CS471: Scientific Computing: Homework 2 Report

Authors: Brendan Donohoe, Tim Chavez

Date: September 14, 2016

At first, the implementation of Newton's method caused the program to iterate over Newton's method 10 times, which was not enough for some functions, but too much for others. We modified the implementation to stop iterating when the approximation of the error, given by $|x_{n+1} - x_n|$, is less than 10^{-15}. Additionally, we modified the write statement to include the ratios for the error at a given iteration over the error of the previous iteration. (See Appendix A for code).

$f(x) = x$

	x[n]	Eabs(x[n])
n = 0	1	-1
n = 1	0	1
n = 2	0	0

$f(x) = x^2$

	x[n]	Eabs(x[n])	Linear Err Ratio	Quad Err Ratio
n = 0	1	-1	-1	-1
n = 1	0.5000	0.5000	-1	-1
n = 2	0.2500	0.2500	0.5000	1
n = 3	0.1250	0.1250	0.5000	2
n = 4	0.0625	0.0625	0.5000	4
n = 5	0.0313	0.0313	0.5000	8
n = 6	0.0156	0.0156	0.5000	16
n = 7	0.0078	0.0078	0.5000	32
n = 8	0.0039	0.0039	0.5000	64
n = 9	0.0020	0.0020	0.5000	128
n = 10	9.7656e-04	9.7656e-04	0.5000	256
n = 11	4.8828e-04	4.8828e-04	0.5000	512
n = 12	2.4414e-04	2.4414e-04	0.5000	1024
n = 13	1.2207e-04	1.2207e-04	0.5000	2048
n = 14	6.1035e-05	6.1035e-05	0.5000	4096
n = 15	3.0518e-05	3.0518e-05	0.5000	8192
n = 16	1.5259e-05	1.5259e-05	0.5000	16384
n = 17	7.6294e-06	7.6294e-06	0.5000	32768
n = 18	3.8147e-06	3.8147e-06	0.5000	65536
n = 19	1.9073e-06	1.9073e-06	0.5000	131072
n = 20	9.5367e-07	9.5367e-07	0.5000	262144
n = 21	4.7684e-07	4.7684e-07	0.5000	524288
n = 22	2.3842e-07	2.3842e-07	0.5000	1048576
n = 23	1.1921e-07	1.1921e-07	0.5000	2.0972e+06
n = 24	5.9605e-08	5.9605e-08	0.5000	4.1943e+06
n = 25	2.9802e-08	2.9802e-08	0.5000	8.3886e+06
n = 26	1.4901e-08	1.4901e-08	0.5000	1.6777e+07
n = 27	7.4506e-09	7.4506e-09	0.5000	3.3554e+07
n = 28	3.7253e-09	3.7253e-09	0.5000	6.7109e+07
n = 29	1.8626e-09	1.8626e-09	0.5000	134217728
n = 30	9.3132e-10	9.3132e-10	0.5000	268435456
n = 31	4.6566e-10	4.6566e-10	0.5000	536870912
n = 32	2.3283e-10	2.3283e-10	0.5000	1.0737e+09
n = 33	1.1642e-10	1.1642e-10	0.5000	2.1475e+09
n = 34	5.8208e-11	5.8208e-11	0.5000	4.2950e+09
n = 35	2.9104e-11	2.9104e-11	0.5000	8.5899e+09
n = 36	1.4552e-11	1.4552e-11	0.5000	1.7180e+10
n = 37	7.2760e-12	7.2760e-12	0.5000	3.4360e+10
n = 38	3.6380e-12	3.6380e-12	0.5000	6.8719e+10

n = 39	1.8190e-12	1.8190e-12	0.5000	1.3744e+11
n = 40	9.0949e-13	9.0949e-13	0.5000	2.7488e+11
n = 41	4.5475e-13	4.5475e-13	0.5000	5.4976e+11
n = 42	2.2737e-13	2.2737e-13	0.5000	1.0995e+12
n = 43	1.1369e-13	1.1369e-13	0.5000	2.1990e+12
n = 44	5.6843e-14	5.6843e-14	0.5000	4.3980e+12
n = 45	2.8422e-14	2.8422e-14	0.5000	8.7961e+12
n = 46	1.4211e-14	1.4211e-14	0.5000	1.7592e+13
n = 47	7.1054e-15	7.1054e-15	0.5000	3.5184e+13
n = 48	3.5527e-15	3.5527e-15	0.5000	7.0369e+13
n = 49	1.7764e-15	1.7764e-15	0.5000	1.4074e+14
n = 50	8.8818e-16	8.8818e-16	0.5000	2.8147e+14

$f(x) = \sin(x) + \cos(x^2)$

	x[n]	Eabs(x[n])	Linear Err Ratio	Quad Err Ratio
n = 0	1	-1	-1	-1
n = 1	2.2093	1.2093	-1	-1
n = 2	1.9511	0.2582	0.2135	0.1765
n = 3	1.8815	0.0696	0.2695	1.0439
n = 4	1.8551	0.0264	0.3794	5.4522
n = 5	1.8496	0.0055	0.2081	7.8828
n = 6	1.8494	2.5852e-04	0.0471	8.5658
n = 7	1.8494	5.7379e-07	0.0022	8.5857
n = 8	1.8494	2.8271e-12	4.9270e-06	8.5867
n = 9	1.8494	0	0	0

Both of the functions x and $\sin(x) + \cos(x^2)$ converge quadratically as we can see from the ratio of the error of the current iteration over the square of the previous error. But, the function x^2 has a root at 0 and the multiplicity of its root is 2, meaning that as $f'(x)$ approaches the root, $f'(x)$ grows closer to 0, causing linear convergence.

We used a modified Newton's method to improve the convergence rate of x^2 and the modified method is given by: $x_{n+1} = x_n - m\,(f(x_n) / f'(x_n))$, where m is the multiplicity of the root of the function. (see Appendix A for code).

$f(x) = x^2$

	x[n]	Eabs(x[n])
n = 0	1	-1
n = 1	0	1
n = 2	0	0

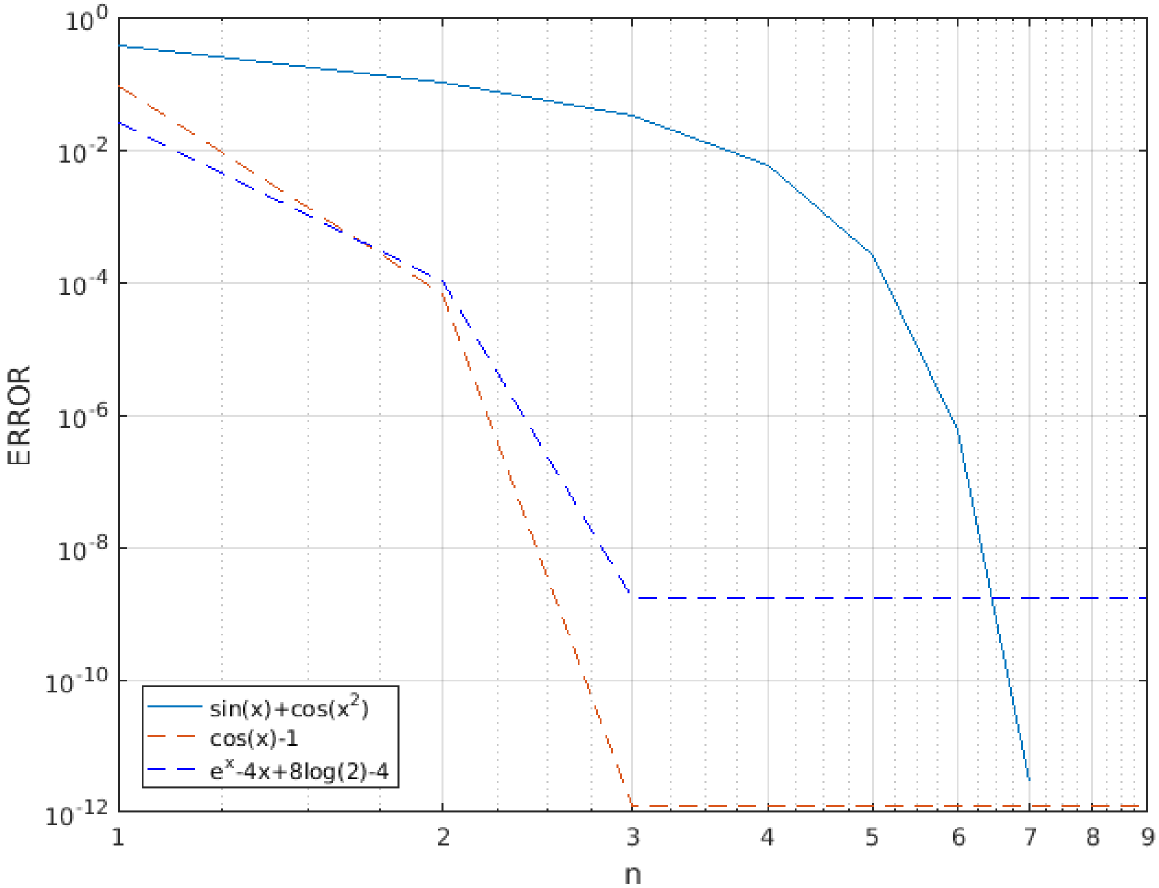
We compared two other functions $\cos(x) - 1$ and $e^x - 4x + 8\,log(2) - 4$, which both have roots at 0 with a multiplicity of 2, against the convergence of $\sin(x) + \cos(x^2)$ in order to test the performance of the modified Newton's method. We plotted the error of all three function in a logarithm plot (see next page) and it's clear that the modified Newton's method converges faster for these two functions than the normal Newton's method does for $\sin(x) + \cos(x^2)$.

$f(x) = \cos(x) - 1$

	x[n]	Eabs(x[n])	Linear Err Ratio	Quad Err Ratio
n = 0	1	-1	-1	-1
n = 1	-0.0926	1.0926	-1	-1
n = 2	6.6236e-05	0.0927	0.0848	0.0776
n = 3	1.2637e-12	6.6236e-05	7.1474e-04	0.0077
n = 4	1.2637e-12	0	0	0

$f(x) = e^x - 4x + 8\,log(2) - 4$

	x[n]	Eabs(x[n])	Linear Err Ratio	Quad Err Ratio
n = 0	1	-1	-1	-1
n = 1	1.4111	0.4111	-1	-1
n = 2	1.3864	0.0247	0.0601	0.1462
n = 3	1.3863	1.0258e-04	0.0042	0.1680
n = 4	1.3863	0	0	0



Appendix A: Code

```
program newton

  implicit none
  double precision :: f,fp,x,xprev,dx,ecurr,eprev,r1,r2,m
  integer :: iter
  character (len=50) :: fst
  fst = 'FFFF'
  if (fst.eq.'x*x' .OR. fst.eq.'cos(x)-1.d0' .OR. fst.eq.'exp(x)-4.d0*x+8.d0*log(2.d0)-4.d0') then
    m = 2.d0
  else
    m = 1.d0
  endif
  x = GGGG
  f = ffun(x)
  xprev = x + 1
  ecurr = 1.d0
  iter = 0
  do while (ecurr > 10.d0**(-15) .AND. (fst /= 'x*x' .OR. ffun(x) /= 0))
     iter = iter + 1
     f = ffun(x)
     fp = fpfun(x)
     dx = -f/fp
     xprev = x
     x = x + (m*dx)
     eprev = ecurr
     ecurr = abs(x - xprev)
     r1 = ecurr/eprev
     r2 = r1/eprev
     write(*,'(A18,I2.2,2(E24.16))') ' FFFF ', iter, x, dx, r1, r2
  end do

contains

  double precision function ffun(x)
    implicit none
    double precision :: x

    ffun = FFFF
  end function ffun

  double precision function fpfun(x)
    implicit none
    double precision :: x

    fpfun = FPFP
  end function fpfun

end program newton
```