APPM 4600 - HW #4 Gustav Cedergrund

1) Assume temp. in C '$T(x,t)$' at distance 'x' meters below surface 't' seconds after cold snap satisfies

$$\frac{T(x,t)-T_s}{T_i-T_s} = \operatorname{erf}\left(\frac{x}{2\sqrt{\alpha t}}\right)$$

Let $T_i = 20°C$, $T_s = -15°C$, $\alpha = 0.138 \cdot 10^{-6}\ m^2/s$

a) How deep should water main be buried so only freeze after 60 days?

Then, we want $T(x,t) = 0°C$ at $t = 60 \cdot 24 \cdot 60^2 = 5184000\,s$

$\therefore$ $\boxed{f(x) = T(x,t) = (T_i - T_s)\operatorname{erf}\left(\frac{x}{2\sqrt{\alpha t}}\right) + T_s = 0}$ at $t = 5184000$

st $f(x) = 0$ is simply a root-finding problem

$\therefore$ $f'(x) = (T_i - T_s) \cdot \frac{2}{\sqrt{\pi}} \cdot e^{-\left(\frac{x}{2\sqrt{\alpha t}}\right)^2} \cdot \frac{1}{2\sqrt{\alpha t}} + 0$

$\boxed{f'(x) = (T_i - T_s) \cdot \frac{1}{\sqrt{\pi \alpha t}} \cdot e^{\frac{-x^2}{4\alpha t}}}$

Plot for $f(x)$ attached

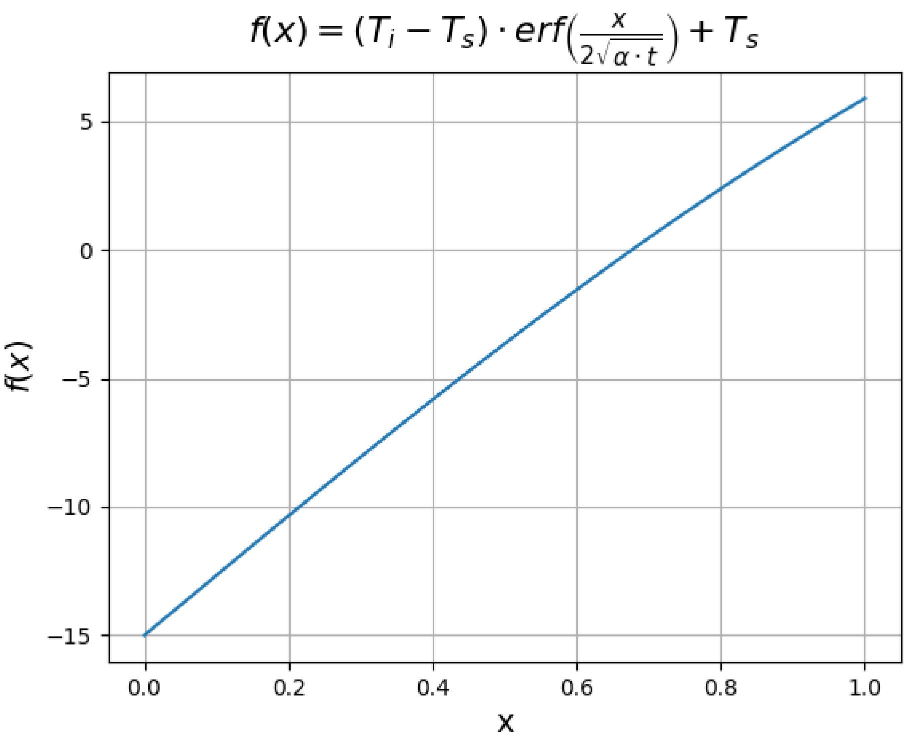
b) Using bisection w/ starting values $a_0 = 0$ and $b_0 = \bar{x}$ ($\bar{x}$ chosen at 1) we get root approximated at $x = 0.676961854481930\,9\,m$
(Full output attached)

c) Using newton's method w/ starting values $x_0 = 0.01m$ and $x_0 = \bar{x} = 1m$, we get an approximate depth of

$x = 0.6769618544819365$ (...9366 for $\bar{x}$)

Both converge after 4 iterations, way faster than that for bisection method, even though it produces a smaller error.
(Full output attached)

Problem 1:




```
(b):

bisection method:
iteration:  1  | curr_root =  0.75
iteration:  2  | curr_root =  0.625
iteration:  3  | curr_root =  0.6875
iteration:  4  | curr_root =  0.65625
iteration:  5  | curr_root =  0.671875
iteration:  6  | curr_root =  0.6796875
iteration:  7  | curr_root =  0.67578125
iteration:  8  | curr_root =  0.677734375
iteration:  9  | curr_root =  0.6767578125
iteration:  10  | curr_root =  0.67724609375
iteration:  11  | curr_root =  0.677001953125
iteration:  12  | curr_root =  0.6768798828125
iteration:  13  | curr_root =  0.67694091796875
iteration:  14  | curr_root =  0.676971435546875
iteration:  15  | curr_root =  0.6769561767578125
iteration:  16  | curr_root =  0.6769638061523438
iteration:  17  | curr_root =  0.6769599914550781
iteration:  18  | curr_root =  0.6769618988037109
iteration:  19  | curr_root =  0.6769609451293945
iteration:  20  | curr_root =  0.6769614219665527
iteration:  21  | curr_root =  0.6769616603851318
iteration:  22  | curr_root =  0.6769617795944214
iteration:  23  | curr_root =  0.6769618391990662
iteration:  24  | curr_root =  0.6769618690013885
iteration:  25  | curr_root =  0.6769618541002274
iteration:  26  | curr_root =  0.676961861550808
iteration:  27  | curr_root =  0.6769618578255177
iteration:  28  | curr_root =  0.6769618559628725
iteration:  29  | curr_root =  0.6769618550315499
iteration:  30  | curr_root =  0.6769618545658886
iteration:  31  | curr_root =  0.676961854333058
iteration:  32  | curr_root =  0.6769618544494733
iteration:  33  | curr_root =  0.676961854507681
iteration:  34  | curr_root =  0.6769618544785772
iteration:  35  | curr_root =  0.6769618544931291
iteration:  36  | curr_root =  0.6769618544858531
iteration:  37  | curr_root =  0.6769618544822151
iteration:  38  | curr_root =  0.6769618544803961
iteration:  39  | curr_root =  0.6769618544813056
iteration:  40  | curr_root =  0.6769618544817604
iteration:  41  | curr_root =  0.6769618544819878
iteration:  42  | curr_root =  0.6769618544818741
iteration:  43  | curr_root =  0.6769618544819309
Number of iterations:  43

the approximate root is 0.6769618544819309
f(root) = -1.1368683772161603e-13
```

```
(c):

newton's method:

initial guess at x=0.01 meters
Found solution after 4 iterations.
the approximate root is 0.6769618544819365
f(root) = -5.329070518200751e-15

initial guess at x=1 meters
Found solution after 4 iterations.
the approximate root is 0.6769618544819366
f(root) = 0.0
```

2)

a) root 'α' has multiplicity 'm' of function $f(x)$ given that $f(x)$ can be written $\underline{f(x) = (x-\alpha)^m g(x)}$

b) For $f(x)$ w/ multiplicity m at root α, note that

$$f(\alpha) = f'(\alpha) = f''(\alpha) = \cdots = f^{m-1}(\alpha) = 0 \; , \; f^{(m)}(\alpha) \neq 0$$

∴ $\underline{\text{Taylor for } f(x) \text{ at root } \alpha}$

$$\hookrightarrow f(x) = 0 + \cdots + 0 + \frac{f^m(x)}{m!}(x-\alpha)^m + \boxed{\frac{f^{m+1}(\xi)}{(m+1)!}(x-\alpha)^{m+1}}$$ ← error term

$$\text{s.t. } f'(x) = \frac{f^m(x)}{m!} \cdot m\,(x-\alpha)^{m-1} + \boxed{\frac{f^{m+1}(\xi)}{(m+1)!}(m+1)(x-\alpha)^m}$$ ← error term

∴ by Newton

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \Rightarrow x_{k+1} - \alpha = x_k - \alpha - \frac{f(x_k)}{f'(x_k)}$$ ← plug in taylor and cancel common terms

$$\approx x_k - \alpha - \frac{(x_k-\alpha)}{m}$$

$$x_{k+1} - \alpha \approx (x_k - \alpha)\left(1 - \frac{1}{m}\right)$$

$$∴ \frac{x_{k+1}-\alpha}{x_k-\alpha} \approx \frac{m-1}{m} < 1 \quad ∴ \underline{\text{linear convergence}}$$

c) If we instead use fixed point w/ $g(x) = x - m\frac{f(x)}{f'(x)}$

we have

$$\underline{x_{k+1} = g(x_k) = x_k - m\frac{f(x_k)}{f'(x_k)}} \Rightarrow x_{k+1} - \alpha = x_k - \alpha - m\frac{f(x_k)}{f'(x_k)}$$

plugging in the same Taylor expansions from (b) we have

$$x_{k+1} - \alpha = x_k - \alpha - m\left(\frac{f^m(x)/m!\,(x-\alpha)^m + f^{m+1}(\xi_k)/(m+1)!\,(x-\alpha)^{m+1}}{f^m(x)/m! \cdot m\,(x-\alpha)^{m-1} + f^{m+1}(\xi_k)/(m+1)!\,(m+1)\,(x-\alpha)^m}\right)$$

$$x_{k+1} - \alpha \approx x_k - \alpha - \cancel{m}\frac{(x_k-\alpha)}{\cancel{m}} \approx x_k - x_k + \alpha - \alpha \approx 0 \neq \lambda$$ ∴ not linear

∴ greater than linear guarantees second-order convergence.

d) part (c) provides a modified method of Newton's that keeps order of convergence as quadratic even if the original function has a multiplicity.

3) Let $\{x_k\}_{k=1}^{\infty}$ be a sequence that converges to α

By definition, $\{x_k\} \to \alpha$ w/ order p given $\exists \lambda$ w/ $0<\lambda<1$ s.t

$$\lim_{k\to\infty} \frac{|x_{k+1}-\alpha|}{|x_k-\alpha|^p} = \lambda$$

Then when k is sufficiently large,

$$|x_{k+1}-\alpha| = \lambda |x_k-\alpha|^p$$

$$\log(|x_{k+1}-\alpha|) = \log(\lambda |x_k-\alpha|^p) = \log(\lambda) + \log(|x_k-\alpha|^p)$$

$$\boxed{\log(|x_{k+1}-\alpha|) = \log(\lambda) + p\log(|x_k-\alpha|)}$$

relating this to slope equation $y=mx+b$,
we see that the relationship b/w $\log(|x_{k+1}-\alpha|)$ and $\log(|x_k-\alpha|)$ is linear w/ order of convergence 'p' being the slope and $\log(\lambda)$ as y-intercept when k is sufficiently large.

---

4) Using $f(x) = e^{3x} - 27x^6 + 27x^4 e^x - 9x^2 e^{2x}$ w/ the three methods we can see different orders of convergence. I modelled convergence using the different methods through coding (code + output attached) and saw that (i) converges linearly while (ii) and (iii) converge closer to quadratically, slightly faster for (ii).
I prefer method (iii) b/c you don't need to calculate $f''(x)$ and convergence is still very quick.

---

5)

a) Plotting error by iteration (output attached) we see that error does decrease as expected, just a lot slower w/ secant method than w/ Newton's

b) As was shown in (3), the slopes of these lines are the order of convergence 'p'. Newton's method has a steeper plot than secant method, showing faster order of convergence.

Problem 4:

```python
# function declarations
f = (
    lambda x: np.exp(3 * x)
    - 27 * x**6
    + 27 * x**4 * np.exp(x)
    - 9 * x**2 * np.exp(2 * x)
)
Df = (
    lambda x: 3 * np.exp(3 * x)
    + (-18 * x**2 - 18 * x) * np.exp(2 * x)
    + (27 * x**4 + 108 * x**3) * np.exp(x)
    - 162 * x**5
)
DDf = (
    lambda x: 9 * np.exp(3 * x)
    + (-36 * x**2 - 72 * x - 18) * np.exp(2 * x)
    + (27 * x**4 + 216 * x**3 + 324 * x**2) * np.exp(x)
    - 810 * x**4
)

tol = 1e-13
x0 = 3
m = 3

print("(i): \nnewton's method with initial guess at x=3\n")
[astar1, iter1] = newton(f, Df, x0, tol, max_iter=1000)
print("the approximate root is", astar1)
print("f(root) =", f(astar1))
print("order of convergence evaluated with alpha = 1:")
print(orderOfConvergence(astar1, iter1, 1))
print("\n")
```

```python
print("(ii):\nmodified newton's method from class x0=3:\n")
mu = lambda x: f(x) / Df(x)
Dmu = lambda x: (Df(x) * Df(x) - f(x) * DDf(x)) / (Df(x) ** 2)
[astar2, iter2] = newton(mu, Dmu, x0, tol, max_iter=1000)
print("the approximate root is", astar2)
print("f(root) =", f(astar2))
print("order of convergence evaluated with alpha = 2:")
print(orderOfConvergence(astar2, iter2, 2))
print("\n")

print("(iii):\nmodified newton's (fixed point) method from (2)\nx0=3 and m=3:\n")
g = lambda x: x - m * f(x) / Df(x)
[astar3, _, iter3] = fixedpt(g, x0, tol, Nmax=1000)
print("Found solution after", len(iter3), "iterations.")
print("the approximate root is", astar3)
print("f(root) =", f(astar3))
print("order of convergence evaluated with alpha = 2:")
print(orderOfConvergence(astar3, iter3[:-1], 2))
print("\n")
```

```
(i):
newton's method with initial guess at x=3

Found solution after 39 iterations.
the approximate root is 3.7330596890832597
f(root) = 0.0
order of convergence evaluated with alpha = 1:
[ 0.50757867  0.74893505  0.73290845  0.71833404  0.70574058  0.69540162
  0.6873103   0.68124106  0.67685341  0.67378749  0.67172564  0.67042018
  0.66969935  0.66946391  0.66968278  0.67039345  0.67170752  0.67382575
  0.67704891  0.68190559  0.68889237  0.69855116  0.71340202  0.7266279
  0.73656509  0.80890729  0.94654482  0.72319919  0.57718158 45.44962427
  0.65812179  0.65390651  0.64732316  0.63656239  0.61948291  0.58897791
  0.53735942  0.47459956  0.        ]


(ii):
modified newton's method from class x0=3:

Found solution after 6 iterations.
the approximate root is 3.733078868957922
f(root) = 0.0
order of convergence evaluated with alpha = 2:
[1.10037411 1.07697453 1.02581073 0.96211941 0.92750855 0.        ]


(iii):
modified newton's (fixed point) method from (2)
x0=3 and m=3:

Found solution after 10 iterations.
the approximate root is 3.7330791332651536
f(root) = 0.0
order of convergence evaluated with alpha = 2:
[4.80523303 0.26108624 0.33601366 0.45114718 0.61916366 0.80643188
 0.90761802 0.        ]
```

Problem 5:

```
Note: using tolerance 10^-3
# of iterations too large otherwise


newton's method with initial guess at x=2:

Found solution after 6 iterations.
the approximate root is 1.134730528343629
f(root) = 6.573836771295305e-05


secant method with x0=2 and x1=1:

root found after 48 iterations.
the approximate root is 1.1346359946857905
f(root) = -0.0009065966333561271
```

```
table with errors:
    Error with newtons method:  Error with secant method:
0                     0.865269                   0.118507
1                     0.545898                   0.103961
2                     0.296008                   0.090919
3                     0.120240                   0.079289
4                     0.026808                   0.068969
5                     0.001623                   0.059853
6                     0.000000                   0.051833
7                     0.000000                   0.044805
8                     0.000000                   0.038665
9                     0.000000                   0.033317
10                    0.000000                   0.028671
11                    0.000000                   0.024645
12                    0.000000                   0.021162
13                    0.000000                   0.018154
14                    0.000000                   0.015561
15                    0.000000                   0.013328
16                    0.000000                   0.011407
17                    0.000000                   0.009757
18                    0.000000                   0.008341
19                    0.000000                   0.007125
20                    0.000000                   0.006084
21                    0.000000                   0.005191
22                    0.000000                   0.004426
23                    0.000000                   0.003772
24                    0.000000                   0.003212
25                    0.000000                   0.002732
26                    0.000000                   0.002322
27                    0.000000                   0.001972
28                    0.000000                   0.001672
29                    0.000000                   0.001416
30                    0.000000                   0.001197
31                    0.000000                   0.001010
32                    0.000000                   0.000850
33                    0.000000                   0.000713
34                    0.000000                   0.000596
35                    0.000000                   0.000497
36                    0.000000                   0.000411
37                    0.000000                   0.000338
38                    0.000000                   0.000276
39                    0.000000                   0.000223
40                    0.000000                   0.000178
41                    0.000000                   0.000139
42                    0.000000                   0.000106
43                    0.000000                   0.000077
44                    0.000000                   0.000053
45                    0.000000                   0.000033
46                    0.000000                   0.000015
47                    0.000000                   0.000000
```