APPM 4600 Lab 4

19 September 2024

1 Prelab

The code for this lab can be seen on github here and is included below.

1. Code for order determination is on github and listed at the end of this document in the function `order_compare`.
2. (a) It takes 12 iterations for fixed point to converge onto the fixed point.
   (b) We find that

$$\alpha \approx 0.99621,$$
$$\lambda \approx 0.12365.$$

This makes sense, as we expect the method to have first order convergence.

2 Aitken's Δ^2 method

1. We have that

$$\frac{p_{n+1}-p}{p_n-p} = \frac{p_{n+2}-p}{p_{n+1}-p},$$

which we can solve as

$$\begin{aligned} 0 &= (p_{n+1}-p)^2 - (p_{n+2}-p)(p_n-p) \\ &= p_{n+1}^2 - 2pp_{n+1} + p^2 - p_{n+2}p_n + pp_{n+2} + pp_n - p^2 \\ &= p_{n+1}^2 - p_{n+2}p_n + p(-2p_{n+1} + p_{n+2} + p_n), \end{aligned}$$

thus,

$$\begin{aligned} p &= \frac{p_{n+1}^2 - p_{n+2}p_n}{p_{n+2} - 2p_{n+1} + p_n} \\ &= p_n - \frac{(p_{n+1}-p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}, \end{aligned}$$

as expected.

2. When we apply Aitken's method to the sequence that we found previously, we converge in 5 iterations and find that

$$\alpha \approx 1.303$$
$$\lambda \approx 0.835,$$

which is super linear convergence ($\alpha > 1$), faster than fixed point iteration.

3 Steffenson's method

1. The implementation of steffenson's method is in the code below and on github.
2. Steffensons's method yields the root

$$r \approx 1.36523001341409,$$

as expected.

3. Steffenson's method converges with

$$\alpha \approx 1.996,$$
$$\lambda \approx 0.001922$$

after 3 iterations. Notice that this appears to be quadratic convergence, which is faster than un modified fixed point's linear convergence.

```
# import libraries
import numpy as np
import matplotlib.pyplot as plt


# run fixed point
def fixedpt(f,x0,tol,Nmax):

    ''' x0 = initial guess'''
    ''' Nmax = max number of iterations'''
    ''' tol = stopping tolerance'''
    x_guess = np.zeros(0)

    count = 0
    while (count <Nmax):
        x_guess = np.append(x_guess, x0)
        count = count +1
        x1 = f(x0)
        if (abs(x1-x0) <tol):
            xstar = x1
            ier = 0
            return [xstar,x_guess,ier]
        x0 = x1

    xstar = x1
    ier = 1
    return [xstar, x_guess, ier]

# run atiken's accelerated method on the sequence p
def atiken(p, tol):
    x = np.zeros(len(p) - 2)
    for n in range(len(p) - 2):
        x[n] = p[n] - (p[n+1] - p[n])**2 / (p[n+2] - 2*p[n+1] + p[n])
        if abs(x[n-1]-x[n]) < tol:
            return x[:n]

    return x

def steffenson(f, x0, tol, Nmax):
    x_guess = np.zeros(0)

    count = 0
    while(count < Nmax):
        x_guess = np.append(x_guess, x0)
        count = count + 1

        a = x0
        b = f(a)
        c = f(b)

        x1 = a - (b-a)**2 / (c - 2*b + a)
```

```
        if abs(x1 - x0) < tol:
            return [x1, x_guess, 0]

        x0 = x1

    return [x1, x_guess, 1]

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]

def question2_2():
    g = lambda x: (10/(x+4))**0.5

    p0 = 1.5
    [xstar, x_guess, ier] = fixedpt(g, p0, 1e-10, 100)
    print("xstar=", xstar)
    print("ier=", ier)
    print("number of guesses=", len(x_guess))
    [fit, alpha, lam] = compute_order(x_guess, xstar)

def question3_3():
    g = lambda x: (10/(x+4))**0.5

    p0 = 1.5
    [xstar, x_guess, ier] = fixedpt(g, p0, 1e-10, 100)
    print("xstar=", xstar)
    print("ier=", ier)
    print("number of guesses=", len(x_guess))
    [fit, alpha, lam] = compute_order(x_guess, xstar)

    print("Atiken's method:")
    x = atiken(x_guess, 1e-10)
    print(x)
    [fit, alpha1, lam1] = compute_order(x, xstar)

def question3_4():
    g = lambda x: (10/(x+4))**0.5

    p0 = 1.5
    [xstar, x_guess, ier] = steffenson(g, p0, 1e-10, 100)
    print("xstar=", xstar)
    print("ier=", ier)
    print("number of guesses=", len(x_guess))
    [fit, alpha, lam] = compute_order(x_guess, xstar)

question3_4()
```