informacijos
sistemų
katedra
1922

KAUNO TECHNOLOGIJOS UNIVERSITETAS
Informatikos fakultetas

P170B115 Skaitiniai metodai ir algoritmai

1 LABORATORINIS DARBAS

2023-10-26

Atliko: IFF-1/6 gr. studentas Lukas Kuzmickas

Priėmė: Doc. Andrius Kriščiūnas

KAUNAS, 2023

Turinys

1.1. Užduotis (10 variantas).

Pagal vidko kodą susirandame mūsų užduoties variantą, t.y. 10 variantas. Pirmosios dalies funkcijos ir metodai (1 pav.).

10	$0.25x^5 + 0.68x^4 - 1.65x^3 - 5.26x^2 - 1.91x + 1.36$	$e^{-x}\cos(x)\sin(x^2 - 1)\,; 7 \le x \le 8$	1, 4

1 pav. Pirmosios dalies daugianaris ir funkcija ir tikslinimo metodai.

Surandame ir mūsų antros dalies funkciją ir duotą intervalą, pagal užduoties variantą (2 pav.).

10	$-79\cos(x) - 11 - 4x$	$-10 \le x \le 0$

2 pav. Antrosios dalies funkcija ir intervalas.

Patikriname, kokius tikslinimo metodus, mums reikės taikyti sprendžiant uždavinius (3 pav.).

Metodo Nr.	Metodo pavadinimas
1	Stygų
2	Pusiaukirtos
3	Niutono (liestinių)
4	Kvazi-Niutono (kirstinių)

3 pav. Tikslinimo metodai.

Mums priklauso Stygų ir Kvazi-Niutono metodai.

1.2. 1 dalis (5 balai)

a) daugianaris $f(x) = 0$;
b) transcendentinė funkcija $g(x) = 0$.

Pirmosios dalies kodas

Main.py

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import fsolve

def f_isvestine(x):

    # Čia apibrėžiama funkcijos išvestinė f'(x)

    return 1.25 * x**4 + 2.72 * x**3 - 4.95 * x**2 - 10.52 * x - 1.91

def g_isvestine(x):

    return (

        -np.exp(-x) * (np.cos(x)**2) * (2 * x * np.sin(x**2 - 1) +
np.cos(x**2 - 1))

    )



# Define the function f(x)

def f(x):

    return (

        0.25 * x**5 +

        0.68 * x**4 -

        1.65 * x**3 -

        5.26 * x**2 -

        1.91 * x +

        1.36



    )



def g(x):
```

```
    return (
        np.exp(-x) * np.cos(x) * np.sin(x**2 - 1)
    )
#grubus_x = [-22.08, 22.08]
#grubus_y = [f(x_val) for x_val in grubus_x]
#tikslus_x = [0, 5.58]
#tikslus_y = [f(x_val) for x_val in tikslus_x]
# Create an array of x-values
#x = np.linspace(-23, 23, 1000)
# Calculate the corresponding y-values
#y = f(x)
# Create the plot
#plt.figure(figsize=(8, 6))
#plt.plot(x, y, label='f(x)', color='blue')
#plt.xlabel('x')
#plt.ylabel('f(x)')
#plt.title('f(x) daugianaris, grubus intervalas [-22.08;22.08]')
#plt.grid(True)
#plt.scatter(grubus_x, grubus_y, color='red', marker='o', label='Grubus
įvertinimas')
#plt.scatter(tikslus_x, tikslus_y, color='green', marker='o',
label='Tikslus įvertinimas')
#plt.legend()
#plt.show()
# Pradiniai intervalo taškai
x_start_f = -22.08
x_end_f = 22.08
x_start_g = 7
x_end_g = 8
# Žingsnio dydis
step_size = 0.1  # Pasirinkite tinkamą žingsnio dydį
```

```
def findIntervals(function, x_start, x_end, step):
 intervals = []
 x_prev = x_start
 sign_prev = function(x_prev) > 0
 x = x_start + step
 while x < x_end:
  sign_current = function(x) > 0
  if sign_current != sign_prev:
   intervals.append((x_prev, x))
   sign_prev = sign_current
   x_prev = x
  x += step

 return intervals
intervals = []
intervals = findIntervals(g, x_start_g, x_end_g, step_size)
print("Funkcijos intervalai:")
for interval in findIntervals(g, x_start_g, x_end_g, step_size):
    print(interval)
def chordMethod(function, intervals, tolerance=1e-6, max_iterations=10000):
 results = []
 for interval in intervals:
  iteration = 0
  pradinis_artinys = interval[0]
  xn = interval[0]
  xn1 = interval[1]
  xm = 0
  while iteration < max_iterations:
      k = abs(function(xn) / function(xn1))
      xm = (xn + k*xn1) / (1 + k)
```

```
      if np.sign(function(xm)) == np.sign(function(xn)):
       xn = xm
      else:
       xn1 = xm
       iteration += 1
      if xm != 0:
       if abs(function(xm)) < tolerance and (xn1-xn) / abs(xm) < tolerance:
        break
       elif abs(function(xm)) < tolerance and (xn1-xn) < tolerance:
        break
  results.append(xm)
  print("Pradinis artinys: " , pradinis_artinys)
  print("Funkcijos reikšmė šaknyje: ", function(xm))
  print("Tikslumas: ", tolerance)
  print("Rasta šaknis: ", xm)
  print("Iteracijų skaičius: ", iteration)
 return results
#chordMethod(g, intervals)
def secantMethod(function, intervals, tolerance=1e-6,
max_iterations=10000):
 results = []
 for interval in intervals:
  iteration = 0
  pradinis_artinys = interval[0]
  x0 = interval[0]
  x1 = interval[1]
  while abs(x1 - x0) > tolerance and iteration < max_iterations:
   x_next = x1 - ((x1 - x0) / (function(x1) - function(x0))) * function(x1)
   x0 = x1
   x1 = x_next
```

```
    iteration += 1
    if iteration == max_iterations:
     print("Buvo pasiektas maksimalus iteraciju skaicius")
    print("Pradinis artinys: " , pradinis_artinys)
    print("Funkcijos reikšmė šaknyje: ", function(x1))
    print("Tikslumas: ", tolerance)
    print("Rasta šaknis: ", x1)
    print("Iteracijų skaičius: ", iteration)
    results.append(x1)
  return results
secantMethod(g, intervals)
# Spausdiname šaknų intervalus
# #surašome koeficientus
# x0 = [0.25,
#         0.68,
#         1.65,
#         5.26,
#         1.91,
#         1.36]
# roots = np.roots(x0)
#
# print("Roots funkcijos rezultatai f(x):")
# print(roots)
#
#
# # Nustatome saknų intervalą
# saknu_intervalas = [7.1, 7.3, 7.5, 7.7, 7.8, 7.9]
#
# # Sukuriame daugianarį su g(x) kaip koeficientų reikšmėmis
# coefficients = [g(x) for x in saknu_intervalas]
```

```
#

# # Surandame šaknis

# roots = np.roots(coefficients)

#

# # Tikriname, ar šaknys yra arti nurodyto saknų intervalo

# for i, root in enumerate(roots):

#     if saknu_intervalas[i] <= root <= saknu_intervalas[i+1]:

#         print(f"Šaknis {root} yra intervalo ({saknu_intervalas[i]},
{saknu_intervalas[i+1]}) ribose.")

#     else:

#         print(f"Šaknis {root} nėra intervalo ({saknu_intervalas[i]},
{saknu_intervalas[i+1]}) ribose.")

delta = 0.1

eps = 1e-2

#print("g(x) funkcijos roots() rezultatai")

#koeficientai = [-11.5726691358786, 225763819.477356, 13421271.3492355, -
800849.09789271, 64435.9856506977, - 95.9874712193938]

#print(np.roots(koeficientai))

#for root in roots:

#    print(fsolve(g, root["xMin"], xtol=1e-12))
```

1.2.1. (tik lygčiai su daugianariu $f(x)$) Nustatykite daugianario $f(x)$ šaknų intervalą, taikydami „grubų“ ir tikslesnį įverčius. Grafiškai pavaizduokite apskaičiuotų šaknų intervalo galus.

Grubus įvertis :

$$f(x) = a_n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0, \qquad a_n > 0$$

Lygties f(x) = 0 šaknų grubus įvertis:

Intervalų apskaičiavimui naudojame intervalus:

a_5	a_4	a_3	a_2	a_1	a_0
0.25	**0.68**	**-1.65**	**-5.26**	**-1.91**	**1.36**

Daugianario eilės numeris: n=5.

$$R = 1 + \frac{\max\limits_{0 \le i \le 4}\{|0.68|, |-1.65|, |-5.26|, |-1.91|, |1.36|\}}{0.25} = 1 + \frac{5.26}{0.25} = 22.04$$

Grubus šaknies lygties šaknų intervalų įvertis (-22.04; 22.04).

Skaičiuojame tikslų įvertį teigiamoms šaknims:

Išrenkame absoliutine verte didžiausią neigiamą koeficientą

$$B = \max_{0 \le i \le 4}\{|-1.65|, |-5.26|, |-1.91|\} = 5.26$$

Apskaičiuojame didžiausią neigiamo koeficiento indeksą (nevertiname koeficiento prie aukščiausio laipsnio).

$$k = 5 - \max_{0 \le i \le 4}\{3,2,1\} = 5 - 3 = 2$$

Apskaičiuojame tikslesnio įverčio viršutinis rėžis:

$$\mathrm{R_{teig}} = 1 + \sqrt[2]{\frac{5.26}{0.25}} = 5{,}58$$

Skaičiuojame tikslų įverti neigiamoms reikšmėms, rasdami apatinį tikslų įvertį:

Nagrinėjame daugianarį *-f(-x),* kadangi mūsų daugianaris yra nelyginis, todėl koeficientai yra pakeičiami.

a_5	a_4	a_3	a_2	a_1	a_0
0.25	**-0.68**	**-1.65**	**5.26**	**-1.91**	**-1.36**

Apskaičiuojame R_{neig} .

$$B = \max_{0 \leq i \leq 4}\{|-0.68|, |-1.65|, |-1.91|, |-1.36|\} = 1.91$$

$$k = 5 - \max_{0 \leq i \leq 4}\{4,3,1,0\} = 5 - 4 = 1$$

$$\mathrm{R_{neig}} = 1 + \sqrt[1]{\frac{1.91}{0.25}} = 8{,}64$$

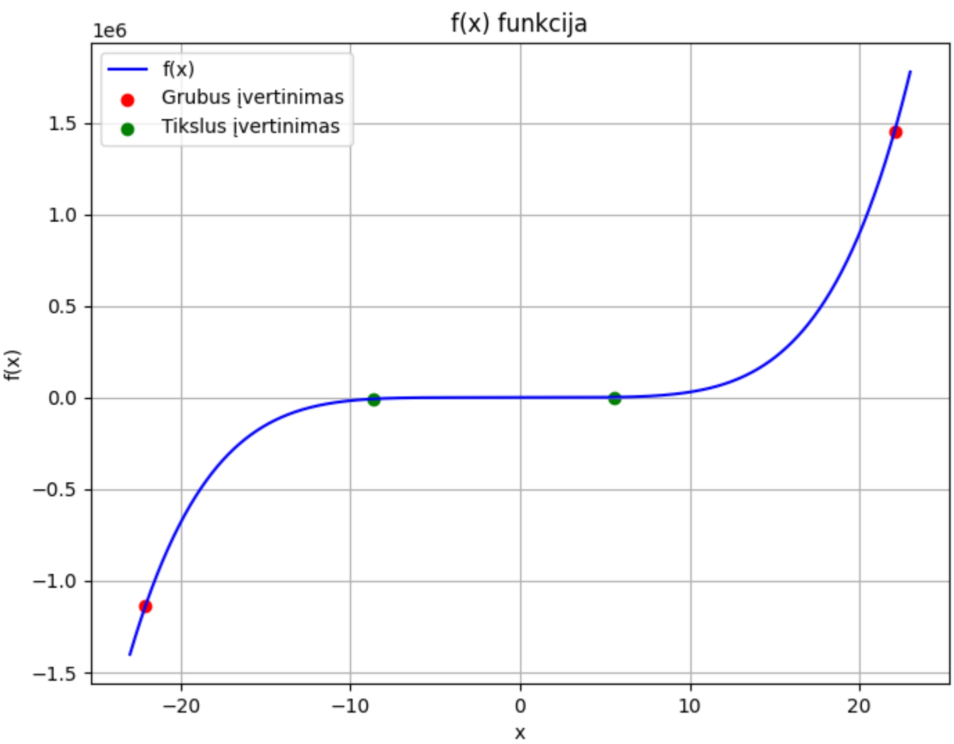
Galutinis rėžių įvertinimas:

$- min(22.04; 8{,}64) \leq x \leq \min(22.04; 5.58)$

$$\mathbf{-8{,}64 \leq x \leq 5.58}$$

Naudodami Python programavimo kalbą, mes galime nubraižyti šią funkciją ir pavaizduoti šių šaknų galus.

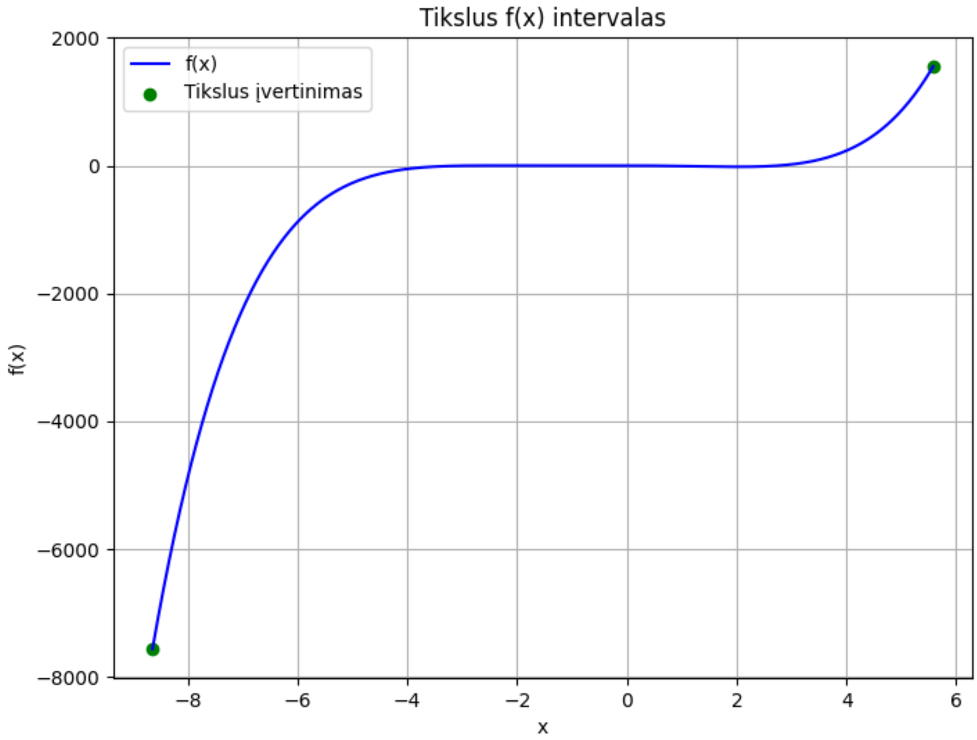
Pavaizduojame šiuos šaknų intervalus grubus atvejis ir tikslus atvejis (4 pav.).



4 pav. F(x) funkcijos grubus ir tikslus šaknų intervalas.

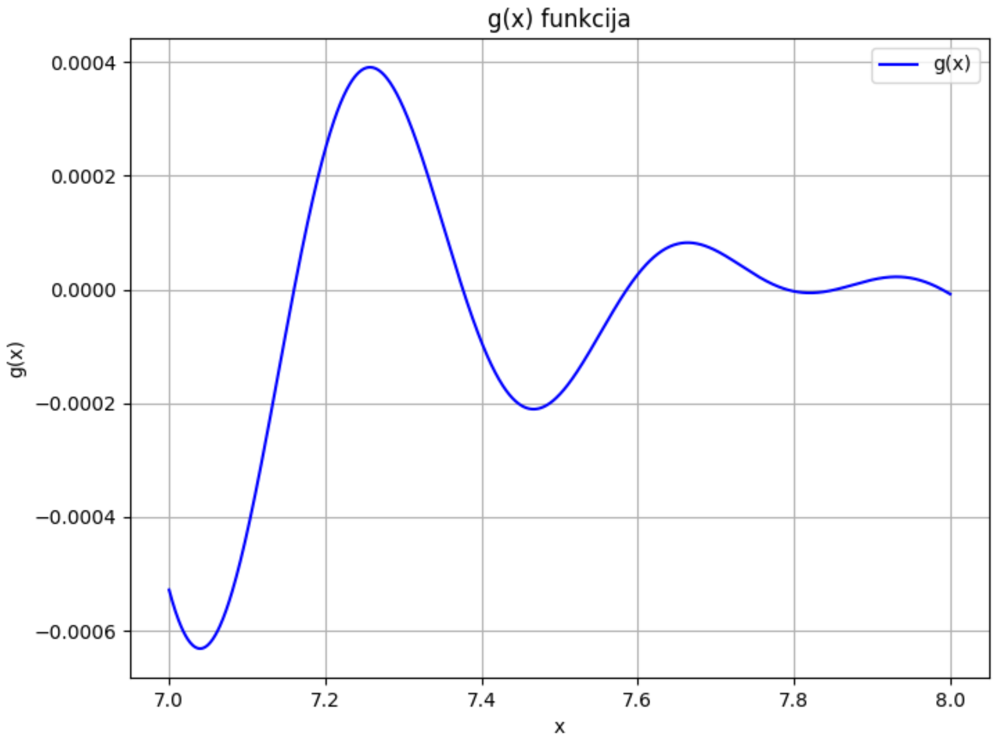
1.2.2. Daugianarį $f(x)$ grafiškai pavaizduokite nustatytame šaknų intervale. Funkciją $g(x)$ grafiškai pavaizduokite užduotyje nurodytame intervale. Esant poreikiui, grafikų ašis pakeiskite taip, kad būtų aiškiai matomos funkcijų šaknys.

Daugianarį f(x) grafiškai pavaizduojame nustatytame šaknų intervale (5 pav.).



5 pav. Grafinis f(x) funkcijos pavaizdavimas nustatytame intervale.

Funkciją g(x) grafiškai atvaizduojame jau duotame intervale (6 pav.).



6 pav. Grafiškas funkcijos g(x) atvaizdavimas.

1.2.3. Naudodami skenavimo algoritmą su nekintančiu skenavimo žingsniu atskirkite šaknų intervalus. Daugianariui skenavimo intervalas parenkamas pagal įverčių reikšmes, funkcija skenuojama užduotyje nurodytame intervale. Šaknies atskyrimo intervalai gali būti naudojami kaip pradiniai intervalai (artiniai) šaknų tikslinimui.

Naudojame skenavimo algoritmą su nekintančiu skenavimo žingsniu f(x) daugianario:

Bandome nustatyti daugianario f(x) šaknis, intervalo [-22.04; 22.04] ribose. Naudojame nekintantį skenavimo žingsnį. Ieškome vietų, kur funkcijos reikšmė keičiasi iš neigiamos į teigiamą arba atvirkščiai.

```
def findIntervals(function, x_start, x_end, step):
 intervals = []
 x_prev = x_start
 sign_prev = function(x_prev) > 0
 x = x_start + step
 while x < x_end:
  sign_current = function(x) > 0
  if sign_current != sign_prev:
   intervals.append((x_prev, x))
   sign_prev = sign_current
   x_prev = x
```

```
    x += step

  return intervals
```

Įvykdę kodą gauname, kad mūsų daugianario f(x) šaknies intervalai yra:

```
Funkcijos intervalai:
(-22.04, -2.7799999999999545)
(-2.7799999999999545, -1.8799999999999537)
(-1.8799999999999537, -0.979999999999953)
(-0.979999999999953, 0.4200000000000469)
(0.4200000000000469, 2.8200000000000482)
```

Naudojame skenavimo algoritmą su nekintančiu skenavimo žingsniu g(x) funkcijos:

Bandome nustatyti funkcijos g(x) šaknis, intervalo [7; 8] ribose. Naudojame nekintantį skenavimo žingsnį. Ieškome vietų, kur funkcijos reikšmė keičiasi iš neigiamos į teigiamą arba atvirkščiai.

Taikome ta patį algoritmą tik pakeisdami intervalą iš [7; 8] ir naudodami pačią funkcijos išraišką.

Gauname šaknų intervalus funkcijos g(x):

```
Funkcijos intervalai:
(7, 7.199999999999999)
(7.199999999999999, 7.399999999999999)
(7.399999999999999, 7.599999999999998)
(7.599999999999998, 7.799999999999997)
(7.799999999999997, 7.899999999999997)
(7.899999999999997, 7.9999999999999964)
```

1.2.4. Skenavimo metodu atskirtas daugianario ir funkcijos šaknis tikslinkite užduotyje nurodytais metodais. Užrašykite skaičiavimų pabaigos sąlygas. Skaičiavimų rezultatus pateikite lentelėje, kurioje nurodykite šaknies tikslinimui naudojamą metodą, pradinį artinį ar intervalą, gautą sprendinį (šaknį), funkcijos reikšmę šaknyje, tikslumą, iteracijų skaičių. Palyginkite, kuris metodas randa sprendinį su mažesniu iteracijų skaičiumi.

Daugianario šaknies tikslinimas:

Daugianario šaknies tikslinimui naudosime Stygų ir Kvazi-Niutono (kirstinių) metodus.

Stygų metodo taikymas:

```
def chordMethod(function, intervals, tolerance=1e-6, max_iterations=10000):
 results = []
 for interval in intervals:
  iteration = 0
  pradinis_artinys = interval[0]
  xn = interval[0]
  xn1 = interval[1]
  xm = 0
  while iteration < max_iterations:
      k = abs(function(xn) / function(xn1))
      xm = (xn + k*xn1) / (1 + k)
      if np.sign(function(xm)) == np.sign(function(xn)):
       xn = xm
      else:
       xn1 = xm
       iteration += 1
      if xm != 0:
       if abs(function(xm)) < tolerance and (xn1-xn) / abs(xm) < tolerance:
        break
       elif abs(function(xm)) < tolerance and (xn1-xn) < tolerance:
        break
  results.append(xm)
  print("Pradinis artinys: " , pradinis_artinys)
  print("Funkcijos reikšmė šaknyje: ", function(xm))
  print("Tikslumas: ", tolerance)
  print("Rasta šaknis: ", xm)
  print("Iteracijų skaičius: ", iteration)
 return results
```

Kvazi-Niutono metodo taikymas:

```
def secantMethod(function, intervals, tolerance=1e-6,
max_iterations=10000):
 results = []
 for interval in intervals:
  iteration = 0
  pradinis_artinys = interval[0]
  x0 = interval[0]
  x1 = interval[1]
  while abs(x1 - x0) > tolerance and iteration < max_iterations:
   x_next = x1 - ((x1 - x0) / (function(x1) - function(x0))) * function(x1)
   x0 = x1
   x1 = x_next
   iteration += 1
   if iteration == max_iterations:
    print("Buvo pasiektas maksimalus iteraciju skaicius")
   print("Pradinis artinys: " , pradinis_artinys)
   print("Funkcijos reikšmė šaknyje: ", function(x1))
   print("Tikslumas: ", tolerance)
   print("Rasta šaknis: ", x1)
   print("Iteracijų skaičius: ", iteration)
   results.append(x1)
 return results
```

1.2.5. Gautas šaknų reikšmes patikrinkite naudodami išorinius išteklius (pvz., funkcijas roots arba fzero, tinklapį wolframalpha.com ir t.t.) ir pateikite patikrinimo rezultatus.

Gautas šaknų reikšmes galime patikrinti naudodami išorinius išteklius. Kadangi visą kodą realizavome naudodami Python, patogu bus patikrinti naudojant funkcijas roots ir fsolve.

Paduodami funkcijai roots, savo daugianario koeficientus, mes gauname atsakymą.

Iš roots() funkcijos sužinome, kad mūsų daugianaris f(x), turi ir kompleksinių šaknų, bet vieną realią šaknį -2.79290844, kuri yra artima mūsų pirmai surastai šakniai.

Roots() rezultatai:

```
Roots funkcijos rezultatai f(x):
[-2.79290844+0.j          0.19872399+2.56899735j  0.19872399-
2.56899735j
 -0.16226977+0.51676308j -0.16226977-0.51676308j]
```

G(x) funkcijos šaknims rasti naudojame Fsolve() funkciją.

Fsolve() rezultatai:

```
Fsolve funkcijos rezultatai g(x):
Roots: [6.93713844 7.15999179 7.37611518 7.58608382 7.79039539
7.85398163]
```

Gauname tokius rezultatus:

Metodas	Funkcija	Pradinis artinys	Funkcijos reikšmė šaknyje	Tikslumas	Rasta šaknis	Iteracijų skaičius
Stygų	F(x)	-22.04	0.19552190545173675	1e-06	-2.8410757354794507	10000
Stygų	F(x)	-2.7799999999999545	2.220446049250313e-16	1e-06	-1.9049252098146157	9
Stygų	F(x)	-1.8799999999999537	0	1e-06	-1.0421914900810698	1
Stygų	F(x)	-0.979999999999953	0	1e-06	0.3480039755545356	1

Stygų	F(x)	0.4200000000000469	1.0436096431476471e-14	1e-06	2.746252213995766	1
Stygų	G(x)	7	-4.505375817352618e-18	1e-06	7.159991791715734	10
Stygų	G(x)	7.199999999999999	-4.234494384754828e-19	1e-06	7.376115177451236	10000
Stygų	G(x)	7.399999999999999	-1.2505684447671525e-18	1e-06	7.5860838226726885	13
Stygų	G(x)	7.599999999999998	3.4133879793983884e-19	1e-06	7.790395395498617	18
Stygų	G(x)	7.799999999999997	1.8948117526616211e-19	1e-06	7.853981633974484	1
Stygų	G(x)	7.899999999999997	-5.386329945130668e-19	1e-06	7.989483905221656	1
Kirstinių	F(x)	-22.04	1.84297022087776e-14	1e-06	-2.867139489654614	7
Kirstinių	F(x)	-2.7799999999999545	-1.4408474413585282e-12	1e-06	-1.9049252098139522	5
Kirstinių	F(x)	-1.8799999999999537	-1.5232259897857148e-13	1e-06	-1.0421914900811433	8
Kirstinių	F(x)	-0.979999999999953	2.9531932455029164e-14	1e-06	-1.9049252098146292	12
Kirstinių	F(x)	0.4200000000000469	-2.886579864025407e-15	1e-06	0.34800397555453605	9
Kirstinių	G(x)	7	6.512618110312574e-13	1e-06	7.159991791807239	5
Kirstinių	G(x)	7.199999999999999	-6.640429128006808e-13	1e-06	7.376115177607585	5
Kirstinių	G(x)	7.399999999999999	8.294031075329004e-18	1e-06	7.586083822672693	6
Kirstinių	G(x)	7.599999999999998	-4.668647111139947e-15	1e-06	7.790395395510017	6
Kirstinių	G(x)	7.799999999999997	-4.1688484519377415e-15	1e-06	7.853981633961678	8
Kirstinių	G(x)	7.899999999999997	-1.6714367796808166e-16	1e-06	7.989483905221883	6
Roots()	F(x)				-2.79290844	
Fsolve()	G(x)				7.15999179	
Fsolve()	G(x)				7.37611518	
Fsolve()	G(x)				7.58608382	
Fsolve()	G(x)				7.79039539	
Fsolve()	G(x)				7.85398163	

Fsolve()	G(x)				6.93713844	

Iš lentelės galime pamatyti, kad Kvazi-Niutono metodas f(x) daugianariui ir g(x) funkcijai randą šaknį per mažesnį iteracijų skaičių, tačiau dalinai nukenčia pats tikslumas. Patys roots() ir fsolve() rezultatai dalinai atitinka mūsų jau surastas šaknis.

1.3. 2 dalis (5 balai)

2 dalis (5 balai). 3 lentelėje pateiktą funkciją $h(x)$ išskleiskite Teiloro eilute (TE) nurodyto intervalo vidurio taško aplinkoje. Nustatykite TE narių skaičių, su kuriuo visos TE šaknys esančios nurodytame intervale, skiriasi nuo funkcijos $h(x)$ šaknų ne daugiau negu |1e-4|. Tiek pateiktos funkcijos $h(x)$ šaknis, tiek TE šaknis raskite antru iš pirmoje dalyje realizuotų skaitinių metodų (Niutono arba Kvazi-Niutono, priklausomai nuo varianto). Darbo ataskaitoje pateikite:

1. tarpinius grafikus, kai drauge su pateikta funkcija $h(x)$ nurodytame intervale atvaizduojama TE, kai jos narių skaičius lygus 3, 4 ir 5.
2. grafiką, kuriame pavaizduotas reikalaujamą tikslumą užtikrinantis pagal TE sudarytas daugianaris, drauge pateikiant ir funkcijos $h(x)$ grafiką;
3. nustatytos reikalaujamą tikslumą užtikrinančios TE analitinę išraišką daugianario pavidalu;
4. grafikus, pagal kuriuos būtų galima įvertinti, kaip gerėjo sprendinys priklausomai nuo TE narių skaičiaus:
   a) grafikas, kuris nurodo visą randamų šaknų skaičių nagrinėjamame intervale (ox-TE eilė, oy – šaknų skaičius);
   b) atskiri grafikai kiekvienai šakniai, kuriuose oy ašyje pateikti tikslumo įverčiai tarp $h(x)$ apskaičiuotos šaknies ir artimiausios TE šaknies, o ox ašyje TE narių skaičiai.

7 pav. Antrosios dalies uždaviniai.

10	$-79\cos(x) - 11 - 4x$	$-10 \le x \le 0$

8 pav. Antrosios dalies funkcija ir intervalas.

Antrosios dalies kodas:

Main.py
```
import numpy as np
import matplotlib.pyplot as plt
import math
import sympy
class Root_with_differences:
    def __init__(self, root):
        self.root = root
        self.differences = []

    def graph_b(self):
        plt.figure()
        plt.plot(range(len(self.differences)), self.differences)
        plt.xlabel("TE eilė")
        plt.ylabel("Skirtumas tarp hx ir artimiausios TE šaknies")
        plt.title(f"{self.root} Šaknies pagerėjimo grafikas")
        plt.grid()
        plt.show()
def hx(x):
    return -79 * np.cos(x) - 11 - 4*x
def dhx(x):
    return 79 * np.sin(x) - 4
def root_intervals(f, min, max, h):
    intervals = []
    while min < max:
        if np.sign(f(min)) != np.sign(f(min + h)):
            plt.plot([min], [0], 'or')
            plt.plot([min + h], [0], 'og')
```

```
            intervals.append({"xMin": round(min, 2), "xMax": 
round(min + h, 2)})
        min += h
    return intervals
def secantMethod(function, intervals, tolerance=1e-6, 
max_iterations=10000):
 roots = []
 for interval in intervals:
  iteration = 0
  pradinis_artinys = interval['xMin']
  x0 = interval['xMin']
  x1 = interval['xMax']
  while abs(x1 - x0) > tolerance and iteration < max_iterations:
   x_next = x1 - ((x1 - x0) / (function(x1) - function(x0))) * 
function(x1)
   x0 = x1
   x1 = x_next
   iteration += 1
   if iteration == max_iterations:
    print("Buvo pasiektas maksimalus iteraciju skaicius")
   roots.append(x1)
 return roots
def newton(f, df, close_points, eps):
    roots = []
    for point in close_points:
        xi = point
        while math.fabs(f(xi)) > eps:
            xi = xi - (f(xi) / df(xi))
        roots.append(xi)
    return roots
def check_for_close_roots(f, roots, eps, min, max, step):
    x = sympy.symbols('x')
    df = f.diff(x)
    df_lamdified = sympy.lambdify(x, df, 'numpy')
    f_lambdified = sympy.lambdify(x, f, 'numpy')
    intervals = root_intervals(f_lambdified, min, max, step)
    close_points_arr = []
    for interval in intervals:
        close_points_arr.append(interval['xMin'])
    plt.clf()
    newton_roots = secantMethod(f_lambdified, intervals)
    count_close_roots = 0
    for root in roots:
        for newton_root in newton_roots:
            if math.fabs(newton_root - root) <= eps:
                count_close_roots += 1
                plt.plot([newton_root], [0], 'or')
                plt.plot([root], [0], 'og')
                break
    return count_close_roots
def get_all_roots(f, x_min, x_max, step):
    x = sympy.symbols('x')
    f_lambdified = sympy.lambdify(x, f, 'numpy')
    return len(root_intervals(f_lambdified, x_min, x_max, step))
def find_differences_between_roots(f, roots, eps, x_min, x_max, 
step):
    differences = []
    x = sympy.symbols('x')
    df = f.diff()
    df_lambdified = sympy.lambdify(x, df, 'numpy')
    f_lambdified = sympy.lambdify(x, f, 'numpy')
    intervals = root_intervals(f_lambdified, x_min, x_max, step)
```

```
    close_points = []
    for interval in intervals:
        close_points.append(interval['xMin'])
    newton_roots = secantMethod(f_lambdified, intervals)
    for root in roots:
        min = x_max-x_min
        for newton_root in newton_roots:
            temp = math.fabs(newton_root-root)
            if temp < min:
                min = temp
        if min == x_max-x_min:
            min = 0
        differences.append({"root": root, "min_diff": min})
    return differences

def taylor(function, x0, roots, eps, x_min, x_max, step):
    x, f, fp = sympy.symbols(('x', 'f', 'fp'))
    all_roots_found = []
    all_differences_for_roots = []
    for root in roots:
        all_differences_for_roots.append(Root_with_differences(root))
    x_vals = np.arange(x_min, x_max + step, step)
    f = function
    f_lambdified = sympy.lambdify(x, function, 'numpy')
    f_values = f_lambdified(x_vals)
    max_iteration = 200
    fp = f.subs(x, x0)
    i = 0
    while i < max_iteration + 1 and len(roots) !=
check_for_close_roots(fp, roots, eps, x_min, x_max, step):
        i += 1
        f = f.diff(x)
        fp = fp + f.subs(x, x0) / math.factorial(i) * (x - x0) ** i
        all_roots_found.append(get_all_roots(fp, x_min, x_max, step))
        differences = find_differences_between_roots(fp, roots, eps,
x_min, x_max, step)
        for difference in differences:
            for all_differences_for_root in
all_differences_for_roots:
                if difference["root"] ==
all_differences_for_root.root:

all_differences_for_root.differences.append(difference["min_diff"])
    fp_lambdified = sympy.lambdify(x, fp, 'numpy')
    fp_values = np.array([fp_lambdified(val) for val in x_vals])
    plt.plot(x_vals, f_values, label='-79 * np.cos(x) - 11 - 4*x')
    plt.plot(x_vals, fp_values, label=f'TE - {i}')
    plt.xlim([-10, 10])
    plt.ylim([-10, 10])
    plt.plot([x0], [0], 'om', label="mid")
    plt.legend()
    plt.grid()
    print(f"daugianario išraiška - {fp}")
    plt.show()
    graph_a(all_roots_found)
    for root_with_diff in all_differences_for_roots:
        root_with_diff.graph_b()
    return fp_lambdified
def graph_a(roots_count):
    plt.figure()
    plt.plot(range(len(roots_count)), roots_count)
    plt.xlabel("Teiloro eilutė")
```

```
    plt.ylabel("Šaknų kiekis")
    plt.title("Šaknų kiekio priklausomybe nuo Teiloro eilutės")
    plt.grid()
    plt.show()

eps = 1e-12
eps2 = 1e-4
dx = 0.01
h = 0.1
max = 0
min = -10
mid = (max + min) / 2
all_x = np.arange(min, max + dx, dx)
all_y = hx(all_x)

intervals = root_intervals(hx, min, max, h)
for item in intervals:
    print(f"Artinys : [{item['xMin']} ; {item['xMax']}]")

# niutono-kvazi metodas
h_function_roots = secantMethod(hx, intervals)
print("-79 * np.cos(x) - 11 - 4*x šaknys Kvazi-Niutono metodu")
for root in h_function_roots:
    print(root)

# TE
x, f = sympy.symbols(('x', 'f'))
f = -79 * sympy.cos(x) - 11 - 4*x
taylor(f, mid, h_function_roots, eps2, min, max, dx)
```

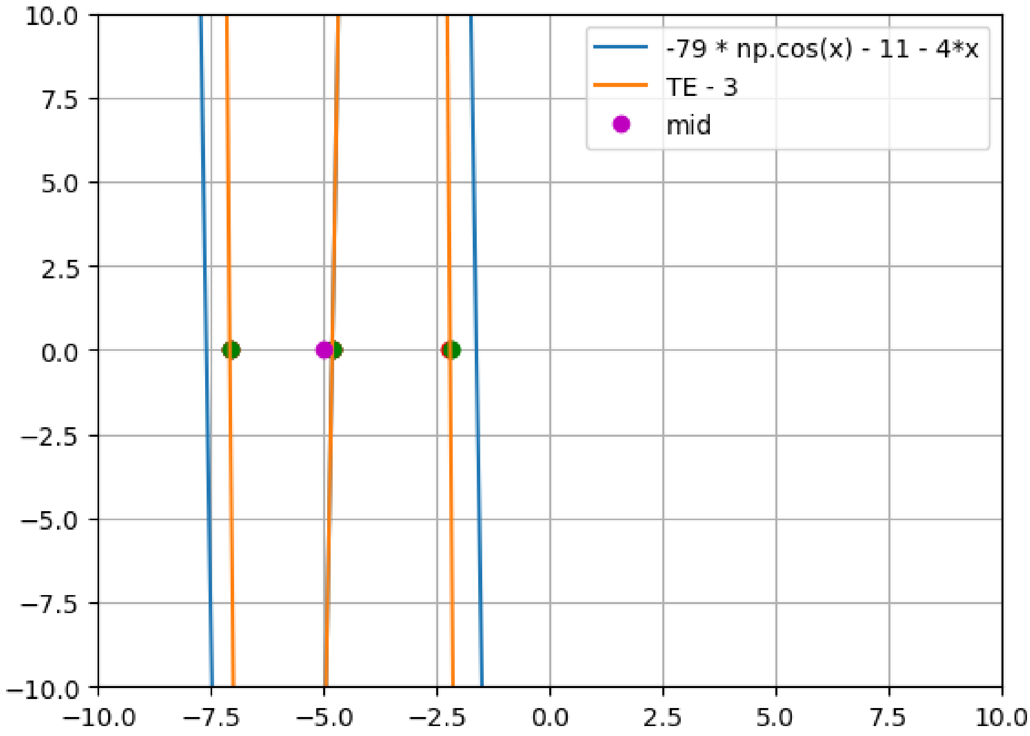
Naudodami kirstinių metodą, suraskime visas h(x) funkcijos šaknis (9 pav.).

```
Artinys : [-7.7 ; -7.6]
Artinys : [-4.9 ; -4.8]
Artinys : [-1.7 ; -1.6]
-79 * np.cos(x) - 11 - 4*x šaknys Kvazi-Niutono metodu
-7.605527209892827
-7.605582797928827
-7.605582760248838
-4.817348023907869
-4.817252134913555
-4.817252048172876
-1.627723051582828
-1.6276546171146777
-1.6276545741375057
```

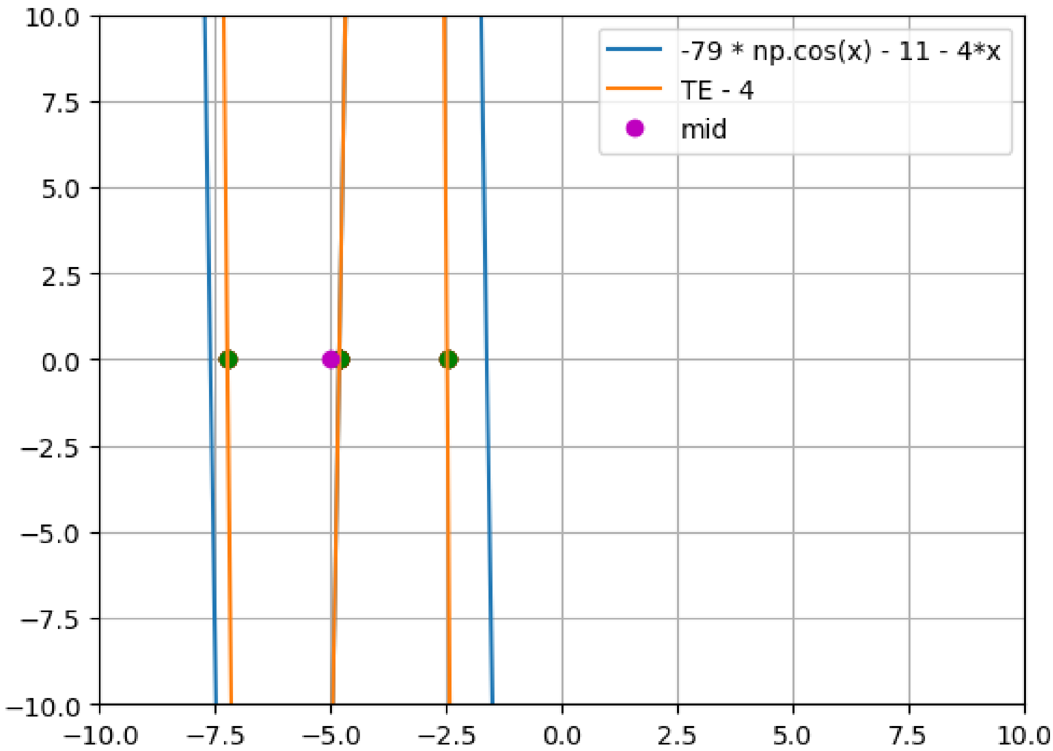
9 pav. Šaknys surastos Kvazi-Niutono metodu.

1.3.1. tarpinius grafikus, kai drauge su pateikta funkcija $h(x)$ nurodytame intervale atvaizduojama TE, kai jos narių skaičius lygus 3, 4 ir 5.

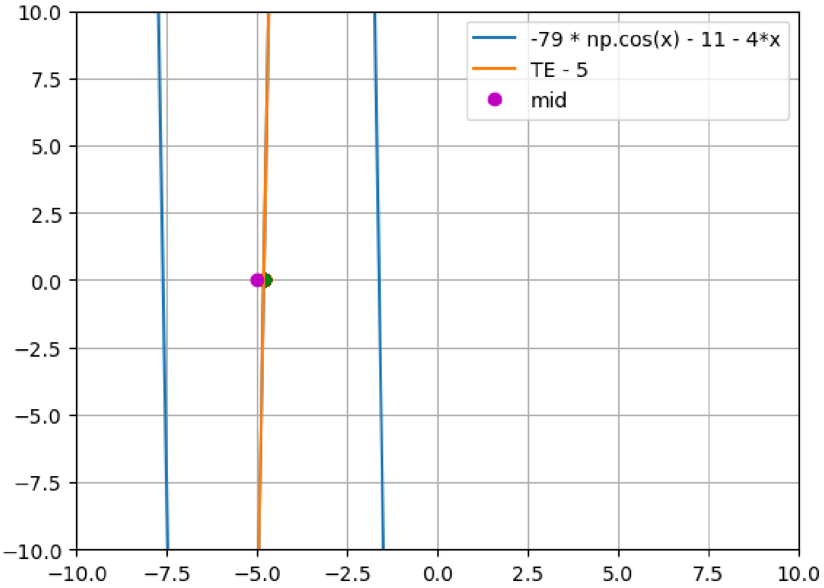
Braižome tarpinius grafikus, kai Teiloro eilutės skaičius yra 3,4,5.



10 pav. Teiloro eilutės skaičius yra lygus 3.



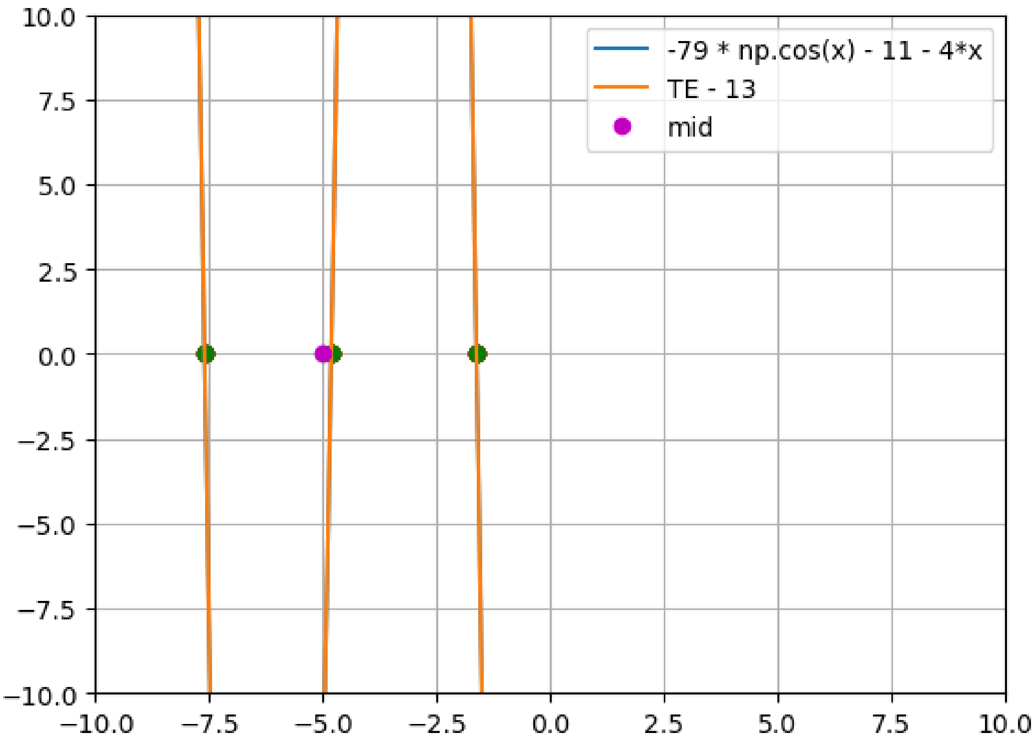
11 pav. Teiloro eilutės skaičius yra lygus 4.



12 pav. Teiloro eilutės skaičius yra lygus 5.

1.3.2. grafiką, kuriame pavaizduotas reikalaujamą tikslumą užtikrinantis pagal TE sudarytas daugianaris, drauge pateikiant ir funkcijos $h(x)$ grafiką;

Pateikiame grafiką kuriame, pavaizduojame reikalaujamą tikslumą užtikrinantis TE sudarytas daugianaris ir funkcijos h(x) grafikas.



13 pav. TE daugianario grafikas.

1.3.3. nustatytos reikalaujamą tikslumą užtikrinančios TE analitinę išraišką daugianario pavidalu;

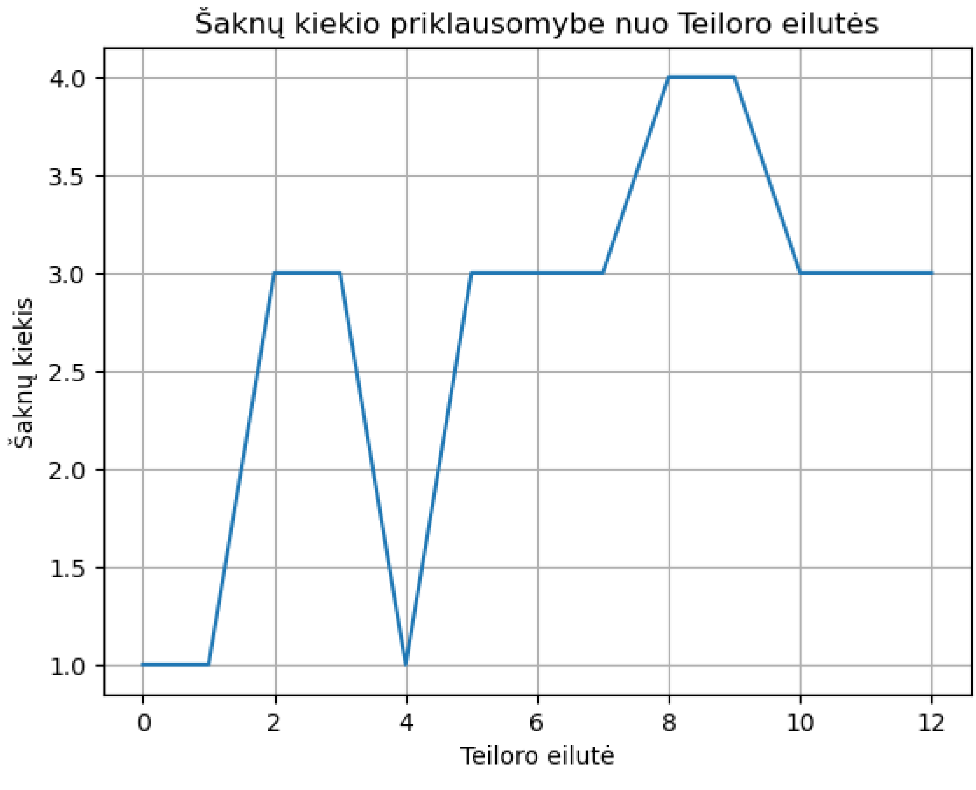
```
daugianario išraiška - 71.7550176983879*x + 14.8505023202833*(0.2*x + 1)**13 - 11.4217230100709*(0.2*x + 1)**12 - 92.6671344785679*(0.2*x + 1)**11 + 60.3066974931744*(0.2*x + 1)**10 + 407.735391705699*(0.2*x + 1)**9 - 217.104110975428*(0.2*x + 1)**8 - 1174.2779281124l*(0.2*x + 1)**7 + 486.313208584958*(0.2*x + 1)**6 + 1972.78691922885*(0.2*x + 1)**5 - 583.57585030195*(0.2*x + 1)**4 - 1578.22953538308*(0.2*x + 1)**3 + 280.116408144936*(0.2*x + 1)**2 + 345.365775840345
```

14 pav. Sugeneruota daugianario išraiška.

1.3.4. grafikus, pagal kuriuos būtų galima įvertinti, kaip gerėjo sprendinys priklausomai nuo TE narių skaičiaus: a) grafikas, kuris nurodo visą randamų šaknų skaičių nagrinėjamame intervale (ox-TE eilė, oy – šaknų skaičius); b) atskiri grafikai kiekvienai šakniai, kuriuose oy ašyje pateikti tikslumo įverčiai tarp $h(x)$ apskaičiuotos šaknies ir artimiausios TE šaknies, o ox ašyje TE narių skaičiai.

a) grafikas, kuris nurodo visą randamų šaknų skaičių nagrinėjamame intervale (ox-TE eilė, oy – šaknų skaičius);

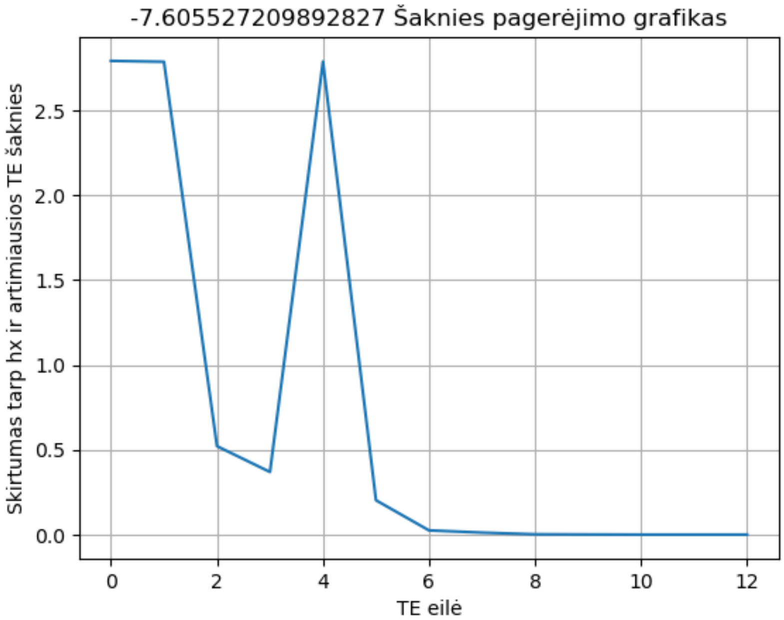
Grafikas rodo, kiek tam tikroje teiloro eilutėje šaknų sutampa su pagrindine funkcija.



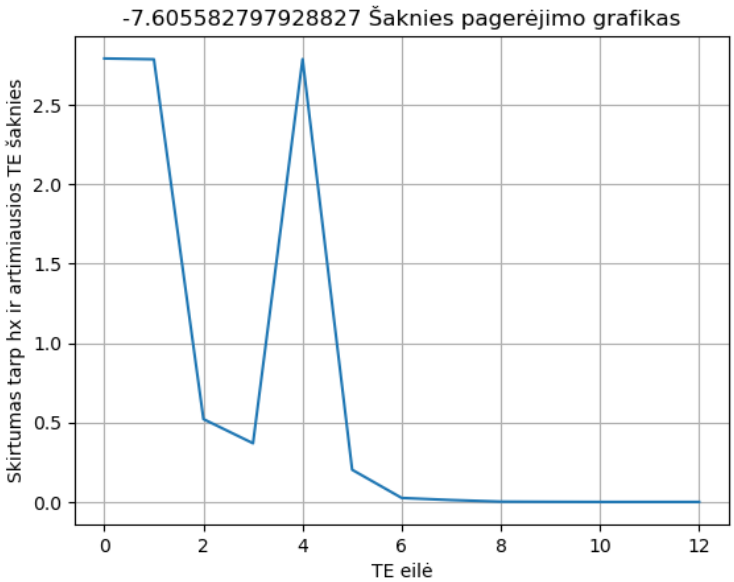
15 pav. Teiloro eilutės kiekio priklausomybės nuo šaknų skaičiaus grafikas.

b) atskiri grafikai kiekvienai šakniai, kuriuose oy ašyje pateikti tikslumo įverčiai tarp $h(x)$ apskaičiuotos šaknies ir artimiausios TE šaknies, o ox ašyje TE narių skaičiai.

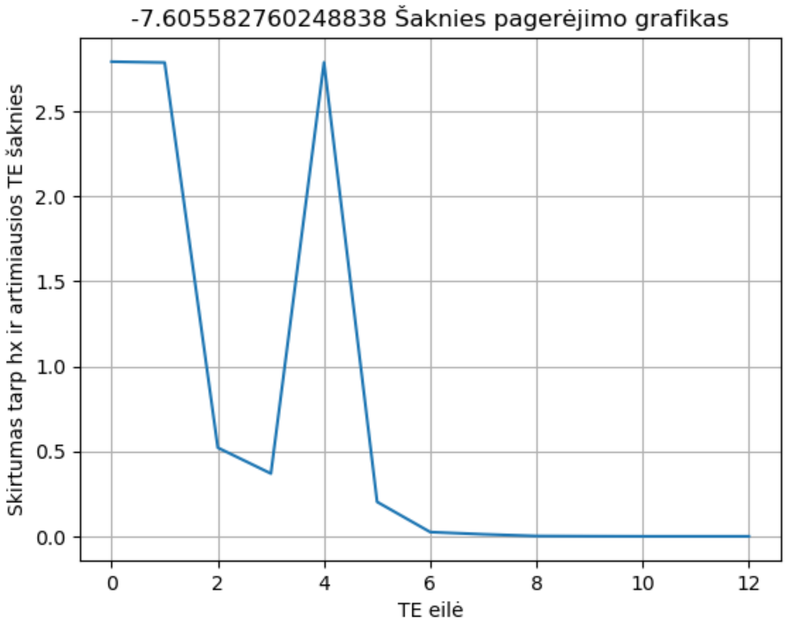
Grafikas rodo pačios hx funkcijos teiloro šaknies aproksimacijos tikslumą, kuo skirtumas mažesnis, tuo labiau tikslesnė aproksimacija.



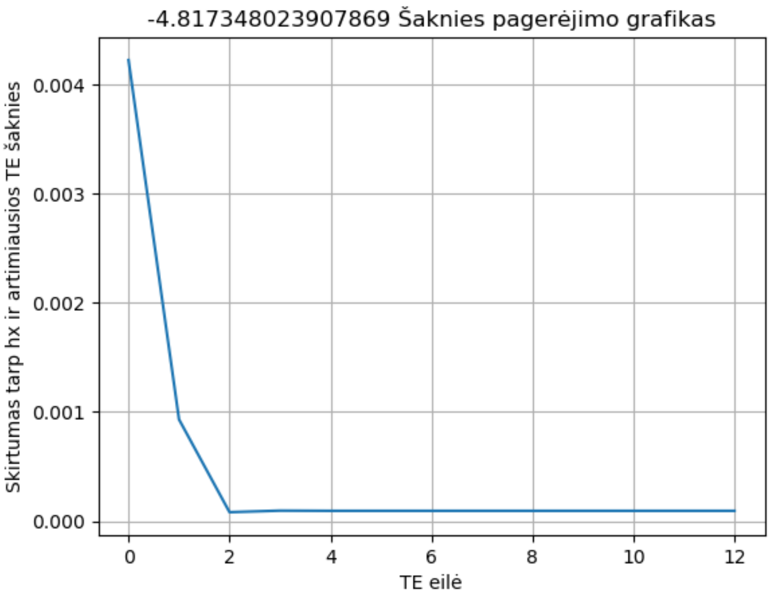
16 pav. Pirmosios šaknies grafikas.



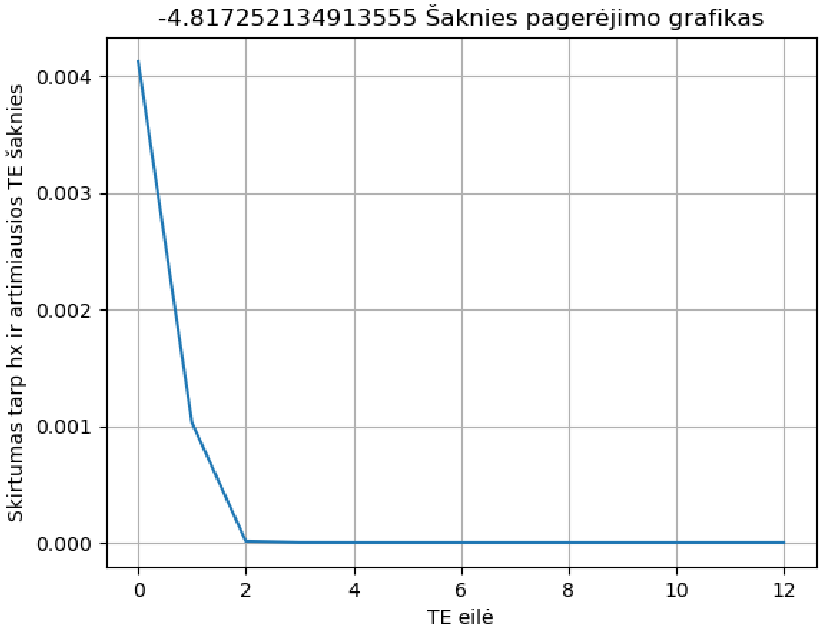
17 pav. Antrosios šaknies grafikas.



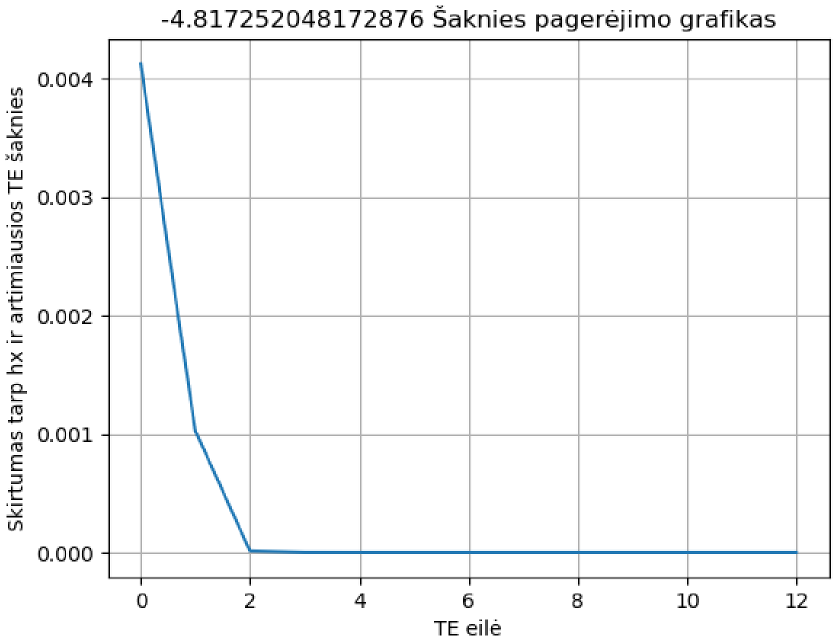
18 pav. Trečiosios šaknies grafikas.



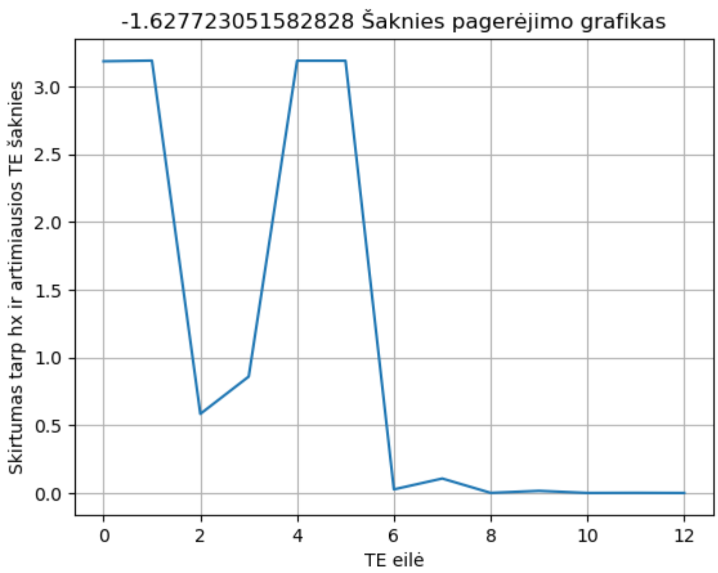
19 pav. Ketvirtosios šaknies grafikas.



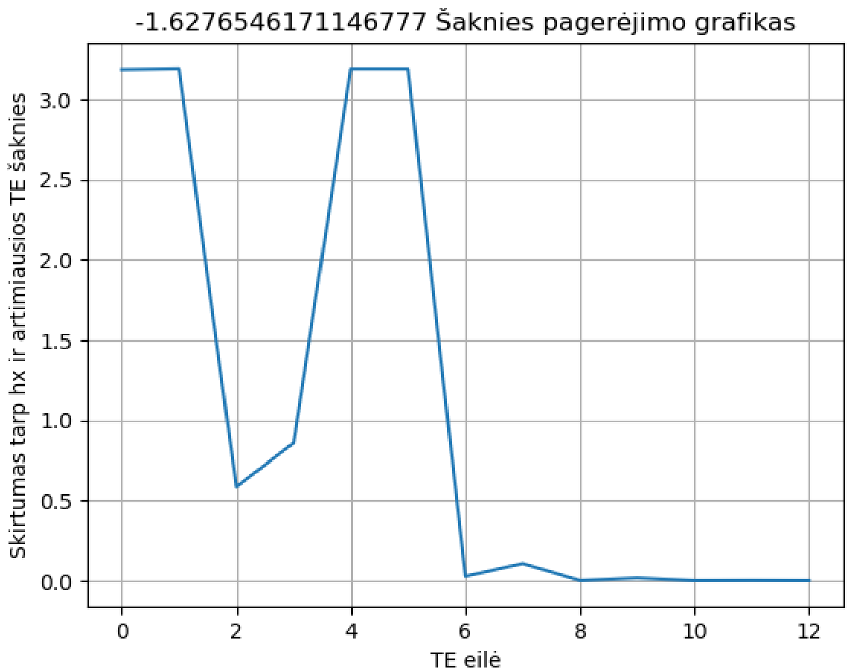
20 pav. Penktosios šaknies grafikas.



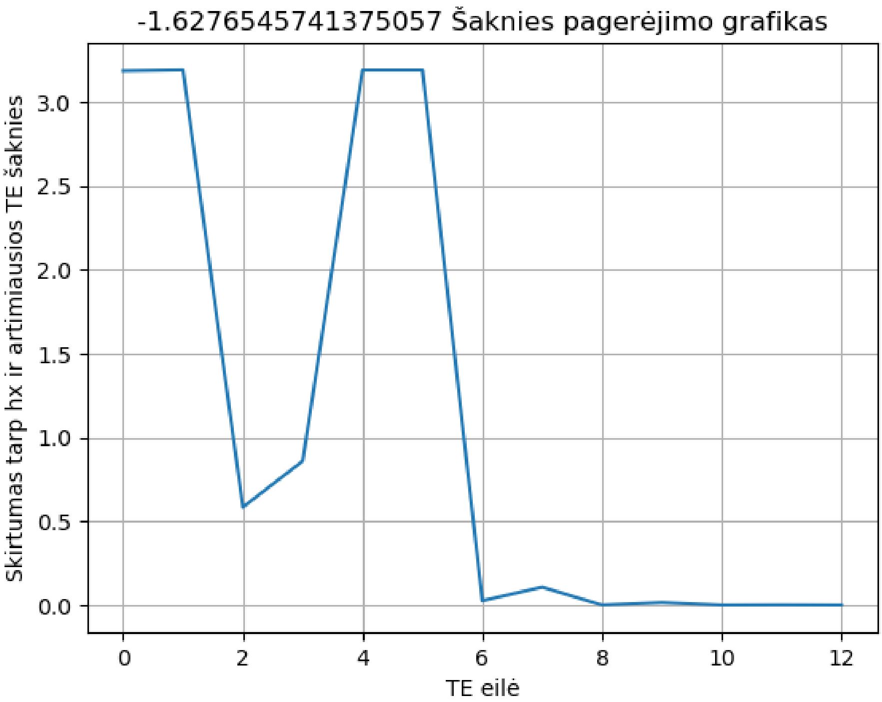
21 pav. Šeštosios šaknies grafikas.



22 pav. Septintosios šaknies grafikas.



23 pav. Aštuntosios šaknies grafikas.



24 pav. Devintosios šaknies grafikas.

2. Literatūra

1. „Moodle“ aplinkoje esantis modulis „Skaitiniai metodai ir algoritmai“
https://moodle.ktu.edu/course/view.php?id=7639