivative of $f(x) = \frac{e^{x}}{1 + x^2}$');
end
legend({'$f''''(x)$'}, 'Location', 'best');

figure(3);
fp_err = abs(fp2 - fp_exact);
fp_err(fp_err == 0) = eps;
loglog(dx, fp_err, '-o');
grid minor;
xlabel('$\Delta x$');
ylabel('Error');
if test == 1
    title('Error for First Derivative of $f(x) = \frac{e^{x}}{1 + x + x^2 / 2 + x^3 / 3 + x
        ^4 / 4}$ (REPLACE THIS WITH YOUR OWN PLOT)');
else
    title('Error for First Derivative of $f(x) = \frac{e^{x}}{1 + x^2}$');
end
legend({'$f''(x)$'}, 'Location', 'best');

figure(4);
% Put your code for 3 for calculating the error of second derivative using central
    difference
fpp_err = abs(fpp2 - fpp_exact);
fpp_err(fpp_err == 0) = eps;
loglog(dx, fpp_err, '-o');
grid minor;
```

```matlab
xlabel('$\Delta x$');
ylabel('Error');
if test == 1
    title('Error for Second Derivative of $f(x) = \frac{e^{x}}{1 + x + x^2 / 2 + x^3 / 3 + x
        ^4 / 4}$ (REPLACE THIS WITH YOUR OWN PLOT)');
else
    title('Error for Second Derivative of $f(x) = \frac{e^{x}}{1 + x^2}$');
end
legend({'$f''''(x)$'}, 'Location', 'best');
```

- Figure files: `hw6_p3_1.pdf` (Figure 2) and `hw6_p3_2.pdf` (Figure 3).



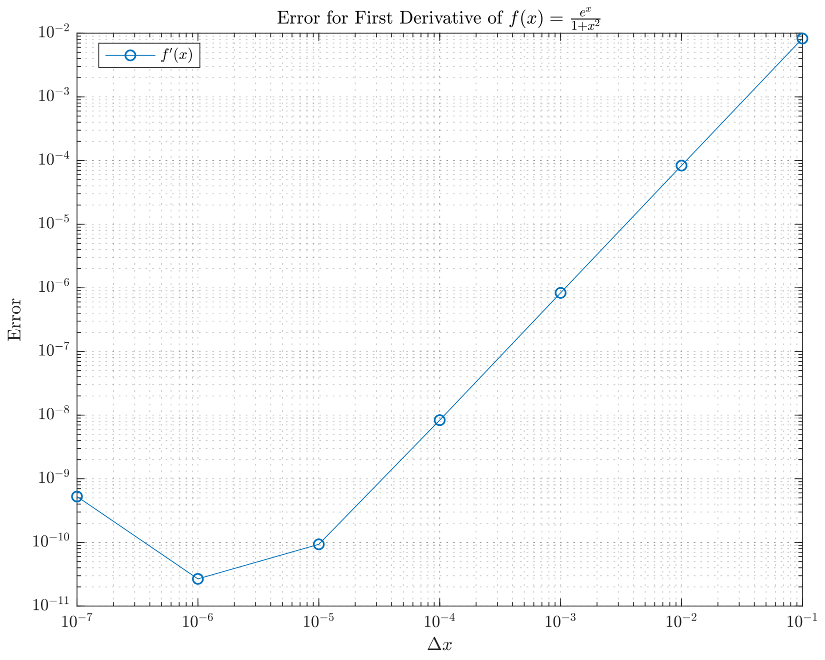
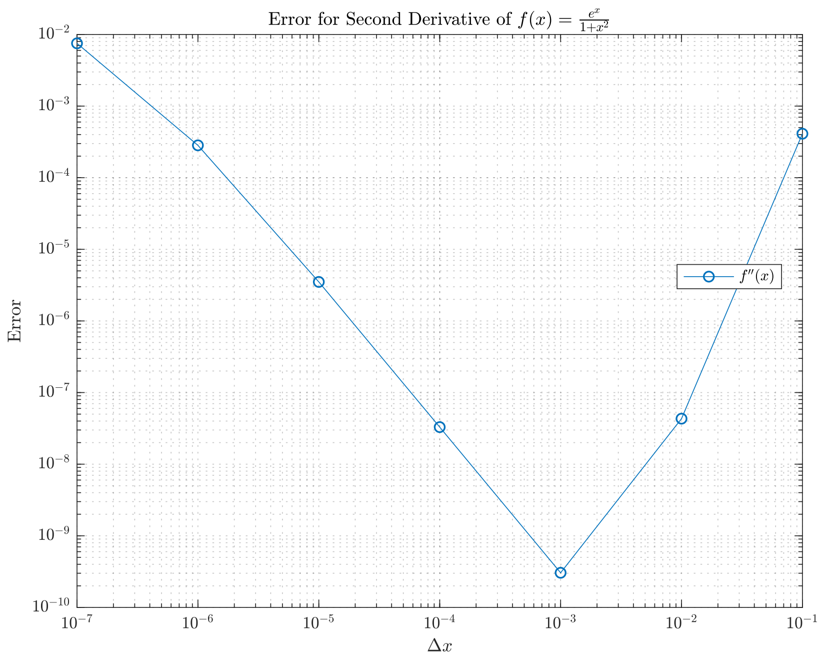


Figure 2: Error vs. Step Δx for First Derivative $f'(0)$

$\square$



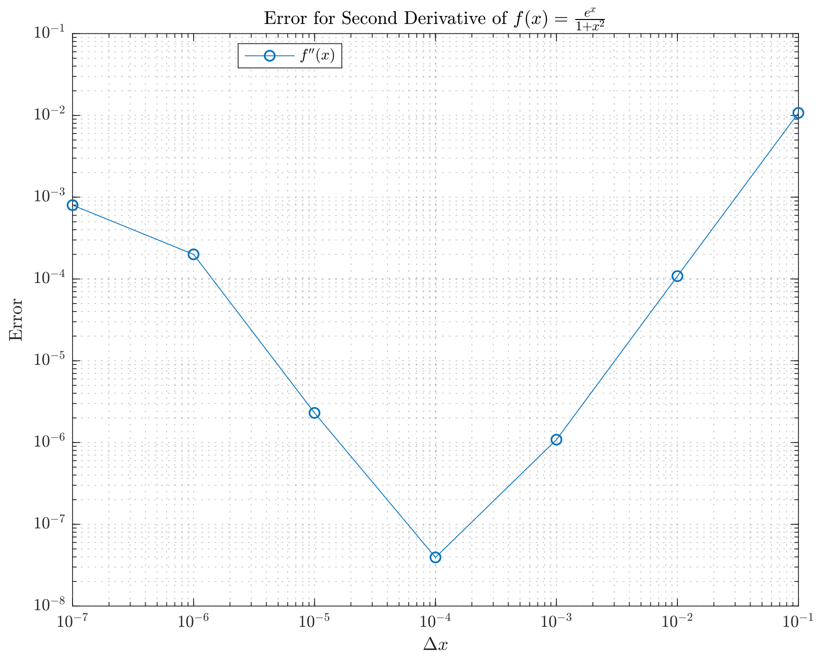


Figure 3: Error vs. Step Δx for Second Derivative $f''(0)$

Problem 4. Using the Taylor series method set up in class (see section 7.2, page 117 in the typed notes) find a formula that approximates the values for $f'(x)$ and $f''(x)$ with the highest accuracy possible using a linear combination (weighted sum) of the values:

$$f(x-2h), \quad f(x-h), \quad f(x+2h), \quad f(x+3h),$$

where $h = \Delta x$ is the step size. Do this calculation by hand.

Solution. Results:

```

A =

    1.0000    1.0000    1.0000    1.0000
   -2.0000   -1.0000    2.0000    3.0000
    2.0000    0.5000    2.0000    4.5000
   -1.3333   -0.1667    1.3333    4.5000

--------------- Problem 4 ---------------
c1 =

      -1/20
      -1/3
       7/12
      -1/5


c2 =

       2/5
      -1/2
      -1/108086391056891904
       1/10


c1 =

  -0.050000000000000
  -0.333333333333333
   0.583333333333333
  -0.200000000000000


c2 =

   0.400000000000000
  -0.500000000000000
  -0.000000000000000
   0.100000000000000
```

```
% MATH 3340, Fall 2020
% Homework 6, Problem 4
% Author: Libao Jin
% Date: 10/28/2020

clear; close all; clc;
% Change default text interpreter to LaTeX
```

```matlab
set(groot,'defaultTextInterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex')

% d = [-1 0 2 3];
% A = [1, 1, 1, 1;
%      -1, 0, 2, 3;
%      1/2, 0, 2, 9/2;
%      -1/6, 0, 8/6, 27/6];

d = [-2 -1 2 3];
n = length(d);
A = zeros(n);
for i = 1:n
    A(i, :) = d.^(i - 1) / factorial(i - 1);
end
A

b1 = [0 1 0 0]';
b2 = [0 0 1 0]';

fprintf('--------------- Problem 4 ---------------');
format rat;
c1 = A \ b1
c2 = A \ b2
format long;
c1 = A \ b1
c2 = A \ b2
```

□