Универзитет у Београду

Електротехнички факултет

Одабрана поглавља нумеричке анализе

Пројектни задатак 3

Миксовано тригонометријско полиномске неједнакости

Студент:

Александра Богићевић 0390/17

Београд, школска година 2021/2022

Миксовано тригонометријско полиномске неједнакости – МТП неједнакости

МТП функција представља функцију

$$f(x) = \sum_{i=1}^{n} \alpha_i x^{p_i} cos^{q_i} x \, sin^{r_i} x$$

за $\alpha_i \epsilon R \backslash \{0\}$ и $p_i, q_i, r_i \epsilon N_0 \{i = 1, \dots, n\}$ за вредности аргумента $x\epsilon(0, c)$, при стандардној вредности $c = \pi/2$. За МТП функцију *f* основни проблем наниже апроксимације је да се одреди полином P такав да

$$f(x) > P(x)$$

за $x\epsilon(0, c)$. Уколико за полином *P* важи полиномска неједнакост

$$P(x) > 0$$

за $x\epsilon(0, c)$, тада за МТП функцију *f* важи МТП неједнакост

$$f(x) > 0$$

за $x\epsilon(0, c)$.

Користићемо Маклоренове полиноме тригонометријских функција cos и sin у циљу одређивања навиших и нанижих полиномских апроксимација МТП функција. Користимо ознаку $T_k^{\varphi,a}(x)$ за Тејлоров полином степена *k* функције φ у тачки a и специјално за $a = 0$ такав полином уобичајено називамо Маклоренов полином.

$$T_{2n}^{\cos,0}(x) = \sum_{k=0}^{n} (-1)^k \frac{x^{2k}}{(2k)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!}$$

$$T_{2n+1}^{\sin,0}(x) = \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^{n+1} \frac{x^{2n+1}}{(2n+1)!}$$

за $n\epsilon N_0$ и $x\epsilon(0, c)$.

Теорема 1:

За Маклоренове полиноме одговарајућег степена *cos* и *sin* функција важе неједнакости:

$$\left\{\begin{array}{ccccc} T_{4s+2}^{\cos,0}(x)=\sum_{i=0}^{2s+1}\frac{(-1)^i}{(2i)!}x^{2i} & < \cos x < & \sum_{i=0}^{2s}\frac{(-1)^i}{(2i)!}x^{2i}=T_{4s}^{\cos,0}(x) \\ T_{4r+3}^{\sin,0}(x)=\sum_{i=0}^{2r+1}\frac{(-1)^i}{(2i+1)!}x^{2i+1} & < \sin x < & \sum_{i=0}^{2r}\frac{(-1)^i}{(2i+1)!}x^{2i+1}=T_{4r+1}^{\sin,0}(x) \end{array}\right\} \quad (r,s\in N_0),$$

за реалне вредности аргумента x.

Теорема 2:

За ма коју МТП функцију

$$f(x)=\sum_{i=1}^{n}\alpha_i x^{p_i}cos^{q_i}x\ sin^{r_i}x$$

постоји полином P као нанижа полиномска апроксимација МТП функције *f* таква да важи

$$f(x) > P(x)$$

за вредности аргумента $x\epsilon(0,c)$.

Задатак:

За погодно изабрану МТП функцију $f:(0,c)\to R$ доказати МТП неједнакости $f(x)>0$ над $(0,c)$, одређујући позитивну нанижу полиномску апроксимацију $P(x)>0$ над $(0,c)$.

Напомена:

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

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

Решење:

За одређивање наниже полиномске апроксимације *P(x)* користићемо **метод директног поређења**.

Метод директног поређења се користи за класу простих МТП функција облика

$$f(x) = \sum_{i=1}^{n} \alpha_i x^{p_i} cos^{q_i} x \; + \sum_{j=1}^{m} \beta_i x^{p_j} \, sin^{r_j} x$$

за $\alpha_i, \beta_i \epsilon R \backslash \{0\}$ и $p_i,\ q_i,\ p_j, r_j \epsilon N_0 \{i = 1, \dots,\ n \wedge j = 1, \dots,\ m\}$ за вредности аргумента $x\epsilon(0, c)$, при стандардној вредности $c = \pi/2$. У циљу одређивања полинома *P(x)* могуће је користити процене:

$$\left\{ \begin{array}{ll} \alpha_i > 0: & \cos x > T_{4s+2}^{\cos,0}(x), \\ \alpha_i < 0: & \cos x < T_{4s}^{\cos,0}(x), \\ \beta_j > 0: & \sin x > T_{4r+3}^{\sin,0}(x), \\ \beta_j < 0: & \sin x < T_{4r+1}^{\sin,0}(x); \end{array} \right\}$$

за $x\epsilon(0, c)$.

Напомена 1 - Ограничење методе директног поређења:

Приметимо да

$$cosx > T_{4s+2}^{cos,0}(x)$$

и при том $T_{4s+2}^{cos,0}(x)$ има јединствен корен $c_s \in (0, \pi/2)$. Стога за сабирке са позитивним коефицијентима уз парне степене $cos^{2l} x$ $(q_i = 2l)$ важи

$$cos^{2l} x \; \geq \; (T_{4s+2}^{cos,0}(x))^{2l}$$

Напомена 2 - Превазилажење ограничења методе директног поређења:

Остварује се трансформацијом просте МТП функције тако што се за сабирке са позитивним коефицијентима уз парне степене $cos^{2l} x$ $(q_i = 2l)$ примени трансформација

$$\cos^{2\ell} x = (1 - \sin^2 x)^{\ell} = \sum_{k=0}^{\ell} \binom{\ell}{k} (-1)^k \sin^k x \, .$$

Да бисмо израчунали Маклоренов полином степена *n* користимо функцију *maclaurin*, којој прослеђујемо врсту функцију *f* и степен *n*.

```
x = sp.symbols('x')
```

```
def maclaurin(f, n):
    sign = -1
    if f == 'cos':
        n = (int)(n / 2)
        T = sp.Poly(1, x) # inicijalizujemo na T = 1
        for k in range(1, n+1):
            T = T.add(sp.Poly(sign**k * x**(2*k) / math.factorial((2*k))))

    elif f == 'sin':
        n = (int)((n - 1) / 2)
        T = sp.Poly(x, x) # inicijalizujemo na T = x
        for k in range(1, n+1):
            T = T.add(sp.Poly(sign**k * x**(2*k+1) / math.factorial((2*k+1))))

    return T
```

Користећи функцију која је детаљно објашњена у другом пројектном задатку, сада само оптималније написана, на основу Штурмове теореме одређујемо да ли полиномска функција има нуле на задатом сегменту. Уколико на датом сегменту постоје нуле, повратна вредност функције је *true*, уколико нема, повратна вредност је *false*.

```
def ima_nule(p):
    # pravljenje sturmovog niza
    Pdiff = sp.diff(P, x)
    Q = sp.gcd(P, Pdiff)
    P0 = sp.div(P, Q)[0]
    P1 = sp.diff(P0, x)
    p_arr = []
    p_arr.append(P0)
    p_arr.append(P1)
    while sp.degree(P0) != 1:
        p_i = -sp.rem(P0, P1)
        P0 = P1
        P1 = p_i
        p_arr.append(p_i)
```

```
    # odredjujemo broj nula na intervalu [a, b]
    Va = 0
    Vb = 0
    a_last_sign = -1
    b_last_sign = -1
```

```
    for p in p_arr:
        if sp.degree(p) != 0:
            pp = sp.Poly(p)
            p_a = pp.subs(x, a)
            p_b = pp.subs(x, b)
        else:
            p_a = p.as_expr()
            p_b = p.as_expr()

        if p_a > 0:
            if a_last_sign == -1:
                a_last_sign = 1
            elif a_last_sign == 0: #poslednje je bio negativan broj
                Va = Va + 1 #doslo je do promene
                a_last_sign = 1 #azuriramo znak
        elif p_a < 0:
            if a_last_sign == -1:
                a_last_sign = 0
            elif a_last_sign == 1: #poslednje je bio negativan broj
                Va = Va + 1 #doslo je do promene
                a_last_sign = 0 #azuriramo znak

        if p_b > 0:
            if b_last_sign == -1:
                b_last_sign = 1
            elif b_last_sign == 0: #poslednje je bio negativan broj
                Vb = Vb + 1 #doslo je do promene
                b_last_sign = 1 #azuriramo znak
        elif p_b < 0:
            if b_last_sign == -1:
                b_last_sign = 0
            elif b_last_sign == 1: #poslednje je bio negativan broj
                Vb = Vb + 1 #doslo je do promene
                b_last_sign = 0 #azuriramo znak

```

```

    N = Va - Vb
    print("Broj nula: N = " + str(N))
    if N > 0:
        return True
    else:
        return False
```

Имајући ове две функције, можемо да напишемо функцију која представља метод директног поређења.

Пре свега треба пронаћи вредности α и β, тј. одредити знак синуса и косинуса. То радимо уз помоћ метода *find_alfa, find_beta* које као аргумент примају функцију, а повратна вредност им је вредност α и β. Те вредности уноси корисник због немогућности окружења да то самостално одреди. Након тога обављамо поређење и извршавамо одговарајуће замене у циљу одређивања одговарајућихј нанижних апроксимација функције.

Позивањем функције *ima_nule* одређујемо да ли је прослеђени пар *k1,k2* одговарајући.

- Уколико функција има нуле, значи да није позитивна на целом сегменту и да није одговарајућа апроксимација.
- Уколико нема нуле, посматрамо вредност функције
    - Уколико је негативна – апроксимација није одоварајућа
    - Уколико је позитивна – апроксимација је одговарајућа

```
def ispitaj(f, k1, k2):
    alfa = find_alfa(f)
    beta = find_beta(f)

    if alfa > 0: # pozitivno je -> gledamo od cega je cos veci
        cos_index = 4 * k1 + 2
    elif alfa < 0: # negativno je -> gledamo od cega je cos manji
        cos_index = 4 * k1

    if beta > 0: # pozitivno je -> gledamo od cega je sin veci
        sin_index = 4 * k2 + 3
    elif beta < 0: # negativno je -> gledamo od cega je sin manji
        sin_index = 4 * k2 + 1

    p = f.subs(sp.cos(x), maclaurin("cos", cos_index).as_expr())
    p = p.subs(sp.sin(x), maclaurin("sin", sin_index).as_expr())


    if ima_nule(p):
        # AKO FUNKCIJA IMA NULE NA DATOM SEGMENTU - NE DOKAZUJE POZITIVNOST
        print("[NOT OK] P[" + str(k1) + "," + str(k2)
              + "](x) ne dokazuje pozitivnost MTP funkcije nad (0, Pi/2) jer ima nule na tom segmentu")
    else:
        # AKO FUNKCIJA NEMA NULE - ODREDJUJEMO ZNAK FUNKCIJE
        fSign= sp.Poly(p).subs(x, 1)
        if fSign < 0:# AKO JE NEGATIVNA - NE DOKAZUJE POZITIVNOST
            print("[NOT OK] P[" + str(k1) + "," + str(k2)
              + "](x) ne dokazuje pozitivnost MTP funkcije nad (0, Pi/2) jer je negativna na zadatom domenu")
        else: # AKO JE POZITIVNA - DOKAZUJE POZITIVNOST
            print("[OK] P[" + str(k1) + "," + str(k2)
              + "](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).")

    return p
```

У главном делу програма уносимо у поље *f* вредност функције чију апроксимацију желимо да нађемо и позивамо функцију *ispitaj*. На крају добијене вредности представљамо на графику.

```python
def find_alfa(f): #uz cos
    return ??

def find_beta(f): # uz sin
    return ??

f = ???
```

```python
p1 = ispitaj(f, 0, 0)
p2 = ispitaj(f, 0, 1)
p3 = ispitaj(f, 1, 0)
p4 = ispitaj(f, 1, 1)

graf0 = sp.plot(f, xlim=[-1, 2], ylim=[-1.25, 5], show=False, label="f(x)")
graf1 = sp.plot(p1.as_expr(), xlim=[-1, 2], ylim=[-1.25, 5], show=False, label="$P_{0,0}$")
graf2 = sp.plot(p2.as_expr(), xlim=[-1, 2], ylim=[-1.25, 5], show=False, label="$P_{0,1}$")
graf3 = sp.plot(p3.as_expr(), xlim=[-1, 2], ylim=[-1.25, 5], show=False, label="$P_{1,0}$")
graf4 = sp.plot(p4.as_expr(), xlim=[-1, 2], ylim=[-1.25, 5], show=False, label="$P_{1,1}$")

graf0.append(graf1[0])
graf0.append(graf2[0])
graf0.append(graf3[0])
graf0.append(graf4[0])

graf0.legend = True

graf0.show()
```

Пример 1

У првом примеру узимамо следеће вредности:

$$f(x) = x^3 \sin x - x \cos^3 x + x - \frac{3}{2}x^3 + \frac{3}{32}x^4$$

Унос у програм:

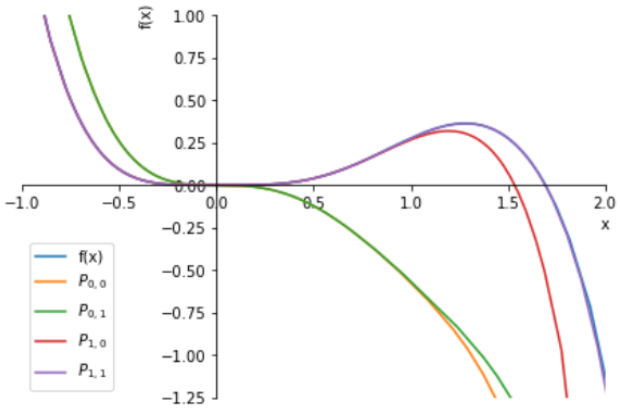
```python
def find_alfa(f): #uz cos
    return -1

def find_beta(f): # uz sin
    return 1

f = x**3 * sp.sin(x) - x * sp.cos(x)**3 + x - 3/2 * x**3 + 3/32 * x**4
```

Излаз програма:

```
Broj nula: N = 0
[NOT OK] P[0,0](x) ne dokazuje pozitivnost MTP funkcije nad (0, Pi/2) jer je negativna na zadatom domenu
Broj nula: N = 0
[NOT OK] P[0,1](x) ne dokazuje pozitivnost MTP funkcije nad (0, Pi/2) jer je negativna na zadatom domenu
Broj nula: N = 1
[NOT OK] P[1,0](x) ne dokazuje pozitivnost MTP funkcije nad (0, Pi/2) jer ima nule na tom segmentu
Broj nula: N = 0
[OK] P[1,1](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).
```



Пример 2

У другом примеру узимамо следеће вредности:

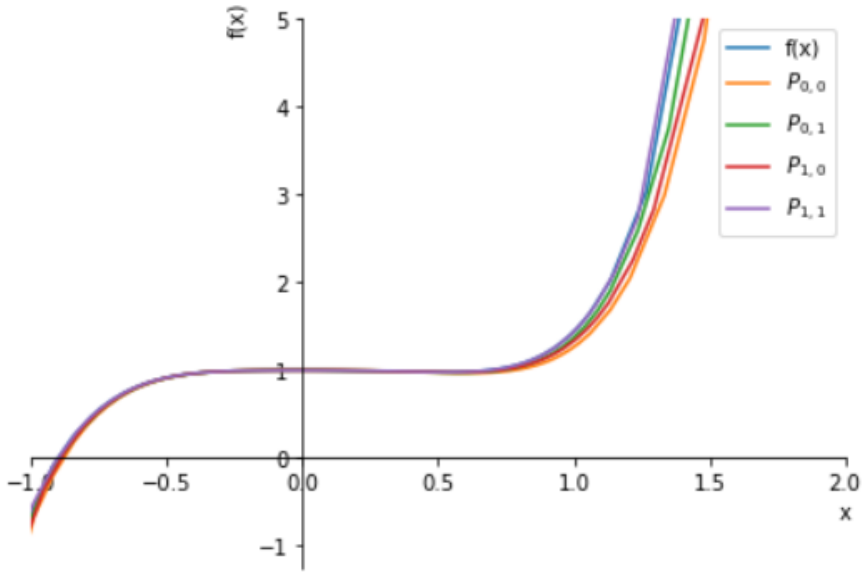
$$g(x) = -\frac{4}{6}\,x^3\,\sin(x) - 3\,x^2 \cdot \frac{\cos(x)}{20} + x^5 + \frac{8}{98}\,x^2 + 1$$

Унос у програм:

```
:
1  def find_alfa(f): # uz cos
2      return -1
3
4  def find_beta(f): # uz sin
5      return -1
6
7  f = -4/6*x**3*sp.sin(x) - 3*x**2 * sp.cos(x)/20 + x**5 + 8/98*x**2 + 1
```

Излаз програма:

```
Broj nula: N = 0
[OK] P[0,0](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).
Broj nula: N = 0
[OK] P[0,1](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).
Broj nula: N = 0
[OK] P[1,0](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).
Broj nula: N = 0
[OK] P[1,1](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).
```



Пример 3

У трећем примеру узимамо следеће вредности:

$$f(x) = x^3 \sin(x) - \frac{x}{5} \cos^3(x) + x \frac{6}{9} - \frac{6}{5} x^3 + \frac{9}{59} x^2$$

Унос у програм:

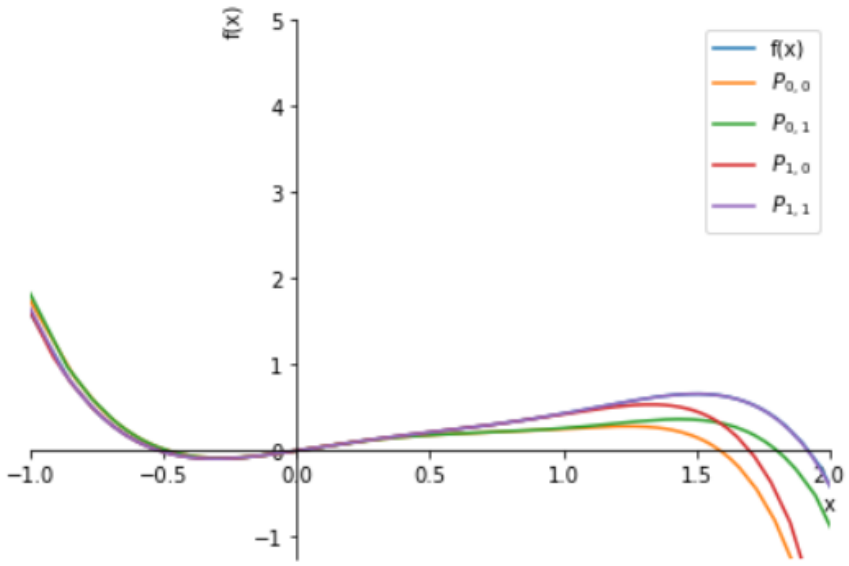
```
def find_alfa(f): # uz cos
    return -1

def find_beta(f): # uz sin
    return 1

f = x**3 * sp.sin(x) - x/5 * sp.cos(x)**3 +x*6/9 - 6/5*x**3 + 9/59*x**2
```

Излаз програма:

```
Broj nula: N = 0
[OK] P[0,0](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).
Broj nula: N = 0
[OK] P[0,1](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).
Broj nula: N = 0
[OK] P[1,0](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).
Broj nula: N = 0
[OK] P[1,1](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).
```



Пример 4

У четвртом примеру узимамо следеће вредности:

$$f(x) = x^6 \sin(x) - \frac{x}{5}\cos^3(x) - \frac{6}{5}x^3 + \frac{5}{9} - x^4 + 1$$

Унос у програм:

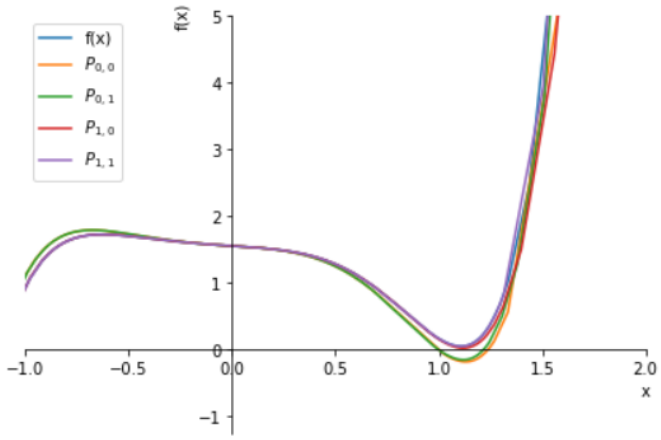
```
def find_alfa(f): #uz cos
    return -1

def find_beta(f): # uz sin
    return 1

f = x**6 * sp.sin(x) - x/5 * sp.cos(x)**3 - 6/5* x**3 + 5/9 - x**4 + 1
```

Излаз програма:

```
Broj nula: N = 2
[NOT OK] P[0,0](x) ne dokazuje pozitivnost MTP funkcije nad (0, Pi/2) jer ima nule na tom segmentu
Broj nula: N = 2
[NOT OK] P[0,1](x) ne dokazuje pozitivnost MTP funkcije nad (0, Pi/2) jer ima nule na tom segmentu
Broj nula: N = 0
[OK] P[1,0](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).
Broj nula: N = 0
[OK] P[1,1](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).
```



Прилог

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

# In[1]:


import numpy as np
import math
import sympy as sp
import matplotlib.pyplot as plt

from colorama import Fore


# In[2]:


x = sp.symbols('x')


# In[3]:


def maclaurin(f, n):
    sign = -1
    if f == 'cos':
        n = (int)(n / 2)
        T = sp.Poly(1, x) # inicijalizujemo na T = 1
        for k in range(1, n+1):
            T = T.add(sp.Poly(sign**k * x**(2*k) / math.factorial((2*k))))

    elif f == 'sin':
        n = (int)((n - 1) / 2)
        T = sp.Poly(x, x) # inicijalizujemo na T = x
        for k in range(1, n+1):
            T = T.add(sp.Poly(sign**k * x**(2*k+1) /
math.factorial((2*k+1))))

    return T


# In[4]:


def ima_nule(P):
    # pravljenje sturmovog niza
    Pdiff = sp.diff(P, x)
    Q = sp.gcd(P, Pdiff)
    P0 = sp.div(P, Q)[0]
    P1 = sp.diff(P0, x)
    p_arr = []
    p_arr.append(P0)
```

```
p_arr.append(P1)
while sp.degree(P0) != 1:
    p_i = -sp.rem(P0, P1)
    P0 = P1
    P1 = p_i
    p_arr.append(p_i)

# odredjujemo broj nula na intervalu [a, b]
a = 0.001
b = np.pi / 2
Va = 0
Vb = 0
a_last_sign = -1
b_last_sign = -1
for p in p_arr:
    if sp.degree(p) != 0:
        pp = sp.Poly(p)
        p_a = pp.subs(x, a)
        p_b = pp.subs(x, b)
    else:
        p_a = p.as_expr()
        p_b = p.as_expr()

    if p_a > 0:
        if a_last_sign == -1:
            a_last_sign = 1
        elif a_last_sign == 0: #poslednje je bio negativan broj
            Va = Va + 1 #doslo je do promene
            a_last_sign = 1 #azuriramo znak
    elif p_a < 0:
        if a_last_sign == -1:
            a_last_sign = 0
        elif a_last_sign == 1: #poslednje je bio negativan broj
            Va = Va + 1 #doslo je do promene
            a_last_sign = 0 #azuriramo znak

    if p_b > 0:
        if b_last_sign == -1:
            b_last_sign = 1
        elif b_last_sign == 0: #poslednje je bio negativan broj
            Vb = Vb + 1 #doslo je do promene
            b_last_sign = 1 #azuriramo znak
    elif p_b < 0:
        if b_last_sign == -1:
            b_last_sign = 0
        elif b_last_sign == 1: #poslednje je bio negativan broj
            Vb = Vb + 1 #doslo je do promene
            b_last_sign = 0 #azuriramo znak


N = Va - Vb
print("Broj nula: N = " + str(N))
if N > 0:
    return True
else:
    return False
```

```
# alfa uz cos, beta uz sin


# In[5]:


def ispitaj(f, k1, k2):
    alfa = find_alfa(f)
    beta = find_beta(f)

    if alfa > 0: # pozitivno je -> gledamo od cega je cos veci
        cos_index = 4 * k1 + 2
    elif alfa < 0: # negativno je -> gledamo od cega je cos manji
        cos_index = 4 * k1

    if beta > 0: # pozitivno je -> gledamo od cega je sin veci
        sin_index = 4 * k2 + 3
    elif beta < 0: # negativno je -> gledamo od cega je sin manji
        sin_index = 4 * k2 + 1

    p = f.subs(sp.cos(x), maclaurin("cos", cos_index).as_expr())
    p = p.subs(sp.sin(x), maclaurin("sin", sin_index).as_expr())


    if ima_nule(p):
        # AKO FUNKCIJA IMA NULE NA DATOM SEGMENTU - NE DOKAZUJE POZITIVNOST
        print("[NOT OK] P[" + str(k1) + "," + str(k2)
              + "](x) ne dokazuje pozitivnost MTP funkcije nad (0, Pi/2) jer
ima nule na tom segmentu")
    else:
        # AKO FUNKCIJA NEMA NULE - ODREDJUJEMO ZNAK FUNKCIJE
        fSign= sp.Poly(p).subs(x, 1)
        if fSign < 0:# AKO JE NEGATIVNA - NE DOKAZUJE POZITIVNOST
            print("[NOT OK] P[" + str(k1) + "," + str(k2)
              + "](x) ne dokazuje pozitivnost MTP funkcije nad (0, Pi/2) jer
je negativna na zadatom domenu")
        else: # AKO JE POZITIVNA - DOKAZUJE POZITIVNOST
            print("[OK] P[" + str(k1) + "," + str(k2)
              + "](x) dokazuje pozitivnost MTP funkcije nad (0, Pi/2).")

    return p


# ## Primer 1

# In[ ]:


def find_alfa(f): #uz cos
    return -1

def find_beta(f): # uz sin
    return 1

f = x**3 * sp.sin(x) - x * sp.cos(x)**3 + x - 3/2 * x**3 + 3/32 * x**4
```

```
# ## Primer 2

# In[20]:


def find_alfa(f): # uz cos
    return -1

def find_beta(f): # uz sin
    return -1

f = -4/6*x**3*sp.sin(x) - 3*x**2 * sp.cos(x)/20 + x**5 + 8/98*x**2 + 1


# ## Primer 3

# In[30]:


def find_alfa(f): # uz cos
    return -1

def find_beta(f): # uz sin
    return 1

f = x**3 * sp.sin(x) - x/5 * sp.cos(x)**3 +x*6/9 - 6/5*x**3 + 9/59*x**2


# ## Primer 4

# In[33]:


def find_alfa(f): #uz cos
    return -1

def find_beta(f): # uz sin
    return 1

f = x**6 * sp.sin(x) - x/5 * sp.cos(x)**3 - 6/5* x**3 + 5/9 - x**4 + 1


# # GLAVNI PROGRAM

# In[34]:


p1 = ispitaj(f, 0, 0)
p2 = ispitaj(f, 0, 1)
p3 = ispitaj(f, 1, 0)
p4 = ispitaj(f, 1, 1)

graf0 = sp.plot(f, xlim=[-1, 2], ylim=[-1.25, 5], show=False, label="f(x)")
graf1 = sp.plot(p1.as_expr(), xlim=[-1, 2], ylim=[-1.25, 5], show=False,
label="$P_{0,0}$")
```

```
graf2 = sp.plot(p2.as_expr(), xlim=[-1, 2], ylim=[-1.25, 5], show=False,
label="$P_{0,1}$")
graf3 = sp.plot(p3.as_expr(), xlim=[-1, 2], ylim=[-1.25, 5], show=False,
label="$P_{1,0}$")
graf4 = sp.plot(p4.as_expr(), xlim=[-1, 2], ylim=[-1.25, 5], show=False,
label="$P_{1,1}$")

graf0.append(graf1[0])
graf0.append(graf2[0])
graf0.append(graf3[0])
graf0.append(graf4[0])

graf0.legend = True

graf0.show()
```