CS / MATH 4334 : Numerical Analysis
Homework Assignment 2

Matthew McMillian
mgm160130@utdallas.edu

September 25, 2018

MatLab Problems

Problem 1 : ffalpos.m

```
function c = ffalpos(a, b)
%FFALPOS Summary of this function goes here
%   Detailed explanation goes here
fa = (a-2)^2 + 1 - (1 - (a-2)^2 / 4)^0.5 - 1;
fb = (b-2)^2 + 1 - (1 - (b-2)^2 / 4)^0.5 - 1;
c = b - fb*((b-a)/(fb-fa));
end
```

Problem 1 : falpos.m

```
%find root of f(x) = 0
%using Bisection Method

format long e

% chosen error tolerance (TOL)
TOL = .000001;

% choose max number of iterations
MAXIT = 50;

% initial bracket
% Calcuated from simplifying (f-g)(x)
% There exists a root since f(a)f(b) < 0, and
% f(a) > 0, f(b) < 0;
a = 1;
b = 2;

%keep track of number of iterations
count = 0;

%record iterates - a col vector of MAXIT length
cits = zeros(MAXIT,1);

%evaluate func. at a and b
fa = 4*a^4 - 32*a^3 + 97*a^2 - 132*a + 64;
fb = 4*b^4 - 32*b^3 + 97*b^2 - 132*b + 64;

%stop if not appropriate interval
if sign(fa)*sign(fb) >= 0
    fprintf("Not appropiate interval\n")
    return
```

```
end

%stop loop when error less than TOL or MAXIT reached
while abs(b-a)/2 >= TOL && count < MAXIT

    %get midpoint(root estimate)
    c = ffalpos(a, b);

    %eval. func at midpoint
    fc = 4*c^4 - 32*c^3 + 97*c^2 - 132*c + 64;

    %stop if f(c)=0
    if fc == 0
        break
    end

    %update count
    count = count + 1;

    %add to list of iterates
    cits(count) = c;

    %if sign change between a and c make c the new right endpt
    if sign(fa)*sign(fc)<0

        b = c;

    %if sign chg betw c and b make c the new left endpt
    else

        a = c;

    end

end

%update count
    count = count + 1;

%get final midpoint(root estimate)
    c = ffalpos(a,b);

%add to vector of iterates
cits(count) = c;
```

```
%display error estimate
error = abs(b-a)/2

%display vector of iterates
cits

%display number of iterates
count
```

\>> falpos.m

```
cits =

1.118146029604788e+00
1.062353732855704e+00
1.060497138511485e+00
1.060437095603502e+00
1.060435155605902e+00
1.060435092926107e+00
1.060435090900974e+00
1.060435090835543e+00
1.060435090833429e+00
1.060435090833361e+00
1.060435090833359e+00


count =

11
```

Problem 2 : newt1.m

```
format long e

% Tolerance Level
TOL = 10^(-6);

% Initial Guess
xp = 2;

iterates = [];

% While the relative error is less than the give tolerance, we
    will
% continue to use newton's method.
while 1

    xf = xp - fnewt(xp) / fpnewt(xp);
    iterates = [iterates, xf];

    if(abs((xf-xp)/(xf)) < TOL)
        break
    end

     xp = xf;

end

% Ouprinting the resulting iterates
clc
fprintf("Iterates::")
iterates'
```

Problem 2 : newt2.m

```
format long e

clc

% Tolerance Level
TOL = 10^(-6);

% Initial Guess
xp = 2;
```

```
% Arrays to store the iterates and error values required
iterates = [];
relerr = [];
relerr2 = [];

% ep is 0 for the first iteration
ep = 1;

% While the relative error is less than the give t olerance, we will
% continue to use newton's method.
while 1

    % Newton's Algorithm
    xf = xp - fnewt(xp) / fpnewt(xp);
    iterates = [iterates, xf];

     % Error and Tolerance
    ef = abs((xf-xp)/(xf));
    fprintf("%f\n", ef)
    if(ef < TOL)
        break
    end

    relerr = [relerr, ef/ep];
    relerr2 = [relerr2, ef/ep^2];

    xp = xf;
    ep = ef;

end

% Ouprinting the resulting iterates
clc
fprintf("Iterates::\n")
iterates'

fprintf("Error 1::\n")
relerr'

fprintf("Error 2::\n")
relerr2'

fprintf("Since the derivative of the function is non-zero, we
```

```
    can determine that there are multiple roots.\n")

v = round(iterates(length(iterates)), 3);
mult = (147500 * (1 + (v/12))^58)/2*fpnewt(v);

rofc = (mult-1)/mult;

fprintf("Iterates:: %d\n", v)
fprintf("Multiplicity:: %d\n", mult)
fprintf("Rate of Convergence:: %d\n", rofc)

fprintf("We have determined that we have a rate of convergence
    of approx: 1, therefore we have linear convergence for
    this newton's method.\n")
```

>> newt2.m

a)

1.a) Actual. value of pi from MatLab = 3.141593e+00

1.b) Approx. value of pi using Maclaurin series: 4arctan(1) = 3.141597e+00

2) Absolute Error: 4.140539e-06

3) Relative Error: 1.317974e-06

4) Number of 'k' terms needed to approx. in single percision: 16777216

b)

Eventually the next value in the series becomes extremely small (since k is constantly increasing in the denominator, the next value to be added in the series will be small). This is the result of the phenomenon SWAMPING, since we are trying to add two numbers whose sizes are very different (one large and one extremely small). Therefore, the percision will eventually lose track of the very small values computed due to the rounding and computational limitations.

Problem 3 : hw3.m

```
format long e

clc

x1 = 5;
x2 = 5 + 10^(-10);

fun = @(x)exp(x-1) - 1;
actual = 1;

fprintf("fzero(function1, x) results::\n")
[rootest, fval] = fzero(fun, x1);
backerr = abs(fun(rootest));
forerr = abs(actual - rootest);
fprintf("(x, rootest, fval): %d, %d, %d\n", x1, rootest, fval)
fprintf("(Backward error, Forward error): %d, %d\n", backerr, forerr)

[rootest, fval] = fzero(fun, x2);
backerr = abs(fun(rootest));
forerr = abs(actual - rootest);
fprintf("(x, rootest, fval): %d, %d, %d\n", x2, rootest, fval)
fprintf("(Backward error, Forward error): %d, %d\n\n", backerr, forerr)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fun = @(x)exp(4*x-4) - 2*exp(3*x-3) + 2*exp(x-1) - 1;

fprintf("fzero(function2, x) results::\n")
[rootest, fval] = fzero(fun, x1);
backerr = abs(fun(rootest));
forerr = abs(actual - rootest);
fprintf("(x, rootest, fval): %d, %d, %d\n", x1, rootest, fval)
fprintf("(Backward error, Forward error): %d, %d\n", backerr, forerr)

[rootest, fval] = fzero(fun, x2);
backerr = abs(fun(rootest));
forerr = abs(actual - rootest);
fprintf("(x, rootest, fval): %d, %d, %d\n", x2, rootest, fval)
fprintf("(Backward error, Forward error): %d, %d\n\n", backerr
```

```
    , forerr)

syms f(x)
f(x) = exp(x-1) - 1;

mult1 = 0;
while f(actual) == 0
    f = diff(f, x);
    mult1 = mult1 + 1;
end

f(x) = exp(4*x-4) - 2*exp(3*x-3) + 2*exp(x-1) - 1;

mult2 = 0;
while f(actual) == 0
    f = diff(f, x);
    mult2 = mult2 + 1;
end
fprintf("Multiplicity f1: %d\n", mult1)
fprintf("Multiplicity f2: %d\n", mult2)
fprintf("As the multiplicity increases, stability decreases.\n
    Thus, with multiplicity 1 for the first function, we\n
    have a pretty stable algorithm. However, when we \n
    introduce higher multiplicity in function 2, we start to \n
    lose some of our stability and it begins to slightly
    affect \n our approximations. The inital guess doesn't
    affect the first \n function very much, but you can see the
    increased variablity in \n the seconds function from the
    root difference.\n")
```

>> hw3.m

```
fzero(function1, x) results:: (x, rootest, fval): 5, 1, 0
(Backward error, Forward error): 0, 0
(x, rootest, fval): 5.000000e+00, 1, 0
(Backward error, Forward error): 0, 0

fzero(function2, x) results::
(x, rootest, fval): 5, 9.999996e-01, 0
(Backward error, Forward error): 0, 4.312810e-07
(x, rootest, fval): 5.000000e+00, 1.000003e+00, 0
(Backward error, Forward error): 0, 3.095297e-06

Multiplicity f1: 1
Multiplicity f2: 3
```

As the multiplicity increases, stability decreases. Thus, with multiplicity 1 for the first function, we have a pretty stable algorithm. However, when we introduce higher multiplicity in function 2, we start to lose some of our stability and it begins to slightly affect our approximations. The inital guess doesn't affect the first function very much, but you can see the increased variablity in the seconds function from the root difference.