MEMS 1060 Homework 2

Shane Riley
Undergraduate Research Assistant, University of Pittsburgh
Swanson School of Engineering
3700 O'Hara Street, Benedum Hall of Engineering
Pittsburgh, PA 15261
Email: shane.riley@pitt.edu

1 Problem 1

Given the following system of equations, find x, y, and z using Cramer's Rule:

1. $3x - 2y + 5z = 14$
2. $x - y = -1$
3. $2x + 4z = 14$

So first, we turn our linear system into matrix form, where $Ax = b$:

1. $A = \begin{bmatrix} 3 & -2 & 5 \\ 1 & -1 & 0 \\ 2 & 0 & 4 \end{bmatrix}$
2. $b = \begin{bmatrix} 14 \\ -1 \\ 14 \end{bmatrix}$

We find $det(A) = 6$. Using that, we find each component of the solution vector using Cramer's rule. x is shown as an example:

$$x = \begin{vmatrix} 14 & -2 & 5 \\ -1 & -1 & 0 \\ 14 & 0 & 4 \end{vmatrix} / det(A) = 6/6 = 1 \tag{1}$$

The rest are computed programmatically using MATLAB, yielding $x = 1$, $y = 2$, and $z = 3$.

2 Problem 2

Approximate the function $y = \sin x$ using a Taylor series expansion about $x = \pi/4$, with two, four, and six terms. Plot the function and the approximations using MATLAB in $0 < x < \pi$.

To construct a taylor series approximation of $f(x)$, we use the formula for a Taylor series centered about $x = c$:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(c)}{n!}(x-c)^n \tag{2}$$

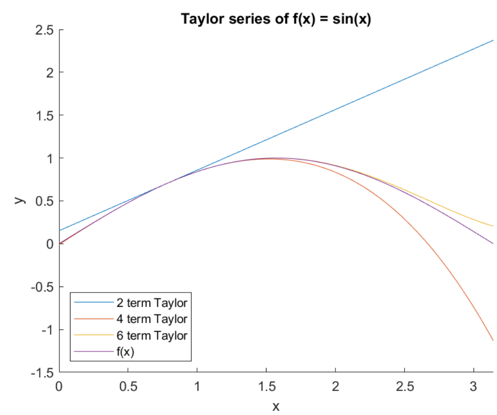

Fig. 1. Taylor series for sin(x).

For $f(x) = \sin x$, we can represent the derivatives at the center as the following, because of the cyclical nature of the derivatives:

$$f^{(n)}(\frac{\pi}{4}) = \frac{\sqrt{2}}{2}(-1)^{n(n-1)/2} \tag{3}$$

The exponent for -1 creates a series of positive and negative signs that will match signs of the subsequent derivatives. To actually implement the taylor series computation, the Taylor series terms are stored as function handles in a cell array, so that they are created only once per function call. X-values for the approximations are a linear space with 100 points, and each point's y-value is computed in order to plot the curves. See the figure, as well as the source code.

3 Problem 3

Apply the intermediate value theorem to show that the function $f(x)$ has a root in the interval $[0, \pi/2]$. Implement a bracketing scheme and an open method to find the root, to a tolerance of 0.001. Discuss assumptions, and compare convergence rates of the two implementations.

Table 1. Root-finding results for f(x).

Method	c	f(c)	Iterations
Newton	0.86552	-0.00014	2
RF	0.86517	0.00093	11

$$f(x) = \cos x - x^3 \tag{4}$$

In order to apply the Intermediate Value Theorem, we must first prove that the function is continuous on the interval. Since $\cos x$ and x^3 are both continuous on their own, the difference between them will also be continuous. We now compute the values of the function at each end of the interval:

$f(0) = 1$
$f(\pi/2) \approx -3.88$

Since 0 falls between $f(0)$ and $f(\pi/2)$ and f is continuous, we prove using the Intermediate Value Theorem that there is an x-value c in $[0, \pi/2]$ such that $f(c) = 0$. This zero will be found to the given tolerance using both the Regula Falsi and Newton's methods.

We seed our Newton's method with the middle x-value for the interval. The x-value is iterated as shown:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{5}$$

For Regula Falsi, our new lower bound (a) will be the x-value of where the connecting line between (a, f(a)) and (b, f(b)) meets the x-axis. Using the formulas for slope and point-slope form of a line where y = 0, the following iteration formula can be constructed:

$$a' = a + \frac{a-b}{\frac{f(b)}{f(a)} - 1} \tag{6}$$

Using both methods for iteration, we find the root:

We notice here that Newton's method converges in this case much more quickly than Regula Falsi. It is worth pointing out, however, that Newton's Method requires an expression for the derivative of f(x) ($f'(x) = -\sin x - 3x^2$) in order to iterate x, while Regula Falsi does not need to use the derivative. Additionally, since the bracket always tightens with every iteration, we can tell intuitively that Regula Falsi will always converge given sufficient iterations, while there exist functions and circumstances that will cause Newton's

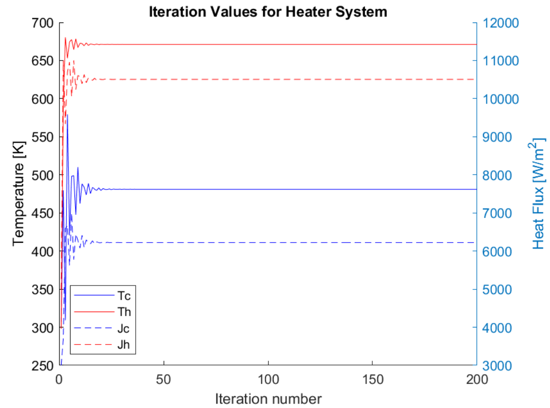

Fig. 2. Values for system DOF's per iteration.

Method to fail. Finally, it is worth pointing out that the interval in this problem only contains one root. For systems with more than one solution, more sophisticated methods will be required.

4 Problem 4 (BONUS)

Use fixed point iteration to solve the nonlinear system of equations, which describes a heater curing a coating panel. Equations are shown below:

1. $5.67 * 10^{-8} T_c^4 + 17.41 T_c - J_c = 5188.18$
2. $J_c - 0.71 J_h + 7.46 T_c = 2352.71$
3. $5.67 * 10^{-8} T_h^4 + 1.865 T_h - J_h = 2250$
4. $J_h - 0.71 J_c + 7.46 T_h = 11093$

To solve the linear system, the fixed-point iteration method is used. The hint iteration functions from the homework are employed for this. See the source code, and the figure. Final results are:

$Tc = 481.0273[K]$
$Th = 671.124[K]$
$Jc = 6222.2251[W/m^2]$
$Jc = 10504.1949[W/m^2]$

Appendix: MATLAB Source code

```
%% MEMS 1060/2060 Homework 2
% Author: Shane Riley
% Date: 2/8/2021
format long
%% Problem 1
% Plot the given function for the given
    span of x
disp(” ”);
disp(”Problem 1”);

A = [...
    3   -2   5
    1   -1   0
```

```matlab
    2   0   4];
b = [...
    14
    -1
    14];

x = myCramer(A,b);
disp("x = " + x(1));
disp("y = " + x(2));
disp("z = " + x(3));


%% Problem 2
disp(" ");
disp("Problem 2");

% Givens
fun = @(x) sin(x);
c = pi/4;
xspan = [0 pi];
numPoints = 100;
x = linspace(xspan(1), xspan(2), 200);
y2 = myTaylorSinPi4(x, 2);
y4 = myTaylorSinPi4(x, 4);
y6 = myTaylorSinPi4(x, 6);

% Plot results
hold on
figure(1)
plot(x,y2)
plot(x,y4)
plot(x,y6)
fplot(fun, xspan);
legend(["2 term Taylor", "4 term Taylor
    ", "6 term Taylor", "f(x)"], '
    Location', 'SW');
xlim(xspan)
xlabel("x")
ylabel("y")
title("Taylor series of f(x) = sin(x)")

% Print figure
print("figure/Taylor", '-dpng');
hold off


%% Problem 3
disp(" ");
disp("Problem 3");

fun = @(x) cos(x) - x^3;
dfun = @(x) -sin(x) - 3*x^2;
tolerance = 0.001;
xspan = [0, pi/2];
nmax = 1000;

disp("f( " + xspan(1) + " ) = " + fun(
    xspan(1)));
disp("f( " + xspan(2) + " ) = " + fun(
    xspan(2)));
[c, n] = myNewton(fun, dfun, xspan,
    tolerance, nmax);
disp("Newton's method after " + n + "
    iterations:");
disp("c = " + c + "; f(c) = " + fun(c));
[c2, n2] = myRegulaFalsi(fun, xspan,
    tolerance, nmax);
disp("Regula Falsi method after " + n2 +
    " iterations:");
disp("c = " + c2 + "; f(c) = " + fun(c2)
    );


%% Problem 4
disp(" ");
disp("Problem 4");

% Values
numIterations = 200;
TcStart = 298;  % K
ThStart = 298;  % K
JcStart = 3000;  % W/m^2
JhStart = 5000;  % W/m^2

start = [...
    TcStart
    ThStart
    JcStart
    JhStart];

% Hint iteration functions
TcNew = @(Jc, Tc) ((Jc - 17.41*Tc +
    5188.18)/(5.67e-8))^(1/4);
ThNew = @(Jh, Th) ((2250 + Jh - 1.865*Th
    )/(5.67e-8))^(1/4);
JcNew = @(Jh, Tc) (2352.71 + 0.71*Jh -
    7.46*Tc);
JhNew = @(Jc, Th) (11093 + 0.71*Jc -
    7.46*Th);

delta = @(arr) [...
    TcNew(arr(3),arr(1))
    ThNew(arr(4),arr(2))
    JcNew(arr(4),arr(1))
    JhNew(arr(3),arr(2))];

values = zeros(4,numIterations);
values(:,1) = start;
for i=1:numIterations
    % Iterate
    values(:,i+1) = delta(values(:,i));
end

% Get final
disp("After " + numIterations + "
    iterations:");
```

```
disp("Tc = " + values(1,end) + " [K]");
disp("Th = " + values(2,end) + " [K]");
disp("Jc = " + values(3,end) + " [W/m^2]");
disp("Jc = " + values(4,end) + " [W/m^2]");

% Make the plot
figure(2)
hold on
plot(values(1,:),'b-');
plot(values(2,:),'r-');
ylabel("Temperature [K]");
yyaxis right
plot(values(3,:),'b--');
plot(values(4,:),'r--');
xlim([0, numIterations]);
title("Iteration Values for Heater System");
xlabel("Iteration number");
ylabel("Heat Flux [W/m^2]");
legend("Tc", "Th", "Jc", "Jh", 'Location', 'SW');

% Print figure
print("figure/Values", '-dpng');
hold off


%% Supporting functions

function x = myCramer(A, b)
% MYCRAMER evaluates x for Ax = b using Cramer's rule
% A: Coefficient matrix
% b: constant vector

% Check dimensions
[nA,mA] = size(A);
assert (nA == mA);
[nB,mB] = size(b);
assert (nB == nA);
assert (mB == 1);

% Preperation
order = nA;
x = zeros(1, order);
detA = det(A);

% Evaluate solution components in order
for i = 1:order
    Abefore = A(:, 1:i-1);
    Aafter = A(:, i+1:order);
    Asub = [Abefore, b, Aafter];
    x(i) = det(Asub) / detA;
end
end

function y = myTaylorSinPi4(x, n)
% MYTAYLORSINPI4 Gets the y values for a taylor series for sinx centered
% about pi/4
% x: vector of inputs
% n: number of terms
% y: vector of outputs
% TODO: GENERALIZE FOR ALL values of c
c = pi/4;

% Expresses value of of the nth derivative of sin at pi/4
sinDerivativeAtPi4 = @(n) (sqrt(2)/2) * (-1) ^ ((n)*(n-1)/2);

% Construct polynomial as terms in a vector
for count = 1:n
    terms{count} = @(x) sinDerivativeAtPi4(count - 1) * (x - c)^(count - 1) / factorial(count - 1);
end

% Using terms vector, compute y values
y = zeros(1, length(x));
for j = 1:length(x)
    xValue = x(j);
    yValue = 0;
    for termNum = 1:length(terms)
        term = @(x) terms{termNum}(x);
        deltaY = term(xValue);
        yValue = yValue + deltaY;
    end
    y(j) = yValue;
end
end

function [x, n] = myRegulaFalsi(f,xspan,t,nmax)
% MYREGULAFALSI finds a root of f between a and b, to a tolerance of t.
% Moves lower bound forward until a solution is found
% using Regula Falsi
% f: function handle
% xspan: upper and lower bounds;
% t: tolerance
% nmax: max number of iterations
% x: root value found
% n: number of iterations required

a = xspan(1);
b = xspan(2);
n = 0;
while abs(f(a)) > t
    % Iterate
```

```
    deltaA = (a - b)/((f(b)/f(a)) - 1);
    a = a + deltaA;
    n = n + 1;
    if n > nmax
        disp("Solution not found within alloted iterations");
        x=NaN;
        return;
    end
end
x = a;
end

function [x, n] = myNewton(f, df, xspan, t, nmax)
% MYNEWTON finds a root of f near a, to a tolerance of t, using Newton's
% Method. Seeds using the midpoint of the interval
% f: function handle
% df: derivative function handle
% xspan: Interval
% t: tolerance
% nmax: max number of iterations
% x: root value found
% n: number of iterations required

xValue = (xspan(2) - xspan(1)) / 2;
n = 0;
while abs(f(xValue)) > t
    % Iterate
    xValue = xValue - (f(xValue)/df(xValue));
    n = n + 1;
    if n > nmax
        disp("Solution not found within alloted iterations");
        x=NaN;
        return;
    end
end
x = xValue;
end
```