KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Modulio P170B115 „Skaitiniai metodai ir algoritmai“

Laboratorinio darbo ataskaita
Pirmas laboratorinis darbas

Dėstytojas
Doc. Kriščiūnas Andrius

Studentas
Rokas Puzonas

KAUNAS, 2023

Turinys

1. Užduoties variantas

Variantas: 20
Metodai: Stygų, Niutono (liestinių)

$$f(x) = 1.34x^4 - 16.35x^3 + 65.13x^2 - 83.45x - 3.27$$
$$g(x) = \sin(x) - \frac{\ln(x)}{2} + 0.1; 0.1 \le x \le 10$$
$$h(x) = -8\sin(x) + 32\sin(10x) - 10;\ 0 \le x\ \le 1$$

2.1 Dalis

Išspręskite netiesines lygtis, kai lygties funkcija yra daugianaris $f(x) = 0$ ir transcendentinė funkcija $g(x) = 0$.

2.1. Daugenario šaknies intervalo nustatymas

„Grubaus“ metodo funkcija:

```python
def grubus_R(LF_terms):
    return 1 + max(abs(ai) for ai in LF_terms[:-1]) / LF_terms[-1]
```

„Tikslesnio“ metodo funkcija:

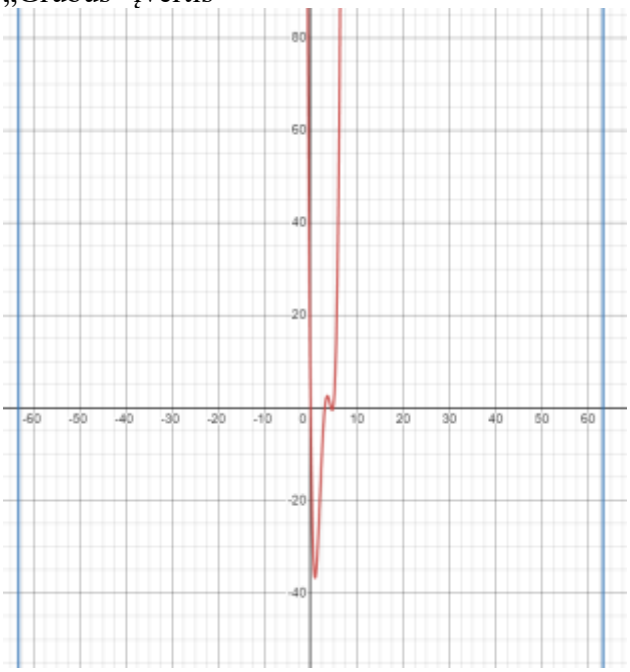
```python
def mul_every_by(numbers: list[float], scalar: float):
    return list(map(lambda a: scalar*a, numbers))

def tikslesnis_R_range(LF_terms: list[float]):
    LF_terms = LF_terms.copy()
    if LF_terms[-1] < 0:
        # f(x) => -f(x)
        LF_terms = mul_every_by(LF_terms, -1)


    k = len(LF_terms) - max(i for i, a in enumerate(LF_terms[:-1]) if a < 0) - 1
    B = max(abs(a) for a in LF_terms[:-1] if a < 0)
    R_teig = 1 + (B / LF_terms[-1]) ** (1/k)

    # f(x) => f(-x)
    for i in range(1, len(LF_terms), 2):
        LF_terms[i] *= -1

    if len(LF_terms) % 2 == 0:
        # f(x) => -f(x)
        LF_terms = mul_every_by(LF_terms, -1)

    if any(True for a in LF_terms[:-1] if a < 0) > 0:
        B = max(abs(a) for a in LF_terms[:-1] if a < 0)
        k = len(LF_terms) - max(i for i, a in enumerate(LF_terms[:-1]) if a < 0) - 1
        R_neig = 1 + (B / LF_terms[-1]) ** (1/k)
    else:
        R_neig = 0

    return (R_neig, R_teig)
```

Bendras intervalo įvertis naudojant „grubų“ ir „tikslesnį“:

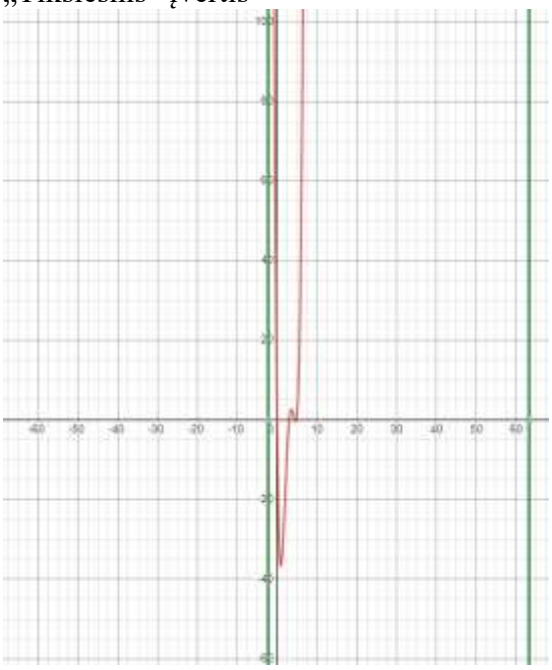
```python
def roots_range(LF_terms: list[float]):
    R = grubus_R(LF_terms)
    R_neig, R_teig = tikslesnis_R_range(LF_terms)
    return (-min(R, R_neig), min(R, R_teig))
```

Programos rezultatas f(x) bendro intervalo įverčiui:
Nuo -2.249 iki 63.276.

„Grubus“ įvertis



„Tikslesnis“ įvertis



2.2. Šaknų atskyrimo intervalų radimas naudojant skenavimo algoritmą

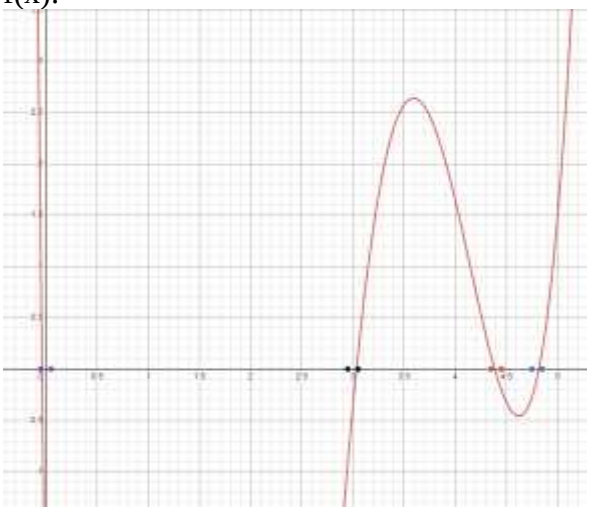
Skenavimo funkcija:

```
def scan_root_ranges(LF, x1, x2, step=0.1):
    x = x1
    while x < x2:
        if np.sign(LF(x)) != np.sign(LF(x+step)):
            yield (x, x+step)
        x += step
```

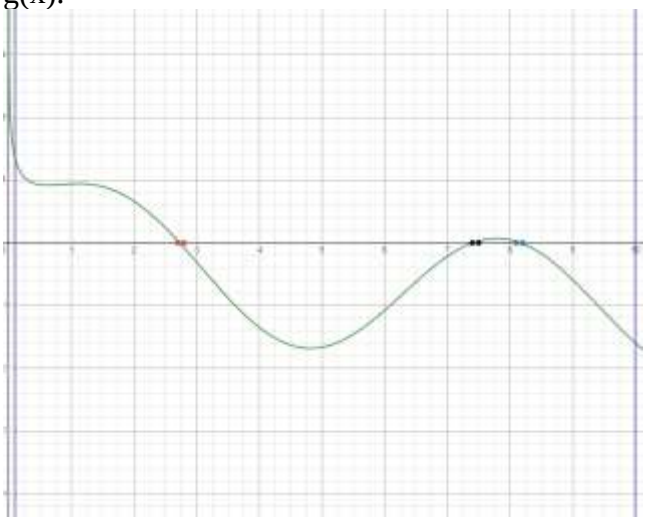
Programos rezultatas f(x) ir g(x) šaknų atskyrimo intervalai naudojant skenavimo algoritmą, skenavimo žingsnio dydis yra 0.1:

Funkcija	Intervalo pradžia	Intervalo pabaiga
f(x)	-0.04985	0.05014
	2.95014	3.05014
	4.35014	4.45014
	4.75014	4.85014
g(x)	2.7	2.8
	7.4	7.5
	8.1	8.2

f(x):



g(x):



2.3. Šaknies tikslinimas

Naudojant stygų metodą tikslinimiui funkcija su visualizacija:

```
def solve_stygos(F, min_x, max_x, view_x, max_iterations=30, epsilon=1e-6, animation_delay=0.05):

    numpy_F = lambda x: F(NumpyMathLibrary, x)

    textbox = create_scroll_textbox(140, 20)
    textbox_append_line(textbox, "-------- Stygu metodas -------------")

    my_plot = MyPlot(animation_delay)
    my_plot.plot_function(view_x[0], view_x[1], numpy_F)
    my_plot.add_zero_line(view_x[0], view_x[1])

    textbox_append_line(textbox, "Saknies reiksmes tikslinimas Stygu metodu:")

    results = []

    for x1, x2 in scan_root_ranges(numpy_F, min_x, max_x):
        start_x1 = x1
        start_x2 = x2
        my_plot.subplot.plot([x1, x2], [0, 0], 'mo--')
        plt.draw()

        f1 = numpy_F(x1)
        f2 = numpy_F(x2)
        my_plot.update_edge_points(x1, f1, x2, f2)

        for i in range(1, max_iterations + 1):
            if f2 == 0:
                f2 += epsilon/2
            k = np.abs(f1 / f2)
            x_middle = (x1 + k * x2) / (1 + k)  # stygos nulio taskas
            middle_value = numpy_F(x_middle)

            my_plot.clear_connecting_line()
            my_plot.update_middle_point(x_middle, middle_value)
```

```
            if np.sign(f1) != np.sign(middle_value):
                x2 = x_middle
                f2 = middle_value
            else:
                x1 = x_middle
                f1 = middle_value

            my_plot.clear_middle_point()
            my_plot.update_edge_points(x1, f1, x2, f2)

            tksl = get_guess_accuracy(x1, x2, f1, f2, epsilon)
            textbox_append_line(textbox, f"{i:2d}. x1={x1:-20g}, x2={x2:-20g}, f1={f1:-20g}, f2={f2:-20g}, tikslumas={tksl:-
20g}")

            if tksl < epsilon:
                break

        tksl = get_guess_accuracy(x1, x2, f1, f2, epsilon)
        if tksl < epsilon:
            print(f"Isspresta: x={x1:g}, f={f1:g}, tikslumas={tksl:g}")
        else:
            print("Tikslumas nepasiektas")

        results.append([start_x1, start_x2, x1, tksl, i])

    return results
```

Naudojant Niutono liestinių metodą tikslinimiui funkcija su visualizacija:

```
def solve_liestiniu(F, min_x, max_x, view_x, epsilon=1e-8, animation_delay=0.5):
    sympy_F = lambda x: F(SympyMathLibrary, x)
    numpy_F = lambda x: F(NumpyMathLibrary, x)

    x_symbol = sympy.symbols("x")
    DF_symbolical = sympy_F(x_symbol).diff(x_symbol)
    DF = sympy.lambdify((x_symbol), DF_symbolical)

    x=np.arange(view_x[0], view_x[1], 0.05); y=numpy_F(x)

    plt.xlabel("x"); plt.ylabel("y"); plt.plot(x, y, 'b'); plt.grid()

    results = []

    for x1, x2 in scan_root_ranges(numpy_F, min_x, max_x):
        start_x = (x1+x2)/2

        xi = start_x
        iterations = 0
        while np.abs(numpy_F(xi)) > epsilon:
            xi_bef = xi
            xi = xi - (1 / DF(xi)) * numpy_F(xi)
            iterations += 1

            plt.plot([xi_bef], numpy_F(xi_bef), 'ob')
            plt.plot([xi], [0], 'or')
            plt.plot([xi_bef, xi], [numpy_F(xi_bef), 0], 'r-')
            plt.plot([xi, xi], [0, numpy_F(xi)], 'g--')
            plt.draw()
            plt.pause(animation_delay)

        results.append((x1, x2, xi, np.abs(numpy_F(xi)), iterations))

    return results
```

Programos šaknų radimo rezultatas naudojant Niutono (liestinių) ir stygų metodus:

Funkcija	Metodas	Pradinis intervalas	Kirtimo taškas	Tikslumas	Iteracijos
f(x)	Stygų	[-0.04985; 0.05014]	-0.0380447	7.51696e-07	10
		[2.95014; 3.05014]	3.03508	2.19478e-16	15

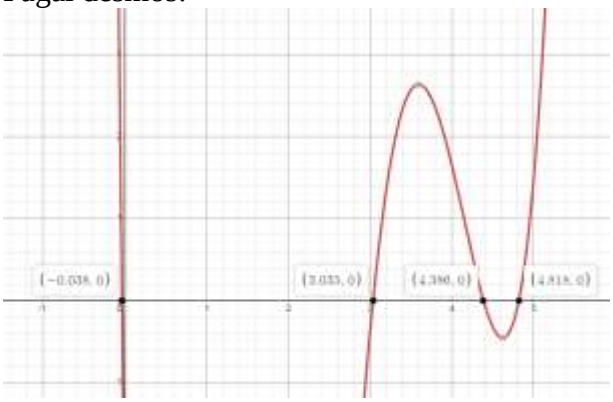
		[4.35014; 4.45014]	4.38623	1.22305e-13	9
		[4.75014; 4.85014]	4.81822	5.71446e-15	14
	Niutono (liestinių)	[-0.04985; 0.05014]	-0.0380447	6.68674e-11	3
		[2.95014; 3.05014]	3.03508	1.41154e-11	3
		[4.35014; 4.45014]	4.38623	1.52323e-13	3
		[4.75014; 4.85014]	4.81822	2.16783e-10	3
g(x)	Stygų	[2.7; 2.8]	2.72811	1.62783e-16	8
		[7.4; 7.5]	7.4	0.000372351	30
		[8.1; 8.2]	8.17066	8.13851e-10	10
	Niutono (liestinių)	[2.7; 2.8]	2.72811	8.13851e-10	2
		[7.4; 7.5]	7.40551	3.70581e-11	3
		[8.1; 8.2]	8.17066	8.10185e-14	3

Iš programos rezultatų galima daryti kelias išvadas:

- Stygų metodas nepasieks kirtimo taško jeigu vienas iš pradinio intervalo galų yra labai arti kirtimo taško
- Stygų metodas geriau veikia kuo arčiau vidurio kirtimo taškas yra pradinio intervalo vidurio, suras kirtimo tašką per mažiau iteracijų
- Niutono (liestinių) metodas suras kirtimo tašką per mažiau iteracijų, nes tame metode funkcijos liestinė gali tiksliai pasakyti, kaip reikia x keisti, kad artėti prie y=0

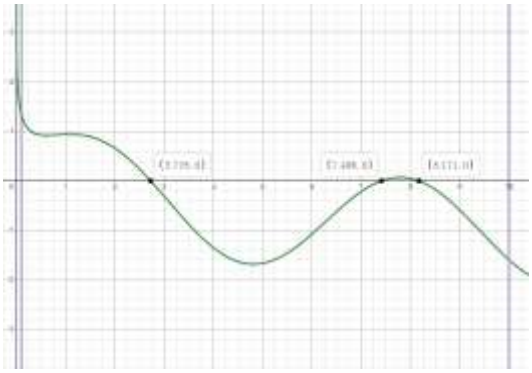
2.4. Gautų šaknų patikrinimas

f(x) šaknų tikrinimas:
Pagal desmos:

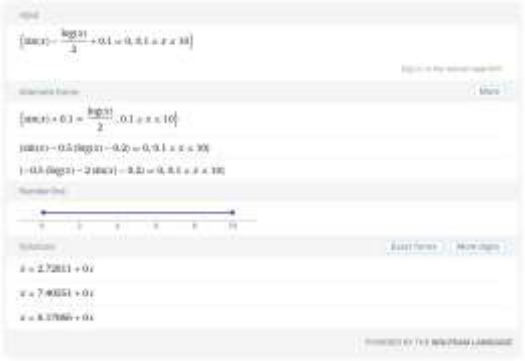


Pagal WolframAlpha:

g(x) šaknų tikrinimas:
Pagal desmos:



Pagal WolframAlpha:

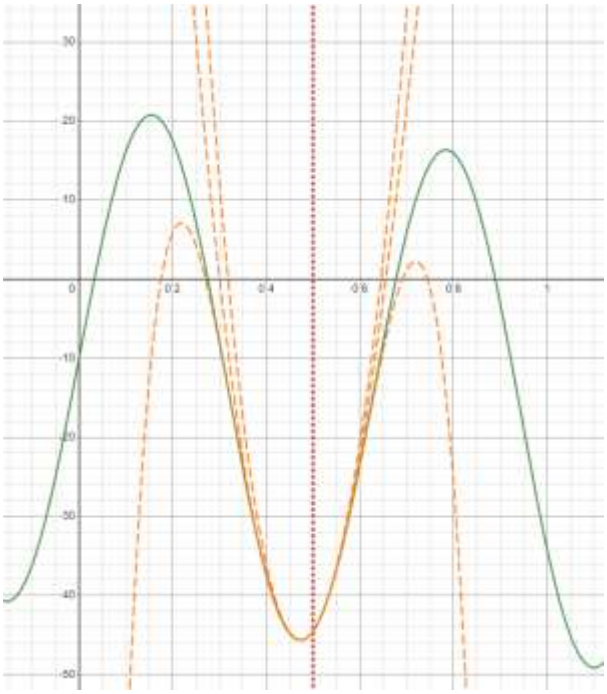


3.2 Dalis

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).

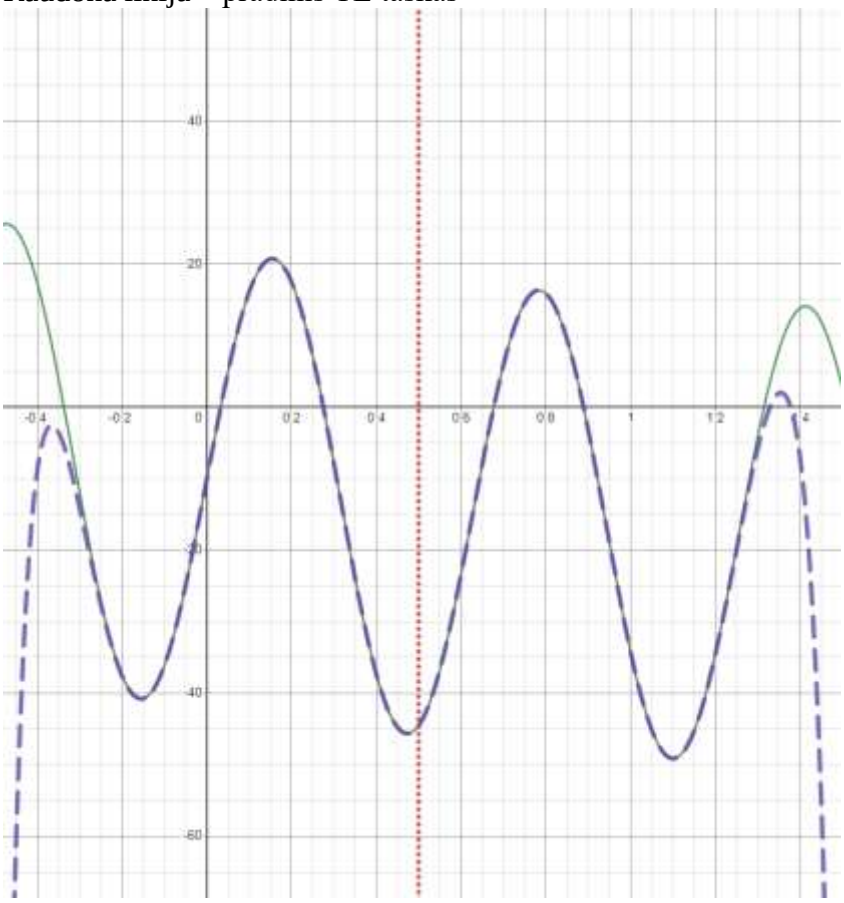
3.1. TE tarpinės funkcijos

Žalia linija – h(x)
Raudona linija – pradinis TE taškas
Geltonos linijos – 3, 4 ir 5 eilės teiloro eilutės



3.2. TE tikslumo atvaizdavimas grafiškai

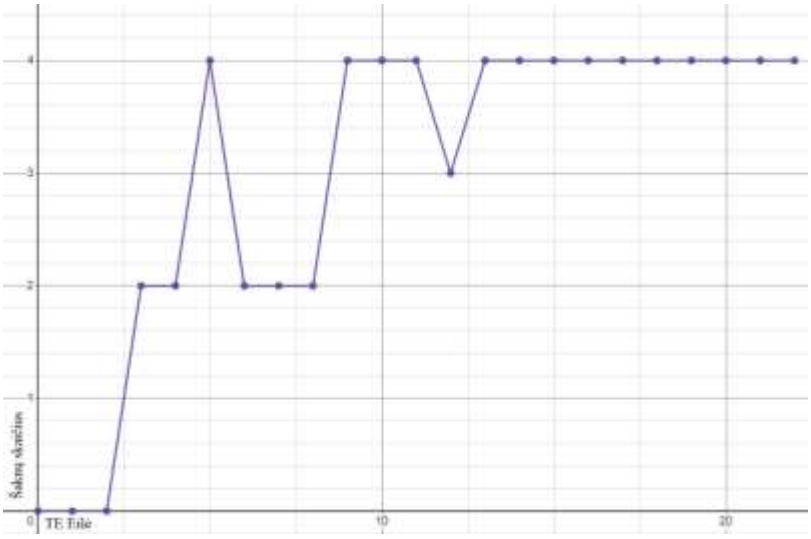
Žalia linija – h(x)
Violetinė linija – 22 TE funkcija
Raudona linija – pradinis TE taškas



3.3. Analitinė išraiška

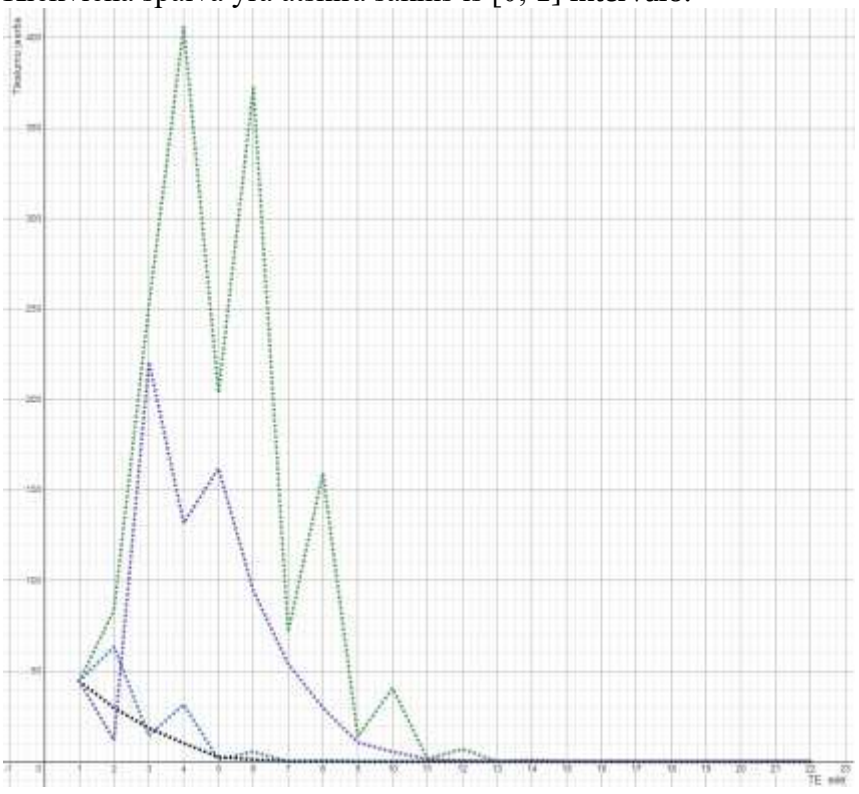
$$
\begin{aligned}
&(177.667303615504\,x^{21}) + (-3126.78135854510\,x^{20}) \\
&+ (21194.0774704520\,x^{19}) + (-77565.9669162660\,x^{18}) \\
&+ (173706.571367708\,x^{17}) + (-257557.626211679\,x^{16}) \\
&+ (289245.368829960\,x^{15}) + (-307120.810591593\,x^{14}) \\
&+ (297668.883815680\,x^{13}) + (-163671.552571374\,x^{12}) \\
&+ (10591.1193766985\,x^{11}) + (-42118.7003803260\,x^{10}) \\
&+ (104538.523032389\,x^{9}) + (-5296.17992616654\,x^{8}) \\
&+ (-62071.4960597741\,x^{7}) + (-312.183401104267\,x^{6}) \\
&+ (26721.8934508589\,x^{5}) + (-7.70217548539222\,x^{4}) \\
&+ (-5331.18733792747\,x^{3}) + (-0.0610647714044319\,x^{2}) \\
&+ (312.002912257649\,x^{1}) + (-10.0000662682033\,x^{0})
\end{aligned}
$$

3.4. Šaknų skaičiaus priklausomybė nuo TE eilės



3.5. Šaknų tikslumo priklausomybė nuo TE eilės

Kiekviena spalva yra atskira šaknis iš [0; 1] intervalo.



4. Programinis kodas

```
import math
import matplotlib.pyplot as plt
import numpy as np
import tkinter as tk
import sympy
from tabulate import tabulate

class MyPlot:
    def __init__(self, animation_delay):
        self.animation_delay = animation_delay
        self.figure = plt.figure()
        self.subplot = self.figure.add_subplot(1, 1, 1)
        self.subplot.set_xlabel('x')
        self.subplot.set_ylabel('y')
        self.m1 = None
        self.m2 = None
        self.st = None
        self.mmid = None

    def plot_function(self, x1, x2, func, color='b'):
        x_range = np.linspace(x1, x2, 500)
        self.subplot.plot(x_range, func(x_range), color)

    def add_zero_line(self, x1, x2):
        self.subplot.plot([x1, x2], [0, 0], 'k.--')
        plt.draw()

    def update_edge_points(self, x1, y1, x2, y2):
        if self.m1:
            self.m1.remove()
        if self.m2:
            self.m2.remove()
        if self.st:
            self.st.remove()

        self.m1, = self.subplot.plot([x1, x1], [0, y1], 'mo--')
        plt.draw()

        self.m2, = self.subplot.plot([x2, x2], [0, y2], 'mo--')
        plt.draw()

        self.st, = self.subplot.plot([x1, x2], [y1, y2], 'k.-.')
        plt.draw()
        plt.pause(self.animation_delay)

    def clear_connecting_line(self):
        if self.st:
            self.st.remove()
            self.st = None
            plt.draw()

    def update_middle_point(self, x, y):
        if self.mmid:
            self.mmid.remove()

        self.mmid, = self.subplot.plot([x, x], [0, y], 'ks--')
        plt.draw()
        plt.pause(self.animation_delay)

    def clear_middle_point(self):
        if self.mmid:
            self.mmid.remove()
            self.mmid = None
            plt.draw()

class StandardMathLibrary:
    sin = math.sin
    cos = math.cos
    log = math.log

class NumpyMathLibrary:
    sin = np.sin
    cos = np.cos
    log = np.log

class SympyMathLibrary:
```

```
    sin = sympy.sin
    cos = sympy.cos
    log = sympy.log

def create_scroll_textbox(width: int, height: int):
    root = tk.Tk()
    root.title("Scroll Text Box")

    frame1 = tk.Frame(root)
    frame1.pack()

    # T = ScrolledText(root, height=h, width=w,wrap=WORD) # jeigu reikia scrolinti tik pagal h
    scrollbar_y = tk.Scrollbar(frame1, orient='vertical')
    scrollbar_y.pack(side=tk.RIGHT, fill=tk.Y)

    scrollbar_x = tk.Scrollbar(frame1, orient='horizontal')
    scrollbar_x.pack(side=tk.BOTTOM, fill=tk.X)

    textbox = tk.Text(frame1,
                      width=width, height=height,
                      yscrollcommand=scrollbar_y.set, xscrollcommand=scrollbar_x.set,
                      wrap=tk.NONE)
    textbox.pack()

    scrollbar_x.config(command=textbox.xview)
    scrollbar_y.config(command=textbox.yview)

    return textbox

def textbox_append_line(textbox: tk.Text, line: str):
    textbox.insert(tk.END, line + "\n")
    textbox.yview(tk.END)

def get_guess_accuracy(x1, x2, f1, f2, eps):
    x_abs_diff = np.abs(x1-x2)
    x_abs_sum = x1+x2
    f1_abs = np.abs(f1)
    f2_abs = np.abs(f2)
    if not np.isscalar(x1):
        x_abs_diff = sum(x_abs_diff)
        x_abs_sum = sum(x_abs_sum)
        f1_abs = sum(f1_abs)
        f2_abs = sum(f2_abs)

    if x_abs_sum > eps:
        return x_abs_diff/(x_abs_sum + f1_abs + f2_abs)
    else:
        return x_abs_diff + f1_abs + f2_abs

def scan_root_ranges(LF, x1, x2, step=0.1):
    x = x1
    while x < x2:
        if np.sign(LF(x)) != np.sign(LF(x+step)):
            yield (x, x+step)
        x += step

def grubus_R(LF_terms):
    return 1 + max(abs(ai) for ai in LF_terms[:-1]) / LF_terms[-1]

def mul_every_by(numbers: list[float], scalar: float):
    return list(map(lambda a: scalar*a, numbers))

def tikslesnis_R_range(LF_terms: list[float]):
    LF_terms = LF_terms.copy()
    if LF_terms[-1] < 0:
        # f(x) => -f(x)
        LF_terms = mul_every_by(LF_terms, -1)


    k = len(LF_terms) - max(i for i, a in enumerate(LF_terms[:-1]) if a < 0) - 1
    B = max(abs(a) for a in LF_terms[:-1] if a < 0)
    R_teig = 1 + (B / LF_terms[-1]) ** (1/k)

    # f(x) => f(-x)
    for i in range(1, len(LF_terms), 2):
        LF_terms[i] *= -1
```

```
    if len(LF_terms) % 2 == 0:
        # f(x) => -f(x)
        LF_terms = mul_every_by(LF_terms, -1)

    if any(True for a in LF_terms[:-1] if a < 0) > 0:
        B = max(abs(a) for a in LF_terms[:-1] if a < 0)
        k = len(LF_terms) - max(i for i, a in enumerate(LF_terms[:-1]) if a < 0) - 1
        R_neig = 1 + (B / LF_terms[-1]) ** (1/k)
    else:
        R_neig = 0

    return (R_neig, R_teig)

def roots_range(LF_terms: list[float]):
    R = grubus_R(LF_terms)
    R_neig, R_teig = tikslesnis_R_range(LF_terms)
    return (-min(R, R_neig), min(R, R_teig))

def get_coeffs(func):
    x_symbol = sympy.symbols("x")
    a: sympy.Poly = sympy.Poly(func(x_symbol), x_symbol)
    coeffs = list(map(float, a.coeffs()))
    coeffs.reverse()
    return coeffs

def solve_stygos(F, min_x, max_x, view_x, max_iterations=30, epsilon=1e-6, animation_delay=0.05):
    numpy_F = lambda x: F(NumpyMathLibrary, x)

    textbox = create_scroll_textbox(140, 20)
    textbox_append_line(textbox, "-------- Stygu metodas ------------")

    my_plot = MyPlot(animation_delay)
    my_plot.plot_function(view_x[0], view_x[1], numpy_F)
    my_plot.add_zero_line(view_x[0], view_x[1])

    textbox_append_line(textbox, "Saknies reiksmes tikslinimas Stygu metodu:")

    results = []

    for x1, x2 in scan_root_ranges(numpy_F, min_x, max_x):
        start_x1 = x1
        start_x2 = x2
        my_plot.subplot.plot([x1, x2], [0, 0], 'mo--')
        plt.draw()

        f1 = numpy_F(x1)
        f2 = numpy_F(x2)
        my_plot.update_edge_points(x1, f1, x2, f2)

        for i in range(1, max_iterations + 1):
            if f2 == 0:
                f2 += epsilon/2
            k = np.abs(f1 / f2)
            x_middle = (x1 + k * x2) / (1 + k)  # stygos nulio taskas
            middle_value = numpy_F(x_middle)

            my_plot.clear_connecting_line()
            my_plot.update_middle_point(x_middle, middle_value)

            if np.sign(f1) != np.sign(middle_value):
                x2 = x_middle
                f2 = middle_value
            else:
                x1 = x_middle
                f1 = middle_value

            my_plot.clear_middle_point()
            my_plot.update_edge_points(x1, f1, x2, f2)

            tksl = get_guess_accuracy(x1, x2, f1, f2, epsilon)
            textbox_append_line(textbox, f"{i:2d}. x1={x1:-20g}, x2={x2:-20g}, f1={f1:-20g}, f2={f2:-20g}, tikslumas={tksl:-
20g}")

            if tksl < epsilon:
                break
```

```
        tksl = get_guess_accuracy(x1, x2, f1, f2, epsilon)
        if tksl < epsilon:
            print(f"Isspresta: x={x1:g}, f={f1:g}, tikslumas={tksl:g}")
        else:
            print("Tikslumas nepasiektas")

        results.append([start_x1, start_x2, x1, tksl, i])

    return results

def solve_liestiniu(F, min_x, max_x, view_x, epsilon=1e-8, animation_delay=0.5):
    sympy_F = lambda x: F(SympyMathLibrary, x)
    numpy_F = lambda x: F(NumpyMathLibrary, x)

    x_symbol = sympy.symbols("x")
    DF_symbolical = sympy_F(x_symbol).diff(x_symbol)
    DF = sympy.lambdify((x_symbol), DF_symbolical)

    x=np.arange(view_x[0], view_x[1], 0.05); y=numpy_F(x)

    plt.xlabel("x"); plt.ylabel("y"); plt.plot(x, y, 'b'); plt.grid()

    results = []

    for x1, x2 in scan_root_ranges(numpy_F, min_x, max_x):
        start_x = (x1+x2)/2

        xi = start_x
        iterations = 0
        while np.abs(numpy_F(xi)) > epsilon:
            xi_bef = xi
            xi = xi - (1 / DF(xi)) * numpy_F(xi)
            iterations += 1

            plt.plot([xi_bef], numpy_F(xi_bef), 'ob')
            plt.plot([xi], [0], 'or')
            plt.plot([xi_bef, xi], [numpy_F(xi_bef), 0], 'r-')
            plt.plot([xi, xi], [0, numpy_F(xi)], 'g--')
            plt.draw()
            plt.pause(animation_delay)

        results.append((x1, x2, xi, np.abs(numpy_F(xi)), iterations))

    return results

def prefix_results(prefix, results):
    return map(lambda line: [*prefix, *line], results)

def main_1(F, G, G_range, F_view_x):
    results = []

    F_min_x, F_max_x = roots_range(get_coeffs(lambda x: F(StandardMathLibrary, x)))
    results.extend(prefix_results(["f(x)", "Stygų"    ], solve_stygos(F, F_min_x, F_max_x, F_view_x)))
    results.extend(prefix_results(["f(x)", "Liestinių"], solve_liestiniu(F, F_min_x, F_max_x, F_view_x)))

    results.extend(prefix_results(["g(x)", "Stygų"    ], solve_stygos(G, G_range[0], G_range[1], G_range)))
    results.extend(prefix_results(["g(x)", "Liestinių"], solve_liestiniu(G, G_range[0], G_range[1], G_range)))

    headers = ["Funkcija", "Metodas", "Skenavimo min", "Skenavimo max", "Kirtimo taškas","Tikslumas", "Iteracijos"]
    print(tabulate(results, headers, tablefmt="grid"))

def print_coeffs_desmos(coeffs):
    coeffs_strs = []
    for i, coeff in enumerate(coeffs):
        x_power = len(coeffs)-i-1
        x_power_str = "^".join(c for c in str(x_power))
        coeffs_strs.append(f"({coeff} * x^{x_power_str})")
    print(" + ".join(coeffs_strs))

def main_2(H, H_range, epsilon=1e-4, animation_delay=0.05):
    standard_H = lambda x: H(StandardMathLibrary, x)
    numpy_H = lambda x: H(NumpyMathLibrary, x)
    sympy_H = lambda x: H(SympyMathLibrary, x)

    my_plot = MyPlot(animation_delay)
```

```
    my_plot.plot_function(H_range[0], H_range[1], numpy_H)
    plt.pause(animation_delay)

    x,f,fp,df=sympy.symbols(('x','f','fp','df'))

    f=sympy_H(x)
    x0=(H_range[0]+H_range[1])/2

    fp=f.subs(x,x0)
    N = 1

    while True:
        f=f.diff(x)
        fp=fp+f.subs(x,x0)/math.factorial(N)*(x-x0)**N;
        N += 1

        a=sympy.Poly(fp,x)
        if abs(a(H_range[0]) - standard_H(H_range[0])) < epsilon and abs(a(H_range[1]) - standard_H(H_range[1])) < epsilon:
            break

        if N % 5 == 0:
            my_plot.plot_function(H_range[0], H_range[1], sympy.lambdify(x, fp, 'numpy'), 'r--')
            plt.pause(animation_delay)

    a=sympy.Poly(fp,x)

    print(N)
    print_coeffs_desmos(a.all_coeffs())
    my_plot.plot_function(H_range[0], H_range[1], sympy.lambdify(x, fp, 'numpy'), 'r--')
    plt.pause(animation_delay)

# Variantas: 20
main_1(
    F=(lambda m, x: 1.34 * (x**4) - 16.35 * (x**3) + 65.13 * (x**2) - 83.45 * (x**1) - 3.27),
    G=(lambda m, x: m.sin(x) - m.log(x)/2 + 0.1),
    G_range=(0.1, 10),
    F_view_x=(-0.5, 5)
)

main_2(
    H=(lambda m, x: -8*m.sin(x) + 32*m.sin(10*x) - 10),
    H_range=(0, 1)
)

input("Baigti darba! Wooo! Press 'ENTER'")
```