# HOMEWORK 4

HUNTER SCHWARTZ

**Problem** (2)**.**

We use the Python code copied on the next page(s) to find the listed solutions:

(a) The real roots of $x^5 - 5x^4 - 8x^3 + 40x^2 - 9x + 45$ are:

- $x = 5$
- $x = 3$
- $x = -3$

(b) The real roots of $1.1x^5 - 5x^4 - 8x^3 + 40x^2 - 9x + 45$ are:

- $x = 4.1858$
- $x = 3.3129$
- $x = -2.9513$

We can note that the roots of part (b) are only slightly different than in part (a), reflecting the slight change in coefficients of their corresponding polynomials.

*Date*: September 30, 2020.

```
class Poly:

    # Takes a list of polynomial coefficients from highest to lowest order

    def __init__(poly, c_list):

        # Degree of polynomial

        poly.degree = len(c_list) - 1


        # Store coefficients

        poly.coefs = c_list


# Implements Horner's method both evaluate polynomials and

# find the remainder left after a root is factored out

def Horner(poly,x):

    q = []

    r = 0

    for i in range(poly.degree+1):

        if i == 0:

            r = poly.coefs[i]

        else:

            q.append(r)

            r = r*x + poly.coefs[i]


    return (Poly(q),r)


# Evaluates polynomial at a point using Horner's method

def val(poly, x):
```

```
    q,r = Horner(poly, x)
    return r

def poly_divide(poly, x):
    q,r = Horner(poly, x)
    return q

# Returns Poly that is derivative of current Poly, simple power rule
def ddx(poly):
    deriv_coefs = []
    for i in range(poly.degree):
        deriv_coefs.append(poly.coefs[i] * (poly.degree-i))
    return Poly(deriv_coefs)

# Returns root found by applying Newton's method, or
# none if no root after maxiters exceeded, which will be
# the case for instance when only complex roots remain
def Newton(poly, x0, tol, maxiters):
    niters = 0
    x = x0
    diff = ddx(poly)
    while abs(val(poly, x)) >= tol and niters < maxiters:
        niters += 1
        x = x - val(poly,x) / val(diff,x)
```

```
    if abs(val(poly, x) < tol):
        return x
    else:
        return None

# Recursively applies Newton's method to find root, then uses
# Horner's method to simplify polynomial
def solve(poly, x0):
    # Init some common params
    tol = 1e-10
    maxiters = 100

    # Don't try to find roots if degree < 1
    if poly.degree < 1:
        return None

    # Solve directly for root if degree = 1
    if poly.degree == 1:
        root = -poly.coefs[1] / poly.coefs[0]

    # Find root if one exists
    root = Newton(poly, x0, tol, maxiters)
    if root == None:
        return None
```

```
    # Horner's method to get polynomial with all the same
    # roots except the one we already have
    q = poly_divide(poly, root)
    other_roots = solve(q, x0)

    # End if no other roots
    if other_roots == None:
        return [root]

    # Refine other roots by solving original poly with
    # approx root as initial condition
    nroots = len(other_roots)
    for i in range(nroots):
        other_roots[i] = Newton(poly, other_roots[i], tol, maxiters)

    # Return list of all roots (so far)
    all_roots = [root] + other_roots

    return all_roots

if __name__ == '__main__':
    x0 = 1

    part_a = [1, -5, -8, 40, -9, 45]
    poly_a = Poly(part_a)
```

```
roots = solve(poly_a, 1)
print(roots)

part_b = [1.1, -5, -8, 40, -9, 45]
poly_b = Poly(part_b)
roots = solve(poly_b, 1)
print(roots)
```