Course Project 1 - Final Report
Math 570
Fall 2023

For this project, Veronika, Adrian, and I chose to create an algorithm that implements Newton's method, Horner's method, and Muller's method to find all roots of a polynomial. This final report will detail how our algorithm works, and how it solved the problem in detail.

Algorithm code

```
def MyRoots(MyMuellers, MyHorners, MySynth, MyNewtonsMethod, a):
    f = a
    p0 = 0
    TOL = 1E-6
    n0 = 1E+6
    realRoots = []          This will be our array of real roots.
    complexRoots = []       This will be our array of complex roots.
    i = 0
This while loop will calculate all the real roots until either the array
of coefficients is equal to 2, or Newton's method returns failure.
    while(len(f)>2):
        if MyHorners(f, p0)[1] == 0:
            Break;
If MyHorners equals 0 then we break the loop and implement mueller's
method. This will show that the derivative equals 0 and the loop breaks.
        cv = MyNewtonsMethod(f, MyHorners, p0, TOL, n0)
        if(isinstance(cv, float) == False):
            break;
        realRoots.append(cv)
        f = MySynth(f, realRoots[i])
        i += 1
This loop finds complex roots, and continues to find roots until either
Mueller's method breaks or the array of coefficients is less than or equal
to 2. This additionally shows that after the root is found, then we
implement the synthetic division algorithm inorder to obtain the rest of
the polynomial where the roots must still be determined.
    i = 0
    while(len(f)>2):
        cv = MyMuellers(p0, f, MyHorners, TOL, n0)
        if(isinstance(cv, complex) == False):
            break;
        complexRoots.append(cv)
        f = MySynth(f, complexRoots[i])
        i += 1
```

This while loop implements Muller's method to further determine the complex roots.

```
    if (isinstance(cv, complex) == False):
        realRoots.append(-1*f[1]/f[0])
    else:
        complexRoots.append(-1*f[1]/f[0])
    return realRoots, complexRoots
```

This then takes the complex roots found by mullers method.

```
def MyMuellers(p0, f, MyHorners, TOL, N):
    p1 = p0 + 1
    p2 = p0 + 2
    h1 = p1-p0
    h2 = p2-p1
    g1 = (MyHorners(f, p1)[0] - MyHorners(f, p0)[0])/h1
    g2 = (MyHorners(f, p2)[0] - MyHorners(f, p1)[0])/h2
    d = (g2 - g1)/(h2 + h1)
    i = 3
    while (i <= N):
        b = g2 + h2 * d
        e = (b**2 - 4 * MyHorners(f, p2)[0] * d)**(1/2)
        if (abs(b - e) < abs(b + e)):
            ee = b + e
        else:
            ee = b - e
        h = (-2 * MyHorners(f, p2)[0])/ee
        p = p2 + h
        if (abs(h) < TOL):
            return p
        p0 = p1
        p1 = p2
        p2 = p
        h1 = p1 - p0
        h2 = p2 - p1
        g1 = (MyHorners(f, p1)[0] - MyHorners(f, p0)[0])/h1
        g2 = (MyHorners(f, p2)[0] - MyHorners(f, p1)[0])/h2
        d = (g2 - g1)/(h2 + h1)
        i = i+1
    return("Method Failed")
```

This part of the code implements Mueller's Method. Specifically, it takes inputs p0, f, MyHorners, a Tolerance, and the maximum number of iterations n0. Then it gives outputs the complex roots of a given polynomial.

```
def MyHorners(a, x):
```

```
    n = len(a)-1
    y = z = a[0]
    for i in range(1, n):
        y = x * y + a[i]
        z = x * z + y
    y = x*y+a[n]
    return(y, z)
```

This part of the code implements Horner's method. Specifically, it takes inputs a and x. Then outputs the polynomials y = f(x) and z = f'(x).

```
def MySynth(a, r):
    n = len(a)
    if (isinstance(r, complex) == True):
        b = np.zeros(n - 1, dtype = complex)
    else:
        b = np.zeros(n - 1)
    i  = 1
    b[0] = a[0]
    while (i < n-1):
        b[i] = a[i] + r * b[i - 1]
        i = i+1
    return b
```

This part of the code implements synthetic division for the polynomial that is inputted. We created two arrays for the synthetic division since the first array will store the complex values, and the second array will store the real values of our function.

```
def MyNewtonsMethod(a, MyHorners, p0, TOL, N0):
    i = 1
    while i <= N0:
        p = p0 - MyHorners(a, p0)[0]/MyHorners(a, p0)[1]
        if abs(p - p0) < TOL:
            return p
        i += 1
        p0 = p
    return("Method failed.")
```

This part of the code implements an updated Newton's method in order to input an array of coefficients of polynomials which then outputs the real root. With this, we used Horner's method to calculate f(x) and f'(x) to then calculate p.

End of Code

Examples of polynomials with code

Example 1: $f(x) = x^2 + x + 1$

```
c = MyRoots(MyMuellers, MyHorners, MySynth, MyNewtonsMethod, [1, 1, 1], 0)
print("Real Roots: ", c[0])
print("Complex Roots: ", c[1])
```

```
Output:
Real Roots:  []
Complex Roots:
[(-0.5-0.8660254037844387j), (-0.5+0.8660254037844387j)]
```

Example 2: $f(x) = x^2 - 1$

```
c = MyRoots(MyMuellers, MyHorners, MySynth, MyNewtonsMethod, [1, 0, -1], 1)
print("Real Roots: ", c[0])
print("Complex Roots: ", c[1])
```

```
Output:
```
Real Roots: [1.0, -1.0]
Complex Roots: []

Example 3: $f(x) = x^2 + 2x + 1$

```
e = MyRoots(MyMuellers, MyHorners, MySynth, MyNewtonsMethod, [1, 2, 1], 1)
print("Real Roots: ", e[0])
print("Complex Roots: ", e[1])
```

Output :
Real Roots: [-0.9999923706054688, -1.0000076293945312]
Complex Roots: []

Example 4: $f(x) = 6x^4 - x^3 - 198x^2 + 256x + 672$

```
f = MyRoots(MyMuellers, MyHorners, MySynth, MyNewtonsMethod, [6, -1, -198,
256, 672], 1)
print("Real Roots: ", f[0])
print("Complex Roots: ", f[1])
```

```
Output:
```
Real Roots: [3.9999999999999996, -6.0, -1.3333333333333355, 3.500000000000002]
Complex Roots: []

Conclusion

Overall, our code is able to calculate the all real and complex roots of a polynomial given the coefficients of the polynomial and the initial approximation p0. From our previous project update, we learned how the Mueller's method works, updated the Newton's method to operate with our given arrays, updated Horner's method to correctly approximate f(x) and f'(x), then used all of this in conjunction with one another to approximate the roots of a polynomial.

Eric Van Amber
Veronika Titarchuk
Adrian Nguyen