Rootfinding

Matthew Hefner

November 7, 2018

Part 1

Problem 1 (8, page 123).

Maximum error of a root estimation by the bisection method is given by $\frac{b-a}{2}$. So we want $\frac{b-a}{2} = 10^{-8}$ on some interval $[a, b]$ found after n iterations of the method. The interval is divided by two at each iteration, so we want to find n where

$$\frac{b-a}{2^n} = 10^{-8}$$

thus

$$\frac{0.9}{2^n} = 10^{-8} \implies 2^n = \frac{0.9}{10^{-8}} \implies n = \log_2(\frac{0.9}{10^{-8}}) \approx 26.42342167$$

So we would need 27 steps of the bisection method.

Problem 2 (6, page 136).

Calculator results:

```
## Five steps, m = 8

## X1 = 1.0875000000000001
## X2 = 1.0765625
## X3 = 1.0669921875000001
## X4 = 1.0586181640625
## X5 = 1.0512908935546876

## Five steps, m = 12

## X0 = 1.0916666666666668
## X1 = 1.0840277777777778
## X2 = 1.077025462962963
## X3 = 1.070606674382716
## X4 = 1.0647227848508232
```

The actual root is clearly 1. This 'sluggishness' occurs because $f'(r) = 0$, meaning r is not a **simple zero**, and thus the sequence does not converge quadratically.

Problem 3 (13, parts a and b)

Part a

$$x_{n+1} = x_n - \frac{(x_n)^3 - R}{3(x_n)^2}$$

Letting $R = 10$ and $x_0 = \frac{10}{3}$:

$$x_1 = \frac{10}{3} - \frac{(\frac{10}{3})^3 - (10)^{\frac{1}{3}}}{3(\frac{10}{3})^2} = \frac{20}{9} + \frac{3(10)^{\frac{1}{3}}}{100} \approx 2.28685526$$

$$x_2 \approx 2.28685526 - \frac{(2.28685526)^3 - (10)^{\frac{1}{3}}}{3(2.28685526)^2} \approx 1.66189047$$

Part b

$$x_{n+1} = x_n - \frac{\frac{1}{(x_n)^3} - \frac{1}{R}}{-\frac{3}{(x_n)^4}}$$

Clearly, neither R nor x_n can be zero.

$$x_1 = \frac{10}{3} - \frac{\frac{1}{(\frac{10}{3})^3} - \frac{1}{(10)^{\frac{1}{3}}}}{-\frac{3}{(\frac{10}{3})^4}} \approx -14.65674418$$

$$x_2 \approx -14.65674418 - \frac{\frac{1}{(-14.65674418)^3} - \frac{1}{(10)^{\frac{1}{3}}}}{-\frac{3}{(-14.65674418)^4}} \approx -7159.496803$$

It's running away!

Problem 4 (2, page 149)

$$x_2 = 1 - \frac{1-0}{((1)^3 - 2(1) + 2) - ((0)^3 - 2(0) + 2)} \left((1)^3 - 2(1) + 2\right) = 1 - (-1)(1) = 2$$

Part 2

Problem 5

```
import math
def secant(a, b, nmax, eps):
  fa = f(a)
  fb = f(b)
  if abs(fa) > abs(fb):
    temp = fa
    fa = fb
    fb = temp
    temp = a
    a = b
    b = temp
  print("0", a, fa)
  print("1", b, fb)
  for n in range(2, nmax):
    if abs(fa) > abs(fb):
      temp = fa
      fa = fb
      fb = temp
      temp = a
      a = b
      b = temp
    d = (b-a)/(fb-fa)
    b = a
    fb = fa
```

```
    d = d * fa
    if abs(d) < eps:
      print("Convergence")
      return
    a = a - d
    fa = f(a)
    print(n, a, fa)
def f(x):
  return((x**3.0) - (2.0 * math.sin(x)))
secant(-2.0, 3.0, 100, 10**-10)
```

```
## 0 -2.0 -6.181405146348636
## 1 3.0 26.717759983880267
## 2 -1.060552886086926 0.552370638480032
## 3 -1.1376155948423217 0.3430032875673885
## 4 -1.2638662367725322 -0.11231550610814733
## 5 -1.2327234310244901 0.013535932293617714
## 6 -1.2360729907545351 0.000435665329169721
## 7 -1.2361843844418794 -1.792370165620838e-06
## 8 -1.2361839280349578 2.355919903607173e-10
## Convergence
```

```
def f(x):
  return(2.0 - ((x**2.0) * math.exp(-.385 * x)))
secant(1.0, 12.0, 100, 10**-10)
```

```
## 0 12.0 0.5811973671890349
## 1 1.0 1.3195493637954123
## 2 20.65870352956822 1.8500311151602353
## 3 8.033825784739705 -0.9279390358254522
## 4 10.47255025647924 0.054374932882473415
## 5 10.31489756009281 -0.0055992020355266625
## 6 10.32961605998814 3.826707949072805e-05
## 7 10.329516150969752 2.6064866265329556e-08
## 8 10.329516082872312 -1.2256862191861728e-13
## Convergence
```

```
secant(1.0, 4.0, 100, 10**-10)
```

```
## 0 1.0 1.3195493637954123
## 1 4.0 -1.430097622831647
## 2 2.4396932081242317 -0.3267273488211906
## 3 2.1539653341882223 -0.024543648029666354
## 4 2.1307582440911017 0.0010663266429091767
## 5 2.1317245214416785 -2.903867323489351e-06
## 6 2.1317218971794665 -3.389466485259618e-10
## 7 2.1317218968731204 2.220446049250313e-16
## Convergence
```

(a)

Maple gives the zeros:

$\{x = 1.236183928\}$, $\{x = -1.236183928\}$, $\{x = 0.\}$

The secant code above gives:

-1.2361839280349578

(b)

Maple gives the zeros:

{x = -1.136354516}, {x = 2.131721897}, {x = 10.32951608}

Only the last two of these are within the interval.

The secant code above gives:

10.329516082872312

(c)

Maple gives the zeros:

{x = -1.136354516}, {x = 2.131721897}, {x = 10.32951608}

Only the second of these is within our interval.

The secant code above gives:

2.1317218968731204

Problem 6

```
import math
def f(x):
  return(math.exp(0.2 * x) - x ** 2)
def bisect(a, b, nmax, eps):
  fa = f(a)
  fb = f(b)
  print("Begin")
  if fa * fb > 0:
    print("\ta =", a)
    print("\tb =", b)
    print("\tfa =", fa)
    print("\tfb =", fb)
    print("\t\tThe function has the same signs at a and b.  End")
    return
  error =  b - a
  for n in range(0, nmax + 1):
    error = error / 2.0
    c = a + error
    fc = f(c)
    print("\tn =", n)
    print("\t\tc = ", c)
    print("\t\tfc =", fc)
    print("\t\tError =", error)
    if abs(error) < eps:
      print("\t\t\tConvergence!  End")
      return
    if fa * fc < 0:
      b = c
      fb = fc
    else:
      a = c
      fa = fc
```

```
# Secant was defined above and is still in the runtime environment
secant(0, 2, 100, 10**-10)
```

```
## 0 0 1.0
## 1 2 -2.5081753023587297
## 2 0.5700969386151529 0.7957633345082045
## 3 2.7913545163028255 -6.044012001948986
## 4 0.8285257977329032 0.4937700748148198
## 5 1.2510664770528246 -0.2808680073787946
## 6 1.0978618239831013 0.040243393671717254
## 7 1.117062248339426 0.00250810147246483
## 8 1.1183384174053965 -2.547890963855437e-05
## 9 1.1183255836320751 1.5807847253412888e-08
## 10 1.1183255915895796 9.947598300641403e-14
## Convergence
```

```
bisect(0, 2, 100, 10**-10)
```

```
## Begin
##  n = 0
##      c =  1.0
##      fc = 0.22140275816016985
##      Error = 1.0
##  n = 1
##      c =  1.5
##      fc = -0.9001411924239968
##      Error = 0.5
##  n = 2
##      c =  1.25
##      fc = -0.2784745833122586
##      Error = 0.25
##  n = 3
##      c =  1.125
##      fc = -0.013302283808135584
##      Error = 0.125
##  n = 4
##      c =  1.0625
##      fc = 0.10785986356528476
##      Error = 0.0625
##  n = 5
##      c =  1.09375
##      fc = 0.04823104526609523
##      Error = 0.03125
##  n = 6
##      c =  1.109375
##      fc = 0.017702425571107794
##      Error = 0.015625
##  n = 7
##      c =  1.1171875
##      fc = 0.0022595797099234094
##      Error = 0.0078125
##  n = 8
##      c =  1.12109375
##      fc = -0.00550647514016589
##      Error = 0.00390625
```

```
##  n = 9
##      c =  1.119140625
##      fc = -0.0016197284505969911
##      Error = 0.001953125
##  n = 10
##      c =  1.1181640625
##      fc = 0.0003208554504539052
##      Error = 0.0009765625
##  n = 11
##      c =  1.11865234375
##      fc = -0.0006492040454562087
##      Error = 0.00048828125
##  n = 12
##      c =  1.118408203125
##      fc = -0.0001641161837746541
##      Error = 0.000244140625
##  n = 13
##      c =  1.1182861328125
##      fc = 7.83841617804093e-05
##      Error = 0.0001220703125
##  n = 14
##      c =  1.11834716796875
##      fc = -4.286237888795341e-05
##      Error = 6.103515625e-05
##  n = 15
##      c =  1.118316650390625
##      fc = 1.7761799473658968e-05
##      Error = 3.0517578125e-05
##  n = 16
##      c =  1.1183319091796875
##      fc = -1.2550062700400488e-05
##      Error = 1.52587890625e-05
##  n = 17
##      c =  1.1183242797851562
##      fc = 2.6059251383436788e-06
##      Error = 7.62939453125e-06
##  n = 18
##      c =  1.1183280944824219
##      fc = -4.972054593155306e-06
##      Error = 3.814697265625e-06
##  n = 19
##      c =  1.118326187133789
##      fc = -1.1830611803542723e-06
##      Error = 1.9073486328125e-06
##  n = 20
##      c =  1.1183252334594727
##      fc = 7.11432865729833e-07
##      Error = 9.5367431640625e-07
##  n = 21
##      c =  1.1183257102966309
##      fc = -2.3581393571170395e-07
##      Error = 4.76837158203125e-07
##  n = 22
##      c =  1.1183254718780518
```

```
##      fc = 2.3780952052021576e-07
##      Error = 2.384185791015625e-07
##  n = 23
##      c =  1.1183255910873413
##      fc = 9.978062820437117e-10
##      Error = 1.1920928955078125e-07
##  n = 24
##      c =  1.118325650691986
##      fc = -1.1740806127313874e-07
##      Error = 5.960464477539063e-08
##  n = 25
##      c =  1.1183256208896637
##      fc = -5.82051267183914e-08
##      Error = 2.9802322387695312e-08
##  n = 26
##      c =  1.1183256059885025
##      fc = -2.8603659885106936e-08
##      Error = 1.4901161193847656e-08
##  n = 27
##      c =  1.118325598537922
##      fc = -1.3802926801531612e-08
##      Error = 7.450580596923828e-09
##  n = 28
##      c =  1.1183255948126316
##      fc = -6.40256025974395e-09
##      Error = 3.725290298461914e-09
##  n = 29
##      c =  1.1183255929499865
##      fc = -2.7023769888501192e-09
##      Error = 1.862645149230957e-09
##  n = 30
##      c =  1.1183255920186639
##      fc = -8.522853534032038e-10
##      Error = 9.313225746154785e-10
##  n = 31
##      c =  1.1183255915530026
##      fc = 7.276046432025396e-11
##      Error = 4.656612873077393e-10
##  n = 32
##      c =  1.1183255917858332
##      fc = -3.897624445414749e-10
##      Error = 2.3283064365386963e-10
##  n = 33
##      c =  1.118325591669418
##      fc = -1.5850099011061047e-10
##      Error = 1.1641532182693481e-10
##  n = 34
##      c =  1.1183255916112103
##      fc = -4.287037391748072e-11
##      Error = 5.820766091346741e-11
##          Convergence!  End
```

The secant method converges to 1.1183255916 (1.11832559158) in 10 steps, and the bisect method converges to 1.1183255916 (1.1183255916112103) in 34 steps. Both take signifigantly longer than Newton's method's 4

step convergence.