APPM 4600 Homework 4

27 September 2024

1. The code used in this question is listed at the end of the question.

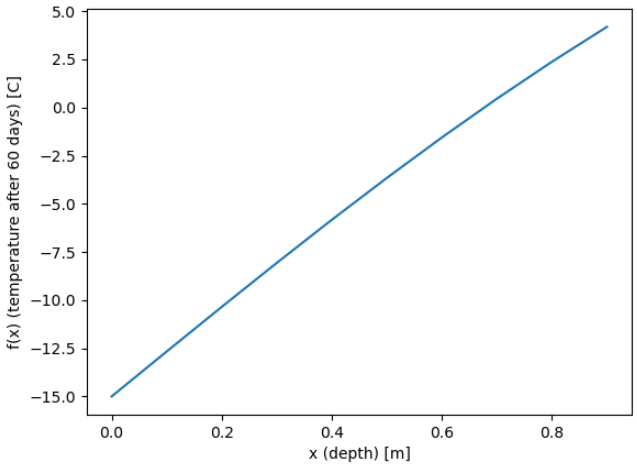
   (a) We are interested in the depth at which the temperature reaches 0 after 60 days. If $T(x,t)$ is the temperature at depth x and time t, we are interested in the root of the function

   $$f(x) = T(x, t = 60 \text{ days}) = (T_i - T_s)\text{erf}\left(\frac{x}{2\sqrt{\alpha(60 \text{ days})}}\right) + T_s.$$

   We have derivative

   $$f'(x) = (T_i - T_s)\frac{e^{-\frac{x^2}{4\alpha(60 \text{ days})}}}{\sqrt{\pi}\sqrt{\alpha(60 \text{ days})}}.$$

   This function is plotted over $x \in [0,1]$ below.



   (b) Bisection over $[0,1]$ yields an approximate root of

   $$x^* \approx 0.6769618544819309 \text{ m}.$$

   (c) Newton's Method starting at $x_0 = 0.01$ yields an approximate root

   $$x^* \approx 0.6769618544819365.$$

   As expected, this is within ϵ of the root we found via bisection.
   If we start at $x_0 = \bar{x}$, it is possible that the function's derivative will evaluate to zero and Newton will be unable to find the root. If we evaluate with $x_0 = 1$, Newton successfully finds the same root as above.

```
#! /usr/bin/env python3

import numpy as np
import scipy.special
import math
import matplotlib.pyplot as plt

Ti = 20
Ts = -15
a = 0.138e-6
```

```
t = 60 * 24 * 60 * 60

# function of interest
def f(x):
    return (Ti - Ts) * scipy.special.erf(x / (2*np.sqrt(a*t))) + Ts

# derivative of f
def df(x):
    return (Ti-Ts) * np.exp(-np.square(x) / (4*a*t)) / (np.sqrt(math.pi * a * t))

# perform bisection (provided)
def bisection(f,a,b,tol):

#    Inputs:
#     f,a,b       - function and endpoints of initial interval
#       tol  - bisection stops when relative interval length < tol

#    Returns:
#       astar - approximation of root
#       ier   - error message
#             - ier = 1 => Failed
#             - ier = 0 == success

#     first verify there is a root we can find in the interval

    fa = f(a)
    fb = f(b);
    if (fa*fb>0):
       ier = 1
       astar = a
       return [astar, ier]

#   verify end points are not a root
    if (fa == 0):
      astar = a
      ier =0
      return [astar, ier]

    if (fb ==0):
      astar = b
      ier = 0
      return [astar, ier]

    count = 0
    d = 0.5*(a+b)
    while (abs(d-a) > tol):
      fd = f(d)
      if (fd ==0):
        astar = d
        ier = 0
        return [astar, ier]
      if (fa*fd<0):
         b = d
      else:
        a = d
        fa = fd
      d = 0.5*(a+b)
      count = count +1
#       print('abs(d-a) = ', abs(d-a))
```

```
    astar = d
    ier = 0
    return [astar, ier]

#perform newton's method (provided)
def newton(f,fp,p0,tol,Nmax):
    """
    Newton iteration.
    Inputs:
    f,fp - function and derivative
    p0 - initial guess for root
    tol - iteration stops when p_n,p_{n+1} are within tol
    Nmax - max number of iterations
    Returns:
    p - an array of the iterates
    pstar - the last iterate
    info - success message
    - 0 if we met tol
    - 1 if we hit Nmax iterations (fail)
    """
    p = np.zeros(Nmax+1);
    p[0] = p0
    for it in range(Nmax):
        p1 = p0-f(p0)/fp(p0)
        p[it+1] = p1
        if (abs(p1-p0) < tol):
            pstar = p1
            info = 0
            return [p,pstar,info,it]
        p0 = p1
    pstar = p1
    info = 1
    return [p,pstar,info,it]

# plot f
t_sample = np.arange(0, 1, 0.1)
plt.plot(t_sample, f(t_sample))
plt.xlabel("x (depth) [m]")
plt.ylabel("f(x) (temperature after 60 days) [C]")
plt.savefig("hw4_1.png")

# find root via bisection
tol = 1e-13
[xstar, ier] = bisection(f, 0, 1, tol)
print("xstar=", xstar, ", ier=", ier)

# find root via Newton
[iters, xstar1, ier, it] = newton(f, df, 0.01, tol, 100)
print("xstar1=", xstar1, ", ier=", ier)

# newton starting at x=10
[iters, xstar2, ier, it] = newton(f, df, 10, tol, 100)
print("xstar2=", xstar2, ", ier=", ier)
```

2. (a) A function $f(x)$ has a root α of multiplicity m if there exists some $q(x)$ such that

$$f(x) = (x-\alpha)^m q(x),$$

where $\lim_{x\to\alpha} q(x) \neq 0$.

We have that $f(x)$ has a root α of multiplicity m iff

$$0 = f(\alpha) = f'(\alpha) = \cdots = f^{(m-1)}(\alpha) \text{ and } f^{(m)}(\alpha) \neq 0.$$

(b) Let the root α have multiplicity $m \geq 2$. The Newton step is

$$x_{n+1} = g(x_n) = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Notice that

$$g'(x) = 1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} = \frac{f(x)f''(x)}{f'(x)^2}.$$

Since the root has multiplicity m, we must have $f(x) = (x-\alpha)^m q(x)$, where $q(\alpha) \neq 0$, so

$$\begin{aligned} g'(x) &= \frac{(x-\alpha)^m q(x)((x-\alpha)^m q''(x) + 2m(x-\alpha)^{m-1}q'(x) + m(m-1)(x-\alpha)^{m-2}q(x))}{(m(x-\alpha)^{m-1}q(x) + (x-\alpha)^m q'(x))^2} \\ &= \frac{(x-\alpha)^m q(x)((x-\alpha)^m q''(x) + 2m(x-\alpha)^{m-1}q'(x) + m(m-1)(x-\alpha)^{m-2}q(x))}{(x-\alpha)^{2m-2}(mq(x) + (x-\alpha)q'(x))^2} \\ &= \frac{q(x)((x-\alpha)^2 q''(x) + 2m(x-\alpha)q'(x) + m(m-1)q(x))}{(mq(x) + (x-\alpha)q'(x))^2} \end{aligned}$$

Near $x = \alpha$, we have $q(x) \neq 0$ and therefore

$$\begin{aligned} g'(x) &\approx \frac{q(\alpha)((x-\alpha)^2 q''(\alpha) + 2m(x-\alpha)q'(\alpha) + m(m-1)q(\alpha))}{(mq(\alpha) + (x-\alpha)q'(\alpha))^2} \\ &\approx \frac{m(m-1)q(\alpha)^2}{m^2 q(\alpha)^2} \\ &\approx \frac{m(m-1)}{m^2} < 1. \end{aligned}$$

Thus, Newton's method converges in linear order near the root.

(c) The Newton step is

$$x_{n+1} = g(x_n) = x_n - m\frac{f(x_n)}{f'(x_n)}.$$

Observe that

$$g'(x) = 1 - m\frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} = \frac{(1-m)f'(x)^2 + mf(x)f''(x)}{f'(x)^2}.$$

As before, we have $f(x) = (x-\alpha)^m q(x)$, where $q(\alpha) \neq 0$. After some algebra, we obtain

$$g'(x) = (1-m)(x-a)^{2m}\left(\frac{mq(x)}{x-a} + q'(x)\right)^2 + \frac{mq(x)\left(2m(x-a)q'(x) + (a-x)^2 q''(x) + (m-1)mq(x)\right)}{\left((x-a)q'(x) + mq(x)\right)^2}.$$

Near $x = \alpha$, we have $q(x) \neq \alpha$ and therefore

$$\begin{aligned} g'(x) &\approx (1-m)(x-a)^{2m}\left(\frac{mq(\alpha)}{x-a}\right)^2 m + \frac{mq(\alpha)\left((m-1)mq(\alpha)\right)}{\left(mq(\alpha)\right)^2} \\ &\approx (1-m) + \frac{m^2(m-1)q(\alpha)^2}{m^2q(\alpha)^2} \\ &\approx (1-m) + m - 1 \\ &\approx 0. \end{aligned}$$

Thus, the modified method converges in second order near the root.

(d) Part (c) provides that the modified Newton iteration can be used to restore quadratic convergence of the method for roots of multiplicity more than 1 if the multiplicity of the root is known.

3. Let $\{x_k\}_{k=1}^{\infty}$ be a sequence that converges to α with order p. By definition,

$$\lim_{n\to\infty} \frac{|x_{n+1}-\alpha|}{|x_n-\alpha|^p} = \lambda,$$

where $\lambda \neq 0$. We have

$$\ln\left(\lim_{n\to\infty} \frac{|x_{n+1}-\alpha|}{|x_n-\alpha|^p}\right) = \lim_{n\to\infty} \ln\left(\frac{|x_{n+1}-\alpha|}{|x_n-\alpha|^p}\right) = \lim_{n\to\infty} \left(\ln\left(|x_{n+1}-\alpha|\right) - p\ln\left(|x_n-\alpha|\right)\right) = \ln\lambda,$$

that is,

$$\frac{\ln\left(|x_{n+1}-\alpha|\right) - \ln\lambda}{\ln\left(|x_n-\alpha|\right)} \to p,$$

i.e. a plot of $\ln\left(|x_{n+1}-\alpha|\right)$ against $\ln\left(|x_n-\alpha|\right)$ has slope p.

4. The code used in this question is listed below.

   (a) We consider finding the root of

   $$f(x) = e^{3x} - 27x^6 + 27x^4e^x - 9x^2e^{2x}$$

   over (3, 5). We have derivative

   $$f'(x) = 3(e^x - 6x)(e^x - 3x^2)^2.$$

   We find numerically that the approximate root is

   $$x \approx 3.7330696105827834,$$

   which is the root of the second term $(e^x - 3x^2)$ in $f'(x)$. Thus, this root has multiplicity at least 2 and Newton's method converges in first order. Indeed, numerically calculating the order of convergence yields $\alpha \approx 1.03$.

   (b) We have function

   $$g(x) = \frac{f(x)}{f'(x)} = \frac{e^{3x} - 27x^6 + 27x^4e^x - 9x^2e^{2x}}{3(e^x - 6x)(e^x - 3x^2)^2},$$

   with derivative

   $$g'(x) = \frac{6x^2 + e^x(x^2 - 4x + 2)}{(e^x - 6x)^2}.$$

   We apply Newton's method to $g(x)$ and get the approximate root

   $$x \approx 3.733098888224901,$$

   which is close to before, as expected. The modified method has effectively reduced the multiplicity of the root by 1, so we expect it to converge quadratically. Indeed, numerically calculating the order of convergence yields $\alpha \approx 1.93$.

   (c) We apply the Newton step

   $$x_{n+1} = x_n - m\frac{f(x)}{f'(x)},$$

   where m is the multiplicity of the root. Here, our root has multiplicity $m = 3$. Applying the modified method yields the approximate root

   $$x \approx 3.7330818885939974,$$

   which is close to before, as expected. We expect the modified method to converge quadratically, which it numerically appears to.

   The modified method from question 2 is a useful method, as it preserves quadratic convergence while only requiring the calculations of $f(x)$ and $f'(x)$ (as in the regular Newton method).

```
#! /usr/bin/env python3

import numpy as np
import scipy.special
import math
import matplotlib.pyplot as plt

# function of interest
def f(x):
    return np.exp(3*x) - 27*(x**6) + 27*(x**4)*np.exp(x) - 9*(x**2)*np.exp(2*x)

# derivative of f
```

```
def df(x):
    return 3*(np.exp(x) - 6*x)*(np.exp(x) - 3*x**2)**2


def g(x):
  return f(x) / df(x)

def dg(x):
  return (6*x**3 + np.exp(x)*(x**2-4*x+2)) / ((np.exp(x) - 6*x)**2)


#perform newton's method (provided)
def newton(f,fp,m,p0,tol,Nmax):
    """
    Newton iteration.
    Inputs:
    f,fp - function and derivative
    p0 - initial guess for root
    tol - iteration stops when p_n,p_{n+1} are within tol
    Nmax - max number of iterations
    Returns:
    p - an array of the iterates
    pstar - the last iterate
    info - success message
    - 0 if we met tol
    - 1 if we hit Nmax iterations (fail)
    """
    p = np.zeros(1);
    p[0] = p0
    for it in range(Nmax):
        p1 = p0-m*f(p0)/fp(p0)
        if (abs(p1-p0) < tol):
            pstar = p1
            info = 0
            return [p,pstar,info,it]
        p = np.append(p, p0)
        p0 = p1
    pstar = p1
    info = 1
    return [p,pstar,info,it]

def compute_order(x, xstar):
    diff1 = np.abs(x[1::] - xstar)
    diff2 = np.abs(x[0:-1]-xstar)
    fit = np.polyfit(np.log(diff2.flatten()), np.log(diff1.flatten()),1)
    print('the order of the equation is')
    print("lambda = " + str(np.exp(fit[1])))
    print("alpha = " + str(fit[0]))

    alpha = fit[0]
    l = np.exp(fit[1])

    return [fit, alpha, l]

tol = 1e-10

# find root via Newton
[iters, xstar1, ier, it] = newton(f, df, 1, 4, tol, 100)
print("xstar1=", xstar1, ", ier=", ier, "iters=", iters)
```

```
compute_order(iters, xstar1)

# find root via modified Newton (ii)
[iters, xstar1, ier, it] = newton(g, dg, 1, 4, tol, 100)
print("xstar1=", xstar1, ", ier=", ier, "iters=", iters)
compute_order(iters, xstar1)

# find root via modified Newton (iii)
[iters, xstar1, ier, it] = newton(f, df, 3, 3, tol, 100)
print("xstar1=", xstar1, ", ier=", ier, "iters=", iters)
compute_order(iters, xstar1)
```

5. The code used in this question is listed below.

   (a) We have the function

   $$f(x) = x^6 - x - 1,$$

   with derivative

   $$f'(x) = 6 * x^5 - 1.$$

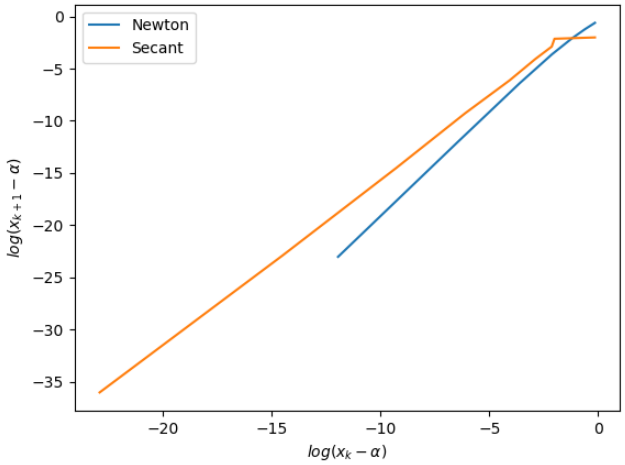
   Applying Newton's method with $x_0 = 2$ yields the root $x \approx 1.1347241384015194$. The error at each step is shown below in the table. Beyond the first few iterations, each iteration gives about twice as many correct digits as the previous, suggesting quadratic convergence.

n	Absolute Error
1	0.8652758615984806
2	0.5459041338497894
3	0.296014849837543
4	0.12024681770791701
5	0.026814294371793723
6	0.0016291357689859343
7	6.389942109663593e-06
8	9.870171346904044e-11
9	0.0

   Applying secant method with $x_0 = 3$, $x_1 = 2$ yields the root $x \approx 1.1347241384015194$, as before. The error at each step is plotted below. The convergence appears to be superlinear but not quadratic—each iteration gives more than a fixed number of new digits but doesn't double the number.

n	Absolute Error
1	0.8652758615984806
2	0.13472413840151942
3	0.11859510614345492
4	0.05585363027511803
5	0.01706830745996779
6	0.0021925881853863682
7	9.266960333342844e-05
8	4.924528145267004e-07
9	1.1030354407637333e-10
10	2.220446049250313e-16
11	0.0

   (b) A plot of $\ln|x_{k+1} - \alpha|$ vs $\ln|x_k - \alpha|$ is below. As we found in question 3, the slope of this plot is the approximate convergence order of the method. As expected, the plot for Newton has a slope of 2 while the line for secant has a slightly lower slope of 1.569. We showed in class that we expect secant to converge with order $\phi \approx 1.618$.




```
#! /usr/bin/env python3

import numpy as np
import scipy.special
import math
import matplotlib.pyplot as plt

# function of interest
def f(x):
    return x**6 - x - 1

# derivative of f
def df(x):
    return 6*x**5-1

#perform newton's method (provided)
def newton(f,fp,m,p0,tol,Nmax):
    """
    Newton iteration.
    Inputs:
    f,fp - function and derivative
    p0 - initial guess for root
    tol - iteration stops when p_n,p_{n+1} are within tol
    Nmax - max number of iterations
    Returns:
    p - an array of the iterates
    pstar - the last iterate
    info - success message
    - 0 if we met tol
    - 1 if we hit Nmax iterations (fail)
    """
    p = np.zeros(1);
    p[0] = p0
    for it in range(Nmax):
        p1 = p0-m*f(p0)/fp(p0)
        if (abs(p1-p0) < tol):
            pstar = p1
            info = 0
            return [p,pstar,info,it]
        p0 = p1
```

```
        p = np.append(p, p0)
    pstar = p1
    info = 1
    return [p,pstar,info,it]

def secant(f, p0, p1, tol, Nmax):
  p = np.zeros(2)
  p[0] = p0
  p[1] = p1
  for i in range(Nmax):
    # secant step
    p2 = p1 - f(p1) * (p1 - p0) / (f(p1) - f(p0))

    # check tolerance
    if (abs(p2-p1) < tol):
      return [p, p2, 0, it]
    # update values
    p = np.append(p, p2)
    p0 = p1
    p1 = p2

  return [p, p2, 1, it]


def plot_order(x, xstar):
    diff1 = np.abs(x[1::] - xstar)
    diff2 = np.abs(x[0:-1]-xstar)
    plt.plot(np.log(diff2.flatten()), np.log(diff1.flatten()))

tol = 1e-16

# find root via Newton
print("Newton")
[iters1, xstar1, ier, it] = newton(f, df, 1, 2, tol, 100)
print("xstar1=", xstar1, ", ier=", ier, "iters=", iters1)
for i in range(len(iters1)):
  print(i+1, " & ", np.abs(iters1[i] - xstar1), "\\\\")

# find root via secant
print("Secant")
[iters2, xstar2, ier, it] = secant(f, 2, 1, tol, 100)
print("xstar1=", xstar2, ", ier=", ier, "iters=", iters2)
for i in range(len(iters2)):
  print(i+1, " & ", np.abs(iters2[i] - xstar2), "\\\\")

plot_order(iters1, xstar1)
plot_order(iters2, xstar2)
plt.legend(["Newton", "Secant"])
plt.xlabel("$log(x_k - \\alpha)$")
plt.ylabel("$log(x_{k+1} - \\alpha)$")
plt.savefig("hw4_5.png")
```