Универзитет у Београду

Електротехнички факултет

Одабрана поглавља нумеричке анализе

Пројектни задатак 2

Штурмова теорема

Студент:

Александра Богићевић 0390/17

Београд, школска година 2021/2022

Задатак:

Нека је дат реални полином *P(x)* над реалним сегментом [a, b].

(1) А) Одредити Еуклидовим алгоритмом највећи заједнички делилац

$$Q(x) = GCD(P(x), P'(x))$$

Б) За полином $P(x) = P(x)/ Q(x)$, употребом Штурмове теореме, одредити број нула на [a, b].

(2) Примена Штурмове теореме.
Нека је *k* број децимала на који се заокружује неки реалан број. За полином

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

са реалним коефицијентима

$$a_n, a_{n-1}, \dots, a_1, a_0$$

одредити низ наниже заокружених рационалних коефицијената

$$\alpha_n, \alpha_{n-1}, \dots, \alpha_1, \alpha_0$$

осређених по следећим правилима:

- Ако је $a_k = a_0.a_1 \dots a_k a_{k+1} \dots > 0$ тада $\alpha_k = a_0.a_1 \dots a_k$;
- Ако је $a_k = a_0.a_1 \dots a_k a_{k+1} \dots < 0$ тада $\alpha_k = a_0.a_1 \dots a_k'$ где је a_k' навише заокружена цифра (уз евентуално последично заокруживање и претходних цифара за један број навише).

Одредити процедуру за формирање рационалног полинома

$$\mathrm{P}(x) = \alpha_n x^n + \alpha_{n-1} x^{n-1} + \cdots + \alpha_1 x + \alpha_0$$

Нека је [a, b] сегмент са рационалним рубним тачкама. Наћи такав полином P(x) са реалним коефицијентима и број k, да на основу позитивности полинома P(x) над сегментом [a, b] имамо доказ о позитизивности полинома P(x) над сегментом [a, b].

Напомена:

За рачунање решења коришћен је програмски језик *Python* и његове библиотеке *numpy,math, sympy* и *decimal*. Радно окружење је *Jupyter notebook*. На дну документа се налази прилог са целокупним кодом.

```
import numpy as np
import math as m
import sympy as sp
import decimal
```

Решење (1):

Пре свега треба дефинисати полином P(x), као и интервал [a, b]. Полином је кодован у програму, док се интервал уноси са стандардног улаза. Посматраћемо пар примера.

Пример 1

$$P(x) = x^9 - 3x^7 - x^6 + 3x^5 + 3x^4 - x^3 - 3x^2 + 1$$

За вредности интервала уносимо вредности:

a = 0

b = 3

```
x = sp.symbols('x')
P = x**9 - 3*x**7 - x**6 + 3*x**5 + 3*x**4 - x**3 - 3*x**2 + 1
a = input("Uneti vrednost a za interval [a,b]: ")
b = input("Uneti vrednost b za interval [a,b]: ")
```

```
Uneti vrednost a za interval [a,b]: 0
Uneti vrednost b za interval [a,b]: 3
```

```
print("Trazi se broj nula za polinom P(x) = " + str(P) + " na intervalu [" +
      str(a) + ", " + str(b) + "]")
```

```
Trazi se broj nula za polinom P(x) = x**9 - 3*x**7 - x**6 + 3*x**5 + 3*x**4 - x**3 - 3*x**2 + 1 na intervalu [0, 3]
```

Ставку (A) решавамо позивањем уграђене функције за тражење највећег заједничког делиоца (НЗД у наставку) за полином P(x) и P’(x). Добијена вредност се смешта у променљиву Q.

```
Pdiff = sp.diff(P, x)
print("P(x) = " + str(P))
print("P'(x) = " + str(Pdiff))
Q = sp.gcd(P, Pdiff)
print("GCD(P, P') = " + str(Q))
```

Излаз програма:

```
P(x) = x**9 - 3*x**7 - x**6 + 3*x**5 + 3*x**4 - x**3 - 3*x**2 + 1
P'(x) = 9*x**8 - 21*x**6 - 6*x**5 + 15*x**4 + 12*x**3 - 3*x**2 - 6*x
GCD(P, P') = x**5 - x**4 - 2*x**3 + 2*x**2 + x - 1
```

Добивши НЗД полинома P(x) и P'(x) можемо да почнемо решавање ставке (Б).

Први корак Штурмове теореме је формирање низа полинома

$$P_0(x), P_1(x), P_2(x),\ldots,P_i(x)$$

као

$$P_0(x) = P(x)/Q(x),$$
$$P_1(x) = P'(x),$$
$$P_{i+1}(x) = -REM(P_i(x), P_{i-1}(x))$$ редом за $i = 1, 2, \ldots, r-1$ и $P_r(x) = C - Const.$

Почетни полином $P_0(x)$ израчунавамо као количник полинома P(x) и Q(x).

Полином $P_1(x)$ израчунавамо као први извод полинома $P_0(x)$.

```
P0 = sp.div(P, Q)[0]
P1 = sp.diff(P0, x)

print("P0(x) = " + str(P0))
print("P1(x) = " + str (P1))
```

```
P0(x) = x**4 + x**3 - x - 1
P1(x) = 4*x**3 + 3*x**2 - 1
```

Добијене полиноме треба памтити како бисмо их касније користили, тако да уводимо низ полинома у којем чувамо претходно добијене полиноме P_0 и P_1.

```
p_arr = []
p_arr.append(P0)
p_arr.append(P1)
```

Наставак израчунавања полинома може се представити једном петљом.

```
while sp.degree(P0) != 1:
    p_i = -sp.rem(P0, P1)
    P0 = P1
    P1 = p_i
    p_arr.append(p_i)
    print("P" + str(len(p_arr)-1) + "(x) = " + str(p_i))
```

Након извршене петље добијамо следеће:

```
P2(x) = 3*x**2/16 + 3*x/4 + 15/16
P3(x) = -32*x - 64
P4(x) = -3/16
```

Овиме смо добили цео низ полинома и можемо прећи на наредни корак.

Следећи корак Штурмове теореме гласи:

Нека је $\xi \in [a, b]$ означимо $V(\xi)$ број промена знакова у низу $P_0(\xi), P_1(\xi), \ldots, P_r(\xi)$, игноришући евентуално јављање корена полинома у том низу. Тада разлика

$$N = V(a) - V(b)$$

одређује број нула полинома P(x) на сегменту.

Другим речима:

За сваки полином у запамћеном низу полинома $P_0(x)$, $P_1(x)$, $P_2(x)$,...,Pi(x) израчунаваћемо вредност за x=a и израчунати број промена знакова. Резултат ћемо сместити у променљиву Va. Исто то ћемо урадити и за x=b и резултат ћемо сместити у променљиву Vb.

Да бисмо добили број нула почетног полинома израчунаћемо разлику Va и Vb, и то сместити у променљиву N и тиме добити коначно решење.

У коду је то одрађено на следећи начин:

Уводимо нове променљиве

```
Va = 0
Vb = 0
a_last_sign = -1
b_last_sign = -1
```

при чему Va и Vb представљају број промена знакова, и иницијално су постављени на нулу.

Променљиве a_last_sign и b_last_sign су помоћне променљиве за гледање да ли је дошло до промене знака или није. Оне означавају ког знака је последње израчунати полином. Вредност -1 означава иницијалну вредност, тј. да још увек ни један полином није израчунат. Уколико се добије негативна вредност полинома за x=a, a_last_sign се поставља на 0, а уколико се добије позитивна вредност, a_last_sign се поставља на 1. Исто важи и за x=b и b_last_sign.

При израчунавању сваког полинома пореди се његов знак са знаком претходно израчунатог полинома. Уколико је дошло до промене знака, увећава се променљива Va (или Vb) и

ажурира се a_last_sign (или b_last_sign). Уколико није дошло до промене знака промењиве Va и Vb остају непромењене.

Да би замена x=a или x=b уопште била могућа, потребно је да степен полинома буде различит од нуле.

```
for p in p_arr:
    if sp.degree(p) != 0: #polinom stepena razlicitog od 0
        pp = sp.Poly(p)
        p_a = pp.subs(x, a)
        if a_last_sign == -1: #prvi broj u nizu, inicijalizujemo a_last_sign i b_last_sign
            if p_a > 0:
                a_last_sign = 1
            else:
                a_last_sign = 0
        else:
            if p_a > 0: #pozitivan broj
                if a_last_sign == 0: #poslednje je bio negativan broj
                    Va = Va + 1 #doslo je do promene
            else: #negativan broj
                if a_last_sign == 1: #poslednje je bio pozitivan broj
                    Va = Va + 1 #doslo je do promene

            #azuriraj sign
            if p_a > 0:
                a_last_sign = 1
            else:
                a_last_sign = 0


        p_b = pp.subs(x, b)
        if b_last_sign == -1: #prvi broj u nizu, inicijalizujemo a_last_sign i b_last_sign
            if p_b > 0:
                b_last_sign = 1
            else:
                b_last_sign = 0
        else:
            if p_b > 0: #pozitivan broj
                if b_last_sign == 0: #poslednje je bio negativan broj
                    Vb = Vb + 1 #doslo je do promene
            else: #negativan broj
                if b_last_sign == 1: #poslednje je bio pozitivan broj
                    Vb = Vb + 1 #doslo je do promene
            #azuriraj sign
            if p_b > 0:
                b_last_sign = 1
            else:
                b_last_sign = 0
    else:
```

Уколико је степен полинома једнак нули, не врши се замена већ се само посматра знак и ажурирају вредности уколико је потребно.

```
else:
    if a_last_sign == -1: #prvi broj u nizu, inicijalizujemo a_last_sign i b_last_sign
        if p > 0:
            a_last_sign = 1
        else:
            a_last_sign = 0
    else:
        if p > 0: #pozitivan broj
            if a_last_sign == 0: #poslednje je bio negativan broj
                Va = Va + 1 #doslo je do promene
        else: #negativan broj
            if a_last_sign == 1: #poslednje je bio pozitivan broj
                Va = Va + 1 #doslo je do promene
        #azuriraj sign
        if p > 0:
            a_last_sign = 1
        else:
            a_last_sign = 0

    if b_last_sign == -1: #prvi broj u nizu, inicijalizujemo a_last_sign i b_last_sign
        if p > 0:
            b_last_sign = 1
        else:
            b_last_sign = 0
    else:
        if p > 0: #pozitivan broj
            if b_last_sign == 0: #poslednje je bio negativan broj
                Vb = Vb + 1 #doslo je do promene
        else: #negativan broj
            if b_last_sign == 1: #poslednje je bio pozitivan broj
                Vb = Vb + 1 #doslo je do promene
        #azuriraj sign
        if p > 0:
            b_last_sign = 1
        else:
            b_last_sign = 0
```

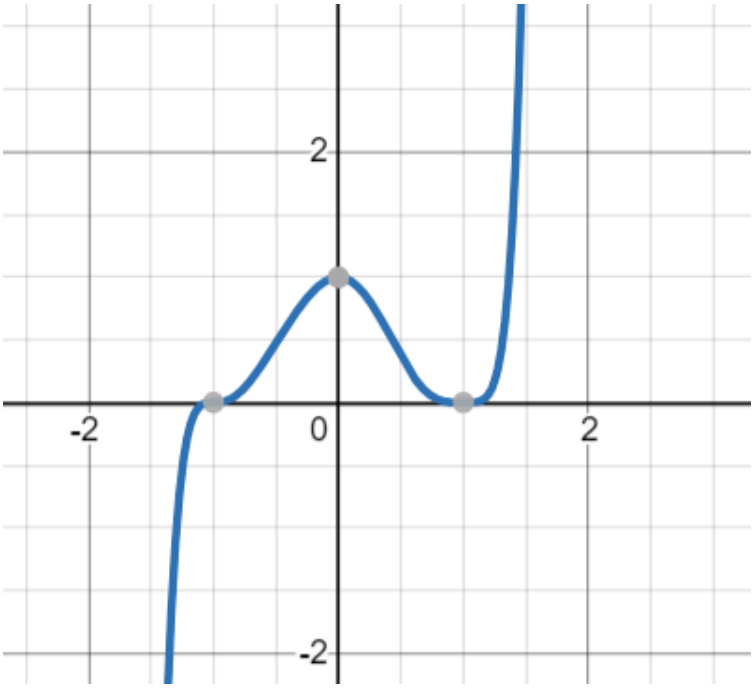
Када смо израчунали вредности Va и Vb можемо да израчунамо и коначни резултат N.

```
print("Va = " + str(Va))
print("Vb = " + str(Vb))
N = Va - Vb
print("Broj nula: N = " + str(N))
```

```
Va = 2
Vb = 1
Broj nula: N = 1
```

Провера решења:

Нуле полинома можемо уочити на следећем графику



Као што се може приметити, на интервалу [0, 3] заиста постоји једна нула.

Пример 2

$$P(x) = x^4 - 3 * x^3 - 5 * x^2 + x + 1$$

За вредности интервала уносимо вредности:

a = -3

b = 4

Излаз програма:

```
Uneti vrednost a za interval [a,b]: -3
Uneti vrednost b za interval [a,b]: 4


Trazi se broj nula za polinom P(x) = x**4 - 3*x**3 - 5*x**2 + x + 1 na intervalu [-3, 4]
```

Тражење НЗД:

```
P(x) = x**4 - 3*x**3 - 5*x**2 + x + 1
P'(x) = 4*x**3 - 9*x**2 - 10*x + 1
GCD(P, P') = 1
```

Формирање низа полинома:

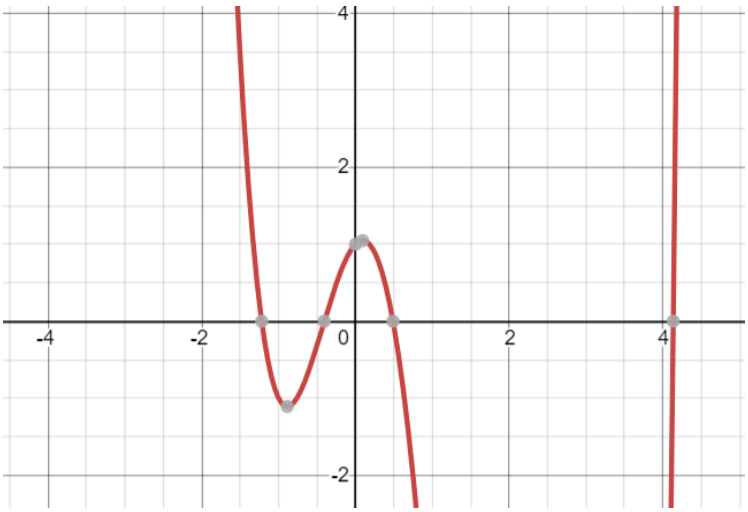
```
P0(x) = x**4 - 3*x**3 - 5*x**2 + x + 1
P1(x) = 4*x**3 - 9*x**2 - 10*x + 1

---

P2(x) = 67*x**2/16 + 9*x/8 - 19/16
P3(x) = 27648*x/4489 + 8336/4489
P4(x) = 54752333/47775744
```

Број нула:

```
Va = 4
Vb = 1
Broj nula: N = 3
```

Провера решења:

Нуле полинома можемо уочити на следећем графику



Као што се може приметити, на интервалу [-3, 4] заиста постоје три нуле.

Пример 3

$$P(x) = 5 * x^6 + 3x^4 - x^3 + 7 * x^2 + 9 * x - 2$$

За вредности интервала уносимо вредности:

a = -2

b = 2

Излаз програма:

```
Uneti vrednost a za interval [a,b]: -2
Uneti vrednost b za interval [a,b]: 2

Trazi se broj nula za polinom P(x) = 5*x**6 + 3*x**4 - x**3 + 7*x**2 + 9*x - 2 na intervalu [-2, 2]
```

Тражење НЗД:

```
P(x) = 5*x**6 + 3*x**4 - x**3 + 7*x**2 + 9*x - 2
P'(x) = 30*x**5 + 12*x**3 - 3*x**2 + 14*x + 9
GCD(P, P') = 1
```

Формирање низа полинома:

```
P0(x) = 5*x**6 + 3*x**4 - x**3 + 7*x**2 + 9*x - 2
P1(x) = 30*x**5 + 12*x**3 - 3*x**2 + 14*x + 9
```

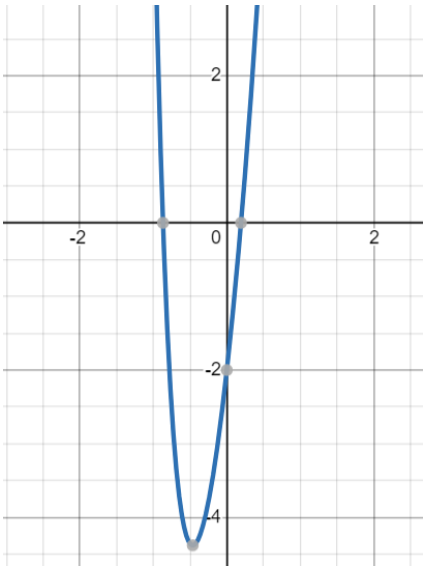
```
P2(x) = -x**4 + x**3/2 - 14*x**2/3 - 15*x/2 + 2
P3(x) = 241*x**3/2 + 298*x**2 + 77*x/2 - 39
P4(x) = 2038565*x**2/174243 + 509576*x/58081 - 172049/58081
P5(x) = 360294701226998*x/4155747259225 - 56400892481187/4155747259225
P6(x) = 29102388498271720500091 25/22350212932327569106372 84
```

Број нула:

```
Va = 4
Vb = 2
Broj nula: N = 2
```

Провера решења:

Нуле полинома можемо уочити на следећем графику



Као што се може приметити, на интервалу [-2, 2] заиста постоје две нуле.

Решење (2):

Посматрамо полиномску функцију са реалним коефицијентима.

Пример 1

$$P(x) = \left(\frac{\pi}{1260} - \frac{1}{420}\right)x^8 + \left(-\frac{\pi^2}{1680} + \frac{\pi}{840}\right)x^7 + \left(-\frac{\pi}{30} + \frac{1}{10}\right)x^6 + \left(-\frac{\pi^2}{60} + \frac{\pi}{30}\right)x^5 + \left(\frac{2\pi}{3} - 2\right)x^4$$

Показаћемо како се може уз нанижу апроксимацију коефицијената разломцима и уз употребу Штурмове теореме доказати P(x)>0 над интервалом (0, 1.35).

Пре свега уносимо полином и вредност k која представља број децимала на који се заокружује неки реалан број.

Израчунавањем вредности P(x) добијамо:

```
print(str(P))
```

```
0.000112375121896661*x**8 - 0.00213477327018439*x**7 - 0.00471975511965976*x**6 - 0.0597736515651629*x**5 + 0.0943951023931953*
x**4
```

Из оваквог полинома издвајамо коефицијенте у један низ.

```
pp = sp.Poly(P)
coefs = pp.all_coeffs()
print(str(coefs))
```

```
[0.000112375121896661, -0.00213477327018439, -0.00471975511965976, -0.0597736515651629, 0.0943951023931953, 0.0, 0.0, 0.0, 0.0]
```

Када смо издвојили коефицијенте, можемо да их скраћујемо.

Пре свега желимо да радимо са бројевима чији је цели део већи од нуле, тј. да децимални број представимо умношком броја 10. Таква трансформација је извршена у следећем делу кода:

```
cnt = 0
my_coefs = []
for c in coefs:
    cnt = 0
    if c != 0:
        if (c < 0):
            neg = True
        else:
            neg = False
        c = abs(c)
        while c < 1:
            c = c * 10
            cnt = cnt + 1
        if neg:
            my_coefs.append((-c, -cnt))
        else:
            my_coefs.append((c, -cnt))
    else:
        my_coefs.append((c, 0))
```

За сваки коефицијент се памти децимални део као и степен умношка.

Извршавањем ове петље добијамо коефицијенте који изгледају овако:

```
for c in my_coefs:
    print(str(c[0]) + "*10**" + str(c[1]))
```

```
1.12375121896661*10**-4
-2.13477327018439*10**-3
-4.71975511965976*10**-3
-5.97736515651629*10**-2
9.43951023931953*10**-2
0.0*10**0
0.0*10**0
0.0*10**0
0.0*10**0
```

Тако да наш полином сада изгледа овако:

```
newPstr = ""
i = len(my_coefs)-1
for c in my_coefs:
    newPstr = newPstr + "(" + str(c[0]) + "*10**" + str(c[1]) + "*x**" + str(i) +")"
    i = i - 1
    if i >= 0:
        newPstr = newPstr + "+"
print(newPstr)
```

```
(1.12375121896661*10**-4*x**8)+(-2.13477327018439*10**-3*x**7)+(-4.71975511965976*10**-3*x**6)+(-5.97736515651629*10**-2*x**5)+
(9.43951023931953*10**-2*x**4)+(0.0*10**0*x**3)+(0.0*10**0*x**2)+(0.0*10**0*x**1)+(0.0*10**0*x**0)
```

Када имамо издвојене коефицијенте, можемо да извршимо скраћивање на k децимала следећим правилом:

- Ако је $a_k = a_0.a_1 \dots a_k a_{k+1} \dots > 0$ тада $\alpha_k = a_0.a_1 \dots a_k$;
- Ако је $a_k = a_0.a_1 \dots a_k a_{k+1} \dots < 0$ тада $\alpha_k = a_0.a_1 \dots a_k{}'$ где је $a_k{}'$ навише заокружена цифра (уз евентуално последично заокруживање и претходних цифара за један број навише).

```
rounded_coefs = []
for c in my_coefs:
    if c[0] > 0:
        decimal.getcontext().rounding = decimal.ROUND_DOWN
        newc = decimal.Decimal(str(c[0]))
        newc = float(round(newc, k))
        print(newc)
        rounded_coefs.append((newc, c[1]))
    else:
        decimal.getcontext().rounding = decimal.ROUND_UP
        newc = decimal.Decimal(str(c[0]))
        newc = float(round(newc, k))
        print(newc)
        rounded_coefs.append((newc, c[1]))
```

```
1.12
-2.14
-4.72
-5.98
9.43
0.0
0.0
0.0
0.0
```

Након заокруживања наш полином изгледа овако:

```
newPstr = ""
i = len(rounded_coefs)-1
for c in rounded_coefs:
    newPstr = newPstr + "(" + str(c[0]) + "*10**" + str(c[1]) + "*x**" + str(i) + ")"
    i = i - 1
    if i >= 0:
        newPstr = newPstr + "+"
print(newPstr)
```

```
(1.12*10**-4*x**8)+(-2.14*10**-3*x**7)+(-4.72*10**-3*x**6)+(-5.98*10**-2*x**5)+(9.43*10**-2*x**4)+(0.0*10**0*x**3)+(0.0*10**0*x**2)+(0.0*10**0*x**1)+(0.0*10**0*x**0)
```

Овако написан полином морамо средити ручно због недотатка програмског језика за представљање разломака.

Коефицијенте претварамо у разломке:

$$
\begin{aligned}
P(x) &= 1.12 \cdot 10^{-4} x^8 - 2.14 \cdot 10^{-3} x^7 - 4.72 \cdot 10^{-3} x^6 - 5.98 \cdot 10^{-2} x^5 + 9.43 \cdot 10^{-2} x^4 \\
&= 1.12 \cdot \frac{1}{10^4} x^8 - 2.14 \cdot \frac{1}{10^3} x^7 - 4.72 \cdot \frac{1}{10^3} x^6 - 5.98 \cdot \frac{1}{10^2} x^5 + 9.43 \cdot \frac{1}{10^2} x^4 \\
&= \frac{112}{10^6} x^8 - \frac{214}{10^5} x^7 - \frac{472}{10^5} x^6 - \frac{598}{10^4} x^5 + \frac{943}{10^4} x^4 \\
&= \frac{112}{1000000} x^8 - \frac{214}{100000} x^7 - \frac{472}{100000} x^6 - \frac{598}{10000} x^5 + \frac{943}{10000} x^4 \\
&= \mathbf{\frac{7}{62500} x^8 - \frac{107}{50000} x^7 - \frac{59}{12500} x^6 - \frac{299}{5000} x^5 + \frac{943}{10000} x^4.}
\end{aligned}
$$

Срећен полином уносимо у програм који смо направили за ставку 1, како бисмо добили број нула полинома на интервалу [0, 1.35]

Излаз програма:

```
Uneti vrednost a za interval [a,b]: 0
Uneti vrednost b za interval [a,b]: 1.35
```

```
Trazi se broj nula za polinom P(x) = 0.000112*x**8 - 0.00214*x**7 - 0.00472*x**6 - 0.0598*x**5 + 0.0943*x**4 na intervalu [0, 1.35]

P(x) = 0.000112*x**8 - 0.00214*x**7 - 0.00472*x**6 - 0.0598*x**5 + 0.0943*x**4
P'(x) = 0.000896*x**7 - 0.01498*x**6 - 0.02832*x**5 - 0.299*x**4 + 0.3772*x**3
GCD(P, P') = 1.0*x**3

P0(x) = 0.000112*x**5 - 0.00214*x**4 - 0.00472*x**3 - 0.0598*x**2 + 0.0943*x
P1(x) = 0.00056*x**4 - 0.00856*x**3 - 0.01416*x**2 - 0.1196*x + 0.0943
```

```
P2(x) = 0.00843028571428571*x**3 + 0.0467022857142857*x**2 + 0.0159685714285714
*x - 0.0720721428571429
P3(x) = -0.0493863269553692*x**2 + 0.0927218268255306*x + 0.00540326270070265
P4(x) = 0.0652308588755418 - 0.134289674120034*x
P5(x) = -0.0387899182667688


Va = 3
Vb = 3
Broj nula: N = 0
```

Добијамо да овај полином нема нуле на том интервалу. На осову тога следи:

P(1,35)=0,0002478.. > 0

Следи закључак

$$(\forall x \in (0, 1.35))\ P(x) > 0.$$

На основу поретка

$$(\forall x \in (0, 1.35))\ P(x) > P(x),$$

Уједно је доказана и позитвност полинома P(x).

Пример 2:

Почетни полином:

$$P = \sqrt{2} * x^5 + \pi * \frac{3}{200} * x^4 - \frac{\sqrt{3}}{60} * x^3 + \frac{2*\pi}{800} * x^2 - 1$$

k = 3

```
x = sp.symbols('x')
P = 2**0.5*x**5 + m.pi*3/200*x**4 - 3**0.5/60*x**3 + 2*m.pi/800*x**2 - 1
k = 3
```

Израчунавањем добијамо:

```
print(str(P))
```

```
1.4142135623731*x**5 + 0.0471238898038469*x**4 - 0.0288675134594813*x**3 + 0.00
785398163397448*x**2 - 1
```

Срећивање коефицијената:

```
(1.41421356237310*10**0*x**5)+(4.71238898038469*10**-2*x**4)+(-2.88675134594813*10**-2*x**3)+(7.85398163397448*10**-3*x**2)+(0.
0*10**0*x**1)+(-1.00000000000000*10**0*x**0)
```

Заокруживање коефицијената на три децимале:

```
(1.414*10**0*x**5)+(4.712*10**-2*x**4)+(-2.887*10**-2*x**3)+(7.853*10**-3*x**2)+(0.0*10**0*x**1)+(-1.0*10**0*x**0)
```

Претварање децималних бројева у разломке:

$$P = 1.414 * x^5 + 4.712 * 10^{-2} * x^4 - 2.887 * 10^{-2} * x^3 + 7.853 * 10^{-3} * x^2 - 1$$

$$P = 1.414 * x^5 + \frac{4.712}{10^2} * x^4 - \frac{2.887}{10^2} * x^3 + \frac{7.853}{10^3} * x^2 - 1$$

$$P = 1.414 * x^5 + \frac{4.712}{100} * x^4 - \frac{2.887}{100} * x^3 + \frac{7.853}{1000} * x^2 - 1$$

$$P = \frac{1414}{1000} * x^5 + \frac{4712}{100000} * x^4 - \frac{2887}{100000} * x^3 + \frac{7853}{1000000} * x^2 - 1$$

$$P = \frac{707}{500} * x^5 + \frac{589}{12500} * x^4 - \frac{2887}{100000} * x^3 + \frac{7853}{1000000} * x^2 - 1$$

Израчунавање нула:

```
Uneti vrednost a za interval [a,b]: 1
Uneti vrednost b za interval [a,b]: 2.35
```

```
Trazi se broj nula za polinom P(x) = 1.414*x**5 + 0.04712*x**4 - 0.02887*x**3 + 0.007853*x**2 - 1 na intervalu [1, 2.35]
```

```
P(x) = 1.414*x**5 + 0.04712*x**4 - 0.02887*x**3 + 0.007853*x**2 - 1.0
P'(x) = 7.07*x**4 + 0.18848*x**3 - 0.08661*x**2 + 0.015706*x
GCD(P, P') = 1.00000000000000
```

```
P0(x) = 1.414*x**5 + 0.04712*x**4 - 0.02887*x**3 + 0.007853*x**2 - 1.0
P1(x) = 7.07*x**4 + 0.18848*x**3 - 0.08661*x**2 + 0.015706*x


P2(x) = 0.0117992355756719*x**3 - 0.00482724733239038*x**2 + 2.09354093352192e-5*x + 1.0
P3(x) = -1.16129907134205*x**2 + 599.18111951309*x + 261.112244913527
P4(x) = -3141.26375247792*x - 1368.74952961889
P5(x) = 0.190704596140051

Va = 2
Vb = 2
Broj nula: N = 0
```

Добијамо:

$$\forall x \in (1, 2.35) \rightarrow P(x)>0$$

Из тога P(x) > P(x) следи:

$$\forall x \in (1, 2.35) \rightarrow P(x) > 0$$

Прилог:

У наставку следе кодови у целости:

Код 1:

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import math as m
import sympy as sp


# P(x) -> pocetni polinom
#
# P = x**9 - 3*x**7 - x**6 + 3*x**5 + 3*x**4 - x**3 - 3*x**2 + 1
#
# P = x**4 - 3*x**3 - 5*x**2 + x + 1
#
# P = 5*x**6 + 3*x**4 - x**3 + 7*x**2 + 9*x - 2
#
# P = (7/62500*x**8)+(-107/50000*x**7)+(-59/12500*x**6)+(-
299/5000*x**5)+(943/10000*x**4)
#
# P = (1.414*10**0*x**5)+(4.712*10**-2*x**4)+(-2.887*10**-
2*x**3)+(7.853*10**-3*x**2)-1
#
# P = (707/500*x**5)+(589/12500*x**4)+(-
2887/100000*x**3)+(7853/1000000*x**2)-1

# In[2]:


x = sp.symbols('x')
P = (707/500*x**5)+(589/12500*x**4)+(-2887/100000*x**3)+(7853/1000000*x**2)-1
a = input("Uneti vrednost a za interval [a,b]: ")
b = input("Uneti vrednost b za interval [a,b]: ")


# In[3]:


print("Trazi se broj nula za polinom P(x) = " + str(P) + " na intervalu [" +
      str(a) + ", " + str(b) + "]")


# Uz pomoc Euklidovog algoritma trazimo GCD za P i P'

# In[4]:


Pdiff = sp.diff(P, x)
print("P(x) = " + str(P))
```

```
print("P'(x) = " + str(Pdiff))
Q = sp.gcd(P, Pdiff)
print("GCD(P, P') = " + str(Q))


# P0 = P/ Q

# In[5]:


P0 = sp.div(P, Q)[0]
P1 = sp.diff(P0, x)

print("P0(x) = " + str(P0))
print("P1(x) = " + str (P1))


# moramo imati niz u kom cemo cuvati sve Pi vrednosti

# In[6]:


p_arr = []
p_arr.append(P0)
p_arr.append(P1)


# In[7]:


while sp.degree(P0) != 1:
    p_i = -sp.rem(P0, P1)
    P0 = P1
    P1 = p_i
    p_arr.append(p_i)
    print("P" + str(len(p_arr)-1) + "(x) = " + str(p_i))


# Kada imamo niz, mozemo da odredimo broj nula na intervalu [a,b] tako sto
cemo odrediti sve Pi(a) i Pi(b) i sracunati razliku broja promene znakova.
# a_last_sign i b_last_sign predstavljaju poslednji znak, i s njim
uporedjujemo sledeci. 0 = negativno, 1 = pozitivno

# In[8]:


Va = 0
Vb = 0
a_last_sign = -1
b_last_sign = -1


# In[9]:


for p in p_arr:
    if sp.degree(p) != 0: #polinom stepena razlicitog od 0
```

```
        pp = sp.Poly(p)
        p_a = pp.subs(x, a)
        if a_last_sign == -1: #prvi broj u nizu, inicijalizujemo a_last_sign
i b_last_sign
            if p_a > 0:
                a_last_sign = 1
            elif p_a < 0:
                a_last_sign = 0
        else:
            if p_a > 0: #pozitivan broj
                if a_last_sign == 0: #poslednje je bio negativan broj
                    Va = Va + 1 #doslo je do promene
            else: #negativan broj
                if a_last_sign == 1: #poslednje je bio pozitivan broj
                    Va = Va + 1 #doslo je do promene

            #azuriraj sign
            if p_a > 0:
                a_last_sign = 1
            elif p_a < 0:
                a_last_sign = 0


        p_b = pp.subs(x, b)
        if b_last_sign == -1: #prvi broj u nizu, inicijalizujemo a_last_sign
i b_last_sign
            if p_b > 0:
                b_last_sign = 1
            elif p_b < 0:
                b_last_sign = 0
        else:
            if p_b > 0: #pozitivan broj
                if b_last_sign == 0: #poslednje je bio negativan broj
                    Vb = Vb + 1 #doslo je do promene
            elif p_b < 0: #negativan broj
                if b_last_sign == 1: #poslednje je bio pozitivan broj
                    Vb = Vb + 1 #doslo je do promene
            #azuriraj sign
            if p_b > 0:
                b_last_sign = 1
            else:
                b_last_sign = 0
    else:
        if a_last_sign == -1: #prvi broj u nizu, inicijalizujemo a_last_sign
i b_last_sign
            if p > 0:
                a_last_sign = 1
            elif p < 0:
                a_last_sign = 0
        else:
            if p > 0: #pozitivan broj
                if a_last_sign == 0: #poslednje je bio negativan broj
                    Va = Va + 1 #doslo je do promene
            elif p < 0: #negativan broj
                if a_last_sign == 1: #poslednje je bio pozitivan broj
                    Va = Va + 1 #doslo je do promene
            #azuriraj sign
```

```
            if p > 0:
                a_last_sign = 1
            elif p < 0:
                a_last_sign = 0

        if b_last_sign == -1: #prvi broj u nizu, inicijalizujemo a_last_sign
i b_last_sign
            if p > 0:
                b_last_sign = 1
            elif p < 0:
                b_last_sign = 0
        else:
            if p > 0: #pozitivan broj
                if b_last_sign == 0: #poslednje je bio negativan broj
                    Vb = Vb + 1 #doslo je do promene
            elif p < 0: #negativan broj
                if b_last_sign == 1: #poslednje je bio pozitivan broj
                    Vb = Vb + 1 #doslo je do promene
            #azuriraj sign
            if p > 0:
                b_last_sign = 1
            elif p < 0:
                b_last_sign = 0


# In[10]:


print("Va = " + str(Va))
print("Vb = " + str(Vb))
N = Va - Vb
print("Broj nula: N = " + str(N))
```

Код 2:

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import math as m
import sympy as sp
import decimal


# P = (m.pi/1260 - 1/420)*x**8 + (-(m.pi**2/1680)+m.pi/840)*x**7 +
\((m.pi/30)+1/10)*x**6 + (-(m.pi**2/60)+m.pi/30)*x**5 +(2*m.pi/3-2)*x**4
#
# P = 2**0.5*x**5 + m.pi*3/200*x**4 - 3**0.5/60*x**3 + 2*m.pi/800*x**2 - 1

# In[2]:


x = sp.symbols('x')
P = 2**0.5*x**5 + m.pi*3/200*x**4 - 3**0.5/60*x**3 + 2*m.pi/800*x**2 - 1
k = 3


# In[3]:


print(str(P))


# In[4]:


pp = sp.Poly(P)
coefs = pp.all_coeffs()
print(str(coefs))


# Sredjivanje koeficijenata na oblik 1,...*10^..
#
# my_coefs predstavlja novi niz koeficijenata, koji se sastoji iz parova
vrednosti
#
# prva vrednost predstavlja koeficijent, a druga umnozak broja 10

# In[5]:


cnt = 0
my_coefs = []
for c in coefs:
    cnt = 0
```

```
    if c != 0:
        if (c < 0):
            neg = True
        else:
            neg = False
        c = abs(c)
        while c < 1:
            c = c * 10
            cnt = cnt + 1
        if neg:
            my_coefs.append((-c, -cnt))
        else:
            my_coefs.append((c, -cnt))
    else:
        my_coefs.append((c, 0))

for c in my_coefs:
    print(str(c[0]) + "*10**" + str(c[1]))


# In[6]:


newPstr = ""
i = len(my_coefs)-1
for c in my_coefs:
    newPstr = newPstr + "(" + str(c[0]) + "*10**" + str(c[1]) + "*x**" +
str(i) +")"
    i = i - 1
    if i >= 0:
        newPstr = newPstr + "+"
print(newPstr)


# Pravilo za zaokruzivanje:
#
# ako je koeficijent pozitivan, zaokruzuje se na donju brojku
#
# ako je negativan, zaokruzuje se na gornju

# In[7]:


rounded_coefs = []
for c in my_coefs:
    if c[0] > 0:
        decimal.getcontext().rounding = decimal.ROUND_DOWN
        newc = decimal.Decimal(str(c[0]))
        newc = float(round(newc, k))
        print(newc)
        rounded_coefs.append((newc, c[1]))
    else:
        decimal.getcontext().rounding = decimal.ROUND_UP
        newc = decimal.Decimal(str(c[0]))
        newc = float(round(newc, k))
        print(newc)
        rounded_coefs.append((newc, c[1]))
```

```
# Ispis zaokruzenog polinoma

# In[8]:


newPstr = ""
i = len(rounded_coefs)-1
for c in rounded_coefs:
    newPstr = newPstr + "(" + str(c[0]) + "*10**" + str(c[1]) + "*x**" +
str(i) + ")"
    i = i - 1
    if i >= 0:
        newPstr = newPstr + "+"
print(newPstr)
```