Numerical Analysis HW 4

Michael Vu

August 31, 2023

Problem 1. To find the real root of the function $f(x) = x^9 + 9x^8 + 36x^7 + 84x^6 + 126x^5 + 126x^4 + 84x^3 + 36x^2 + 9x + 1$, I decided to embark on a few different methods. Those methods are modified Newton's, Secant, and Bisection methods. All three produced some very interesting results. All three methods require us to choose a number or interval close to the actual value of x. Keep in mind, that because this function has a multiplicity of up to 9, the margin of error is $\sqrt[9]{10^{-15}} = 0.0215$. By glancing at $f(x)$, I can see that $x = -1$ is a root, but lets choose numbers close to -1 and see what these codes can do numerically.

Newton's Method:

```
program hw4pr1newtonmod
implicit none
double precision :: x(101), f, fprime, m
integer :: i

x(1) = -1.5d0
m = 9.0d0

do i = 1, 100
x(i+1) = x(i) - m * (f(x(i)) / fprime(x(i)))
print*, i, x(i), f(x(i))

if(f(x(i)) == 0.0d0)then
print*, 'The root of f(x) is', x(i)
end if
```

```
if(f(x(i)) == 0.0d0)exit
if(abs(f(x(i))) < 1.0d0*10.0d0**(-12.0d0))exit
end do

end program

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

f = x**9 + 9.0d0*x**8 + 36.0d0*x**7 + 84.0d0*x**6 + 126.0d0*x**5 &
+ 126.0d0*x**4 + 84.0d0*x**3 + 36.0d0*x**2 + 9.0d0*x + 1

return
end

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

fprime = 9.0d0*x**8 + 72.0d0*x**7 + 252.0d0*x**6 + 504.0d0*x**5 &
+ 625.0d0*x**4 + 500.0d0*x**3 + 252.0d0*x**2 + 72.0d0*x + 9

return
end
```

Here are the first and last 5 lines of the output where $x_1 = -1.5$:

```
i          x(i)                          f(x(i))
1  -1.5000000000000000          -1.9531250000000000E-003
2  -1.5014925373134329          -2.0062279299476182E-003
3  -1.5030179212599786          -2.0618217817905682E-003
4  -1.5045775067422211          -2.1200737956694748E-003
5  -1.5061727314691555          -2.1811659348024981E-003
.
.
.
56  -1.9520253273160899          -0.64244583967646107
```

```
57  -2.1092714675687212          -2.5429661590575900
58  -2.6702176076812827          -101.14805124786859
59 -0.18977864720668991           0.15046419315218215
60 -0.99001505300024639           8.1601392309949006E-015
```

Secant Method:

```
program hw4pr1secant
implicit none
double precision :: p(101), f
integer :: i

p(1) = -1.5d0
p(2) = -0.5d0

do i = 2, 100
p(i+1) = p(i) - (f(p(i))*(p(i)-p(i-1)))/( f(p(i))-f(p(i-1)))
print*, i, p(i), abs(p(i+1)-p(i)), f(p(i))

if(f(p(i)) == 0.0d0)then
print*, 'The root of f(x) is', p(i)
end if

if(f(p(i)) == 0.0d0)exit
end do

end program

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

f = x**9 + 9.0d0*x**8 + 36.0d0*x**7 + 84.0d0*x**6 + 126.0d0*x**5
         + 126.0d0*x**4 + 84.0d0*x**3 + 36.0d0*x**2 + 9.0d0*x + 1

return
end
```

Here are the two lines of the output where $[p_1, p_2] = [-1.5, -0.5]$:

```
i          p(i)                          f(p(i))
2  -0.50000000000000000        1.9531250000000000E-003
3  -1.0000000000000000         0.0000000000000000

The root of f(x) is  -1.0000000000000000
```

Here are the first and lase 5 lines of the output where $[p_1, p_2] = [-2.0, -0.5]$:

```
i          p(i)                          f(p(i))
2  -0.50000000000000000        1.9531250000000000E-003
3  -0.50292397660818711        1.8527010084193277E-003
4  -0.55686780342999798        6.5887002604606611E-004
5  -0.58663916228052448        3.5234162919161281E-004
6  -0.62086009968236455        1.6188083060478947E-004
.
.
.
96  -0.98556636976339351       -1.3322676295501878E-015
97  -0.98556584170596373       -7.9103390504542404E-016
98  -0.98556506992972015       -1.0554057627842894E-014
99  -0.98556590423794077       -9.7283292532779342E-015
100 -0.98557573365075846        6.1062266354383610E-016
```

Bisection Method:

```
program hw4pr1bis
double precision :: a, b, m(101), f
integer :: i

a = -1.3d0
b = -0.5d0

do i = 1, 100
m(i) = (a + b)/2.0d0

if(f(a)*f(m(i)) < 0.0d0)then
```

```
b = m(i)
else
a = m(i)
end if
print*, i, m(i), f(m(i))

if(f(m(i)) == 0.0d0)then
print*, 'The root of f(x) is', m(i)
end if

if(f(m(i)) == 0.0d0)exit
if(abs(f(m(i))) < 1.0d0*10.0d0**(-15.0d0))exit
end do
end program

double precision function f(x)
double precision :: x

f = x**9 + 9.0d0*x**8 + 36.0d0*x**7 + 84.0d0*x**6 + 126.0d0*x**5
         + 126.0d0*x**4 + 84.0d0*x**3 + 36.0d0*x**2 + 9.0d0*x + 1

return
end
```

Here are the three lines of the output where $[a, b] = [-1.3, -0.5]$:

```
i        m(i)                        f(m(i))
1  -0.90000000000000002       9.9999565225661335E-010
2  -1.1000000000000001       -9.9999092340041784E-010
3  -1.0000000000000000        0.0000000000000000
The root of f(x) is  -1.0000000000000000
```

Here are the eleven lines of the output where $[a, b] = [-1.8, -0.3]$:

```
i          m(i)                         f(m(i))
1   -1.0500000000000000       -1.9701670850302833E-012
2   -0.67500000000000004       4.0452954760855295E-005
3   -0.86250000000000004       1.7568076382534770E-008
```

```
4   -0.95625000000000004        5.9280358399860233E-013
5   -1.0031250000000000         5.5441762292218755E-015
6   -1.0265625000000000        -6.2796989830360417E-015
7   -1.0148437500000000         4.6004866582904924E-015
8   -1.0207031250000000        -9.8046570862209137E-015
9   -1.0177734375000000         2.3661628212323649E-015
10  -1.0192382812500000         9.5201624361607173E-015
11  -1.0199707031250000         8.3266726846886741E-016
```

So we find that regardless of the method we use, the final output or number the compiler converges to is -1.0 ± 0.0215, which is within the tolerance of the actual root of -1.

Problem 2. In this scenario, we take a look at the compiler's "intuition" on which precision to use in the following code:

```
program hw4pr2
double precision :: x, y1, y2, y3
integer :: i

do i = 1, 10
x = dble(i)

y1 = 3.0 * x**2 + 5.7 * x - 4.5
y2 = 3.0d0 * x**2 + 5.7d0 * x - 4.5d0
y3 = 3.0d0 * x**2.0d0 + 5.7d0 * x -4.5d0

print*, 'y1 = ', y1
print*, 'y2 = ', y2
print*, 'y3 = ', y3

print*, 'y1-y2 = ', y1-y2
print*, 'y1-y3 = ', y1-y3
print*, 'y2-y3 = ', y2-y3

print*, '-------------------------------------------'
print*
end do
```

```
stop
end program
```

Here we notice the following output where $x = 1$:

```
y1    =    18.899999618530273
y2    =    18.899999999999999
y3    =    18.899999999999999
y1-y2 =   -3.8146972514141453E-007
y1-y3 =   -3.8146972514141453E-007
y2-y3 =    0.0000000000000000
```

We see here the code writer needs to be careful about how they express their numbers. These coefficients seemed to be handled initially as single precision until the compiler realizes it is supposed to be double precision creating a margin of error. The added insurance does not seem to matter for exponents and I suppose this is because the exponents act as a counting process, rather than a calculation. So at least exponents appear to be acting as it ought to.

Problem 3. Next we took a look at a similar scenario, only this time we replace the power of 2 with cube root. Here is the code:

```
program hw4pr3
double precision :: x, y1, y2, y3
integer :: i

do i = 1, 10
x = dble(i)

y1 = 3.0 * x**(1.0/3.0) + 5.7 * x - 4.5
y2 = 3.0d0 * x**(1.0/3.0) + 5.7d0 * x - 4.5d0
y3 = 3.0d0 * x**(1.0d0/3.0d0) + 5.7d0 * x - 4.5d0

print*, 'y1 = ', y1
print*, 'y2 = ', y2
print*, 'y3 = ', y3
```

```
print*, 'y1-y2 = ', y1-y2
print*, 'y1-y3 = ', y1-y3
print*, 'y2-y3 = ', y2-y3

print*, '-------------------------------------------------------'
print*
end do

stop
end program
```

Here we notice the following output $x = 2$:

```
y1    =    10.679762794241581
y2    =    10.679763175711308
y3    =    10.679763149684620
y1-y2 =   -3.8146972691777137E-007
y1-y3 =   -3.5544303855772341E-007
y2-y3 =    2.6026688360047956E-008
```

In the previous problem, we see that exponents are not affected by adding "insurance" to its numbers. However, in this case since the exponent is no longer a whole number, the compiler must compute the fraction before applying its value as an exponent. Therefore, the added insurance is a must. Again, based off of Dr. Glunt's paranoia, the exponents acted just as suspected.