Numerical Analysis HW 3

Michael Vu

August 31, 2023

Problem 1. Reflecting upon bisection, one of its biggest pitfalls in machine computation, is its inability to distinguish or compare itself to another number when both numbers are extremely small. In theory, the program should run for an infinite amount of time, but this is not the case. We find that computations end after about 24 iterations because the machine could not differentiate between such small numbers and it assumes the numbers are the same.

Problem 2. As we look to solve $f(x) = x^3 - 9$, using bisection on the interval $[2, 3]$, we find that trying to obtain x when the successive midpoints differ by less than 10^{-9} to be quite difficult with single precision units. Below is a code for a single precision Fortran program using bisection to solve for $f(x)$ (note: because of the cons of bisection as noted in problem 1, there is also a check within the loop for tolerance):

```
program hw3pr2pt1
real :: a, b, m(30), f
integer :: i

a = 2.0
b = 3.0

do i = 1, 30
m(i) = (a + b)/2.0
if(f(a)*f(m(i)) < 0.0)then
b = m(i)
else
```

```
a = m(i)
end if
print*, i, m(i), abs(m(i)-m(i-1))

if(f(a) == f(m(i)))then
print*, 'You have met the tolerance of single precision'
end if
end do

end program

real function f(x)
real :: x

f = x**3 - 9

return
end
```

Here are a few lines from the beginning and end of the output:

```
1   2.50000000       2.50000000
2   2.25000000      0.250000000
3   2.12500000      0.125000000
4   2.06250000       6.25000000E-02
You have met the tolerance of single precision
5   2.09375000       3.12500000E-02
.
.
.
19   2.08008385       1.90734863E-06
20   2.08008289       9.53674316E-07
You have met the tolerance of single precision
21   2.08008337       4.76837158E-07
You have met the tolerance of single precision
22   2.08008361       2.38418579E-07
You have met the tolerance of single precision
23   2.08008385       2.38418579E-07
24   2.08008385       0.00000000
```

We find $x = 2.08008385$ and differences in successive midpoints with precision of about 10^{-7} which is less than the 10^{-9} precision prescribed in the assignment. The most probably cause of this "lack" of precision is because of the single precision units. Also observe the warning that is given about tolerance. The program found instances where $f(a)f(m(i)) = 0$, cause by multiply two extremely small values that it considered to be 0 (although it did not affect finding the correct answer). Next we will look at the same program with double precision units instead (again, syntax has been implemented to catch tolerance issues with bisection):

```
program hw3pr2pt2
double precision :: a, b, m(30), f
integer :: i

a = 2.0d0
b = 3.0d0

do i = 1, 30
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), abs(m(i)-m(i-1))

if(f(a) == f(m(i)))then
print*, 'You have met the tolerance of double precision'
end if
end do

end program

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

f = x**3.0d0 - 9.0d0
```

```
return
end
```

After running the program, the first and last few output lines are as follows:

```
1    2.5000000000000000         2.5000000000000000
2    2.2500000000000000        0.25000000000000000
3    2.1250000000000000        0.12500000000000000
4    2.0625000000000000         6.2500000000000000E-002
You have met the tolerance of double precision
5    2.0937500000000000         3.1250000000000000E-002
.
.
.
26    2.0800838321447372         1.4901161193847656E-008
27    2.0800838246941566         7.4505805969238281E-009
28    2.0800838209688663         3.7252902984619141E-009
You have met the tolerance of double precision
29    2.0800838228315115         1.8626451492309570E-009
You have met the tolerance of double precision
30    2.0800838237628341         9.3132257461547852E-010
```

Now we find $x = 2.0800838237628341$ and successive midpoints to differ with precision greater than 10^{-9}. This shows how important choosing the correct precision based on allowable tolerances. We also find there are instances when tolerances of bisection are met when choosing new end points for subintervals. We know to set the midpoint as the new right limit when $f(a)f(m(i)) < 0$ or as the left limit when $f(a)f(m(i)) > 0$. The problem with this, as mentioned earlier, is when the two resulting values of the function as so small, whether negative or positive, the machine considers them both 0, and the resulting product is 0. This may become problematic when deciding which side limit to assign the midpoint. In our case, it did not affect the final result and the correct x is achieved. The book mentions another method to compare the two functions by using the signum function.

Problem 3. The textbook says $m = a + \frac{b-a}{2}$ is equivalent to $m = \frac{a+b}{2}$. Simple check by adding the terms verifies this. Using algebra we find:

$$\begin{aligned} m &= a + \frac{b-a}{2} \\ &= \frac{2}{2}(a) + \frac{b-a}{2} \\ &= \frac{2a}{2} + \frac{b-a}{2} \\ &= \frac{2a+b-a}{2} \\ &= \frac{a+b}{2} \end{aligned}$$

Included is the code for the same function as in problem 2, except $m = a + \frac{b-a}{2}$ is used to compute the midpoint:

```
program hw3pr3
double precision :: a, b, m(30), f
integer :: i

a = 2.0d0
b = 3.0d0

do i = 1, 30
m(i) = a + (b - a)/2.0d0
if(f(a)*f(m(i)) < 0.0d0)then
b = m(i)
else
a = m(i)
end if
print*, i, m(i), abs(m(i)-m(i-1))
if(f(a) == f(m(i)))then
print*, 'You have met the tolerance of double precision'
end if
end do

end program
```

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

f = x**3.0d0 - 9.0d0

return
end
```

After running the program and comparing the output to the previous problem, we find the results, including tolerance issues, to be exactly the same. Here are the first and last few lines of the output:

```
1   2.5000000000000000         2.5000000000000000
2   2.2500000000000000        0.25000000000000000
3   2.1250000000000000        0.12500000000000000
4   2.0625000000000000         6.2500000000000000E-002
You have met the tolerance of double precision
5   2.0937500000000000         3.1250000000000000E-002
.
.
.
26   2.0800838321447372         1.4901161193847656E-008
27   2.0800838246941566         7.4505805969238281E-009
28   2.0800838209688663         3.7252902984619141E-009
You have met the tolerance of double precision
29   2.0800838228315115         1.8626451492309570E-009
You have met the tolerance of double precision
30   2.0800838237628341         9.3132257461547852E-010
```

In conclusion, we observe the behavior not to change at all.

Problem 4. On page 44 of our textbook, problem 17 asks us to find a solution for the given equation:

$$x(t) = \frac{g}{2\omega^2} - \left(\frac{e^{\omega t} - e^{-\omega t}}{2} - \sin \omega t \right)$$

We are given $g = -32.17$ ft/s^2 and $t = 1$ second moves the particle 1.7 ft. Plugging in these values and setting the resulting equation to 0, we find:

$$x(w) = -32.17\left(\frac{e^{\omega} - e^{-\omega}}{2} - \sin\omega\right) - 3.4\omega^2$$

Now that the function only depends on ω, we can implement the secant method to find ω such that the function equals 0. This is the code used to find ω by establishing an interval of $[-1, 1]$:

```
program hw3pr4
implicit none
double precision :: p(30), f
integer :: i

p(1) = -1.0d0
p(2) = 1.0d0

do i = 2, 30
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))
end do

end program

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

f = -32.17d0*((dexp(w)-dexp(-w))/2.0d0 - sin(w))-3.4d0*w**2

return
end
```

The resulting output of p_{14} gives us $\omega = -0.31706177453108753$ and successive p_i with differences of 10^{-17} which is greater than the prescribed 10^{-10}. We find that starting on the interval $[-1, 1]$ requires at least p_7 to meet the required difference precision.