Zadanie 13:

13 O. Stosując metodę Laguerre'a wraz ze strategią obniżania stopnia wielomianu i wygładzania, znajdź wszystkie rozwiązania równań

$$243z^7 - 486z^6 + 783z^5 - 990z^4 + 558z^3 - 28z^2 - 72z + 16 = 0 \quad (12a)$$

$$z^{10} + z^9 + 3z^8 + 2z^7 - z^6 - 3z^5 - 11z^4 - 8z^3 - 12z^2 - 4z - 4 = 0 \quad (12b)$$

$$z^4 + iz^3 - z^2 - iz + 1 = 0 \quad (12c)$$

Kod w języku Python:

```
import sys


class Laguerre:
    def __init__(self, wewn):
        self.wewn = wewn

    def findRoots(self, zo, tol):
        zerowe = []
        cutPoly = []  # wielomian zmniejszony
        tmp = self.wewn
        i = 1
        zerowe.append(self.laguerre(zo, self.wewn, tol))
        while i < len(self.wewn) - 1:
            # obnizenie stopnia wielomianu
            cutPoly = self.deflacja(tmp, zerowe[i - 1])
            zerowe.append(self.laguerre(zerowe[i - 1], cutPoly, tol))

            # wygladzanie
            zerowe[i] = self.laguerre(zerowe[i], self.wewn, tol)
            tmp = cutPoly
            i = i + 1

        return zerowe

    def laguerre(self, zo, w, tol):
        z = zo
        temp = 100 + 0j
        while (abs(self.f(w, z) - self.f(w, temp)) > tol):
            temp = z
            # obliczanie wartosci funkcji
            fo = self.f(w, z)
            fprim = self.f_prim(w, z)
            fbis = self.f_bis(w, z)

            # do sprawdzenia znaku + lub -
            plus = fprim + pow((len(w) - 2) * ((len(w) - 2) * pow(fprim, 2) - (len(w) - 1) * fo *
fbis), 0.5)
            minus = fprim - pow((len(w) - 2) * ((len(w) - 2) * pow(fprim, 2) - (len(w) - 1) * fo *
fbis), 0.5)

            if abs(plus) > abs(minus):
                z = z - (fo * (len(w) - 1)) / plus
            else:
                z = z - (fo * (len(w) - 1)) / minus

        return z

    # oblicza wartosc wielomianu w w punkcie x
    def f(self, w, x):
        result = w[0]
        i = 1
        while i < len(w):
            result = result * x + w[i]
            i = i + 1
        return result
```

```
    # oblicza wartosc pochodnej wielomianu w w punkcie x
    def f_prim(self, w, x):
        result = w[0] * (len(w) - 1)
        i = 1
        while i < len(w) - 1:
            result = result * x + w[i] * (len(w) - 1 - i)
            i = i + 1
        return result

    # oblicza wartosc drugiej pochodnej wielomianu w w punkcie x
    def f_bis(self, w, x):
        result = w[0] * (len(w) - 1) * (len(w) - 2)
        i = 1
        while i < len(w) - 2:
            result = result * x + w[i] * (len(w) - 1 - i) * (len(w) - i - 2)
            i = i + 1
        return result

    # oblicza wspolczynniki wielomianu stopnia o 1 nizszego niz last
    def deflacja(self, lastPoly, zo):
        tempPoly = []
        tempPoly.append(lastPoly[0])
        i = 1
        while i < len(lastPoly) - 1:
            tempPoly.append(lastPoly[i] + zo * tempPoly[i - 1])
            i = i + 1
        return tempPoly


# wypisuje znalezione miejsca zerowe
def printFound(wsp, result):
    print("Miejsca zerowych wielomianu o wspolczynnikach", end="")
    print(wsp)
    if len(sys.argv) == 2 and sys.argv[1] == "rounded":
        for i in result:
            if type(i) is complex:
                print("( %09f %09fj)" % (i.real, i.imag))
            else:
                print(round(i, 9))
    elif len(sys.argv) == 2 and sys.argv[1] == "exact":
        for i in result:
            print(i)
    else:
        print("Nie rozpoznano opcji wywolania programu.")

# round-zaokragla miejsca po przecinku do 9 miejsca
# exact-wypisuje liczbe bez zaokraglania

if __name__ == "__main__":
    tolerance = 1e-13
    wspolczynniki = [243, -486, 783, -990, 558, -28, -72, 16]
    lager = Laguerre(wspolczynniki)
    result = lager.findRoots(-1, tolerance)
    print("Pierwszy wielomian")
    printFound(wspolczynniki, result)
    wspolczynniki = [1, 1, 3, 2, -1, -3, -11, -8, -12, -4, -4]
    lager = Laguerre(wspolczynniki)
    result = lager.findRoots(-1, tolerance)
    print("Drugi wielomian")
    printFound(wspolczynniki, result)
    wspolczynniki = [1, 1j, -1, -1j, 1]
    lager = Laguerre(wspolczynniki)
    result = lager.findRoots(-1, tolerance)
    print("Trzeci wielomian")
    printFound(wspolczynniki, result)
```

Wywołanie programu dla wyników dokładnych:

```
python Zadanie13.py exact
```

Wywołanie programu dla wyników przybliżonych:

```
Python Zadanie13.py rounded
```

Wyniki:

- Dokładne:

```
Pierwszy wielomian
Miejsca zerowych wielomianu o wspolczynnikach[243, -486, 783, -990, 558, -28, -72, 16]
-0.3333333333333333
0.3333333333333334
(0.666665513564496+1.996716348656657e-06j)
(0.6666682379802853-3.3020754057399575e-06j)
(0.6666680122812612+2.2852782995941277e-06j)
(1.1713666691650251e-17+1.414213562373095j)
(-3.704825327778471e-17-1.414213562373095j)
Drugi wielomian
Miejsca zerowych wielomianu o wspolczynnikach[1, 1, 3, 2, -1, -3, -11, -8, -12, -4, -4]
(-0.4999999999999999+0.8660254037844387j)
(8.176638159439723e-10+1.000000006771956j)
(1.3621378523211283e-08+0.9999999954410219j)
(-1.4689251317784723e-08-0.9999999996103799j)
(5.752339469421166e-09-1.0000000143812027j)
(1.0262316566802546e-16-1.4142135623730951j)
(-0.5-0.8660254037844386j)
(-1.4142135623730951+0j)
(1.3037806023828376e-16+1.4142135623730951j)
(1.4142135623730951+0j)
Trzeci wielomian
Miejsca zerowych wielomianu o wspolczynnikach[1, 1j, -1, (-0-1j), 1]
(-0.9510565162951535+0.30901699437494745j)
(-0.5877852522924731-0.8090169943749475j)
(0.5877852522924731-0.8090169943749475j)
(0.9510565162951536+0.3090169943749474j)
```

- Zaokrąglone:

```
Pierwszy wielomian
Miejsca zerowych wielomianu o wspolczynnikach[243, -486, 783, -990, 558, -28, -72, 16]
-0.333333333
0.333333333
( 00.666666 00.000002j)
( 00.666668 -0.000003j)
( 00.666668 00.000002j)
( 00.000000 01.414214j)
( -0.000000 -1.414214j)
Drugi wielomian
Miejsca zerowych wielomianu o wspolczynnikach[1, 1, 3, 2, -1, -3, -11, -8, -12, -4, -4]
( -0.500000 00.866025j)
( 00.000000 01.000000j)
( 00.000000 01.000000j)
( -0.000000 -1.000000j)
( 00.000000 -1.000000j)
( 00.000000 -1.414214j)
( -0.500000 -0.866025j)
( -1.414214 00.000000j)
( 00.000000 01.414214j)
( 01.414214 00.000000j)
Trzeci wielomian
Miejsca zerowych wielomianu o wspolczynnikach[1, 1j, -1, (-0-1j), 1]
( -0.951057 00.309017j)
( -0.587785 -0.809017j)
( 00.587785 -0.809017j)
( 00.951057 00.309017j)
```

Aby obliczyć miejsce zerowe korzystamy z poniższej iteracji z wyznaczonym przez nas punktem początkowym z_0:

$$z_{i+1} = z_i - \frac{nP_n(z_i)}{P'_n(z_i) \pm \sqrt{(n-1)((n-1)[P'_n(z_i)]^2 - nP_n(z_i)P''_n(z_i))}}$$

Działanie algorytmu:

1. Oblicz pierwsze miejsce zerowe korzystając z metody Laguarre'a
2. Dokonaj obniżenia (deflacji) wielomianu
3. Oblicz kolejne miejsce zerowe
4. Dokonaj wygładzenia obliczonego miejsca zerowego
5. jeśli ilość obliczonych miejsc zerowych równa się stopniowi wielomianu to zakończ, w przeciwnym razie powrót do punktu 2.

Aby obniżyć stopień wielomianu tworzymy wielomian b stopnia o jeden mniejszego, a jego współczynniki obliczamy z poniższego układu równań metodą forward substitution:

$$\begin{pmatrix} 1 & & & & & \\ -z_0 & 1 & & & & \\ & -z_0 & 1 & & & \\ & & -z_0 & \dots & & \\ & & & -z_0 & 1 & \\ & & & & -z_0 & 1 \end{pmatrix} \begin{pmatrix} b_{n-1} \\ b_{n-2} \\ b_{n-3} \\ \dots \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ \dots \\ a_2 \\ a_1 \end{pmatrix}$$

Aby dokonać wygładzenia podstawiamy nasze znalezione miejsce zerowe jako z_0 do wzoru Laguerre'a dla wielomianu początkowego.
Obliczając wartości wielomianu (jak i jego pierwsze i drugie pochodne) korzystałem z wzoru Hornera.

Omówienie metody:
Metoda Laguerre'a to bardzo szybka i dokładna metoda, która pozwala nam obliczyć wszystkie miejsca zerowe wielomianu(również zespolone). Dzięki temu, że metoda jest iteracyjna możemy wyniki przybliżać coraz bardziej do wartości rzeczywistych obliczonych analitycznie.
Niewielkie błędy w obliczonych miejscach zerowych mogą wynikać z niedokładności języka programowania, niemożliwości zapisywania liczb niewymiernych w nieskończonej dokładności oraz z zaburzeń współczynników.