**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

INFORMATIKOS FAKULTETAS

TAIKOMOSIOS INFORMATIKOS KATEDRA

# SKAITINIAI METODAI IR ALGORITMAI(P170B115)

## 1 LABORATORINIS DARBAS

Varianto Nr. 15

**Atliko:**

IFF-1/8 gr. studentas

Matas Palujanskas

**Priėmė:**

Prof. Rimantas Barauskas

Doc. Andrius Kriščiūnas

KAUNAS

2023

Matas Palujanskas

# Turinys

Matas Palujanskas

# 1. Pirma užduoties dalis

## 1.1 Užduoties sąlygos

**1 dalis (5 balai).** Išspręskite netiesines lygtis (1 ir 2 lentelės):

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

1. **(tik lygčiai su daugianariu** $f(x)$**)** Nustatykite daugianario $f(x)$ šaknų intervalą, taikydami „grubų" ir tikslesnį įverčius. Grafiškai pavaizduokite apskaičiuotų šaknų intervalo galus.
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.
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.
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.
5. Gautas šaknų reikšmes patikrinkite naudodami išorinius išteklius (pvz., funkcijas **roots** arba **fzero**, tinklapį wolframalpha.com ir t.t.) ir pateikite patikrinimo rezultatus.

*1. pav. Pirmojo laboratorinio darbo 1 dalis*

## Užduoties variantas: 15

| | | | |
|---|---|---|---|
| 15 | $2.19x^4 - 5.17x^3 - 7.17x^2 + 15.14x + 1.21$ | $e^{-\left(\frac{x}{2}\right)^2}\sin(2x)$; $-6 \le x \le 6$ | 1, 3 |

*2. pav. Užduoties variantas*

1 lentelė. Netiesinių lygčių sprendimas. Metodai.

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

Bus naudojami stygų ir Niutono (liestinių) metodai.

Matas Palujanskas

## 1.2 Daugianario programos kodas

**LAB1_Daugianaris.ipynb:**

```
import numpy as np
import matplotlib.pyplot as plt
import math
def fx(x):
    return 2.19 * np.power(x, 4) - 5.17 * np.power(x, 3) - 7.17 * np.power(x, 2) +
15.14 * x + 1.21
def root_intervals(f, x_min, x_max, step):
    intervals = []
    current_x = x_min
    while current_x < x_max:
        if np.sign(f(current_x)) != np.sign(f(current_x + step)):
            plt.plot([current_x], [0], 'or')
            plt.plot([current_x + step], [0], 'og')
            intervals.append({"xMin": round(current_x, 2), "xMax": round(current_x +
step, 2)})
        current_x += step
    return intervals



def chords(f, ranges, epsilon, iteration_max):
    roots = []
    for current_range in ranges:
        out_of_range = False
        current_iteration = 0
        x_min = float(current_range["xMin"])
        x_max = float(current_range["xMax"])
        k = np.abs(f(x_min) / f(x_max))
        x_mid = (x_min + k * x_max) / (1 + k)
        x_mid_n1 = x_mid + epsilon * 2  # used to enter while for the first time
        while np.abs(x_mid - x_mid_n1) > epsilon or np.abs(f(x_mid)) > epsilon:  #
absoliutinis sprendinio tikslumo ivertis
            current_iteration += 1
            if current_iteration > iteration_max:
                #print(f"Chords method has reached the maximum iteration count -
{iteration_max}")
                roots.append({"range": current_range, "root": x_mid, "iteration":
current_iteration})
                out_of_range = True
                break
            if np.sign(f(x_min)) == np.sign(f(x_mid)):
                x_min = x_mid
            else:
                x_max = x_mid
                x_mid_n1 = x_mid
                k = np.abs(f(x_min) / f(x_max))
                x_mid = (x_min + k * x_max) / (1 + k)
        if not out_of_range:
            roots.append({"range": current_range, "root": x_mid, "iteration":
current_iteration})
    return roots

def newton_raphson(f, df, initial_guesses, epsilon, iteration_max):
    roots = []
    found_roots = set()  # Saugome rastų šaknų reikšmes
    for guess in initial_guesses:
        current_iteration = 0
        x_n = guess
```

```
        x_n1 = x_n + epsilon * 2  # Naudojama, kad įeitume į while ciklą
        interval = {"xMin": x_n, "xMax": x_n}  # Intervalas pradedamas nuo pradinio
spėjimo
        while np.abs(x_n - x_n1) > epsilon or np.abs(f(x_n)) > epsilon:
            current_iteration += 1
            if current_iteration > iteration_max:
                print(f"Newton-Raphson method has reached the maximum iteration count
- {iteration_max}")
                break
            x_n1 = x_n - f(x_n) / df(x_n)
            x_n = x_n1
            interval["xMin"] = min(interval["xMin"], x_n)
            interval["xMax"] = max(interval["xMax"], x_n)
            if x_n < xmin or x_n > xmax:
                break
        else:
            if x_n >= xmin and x_n <= xmax and round(x_n, 8) not in found_roots:
                roots.append({"range": interval, "root": x_n, "iteration":
current_iteration})
                found_roots.add(round(x_n, 8))
    return roots


if __name__ == "__main__":
    eps = 1e-12
    nitmax = 50
    xmin = -2.9
    xmax = 4.27
    step = 0.3

    dx = 0.05
    x = np.arange(xmin, xmax + dx, dx)
    y = fx(x)

    plt.title("Daugianaris 2.19x^4 - 5.17x^3 - 7.17x^2 + 15.14x + 1.21")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.plot(x, y)
    plt.grid(color='black', linestyle="-", linewidth=0.5)

RootIntervals = root_intervals(fx, xmin, xmax, step)
for item in RootIntervals:
    print(f"Range : [{item['xMin']} ; {item['xMax']}]")
for item in RootIntervals:
    plt.plot([item['xMin'], item['xMax']], [0, 0], 'ro')

coefficients = [2.19, -5.17, -7.17, 15.14, 1.21]
real_roots = np.roots(coefficients)
print("Šaknys, naudojant numpy.roots", real_roots)

chords_roots = chords(fx, RootIntervals, eps, nitmax)
print("Stygų metodas")

for root in chords_roots:
    print( f"Range : [{root['range']['xMin']} ; {root['range']['xMax']} ], root -
{round(root['root'], 8)}, function value at root " f"point = {fx(root['root'])},
iteration = {root['iteration']}")

def df(x):
    return 8.76 * np.power(x, 3) - 15.51 * np.power(x, 2) - 14.34 * x + 15.14
```

```
initial_guesses = np.arange(xmin, xmax, 0.1)  # Generating initial guesses
newton_roots = newton_raphson(fx, df, initial_guesses, eps, nitmax)

print("Niutono (liestinių) metodas")

unique_roots = []
for root in newton_roots:
    if root['root'] not in [r['root'] for r in unique_roots]:
        unique_roots.append(root)

for i, root in enumerate(unique_roots):
    print(f"Root {i + 1}: Range : [{root['range']['xMin']} ;
{root['range']['xMax']}], root - {round(root['root'], 8)}, function value at root
point = {fx(root['root'])}, iteration = {root['iteration']}")
```

## 1.3 Transcendentinės programos kodas

### LAB1_Transcendentine.ipynb :

```
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.optimize import fsolve
#from LAB1_Daugianaris.ipynb import root_intervals, bisection, chords
def root_intervals(f, x_min, x_max, h):
    intervals = []
    x_start = x_min
    while x_start < x_max:
        x_end = x_start + h
        if np.sign(f(x_start)) != np.sign(f(x_end)):
            plt.plot([x_start], [0], 'or')
            plt.plot([x_end], [0], 'og')
            intervals.append({"xMin": round(x_start, 2), "xMax": round(x_end, 2)})
        x_start = x_end
    return intervals
def gx(x):
    #return np.power(math.e, -np.power(x/2,2)) np.sin(2x)
    return np.exp(-np.power(x / 2, 2)) * np.sin(2 * x)

def chords(f, ranges, epsilon, iteration_max):
    roots = []
    reached_max_iteration = False  # Track if the maximum iteration message has been
printed
    for current_range in ranges:
        out_of_range = False
        current_iteration = 0
        x_min = float(current_range["xMin"])
        x_max = float(current_range["xMax"])
        k = np.abs(f(x_min) / f(x_max))
        x_mid = (x_min + k * x_max) / (1 + k)
        x_mid_n1 = x_mid + epsilon * 2  # used to enter while for the first time
        while np.abs(x_mid - x_mid_n1) > epsilon or np.abs(f(x_mid)) > epsilon:
            current_iteration += 1
            if current_iteration > iteration_max:
                if not reached_max_iteration:
                    print(f"Chords method has reached the maximum iteration count -
```

```
{iteration_max}")
                    reached_max_iteration = True  # Set the flag to True
                break  # Break the loop when the maximum iteration count is reached
            if np.sign(f(x_min)) == np.sign(f(x_mid)):
                x_min = x_mid
            else:
                x_max = x_mid
                x_mid_n1 = x_mid
                k = np.abs(f(x_min) / f(x_max))
                x_mid = (x_min + k * x_max) / (1 + k)
        if not out_of_range:
            roots.append({"range": current_range, "root": x_mid, "iteration":
current_iteration})
    return roots

def newton_raphson(f, df, initial_guesses, epsilon, iteration_max):
    roots = []
    found_roots = set()  # Saugome rastų šaknų reikšmes
    for guess in initial_guesses:
        current_iteration = 0
        x_n = guess
        x_n1 = x_n + epsilon * 2  # Naudojama, kad įeitume į while ciklą
        interval = {"xMin": x_n, "xMax": x_n}  # Intervalas pradedamas nuo pradinio
spėjimo
        while np.abs(x_n - x_n1) > epsilon or np.abs(f(x_n)) > epsilon:
            current_iteration += 1
            if current_iteration > iteration_max:
                print(f"Newton-Raphson method has reached the maximum iteration count
- {iteration_max}")
                break
            x_n1 = x_n - f(x_n) / df(x_n)
            x_n = x_n1
            interval["xMin"] = min(interval["xMin"], x_n)
            interval["xMax"] = max(interval["xMax"], x_n)
            if x_n < xmin or x_n > xmax:
                break
        else:
            if x_n >= xmin and x_n <= xmax and round(x_n, 8) not in found_roots:
                roots.append({"range": interval, "root": x_n, "iteration":
current_iteration})
                found_roots.add(round(x_n, 8))
    return roots

step = 0.1
eps = 1e-12
nitmax = 100
dx = 0.05
xmin = -6
xmax = 6

x = np.arange(xmin, xmax + dx, dx)
y = gx(x)

plt.plot(x, y)
plt.title("Transcedentinė funkcija e^-(x/2)^2 sin(2x)")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(color='black', linestyle="-", linewidth=0.5)

RootIntervals = root_intervals(gx, xmin, xmax, step)
for item in RootIntervals:
```

```
    print(f"Range : [{item['xMin']} ; {item['xMax']}]")
for item in RootIntervals:
    plt.plot([item['xMin'], item['xMax']], [0, 0], 'ro')

print("")
chords_roots = chords(gx, RootIntervals, eps, nitmax)
print("Stygų metodas")

for root in chords_roots:
    print( f"Range : [{root['range']['xMin']} ; {root['range']['xMax']} ], root -
{round(root['root'], 8)}, function value at root " f"point = {gx(root['root'])},
iteration = {root['iteration']}")


def df(x):
    return (-x * np.exp(-np.power(x / 2, 2)) * np.cos(2 * x)) + (np.exp(-np.power(x /
2, 2)) * 2 * np.cos(2 * x))

initial_guesses = np.arange(xmin, xmax, 0.1)  # Generating initial guesses
newton_roots = newton_raphson(gx, df, initial_guesses, eps, nitmax)
print("Niutono (liestinių) metodas")
unique_roots = []
for root in newton_roots:
    if root['root'] not in [r['root'] for r in unique_roots]:
        unique_roots.append(root)

for i, root in enumerate(unique_roots):
    print(f"Root {i + 1}: Range : [{root['range']['xMin']} ;
{root['range']['xMax']}], root - {round(root['root'], 8)}, function value at root
point = {gx(root['root'])}, iteration = {root['iteration']}")

print("Šaknys, naudojant scipy.optimize.fsolve: ")
for current_range in RootIntervals:  # Rename the list
    print(fsolve(gx, current_range["xMin"], xtol=1e-12))

plt.show()
```

## 1.4 Daugianario f(x) šaknų intervalo nustatymas

$$f(x) = 2.19x^4 - 5.17x^3 - 7.17x^2 + 15.14x + 1.21 = 0$$

Daugianario eilė n = 4;

Koeficientai:

$$a_4 = 2{,}19;\ a_3 = -5{,}17;\ a_2 = -7{,}17;\ a_1 = 15{,}14;\ a_0 = 1{,}21;$$

- „Grubaus“ įverčio radimas:

$$R = 1 + \frac{\max\limits_{0 \le i \le n-1} |a_i|}{a_n}$$

Matas Palujanskas

$$R = 1 + \frac{15,14}{2,19} \approx \mathbf{7.9}$$

$$Grubus\ įvertis: (-7.9; 7.9)$$

- „Tikslesnio" įverčio radimas:

Teigiamoms šaknims:

$$R_{teig} = 1 + \sqrt[k]{\frac{B}{a_n}}\ ; k = n - \max_{0 \le i \le n-1}(i, a_i < 0)\ ; B = \max_{0 \le i \le n-1}(|a_i|, a_i < 0)$$

$$B = max(|a3|, |a2|) = 7.17;$$

$$k = 4 - \max(3; 2) = 4 - 3 = 1;$$

Tikslesnio įverčio viršutinis rėžis:

$$R_{teig} = 1 + \sqrt[k]{\frac{B}{a_n}} = 1 + \sqrt[1]{\frac{7.17}{2.19}} = 1 + 3.27 = 4.27;$$

Neigiamoms šaknims:

Koeficientai:

$a_4$ =2.19; $a_3$ = 5.17; $a_2$ = 7.17; $a_1$ =-15.14; $a_0$ =-1.21;

$$B = max(|a1|, |a0|) = 15.14;$$

$$k = 4 - \max(1; 0) = 4 - 1 = 3;$$

$$R_{teig} = 1 + \sqrt[k]{\frac{B}{a_n}} = 1 + \sqrt[3]{\frac{15.14}{2.19}} = 1 + 1.9 = 2.9;$$

Galutinis šaknų intervalo įvertis:

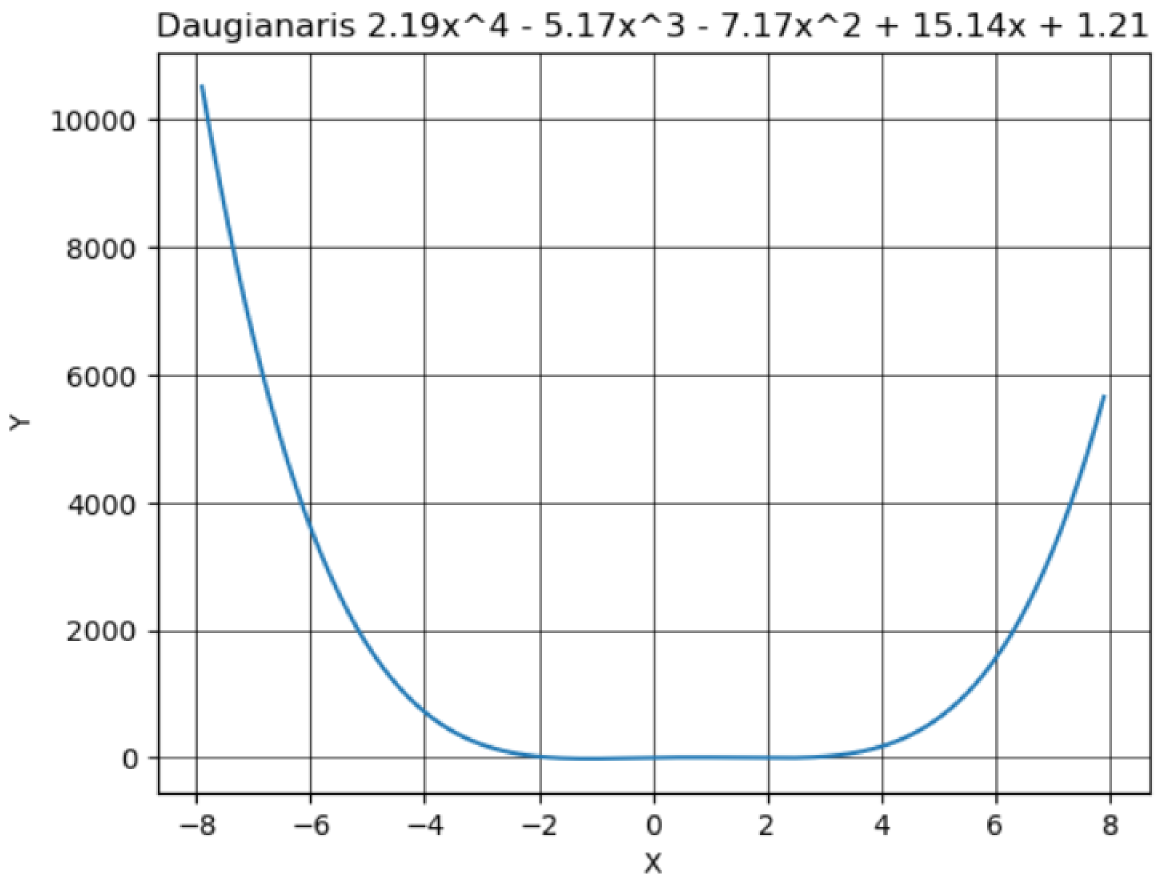
$$-\min(R, R_{neig}) \leq x \leq \min(R, R_{teig}) =>$$

$$-\min(-7.9;\ 2.9) \leq x \leq \min(7.9;\ 46.014) =>$$

$$\mathbf{-2.9 \leq x \leq 4.27}$$

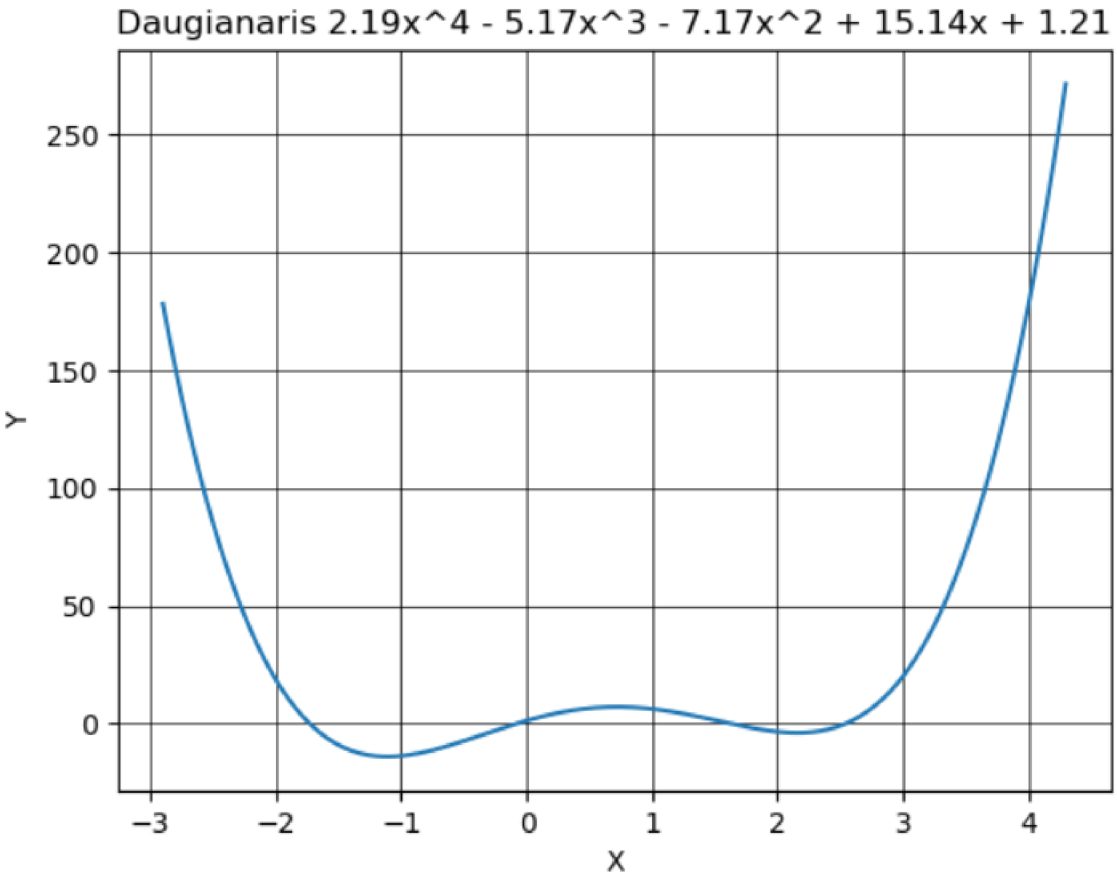
1.5 Grafinis daugianario ir transcendentinės funkcijos atvaizdavimas

**Daugianaris:**




*3. pav. Grafiškai atvaizduotas daugianaris grubiuose -7.9 <= x <= 7.9 rėžiuose*

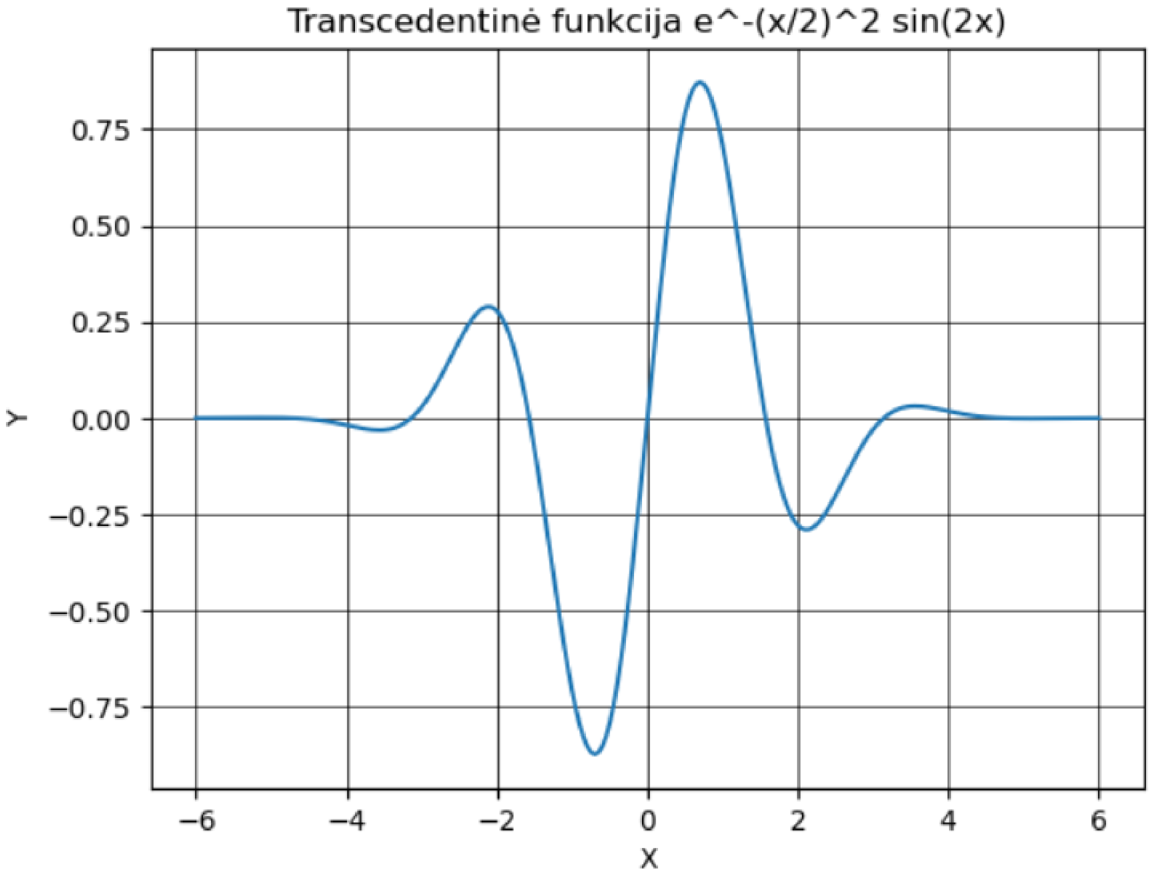
Reikia sumažinti rėžius, atvaizduoti tikslesniuose rėžiuose, kad geriau matytųsi šaknys.




*4. pav. Grafiškai atvaizduotas daugianaris tikslesniuose -2.9 <= x <= 4.27 rėžiuose*

Iš grafiko galime matyti, kad daugianaris turi 4 šaknis.

**Transcendentinė:**




*5. pav. Grafiškas transcendentinės funkcijos atvaizdavimas rėžiuose -6<=x<= 6*

Iš grafiko galime matyti, kad transcendentinė funkcija turi 7 šaknis.

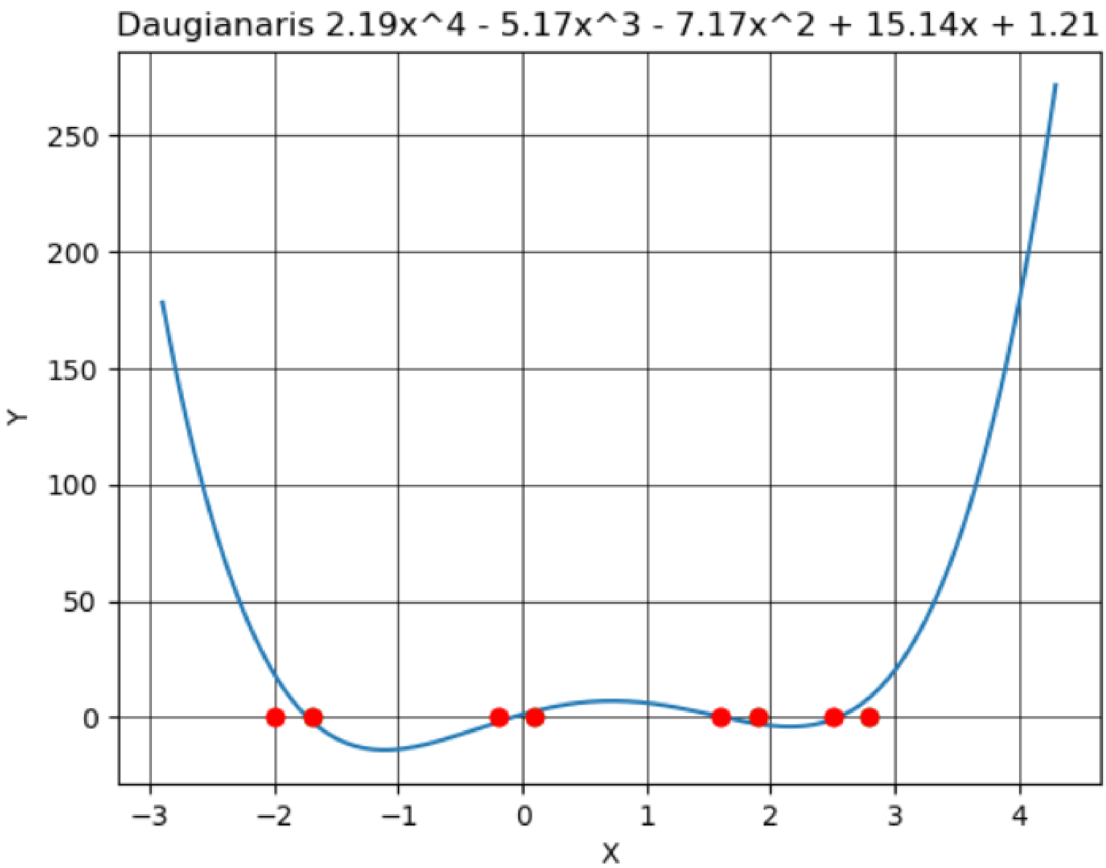
## 1.6 Šaknų intervalai

**Daugianaris:**

```
Range : [-1.9 ; -1.6]
Range : [-0.1 ; 0.2]
Range : [1.4 ; 1.7]
Range : [2.3 ; 2.6]
```

*6. pav. Daugianario šaknies intervalai*

Grafiko rėžiai: [-2.9; 4.27]



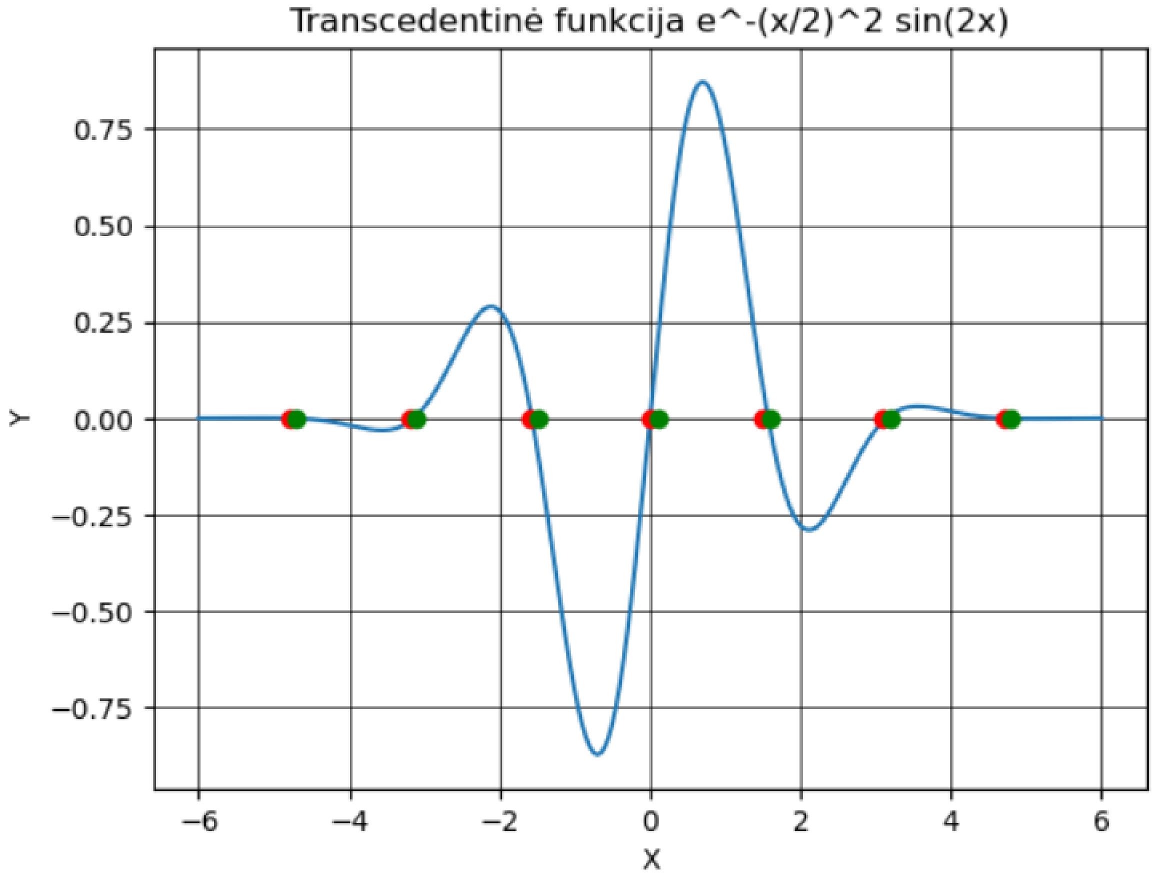

*7. pav. Daugianario šaknų intervalai grafike*

**Transcendentinė funkcija:**

```
Range : [-4.8 ; -4.7]
Range : [-3.2 ; -3.1]
Range : [-1.6 ; -1.5]
Range : [-0.0 ; 0.1]
Range : [1.5 ; 1.6]
Range : [3.1 ; 3.2]
Range : [4.7 ; 4.8]
```

*8. pav. Transcendentinės funkcijos šaknų intervalai*

Grafiko rėžiai: [-6; 6]




*9. pav. Transcendentinės funkcijos šaknų intervalai grafike*

## 1.7 Šaknų tikslinimas stygų ir Niutono (liestinių) metodais

**Daugianaris:**

```
Šaknys, naudojant numpy.roots [-1.731073    2.54706654  1.6219938  -0.07725674]
Chords method has reached the maximum iteration count - 50
Stygų metodas
Range : [-2.0 ; -1.7 ], root - -1.731073, function value at root point = -3.9879211044535623e-13, iteration = 20
Range : [-0.2 ; 0.1 ], root - -0.07725674, function value at root point = 5.597744490160039e-13, iteration = 8
Range : [1.6 ; 1.9 ], root - 1.6219938, function value at root point = -3.2862601528904634e-14, iteration = 5
Range : [2.5 ; 2.8 ], root - 2.5311845, function value at root point = -0.35158151010389194, iteration = 51
Niutono (liestinių) metodas
Root 1: Range : [-2.9 ; -1.7310729963869], root - -1.731073, function value at root point = 2.6645352591003757e-15, iteration = 7
Root 2: Range : [-0.9999999999999982 ; 1.637236084452935], root - 1.6219938, function value at root point = 2.6645352591003757e-15, iteration = 4
Root 3: Range : [-0.9540323871309622 ; 2.8404804291621923], root - 2.54706654, function value at root point = 6.306066779870889e-14, iteration = 10
Root 4: Range : [-0.799999999999998 ; 0.1792021297522678], root - -0.07725674, function value at root point = -1.63202784619898e-13, iteration = 5
```

*10.pav. Daugianario šaknų tikslinimo stygų ir Niutono(liestinių) metodų rezultatai ekrane*

*1. lent. Daugianario šaknų tikslinimo stygų ir Niutono(liestinių) metodų rezultatai*

| Metodas | Intervalas | Šaknis | Funkcijos reikšmė šaknyje | Tikslumas | Iteracijų skaičius |
|---|---|---|---|---|---|
| Stygų | [-2.0; -1.7] | -1.731073 | $-3.987321 * 10^{-13}$ | $1 * 10^{-12}$ | 20 |
| | [-0.2; 0.1] | -0.077256 | $5.597744 * 10^{-13}$ | | 8 |
| | [1.6; 1.9] | 1.621993 | $-3.286260 * 10^{-14}$ | | 5 |
| | [2.5; 2.8] | 2.531184 | $-0.351581 * 10^{-14}$ | | 51 |
| Niutono (liestinių) | [-2.9; -1.7] | -1.731073 | $2.664535 * 10^{-15}$ | | 7 |
| | [-1.0; 1.6] | 1.621993 | $6.306066 * 10^{-15}$ | | 4 |
| | [-0.8; 0.2] | -0.077256 | $-1.632027 * 10^{-14}$ | | 10 |
| | [4.3; 4.6] | 4.37860134 | $-5.329071 * 10^{-13}$ | | 5 |

## Transcendentinė funkcija:

```
Stygų metodas
Range : [-4.8 ; -4.7 ], root - -4.71527, function value at root point = 2.2211265483807025e-05, iteration = 101
Range : [-3.2 ; -3.1 ], root - -3.14548988, function value at root point = -0.0006569655265887052, iteration = 101
Range : [-1.6 ; -1.5 ], root - -1.57231694, function value at root point = 0.0016392069667078553, iteration = 101
Range : [-0.0 ; 0.1 ], root - 0.0, function value at root point = 0.0, iteration = 101
Range : [1.5 ; 1.6 ], root - 1.57079633, function value at root point = -3.0493494716709398e-15, iteration = 9
Range : [3.1 ; 3.2 ], root - 3.14159265, function value at root point = 8.641252612073844e-15, iteration = 9
Range : [4.7 ; 4.8 ], root - 4.71238898, function value at root point = -1.236234853608059e-17, iteration = 8
Niutono (liestinių) metodas
Root 1: Range : [-5.400000000000002 ; -4.712388980498786], root - -4.71238898, function value at root point = 8.856227739258034e-13, iteration = 51
Root 2: Range : [-3.800000000000008 ; -3.1358162595370964], root - -3.14159265, function value at root point = 6.17359444188692e-13, iteration = 44
Root 3: Range : [-2.300000000000013 ; -7.889612493378979e-20], root - -0.0, function value at root point = -1.5779224986757958e-19, iteration = 5
Root 4: Range : [-2.2000000000000135 ; -1.4627800522261998], root - -1.57079633, function value at root point = -9.315815985790062e-13, iteration = 32
```

*11. pav. Transcendentinės funkcijos šaknų tikslinimo stygų ir Niutono(liestinių) metodų rezultatai ekrane*

*2. lent. Transcendentinės funkcijos šaknų tikslinimo stygų ir Niutono(liestinių) metodų rezultatai*

| Metodas | Intervalas | Šaknis | Funkcijos reikšmė šaknyje | Tikslumas | Iteracijų skaičius |
|---|---|---|---|---|---|
| Stygų | [-4.8; -4.7] | -4.71527 | $2.221126 * 10^{-5}$ | $1 * 10^{-12}$ | 101 |
| | [-3.2; -3.1] | -3.145489 | $-0.000656 * 10^{-13}$ | | 101 |
| | [-1.6; -1.5] | -1.572316 | $0.001639 * 10^{-13}$ | | 101 |
| | [-0.0; 0.1] | 0.0 | 0.0 | | 101 |
| | [1.5; 1.6] | 1.570796 | $-3.049349 * 10^{-15}$ | | 9 |
| | [3.1; 3.2] | 3.141592 | $8.641252 * 10^{-15}$ | | 9 |
| | [4.7; 4.8] | 4.712388 | $-1.236234 * 10^{-17}$ | | 8 |
| Niutono (liestinių) | [-5.4; -4.7] | -4.712388 | $8.856227 * 10^{-13}$ | | 51 |
| | [-3.8; -3.1] | -3.141592 | $6.173594 * 10^{-13}$ | | 44 |
| | [-2.3; -7.9] | -0.0 | $-1.577922 * 10^{-19}$ | | 5 |
| | [-2.2; -1.5] | -1.570796 | $-9.315815 * 10^{-13}$ | | 32 |

**Išvada.** Niutono(liestinių) metodas randa sprendinį su mažesniu iteracijų skaičiumi tiek sprendžiant daugianarį, tiek transcendentinę funkciją.

## 1.8 Šaknų reikšmių tikrinimas išoriniais ištekliais

**Daugianaris**:

Šaknų radimui naudojame „numpy“ bibliotekos „roots()“ metodą.

```
Šaknys, naudojant numpy.roots [-1.731073    2.54706654  1.6219938  -0.07725674]
Stygų metodas
Range : [-2.0 ; -1.7 ], root - -1.731073, function value at root point = -3.9879211044535623e-13, iteration = 20
Range : [-0.2 ; 0.1 ], root - -0.07725674, function value at root point = 5.597744490160039e-13, iteration = 8
Range : [1.6 ; 1.9 ], root - 1.6219938, function value at root point = -3.2862601528904634e-14, iteration = 5
Range : [2.5 ; 2.8 ], root - 2.5311845, function value at root point = -0.35158151010389194, iteration = 51
Niutono (liestinių) metodas
Root 1: Range : [-2.9 ; -1.7310729963869], root - -1.731073, function value at root point = 2.6645352591003757e-15, iteration = 7
Root 2: Range : [-0.9999999999999982 ; 1.637236084452935], root - 1.6219938, function value at root point = 2.6645352591003757e-15, iteration = 4
Root 3: Range : [-0.9540323871309622 ; 2.8404804291621923], root - 2.54706654, function value at root point = 6.306066779870889e-14, iteration = 10
Root 4: Range : [-0.799999999999998 ; 0.1792021297522678], root - -0.07725674, function value at root point = -1.63202784619898e-13, iteration = 5
```

*12. pav. Daugianario šaknų palyginimas*

Tikrinimas su wolframalpha.com:

Roots

$x \approx -1.73107$

$x \approx -0.0772567$

$x \approx 1.62199$

$x \approx 2.54707$

*pav. 13 Daugianario šakų palyginimas wolframalpha.com*

**Išvada.** Šaknys Niutono(liestinių) ir stygų metodais, tiek naudojant „roots()“ ir wolframalpha.com gautos vienodos.

### Transcendentinė funkcija:

Šaknų radimui naudojame „scipy.optimize.fsolve()“ metodą

```
Stygų metodas
Range : [-4.8 ; -4.7 ], root - -4.71527, function value at root point = 2.2211265483807025e-05, iteration = 101
Range : [-3.2 ; -3.1 ], root - -3.14548988, function value at root point = -0.0006569655265887052, iteration = 101
Range : [-1.6 ; -1.5 ], root - -1.57231694, function value at root point = 0.0016392069667078553, iteration = 101
Range : [-0.0 ; 0.1 ], root - 0.0, function value at root point = 0.0, iteration = 101
Range : [1.5 ; 1.6 ], root - 1.57079633, function value at root point = -3.0493494716709398e-15, iteration = 9
Range : [3.1 ; 3.2 ], root - 3.14159265, function value at root point = 8.641252612073844e-15, iteration = 9
Range : [4.7 ; 4.8 ], root - 4.71238898, function value at root point = -1.236234853608059e-17, iteration = 8
Niutono (liestinių) metodas
Root 1: Range : [-5.400000000000002 ; -4.712388980498786], root - -4.71238898, function value at root point = 8.856227739258034e-13, iteration = 51
Root 2: Range : [-3.800000000000008 ; -3.1358162595370964], root - -3.14159265, function value at root point = 6.17359444188692e-13, iteration = 44
Root 3: Range : [-2.300000000000013 ; -7.889612493378979e-20], root - -0.0, function value at root point = -1.5779224986757958e-19, iteration = 5
Root 4: Range : [-2.2000000000000135 ; -1.4627800522261998], root - -1.57079633, function value at root point = -9.315815985790062e-13, iteration = 32
Šaknys, naudojant scipy.optimize.fsolve:
[-4.71238898]
[-3.14159265]
[-1.57079633]
[-0.]
[1.57079633]
[3.14159265]
[4.71238898]
```

*13. pav. Transcendentinės funkcijos šaknų palyginimas*

Roots

$x \approx 1.5708\,n\,, \quad n \in \mathbb{Z}$

*pav. 15 Transcendentinės funkcijos šakų palyginimas wolframalpha.com*

**Išvada.** Šaknys Niutono(liestinių) ir stygų metodais, tiek naudojant „scipy.optimize.fsolve()“ ir wolframalpha.com gautos vienodos.

# 2. Antra užduoties dalis

## 2.1 Užduotis:

**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.

*14.*

*pav. Pirmojo laboratorinio darbo 2 dalis*

| | | |
|---|---|---|
| 15 | $154\sin(x) - 9 + 2x^2$ | $-2 \leq x \leq 8$ |

*15. pav. Antros dalies užduoties variantas*

Matas Palujanskas

## 2.2 Antros užduoties programinis kodas

**TE.ipynb:**

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

def root_intervals(f, x_min, x_max, step):
    intervals = []
    current_x = x_min
    while current_x < x_max:
        if np.sign(f(current_x)) != np.sign(f(current_x + step)):
            plt.plot([current_x], [0], 'or')
            plt.plot([current_x + step], [0], 'og')
            intervals.append({"xMin": round(current_x, 2), "xMax": round(current_x +
step, 2)})
        current_x += step
    return intervals

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 154 * np.sin(x) - 9 + 2 * np.power(x,2)

def dhx(x):
    return 154 * np.cos(x) + 4 * x


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, x_min, x_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, x_min, x_max, step)
    close_points_arr = []
    for interval in intervals:
        close_points_arr.append(interval['xMin'])
    plt.clf()
```

```
    newton_roots = newton(f_lambdified, df_lamdified, close_points_arr, eps)
    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 = newton(f_lambdified, df_lambdified, close_points, eps)
    for root in roots:
        min_diff = x_max - x_min
        for newton_root in newton_roots:
            temp_diff = math.fabs(newton_root - root)
            if temp_diff < min_diff:
                min_diff = temp_diff
        differences.append({"root": root, "min_diff": min_diff})
    return differences

def taylor_with_custom_styles(function, x0, roots, orders, 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 = 100
    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"])

    # Calculate the Taylor series with specified orders and derivatives
    taylor_series = []
    for order in orders:
        taylor_expr = fp.series(x, x0, order).removeO()
        taylor_series.append(sympy.lambdify(x, taylor_expr, 'numpy'))

    # Plot the hx(x) function
    plt.figure(figsize=(10, 6))
    plt.plot(x_vals, f_values, label='hx(x) = 154sin(x) - 9 + 2x^2', color='blue')

    # Plot Taylor series approximations with selected numbers of terms
    for i, order in enumerate(orders):
        label = f'TE - {order} terms'
        taylor_values = np.array([taylor_series[i](val) for val in x_vals])
        plt.plot(x_vals, taylor_values, label=label, linestyle='--', linewidth=2)

    # Plot roots as bigger green dots
    for root in roots:
        plt.plot([root], [hx(root)], 'og',  markersize=10, label=f'Root: {root}')

    plt.plot([x0], [0], 'om', label="MID")

    plt.legend()
    plt.grid()
    plt.title("Funkcija hx(x) ir tarpiniai Teiloro grafikai")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()

    graph_a(all_roots_found)  # Plot the number of roots found vs. TE order

    for root_with_diff in all_differences_for_roots:  # Plot improvement in each root
        root_with_diff.graph_b()

    # Print analytical expression
    # Create the analytical expression of the polynomial
    taylor_expr = sympy.expand(fp)
    taylor_expr_str = sympy.pretty(taylor_expr, use_unicode=True)
    print("Daugianario analitinė išraiška:")
    print(taylor_expr_str)

    return taylor_series


def graph_a(roots_count):
    plt.figure()
    plt.plot(range(len(roots_count)), roots_count)
    plt.xlabel("TE eilė")
    plt.ylabel("Šaknų skaičius")
    plt.title("Rastų šaknų skaičiaus priklausomybė nuo TE eilės")
    plt.grid()
    plt.show()
```

```python
# bendri kintamieji
dx = 0.01
h = 0.1
x_max = 8
x_min = -2
mid = (x_max + x_min) / 2
eps = 1e-12
eps2 = 1e-4
all_x = np.arange(x_min, x_max + dx, dx)
all_y = hx(all_x)

# randame artinius
intervals = root_intervals(hx, x_min, x_max, h)
close_points = []
for item in intervals:
    print(f"Artinys : {item['xMin']}")
    close_points.append(item["xMin"])

# niutono metodu randame hx šaknis
h_function_roots = newton(hx, dhx, close_points, eps)
print("154sin(x) - 9 + 2x^2 šaknys Niutono metodu:")
for root in h_function_roots:
    print(root)

# teiloro eilute
x, f = sympy.symbols(('x', 'f'))
f = 154 * sympy.sin(x) - 9 + 2 * np.power(x,2)

selected_orders = [3, 4, 5]
taylor_with_custom_styles(f, mid, h_function_roots, selected_orders, eps2, x_min,
x_max, dx)
```

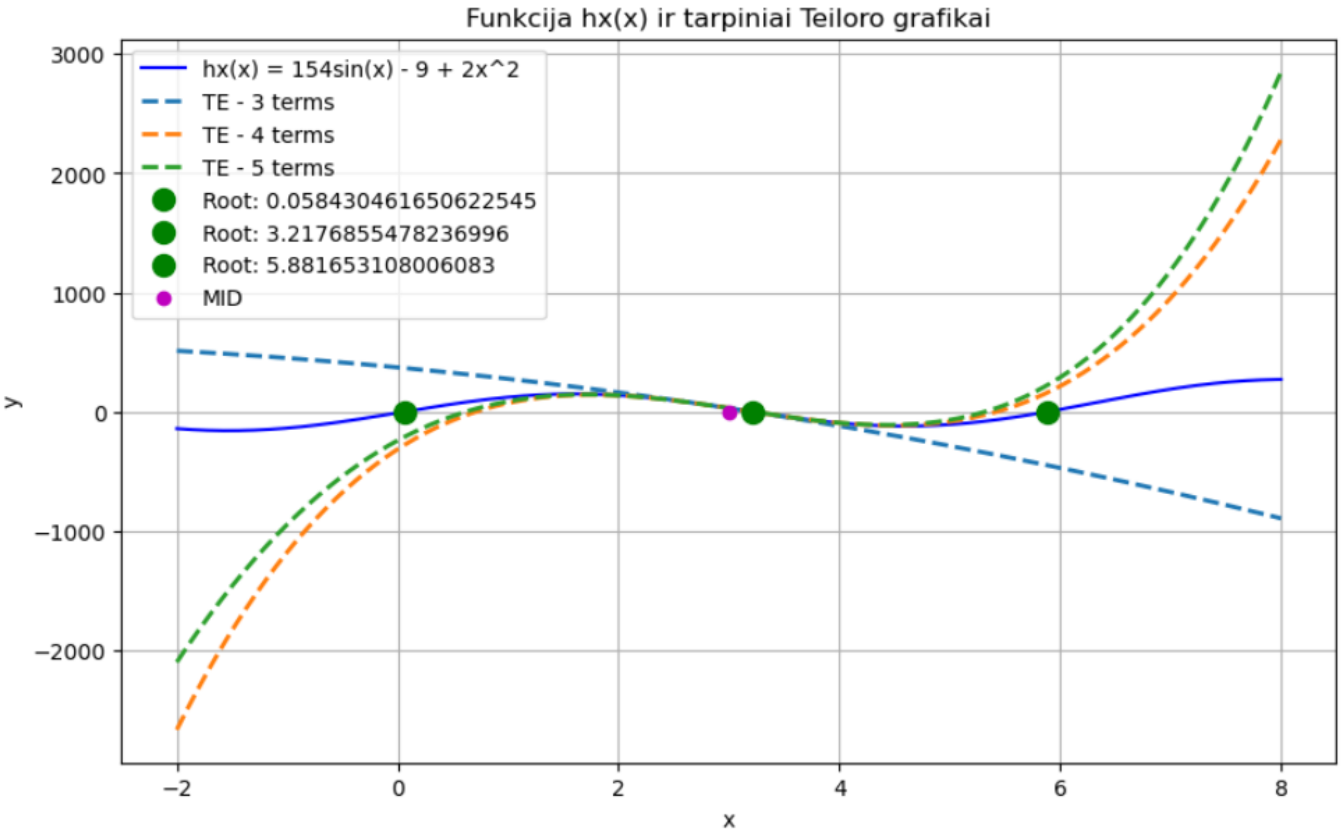
## 2.3 Gautos h(x) funkcijos šaknys Niutono metodu

```
154sin(x) - 9 + 2x^2 šaknys Niutono metodu:
0.058430461650622545
3.2176855478236996
5.881653108006083
```

*16. pav. -154sin(x)-9+2x^2 šaknys Niutono metodu*

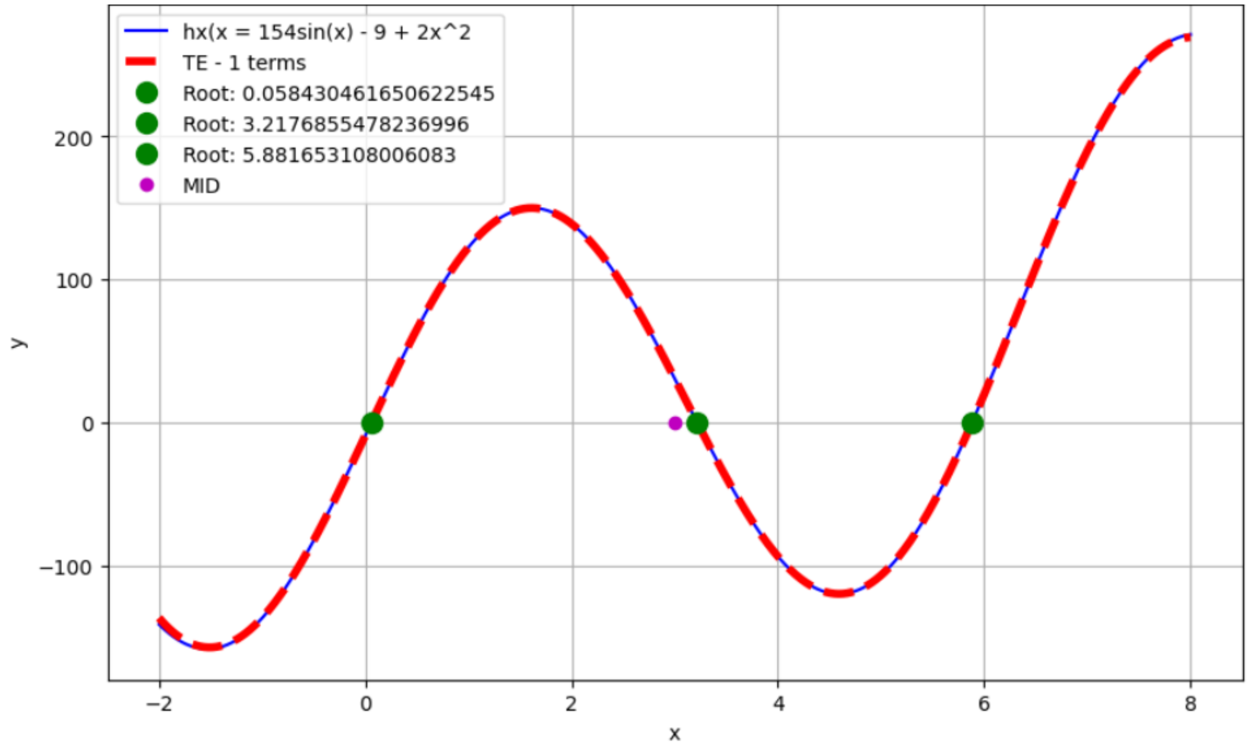
Matas Palujanskas

## 2.4 Grafiškai atvaizduoti tarpiniai grafikai, kai TE narių skaičius 3, 4 ir 5




*17. pav. Tarpiniai grafikai, kai TE narių skaičius 3,4 ir 5*

## 2.5 |1e-4| tikslumą užtikrinantis TE sudarytas daugianaris




*19. pav. tikslumą atitinkančio TE daugianario ir h(x) funkcijų grafikas*

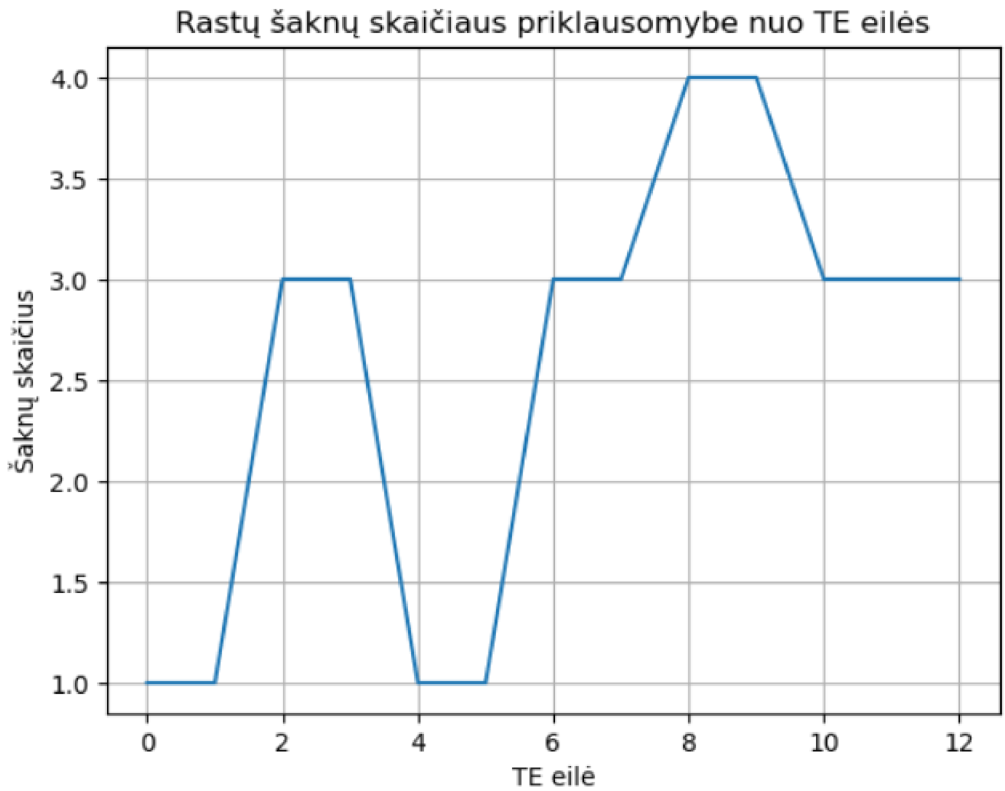
## 2.6 Daugianario analitinė išraiška

```
Daugianario analitinė išraiška:
                       13                          12                          11
- 2.44834326675878e-8·x   + 1.00022424699756e-6·x   - 1.50012876631218e-5·x

                       10                          9                         8
+ 8.39814679946459e-5·x   - 3.73163753920597e-5·x  + 0.00191776620027266·x  -

                    7                        6                      5
0.0366729143340522·x  + 0.0150500731209251·x  + 1.25493792071271·x  + 0.040410

            4                       3                     2
7345031189·x  - 25.7087004791334·x  + 2.03018914450904·x  + 153.986606670068·x

 - 8.99723211282009
```

*20. pav. Daugianario analitinė išraiška*

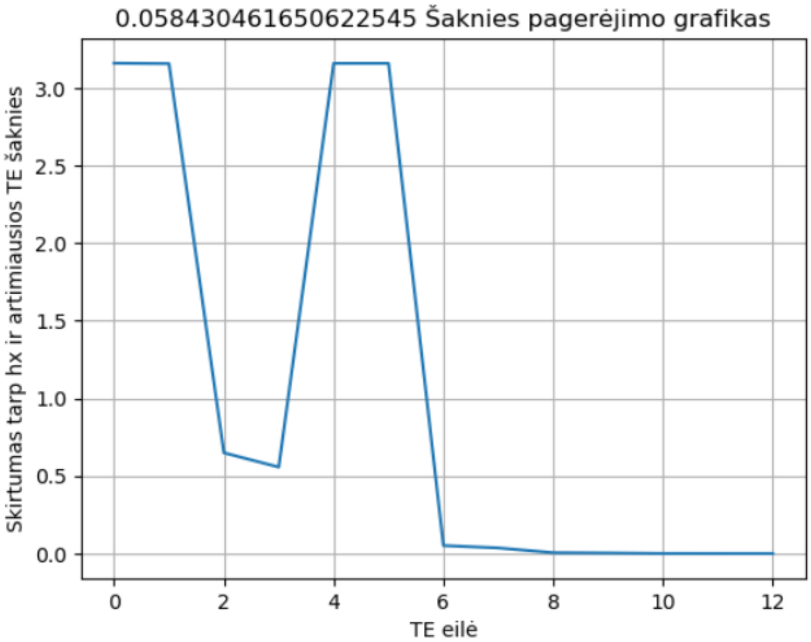
## 2.7 Sprendinių gerėjimo grafikai

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

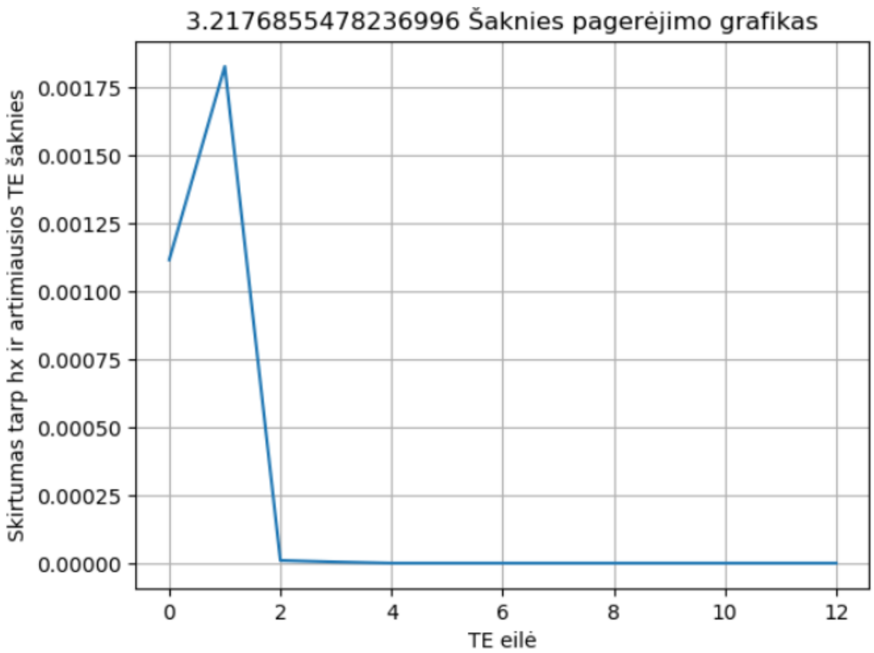



*21. pav. Rastų šaknų skaičiaus priklausomybė nuo TE eilės*

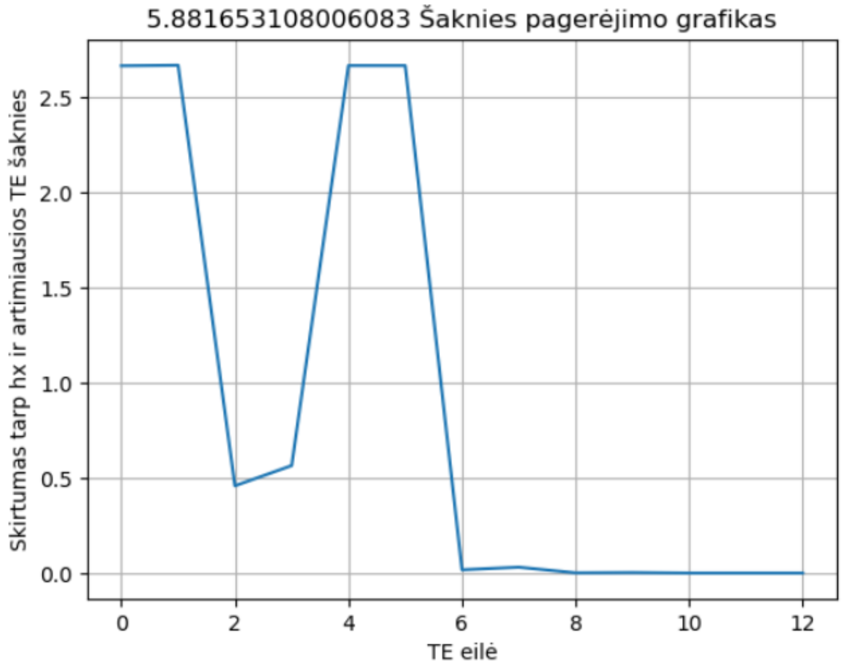
b) Vaizduojami grafikai kiekvienai šakniai, kuriuose oy ašyje pateikti tikslumo įverčiai tarp $h(x)$ apskaičiuotos šaknies ir artimiausios TE šaknies, o ox ašyje TE narių skaičiai.




*22. Pirmos šaknies pagerėjimo grafikas*




*23. pav. Antros šaknies pagerėjimo grafikas*




*24. pav. Trečios šaknies pagerėjimo grafikas*

# 3. Literatūros sąrašas

1. „Skaitiniai metodai ir algoritmai“ „Moodle“ aplinkoje HTTPS://MOODLE.KTU.EDU/COURSE/VIEW.PHP?ID=7639