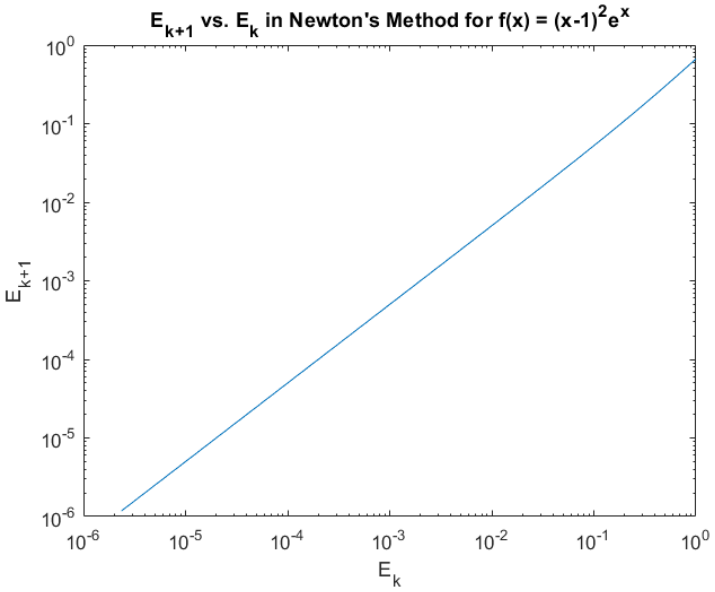
# MACM 316 – Assignment 3

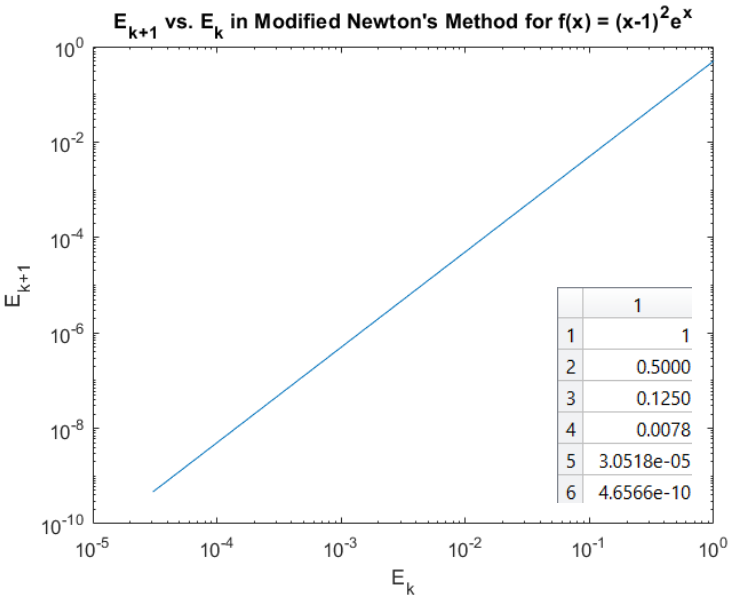
Rajan Grewal 301335629

a)

| | 1 |
|---|---|
| 1 | 2.000000000000000 |
| 2 | 1.666666666666667 |
| 3 | 1.416666666666667 |
| 4 | 1.244252873563218 |
| 5 | 1.135418046418532 |
| 6 | 1.072002807118831 |
| 7 | 1.037252464662030 |
| 8 | 1.018966824905443 |
| 9 | 1.009572502685051 |
| 10 | 1.004809050422313 |
| 11 | 1.002410293083653 |
| 12 | 1.001206597171792 |
| 13 | 1.000603662335630 |
| 14 | 1.000301922242380 |
| 15 | 1.000150983907010 |
| 16 | 1.000075497652110 |
| 17 | 1.000037750250975 |
| 18 | 1.000018875481751 |
| 19 | 1.000009437829946 |
| 20 | 1.000004718937241 |
| 21 | 1.000002359474188 |
| 22 | 1.000001179738486 |

b) For the first iteration, $E_{k+1} \leq \alpha E_k^p$ is not affected by values of $p \geq 1$, and we know p cannot be $< 1$ since the iterations are not converging at a decreasing rate judging from the increasing relative differences in $E_k$ shown on the table. This gives us $\alpha = 2/3$. By plugging in p=1, we can see the inequality holds, but not tightly, so $p = 1$ is a rough estimate.




| | 1 |
|---|---|
| 1 | 1 |
| 2 | 0.5000 |
| 3 | 0.1250 |
| 4 | 0.0078 |
| 5 | 3.0518e-05 |
| 6 | 4.6566e-10 |

c) For the first iteration, $E_{k+1} \leq \alpha E_k^p$ is not affected by values of $p \geq 1$, and we know p cannot be $< 1$ since the iterations are not converging at a decreasing rate, judging from the quickly increasing relative differences in $E_k$ from the table shown. This gives us $\alpha=0.5$ by performing $E_1/E_0$. Plugging in $\alpha = 0.5$, and k=1, as an example, the equation $E_{k+1} \leq \alpha E_k^p = .125 \leq .5(.5)^p = .25 \leq .5^p$. We know p is usually around 1 or 2, so by guess and check (and simply by inspection), plugging in $p = 2$ satisfies the inequality with the tightest possible bound. The modified version managed to find the root in only 5 iterations. This is due to the smaller alpha value and larger order of convergence. The modified method requires f'(x) and can be subject to subtractive cancellation error in the denominator, but it is faster at finding roots and can still work for double roots, unlike the original method.

d) The original bisect2 algorithm does not work because when the if statement checks the condition f(mid) * f(xint(1)) < 0, it always evaluates to false since the function is never negative, making it search on the right each iteration, and return the right endpoint.

```
% a)
f = @(x) (x-1)^2*exp(x);
fp = @(x) (x^2-1)*exp(x);
fpp = @(x) (x^2+2*x-1)*exp(x);
[root, iter, xlist] = newton(f, fp, 2, 1e-6)

% b)
for i = xlist
    ek = abs(i-1)
end
ek1 = circshift(ek,-1);
ek1(22) = 0;
loglog(ek, ek1)
xlabel('E_k')
ylabel('E_k+_1')
title 'E_k+_1 vs. E_k in Newton''s Method for f(x) = (x-1)^2e^x'

% c)
[root, iter, xlist] = mnewton(f, fp, fpp, 2, 1e-6)
for i = xlist
    ekk = abs(i-1)
end
ekk1 = circshift(ekk,-1);
ekk1(6) = 0
loglog(ekk, ekk1)
xlabel('E_k')
ylabel('E_k+_1')
title 'E_k+_1 vs. E_k in Modified Newton''s Method for f(x) = (x-1)^2e^x'
[root, niter, rlist] = bisect2(f, [-15,15], 1e-6)
```

## Modified Newton's Method

```
function [root, iter, xlist] = mnewton( func, pfunc, ppfunc, xguess, tol )
if nargin < 4
  fprintf(1, 'NEWTON: must be called with at least four arguments' );
  error( 'Usage:  [root, niter, xlist] = newton( func, pfunc, xguess, [tol], [mult] )' );
end
if nargin < 5, tol  = 1e-6; end
func = fcnchk( func );
pfunc= fcnchk( pfunc );
x    = xguess;
fx   = feval( func,  x );
fpx  = feval( pfunc, x );
fppx = feval( ppfunc, x);
if( fx == 0 | fpx == 0 )
  error( 'NEWTON: both f and f'' must be non-zero at the initial guess' );
end
xlist= [ x ];
done = 0;
iter = 0;
while( ~done )
  x0  = x;
  x   = x0 - (fx * fpx) / (fpx^2 - (fx*fppx));
  fx  = feval( func,  x );
  fpx = feval( pfunc, x );
  fppx = feval(ppfunc, x);
  if( abs(x-x0) < tol )     % absolute tolerance on x
    done = 1;
  else
    xlist = [ xlist; x ];   % add to the list of x-values
    iter  = iter + 1;
  end
end

root = x;
```