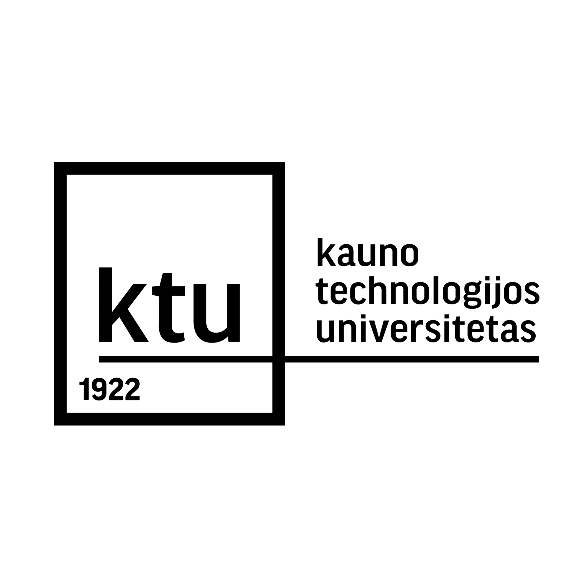
**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

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

**Turinys**

[1. Pirma užduoties dalis 3](#_Toc147067744)

[1.1 Užduoties sąlygos 3](#_Toc147067745)

[1.2 Daugianario programos kodas 4](#_Toc147067746)

[1.3 Transcendentinės programos kodas 6](#_Toc147067747)

[1.4 Daugianario f(x) šaknų intervalo nustatymas 8](#_Toc147067748)

[1.5 Grafinis daugianario ir transcendentinės funkcijos atvaizdavimas 10](#_Toc147067749)

[1.6 Šaknų intervalai 13](#_Toc147067750)

[1.7 Šaknų tikslinimas stygų ir Niutono (liestinių) metodais 15](#_Toc147067751)

[1.8 Šaknų reikšmių tikrinimas išoriniais ištekliais 17](#_Toc147067752)

[2. Antra užduoties dalis 18](#_Toc147067753)

[2.1 Užduotis: 18](#_Toc147067754)

[2.2 Antros užduoties programinis kodas 19](#_Toc147067755)

[2.3 Gautos h(x) funkcijos šaknys Niutono metodu 22](#_Toc147067756)

[2.4 Grafiškai atvaizduoti tarpiniai grafikai, kai TE narių skaičius 3, 4 ir 5 23](#_Toc147067757)

[2.5 |1e-4| tikslumą užtikrinantis TE sudarytas daugianaris 23](#_Toc147067758)

[2.6 Daugianario analitinė išraiška 24](#_Toc147067759)

[2.7 Sprendinių gerėjimo grafikai 24](#_Toc147067760)

[3. Literatūros sąrašas 26](#_Toc147067761)

# **Pirma** **užduoties dalis**

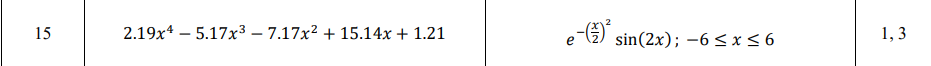
## 1.1 Užduoties sąlygos

Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija, skaičius

Automatiškai sugeneruotas aprašymas

*1. pav. Pirmojo laboratorinio darbo 1 dalis*

**Užduoties variantas: 15**



*2. pav. Užduoties variantas*

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

Bus naudojami stygų ir Niutono (liestinių) metodai.

## 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) = 2.19𝑥4 - 5.17𝑥3 - 7.17𝑥2 + 15.14𝑥 + 1.21 = 0

Daugianario eilė n = 4;

Koeficientai:

𝑎4 =2,19; 𝑎3 = -5,17; 𝑎2 = -7,17; 𝑎1 =15,14; 𝑎0 =1,21;

* „Grubaus“ įverčio radimas:

Paveikslėlis, kuriame yra juodas, tamsa

Automatiškai sugeneruotas aprašymas

* „Tikslesnio” įverčio radimas:

Teigiamoms šaknims:

Paveikslėlis, kuriame yra juodas, tamsa

Automatiškai sugeneruotas aprašymas

Tikslesnio įverčio viršutinis rėžis:

Neigiamoms šaknims:

Koeficientai:

𝑎4 =2.19; 𝑎3 = 5.17; 𝑎2 = 7.17; 𝑎1 =-15.14; 𝑎0 =-1.21;

Galutinis šaknų intervalo įvertis:

− min(𝑅, 𝑅𝑛𝑒𝑖𝑔 ) ≤ 𝑥 ≤ min(𝑅, 𝑅𝑡𝑒𝑖𝑔 ) =>

− min(-7.9; 2.9) ≤ 𝑥 ≤ min(7.9; 46.014) =>

**−2.**𝟗 ≤ 𝒙 ≤ 𝟒**.27**

## 1.5 Grafinis daugianario ir transcendentinės funkcijos atvaizdavimas

**Daugianaris:**

Paveikslėlis, kuriame yra linija, tekstas, Grafikas, diagrama

Automatiškai sugeneruotas aprašymas

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

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

Paveikslėlis, kuriame yra tekstas, linija, Grafikas, diagrama

Automatiškai sugeneruotas aprašymas

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

Iš grafiko galime matyti, kad daugianaris turi 4 šaknis.

**Transcendentinė:**

Paveikslėlis, kuriame yra linija, Grafikas, diagrama, tekstas

Automatiškai sugeneruotas aprašymas

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

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

## 1.6 Šaknų intervalai

**Daugianaris:**

Paveikslėlis, kuriame yra tekstas, Šriftas, tipografija

Automatiškai sugeneruotas aprašymas

1. *pav. Daugianario šaknies intervalai*

Grafiko rėžiai: [-2.9; 4.27]

Paveikslėlis, kuriame yra tekstas, linija, Grafikas, diagrama

Automatiškai sugeneruotas aprašymas

1. *pav. Daugianario šaknų intervalai grafike*

**Transcendentinė funkcija:**

Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija, tipografija

Automatiškai sugeneruotas aprašymas

1. *pav. Transcendentinės funkcijos šaknų intervalai*

Grafiko rėžiai: [-6; 6]

Paveikslėlis, kuriame yra linija, Grafikas, diagrama, tekstas

Automatiškai sugeneruotas aprašymas

1. *pav. Transcendentinės funkcijos šaknų intervalai grafike*

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

**Daugianaris:**

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, dokumentas

Automatiškai sugeneruotas aprašymas

*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:**

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas

Automatiškai sugeneruotas aprašymas

***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.

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas

Automatiškai sugeneruotas aprašymas

*12. pav. Daugianario šaknų palyginimas*

Tikrinimas su wolframalpha.com:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

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

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, dokumentas

Automatiškai sugeneruotas aprašymas

*13. pav. Transcendentinės funkcijos šaknų palyginimas*

Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija, baltas

Automatiškai sugeneruotas aprašymas

pav. 15 Transcendentinės funkcijos šakų palyginimas wolframalpha.com

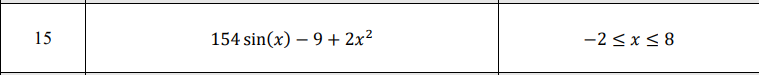
**Išvada.** Šaknys Niutono(liestinių) ir stygų metodais, tiek naudojant „scipy.optimize.fsolve()“ ir wolframalpha.com gautos vienodos.

# 2. **Antra užduoties dalis**

## 2.1 Užduotis:

Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija, skaičius

Automatiškai sugeneruotas aprašymas *14. pav. Pirmojo laboratorinio darbo 2 dalis*



*15. pav. Antros dalies užduoties variantas*

## 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()  
  
  
*# 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

Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija, baltas

Automatiškai sugeneruotas aprašymas

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

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

Paveikslėlis, kuriame yra tekstas, linija, diagrama, Grafikas

Automatiškai sugeneruotas aprašymas

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

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

Paveikslėlis, kuriame yra Grafikas, linija, tekstas, diagrama

Automatiškai sugeneruotas aprašymas

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

## 2.6 Daugianario analitinė išraiška

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

*20. pav. Daugianario analitinė išraiška*

## 2.7 Sprendinių gerėjimo grafikai

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

Paveikslėlis, kuriame yra linija, diagrama, tekstas, Grafikas

Automatiškai sugeneruotas aprašymas

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

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

Paveikslėlis, kuriame yra tekstas, linija, Grafikas, diagrama

Automatiškai sugeneruotas aprašymas

* 1. *Pirmos šaknies pagerėjimo grafikas*

Paveikslėlis, kuriame yra tekstas, linija, Grafikas, diagrama

Automatiškai sugeneruotas aprašymas

* 1. *pav. Antros šaknies pagerėjimo grafikas*

Paveikslėlis, kuriame yra tekstas, linija, diagrama, Grafikas

Automatiškai sugeneruotas aprašymas

* 1. *pav. Trečios šaknies pagerėjimo grafikas*

# 3. Literatūros sąrašas

1. „Skaitiniai metodai ir algoritmai“ „Moodle“ aplinkoje [HTTPS://MOODLE.KTU.EDU/COURSE/VIEW.PHP?ID=7639](https://moodle.ktu.edu/course/view.php?id=7639)