MATH 5610 HW3 SOLUTIONS

Problem 1: See attached code

Note: B&F 2.4.2 b has a typo in function f.
it should be:

$$f(x) = x^6 + 6x^5 + 9x^4 - 2x^3 - 6x^2 + 1$$

Problem 2

(a) if iteration converges, it converges to a fixed point of the function $F(x)$:

$$x_* = \frac{(24x_* + 25/x_*^2)}{25}$$

$$\Rightarrow \frac{x_*}{25} = \frac{1}{x_*^2} \quad \Rightarrow \quad x_*^3 = 25$$

The only real root is $\boxed{x_* = \sqrt[3]{25}}$.

(b) We need to verify:

- $|F'(x)| \leq C < 1$ for $x \in [2,3]$

$$F'(x) = \frac{24}{25} - \frac{1}{2x^3} = \text{monotonically increasing on } [2,3]$$

$$F'(2) = \frac{24}{25} - \frac{1}{2\times 8} \simeq 0.90$$

$$F'(3) = \frac{24}{25} - \frac{1}{2\times 27} \simeq 0.94$$

$\Rightarrow F$ is a contraction on $[2,3]$.

- F maps $[2,3]$ to $[2,3]$:

  Since $F'(x) > 0$ for $x \in [2,3]$,

  F must be monotonically increasing on $[2,3]$

  i.e.:

$$[2,3] \xrightarrow{F} [F(2), F(3)] \simeq [2.17, 2.99] \subset [2,3]$$

Thus by the contractive mapping theorem, F admits a unique fixed point in $[2,3]$ and fixed point iteration converges for any $x_0 \in [2,3]$.

(c) Steffensen's method converges faster.
See attached code & output.

Problem 3 $\quad P(z) = 3z^5 - 7z^4 - 5z^3 + z^2 - 8z + 2$

(a)

	3	−7	−5	1	−8	2
4		12	20	60	244	944
	3	5	15	61	236	**946**

$\Rightarrow$ **$P(4) = 946$**

(b) Applying Horner's algorithm saccesively we get

	3	−7	−5	1	−8	2
4		12	20	60	244	944
	3	5	15	61	236	**946**
4		12	68	332	1572	
	3	17	83	393	**1808**	
4		12	116	796		
	3	29	199	**1189**		
4		12	164			
	3	41	**363**			
4		12				
	3	**53**				

Thus:

$$P(x) = 946 + 1808(x-4) + 1189(x-4)^2 + 363(x-4)^3 + 53(x-4)^4 + 3(x-4)^5$$

(c) $z_1 = z_0 - \dfrac{P(z_0)}{P'(z_0)}$ (Newton update)

From table above: $P(z_0) = 946$, $P'(z_0) = 1808$

Thus:

$$z_1 = 4 - \frac{946}{1808} = \frac{4 \times 904 - 473}{904} = \frac{3143}{904} \approx 3.477$$

Problem 4: See attached code & output.

Problem 5

(a) If z_* is a root of multiplicity m of $p(z)$ then:

$$p(z) = (z - z_*)^m q(z), \text{ with } q(z_*) \neq 0.$$

Thus

$$p^{(n)}(z) = \sum_{k=0}^{n} \binom{n}{k} \left((z - z_*)^m\right)^{(k)} q^{(n-k)}(z)$$

Now note that:

$$\left((z - z_*)^m\right)^{(k)} = \begin{cases} \frac{m!}{(m-k)!}(z - z_*)^{m-k} & \text{if } 0 \leq k < m \\ m! & \text{if } k = m \\ 0 & \text{if } k > m \end{cases}$$

Thus:

$$p^{(n)}(z_*) = 0 \quad \text{if } 0 \leq n \leq m-1$$

but $p^{(m)}(z_*) = m!\, q(z_*)$, w/ $q(z_*) \neq 0$.

QED

(b) Assume p is a poly for which:

$$p(z_*) = p'(z_*) = \cdots = p^{(m-1)}(z_*) = 0$$

but $p^{(m)}(z_*) \neq 0$.

Then its Taylor expansion about $z = z_*$ is:

$$p(z) = \sum_{k=0}^{n} \frac{p^{(k)}(z_*)}{k!} (z - z_*)^k$$

where n = degree of poly. p.

By hypothesis:

$$p(z) = \frac{p^{(m)}(z_*)}{m!} (z - z_*)^m + \sum_{k=m+1}^{n} \frac{p^{(k)}(z_*)}{k!} (z - z_*)^k$$

$$= (z - z_*)^m \underbrace{\left[\frac{p^{(m)}(z_*)}{m!} + \sum_{k=1}^{n-m} \frac{p^{(k+m)}(z_*)}{(k+m)!} (z - z_*)^k \right]}_{= q(z)}$$

$q(z)$ is a poly which does not vanish at $z = z_*$ since:

$$q(z_*) = \frac{p^{(m)}(z_*)}{m!}$$

QED.

```
>> prob1
Part a: NM=   0.73907868 in   17 iter MNM=   0.73908513 in    6 iter
Part b: NM=  -2.87939396 in   15 iter MNM=  -2.87938523 in    5 iter
>> prob2
             fixed point   Steffensens
      root    2.92403530    2.92401774
      iter            14            13
   g evals            13             8
true value: (25)^(1/3) =    2.92401774
>> prob3
----- part a -----
found root r1 =   4.123105626 in 5 iterations
found root r2 =  -4.123105626 in 8 iterations
deflated polynomial is:
   1.000000000000000   4.999999999999526   7.999999999999584

with roots:
 -2.499999999999763 + 1.322875655532586i
 -2.499999999999763 - 1.322875655532586i

the roots according to Matlab are:
 -2.499999999999999 - 1.322875655532298i
 -2.499999999999999 + 1.322875655532298i
 -4.123105625617663
  4.123105625617665

----- part h -----
found root r1 =  0.5857864376 in 5 iterations
found root r2 =   3.414213562 in 6 iterations
found root r3 =             3 in 5 iterations
there are no imaginary roots
the roots according to Matlab are:
   0.585786437626905
   2.999999999999985
   3.414213562373112
```

```
% HW 3 Problem 1
% B&F 2.4.2 and 2.4.4
tol = 1e-5; maxit = 20;

%% Part A
x0 = 0; % initial guess
f  = @(x) 1 - 4*x*cos(x) + 2*x^2 + cos(2*x);
fp = @(x) 4*x - 2*sin(2*x) - 4*cos(x) + 4*x*sin(x);
fpp= @(x) 8*sin(x) - 4*cos(2*x) + 4*x*cos(x) + 4;
xs1 = newton(f,fp,x0,tol,maxit);
xs2 = newton_mult(f,fp,fpp,x0,tol,maxit);
fprintf('Part a: NM=%13.8f in %4d iter MNM=%13.8f in %4d iter\n',...
       xs1(end),length(xs1),xs2(end),length(xs2));

%% Part B
x0 = -3; % initial guess
% NOTE: the function below has a typo in the book
%       the first term in the sum should be x^6 and
%       not x^2
f  = @(x) x^6 + 6*x^5 + 9*x^4 - 2*x^3 - 6*x^2 +1;
fp = @(x) 6*x^5 + 30*x^4 + 36*x^3 - 6*x^2 - 12*x;
fpp= @(x) 30*x^4 + 120*x^3 + 108*x^2 - 12*x - 12;
xs1 = newton(f,fp,x0,tol,maxit);
xs2 = newton_mult(f,fp,fpp,x0,tol,maxit);
fprintf('Part b: NM=%13.8f in %4d iter MNM=%13.8f in %4d iter\n',...
       xs1(end),length(xs1),xs2(end),length(xs2));
```



```
% HW 3 Problem 2

% cube root of 25 (or 25^(1/3)) can be approximated by the following fixed point
% iteration
g = @(x) (x + 25/x^2)/2;

x0    = 5;   % initial guess
maxit = 50;  % max number of iterations
tol   = 1e-4;% tolerance

xs1 = fixedpoint(g,x0,maxit,tol);
xs2 = steffensen(g,x0,maxit,tol);
fprintf('%10s %13s %13s\n', ' ','fixed point','Steffensens');
fprintf('%10s %13.8f %13.8f\n','root',xs1(end),xs2(end));
fprintf('%10s %13d %13d\n','iter',length(xs1),length(xs2));
fprintf('%10s %13d %13d\n','g evals',length(xs1)-1,2*(length(xs2)-1)/3);
% for Steffensen's 2 out of 3 iterations are function evaluations.
fprintf('true value: (25)^(1/3) = %13.8f\n',(25)^(1/3));
```

```
% HW 3 Problem 3
tol = 1e-5;
maxit = 20;
format long

fprintf('----- part a -----\n');
% This defines the polynomial
% p(z) = z^4 + 5 z^3 - 9 z^2 -85 z -136
p = [1 5 -9 -85 -136];


%  try starting at z0 = 5
z0=5;
[r1,k]=polynm(p,z0,tol,maxit);
fprintf('found root r1 = %13.10g in %d iterations\n',r1,k);

%  try starting at z0 = -1
z0=-1;
[r2,k]=polynm(p,z0,tol,maxit);
fprintf('found root r2 = %13.10g in %d iterations\n',r2,k);

% deflate polynomial
[q1,r] = horner(p,r1);
[q2,r] = horner(q1,r2);

fprintf('deflated polynomial is:\n');
disp(q2)

fprintf('with roots:\n');
del = q2(2)^2 - 4*q2(1)*q2(3);
disp([(-q2(2)+sqrt(del))/2/q2(1);(-q2(2)-sqrt(del))/2/q2(1)])

% compare to the roots that Matlab finds:
fprintf('the roots according to Matlab are:\n');
disp(sort(roots(p)));

fprintf('----- part h -----\n');
% This defines the polynomial
% p(z) = z^3 -7 z^2 + 14 z -6
p = [1 -7 14 -6];

% try starting at z0 = 0
z0 = 0;
[r1,k]=polynm(p,z0,tol,maxit);
fprintf('found root r1 = %13.10g in %d iterations\n',r1,k);

% try starting at z0 = 4
z0 = 4;
[r2,k]=polynm(p,z0,tol,maxit);
fprintf('found root r2 = %13.10g in %d iterations\n',r2,k);

% try starting at z0 = 2
z0 = 2.1;
[r3,k]=polynm(p,z0,tol,maxit);
fprintf('found root r3 = %13.10g in %d iterations\n',r3,k);

fprintf('there are no imaginary roots\n');

% compare to the roots that Matlab finds:
fprintf('the roots according to Matlab are:\n');
disp(sort(roots(p)));
```



```
% Newton method to find a root of a function f starting at x0
%
% Inputs
%  f       function we want to find the root of
%  fprime  derivative of f
%  x0      initial iterate
%  tol     desired tolerance or precision
%  maxit   maximum  number of iterations
%
% Outputs
%  xs      iteration history, last iterate is xs(end)
function xs = newton(f,fprime,x0,tol,maxit)

xs = [x0]; % store iterates

x = x0;

for k=1:maxit,
 xnew = x - f(x)/fprime(x);
 xs = [xs xnew];                % store iterates
 if abs(xnew-x)<tol break; end; % check convergence
 x = xnew;                      % prepare next iteration
end;
```

Sep 20, 11 14:20 **newton_mult.m** Page 1/1

```
% Newton method to find a root of a function f starting at x0
%
% Inputs
%  f       function we want to find the root of
%  fp      derivative of f
%  fpp     second derivative of f
%  x0      initial iterate
%  tol     desired tolerance or precision
%  maxit   maximum  number of iterations
%
% Outputs
%  xs      iteration history, last iterate is xs(end)
function xs = newton(f,fp,fpp,x0,tol,maxit)

xs = [x0]; % store iterates

x = x0;

for k=1:maxit,
 xnew = x - f(x)*fp(x)/((fp(x))^2 - f(x)*fpp(x));
 xs = [xs xnew];                 % store iterates
 if abs(xnew-x)<tol break; end; % check convergence
 x = xnew;                       % prepare next iteration
end;
```

Sep 20, 11 14:20 **fixedpoint.m** Page 1/1

```
% Fixed point iteration
% Inputs
%  g     contraction (function taking a real argument and returning a real
%        number)
%  x0    initial iterate
%  maxit maximum number of iterations
%  tol   tolerance for two consecutive iterates
%
% Outputs
%  xs    vector containing the iterates
function xs = fixedpoint(g,x0,maxit,tol)
x = x0;
xs = x; % store iterate
for k = 1:maxit,
 xnew = g(x);
 xs = [xs xnew];                 % store iterate
 if abs(xnew-x)<tol break; end; % check convergence
 x = xnew;                       % prepare next iteration
end;
```

```
% Steffensen's method to accelerate convergence of certain fixed point iteration
s
% Inputs
%  g     contraction (function taking a real argument and returning a real
%        number)
%  x0    initial iterate
%  maxit maximum number of iterations
%        (each Steffensen iteration costs two evaluations of g)
%  tol   tolerance for two consecutive iterates
%
% Outputs
%  xs    vector containing the iterates
function xs = steffesnse(g,x0,maxit,tol);
xs = x0; % store iterate
k=1;
for k = 1:maxit,
 x1 = g(x0);
 x2 = g(x1);
 x3 = x0 - (x1-x0)^2/(x2-2*x1+x0);
 xs = [xs x1 x2 x3];            % store iterates
 if abs(x3 -x0)<tol break; end;  % check convergence
 x0 = x3;                       % prepare next iteration
end;
```



```
% Newton's method using Horner's algorithm to evaluate a polynomial and its
% derivative efficiently
%
% Inputs
%   p      Vector of polynomial coefficients, from highest to lowest degree
%   z0     initial guess
%   tol    tolerance
%   maxit  maximum number of iterations
%
% Outputs
%   z      last iterate
%   k      number of iterations
function [z,k] = polynm(p,z0,tol,maxit)
for k=1:maxit,
 [alpha,beta] = hornernm(p,z0);
 z = z0 - alpha/beta;
 %fprintf('z = %14g + i%14g, |z-z0| = %14g\n',real(z),imag(z),abs(z-z0));
 if abs(z-z0)<tol break; end;
 z0=z;
end;
```

```
% Computes value of a polynomial and its derivative at a point
% using Horner's algorithm twice.
%
% Inputs
%  p    vector of polynomial coefficients, from highest to lowest degree
%  z0   evaluation point
%
% Outputs
%  alpha  p(z0)
%  beta   p'(z0)

function [alpha,beta] = hornernm(p,z0)
n = length(p);
alpha = p(1); beta = 0;
for k=1:n-1,
 beta = alpha + z0*beta;
 alpha = p(k+1)+z0*alpha;
end;
```



```
% Horner's algorithm to divide a polynomial p by the
% linear factor z-z0
%
% i.e.
%
%  p(z) = q(z) (z-z0) + r
%
% Inputs
%  p     vector of polynomial coefficients, from highest to lowest degree
%  z0    root of linear factor
%
%  Outputs
%  q     vector of polynomial coefficients for the quotient polynomial
%  r     residual
function [q,r] = horner(p,z0)
n = length(p);
q = zeros(size(p));
q(1) = p(1);
for k=1:n-1,
 q(k+1) = p(k+1) + z0 * q(k);
end;
r = q(n);
q = q(1:n-1);
```