Kauno technologijos universitetas

Informatikos fakultetas

P170B115 Skaitiniai metodai ir algoritmai

3 projektinė užduotis. Interpoliavimas, aproksimavimas.

Nedas Liaudanskis

Studentas

doc. Čalnerytė Dalia

Dėstytoja

KAUNAS, 2023

Turinys

Įvadas

Interpoliavimas yra matematinis procesas, kuriuo yra konstruojama funkcija arba kreivė, kuri praeina per duotas taškas arba duotus duomenis. Pagrindinė šio proceso idėja yra tai, kad turint baigtinį skaičių duomenų taškų galima sukurti funkciją, kuri praeina per kiekvieną iš šių taškų.

I užduotis. Interpoliavimas daugianariu: Pirmoje dalyje buvo interpoliuojama funkcija, naudojant tolygiai pasiskirstytus ir Čiobyševo abscises. Rezultatai pateikti grafiškai, parodyant duotąją funkciją ir interpoliuojančią funkciją. Rodomi ir tolygiai išskirstyti taškai ir taškai sudėlioti naudojant Čiobyševo abscises.

Norint nustatyti Čiobyševo absicisių intervalus naudojame:

$$x_i = \cos\left(\frac{\pi(2i+1)}{2n}\right), i = 0,1,\dots,n-1;$$
$$X_i = \frac{b+a}{2} + \frac{b-a}{2}x_i, i = 0,1,\dots,n-1;$$

Per funkcijos grafiko taškus ties Čiobyševo abscisėmis pravesta interpoliuojančio daugianario kreivė yra "mažiausiai banguota", palyginus su tos pačios eilės daugianarių kreivėmis, pravestomis per kitaip išdėstytą tą patį skaičių interpoliavimo mazgų.

Funkcija eina per kiekvieną interpoliacijos tašką, skaičiuoja Lagranžo bazinį polinomą tam taškui ir susumuoją rezultatus, kad gautų Lagranžo interpoliacinį polinomą.

Naudojamos formulės, skaičiuojant Lagranžo interpoliavimą:

$$P(x) = \sum_{i=0}^{n-1} y_i * L_i(x);$$
$$L_i(x) = \prod_{j=0, j\neq i}^{n-1} \frac{x - x_j}{x_i - x_j};$$

Variantas:

Var. Nr.	Funkcijos išraiška	Bazinė funkcija
10	$\cos(2\cdot x)\cdot(\sin(3\cdot x)+1{,}5)-\cos\frac{x}{5}; -2 \le x \le 3;$	Vienanarių

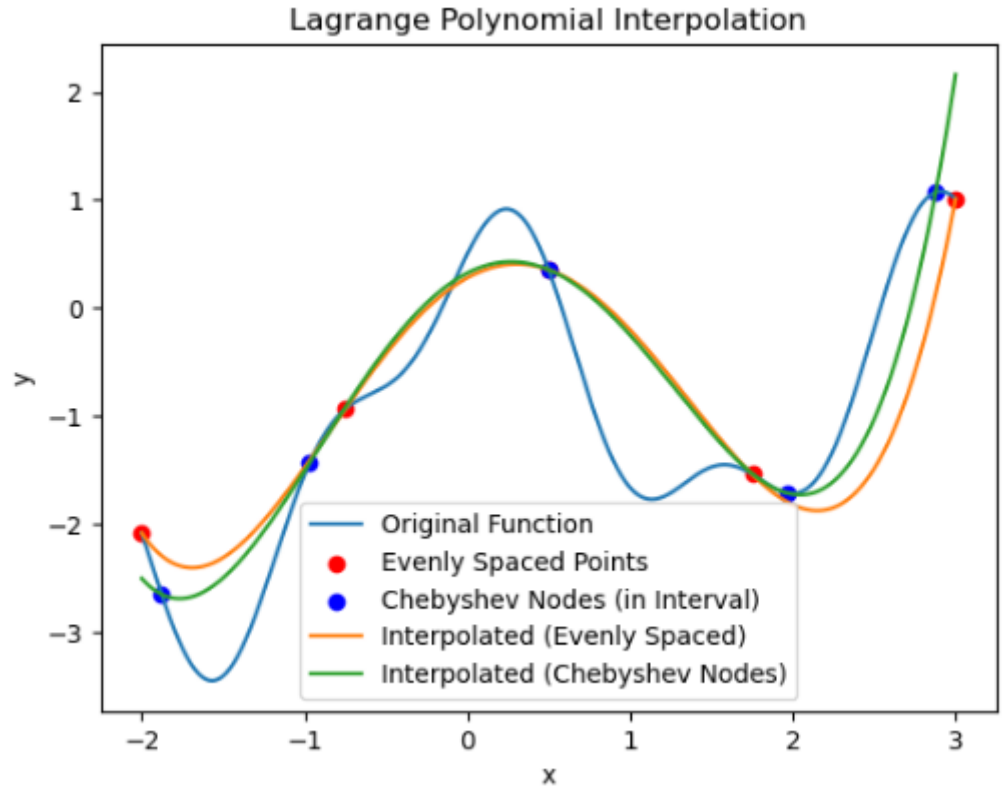
Kodas:

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

# Given function
def f(x):
    return np.cos(2*x) * (np.sin(3*x) + 1.5) - np.cos(x/5)
```

```
# Lagrange polynomial interpolation function
def lagrange_interpolation(x, x_values, y_values):
    result = 0
    n = len(x_values)

    for i in range(n):
        term = y_values[i]
        for j in range(n):
            if i != j:
                term = term * (x - x_values[j]) / (x_values[i] - x_values[j])
        result += term

    return result

# Number of interpolation points
num_points = 5

# Generate evenly spaced points
x_evenly_spaced = np.linspace(-2, 3, num_points)

# Define the interval for Chebyshev nodes
interval_start = -2
interval_end = 3

# Generate Chebyshev nodes within the specified interval
def chebyshev_nodes(num_nodes, start, end):
    k = np.arange(0, num_nodes)
    nodes = 0.5 * (start + end) + 0.5 * (end - start) * np.cos((2*k + 1) * np.pi / (2 * num_nodes))
    return nodes

x_chebyshev = chebyshev_nodes(num_points, interval_start, interval_end)

# Calculate function values at interpolation points
y_evenly_spaced = f(x_evenly_spaced)
y_chebyshev = f(x_chebyshev)

# Perform Lagrange polynomial interpolation
y_plot_evenly_spaced = [lagrange_interpolation(x, x_evenly_spaced, y_evenly_spaced) for x in x_plot]
y_plot_chebyshev = [lagrange_interpolation(x, x_chebyshev, y_chebyshev) for x in x_plot]

# Plot the original function, interpolated functions, and points
plt.plot(x_plot, f(x_plot), label='Original Function')
plt.scatter(x_evenly_spaced, y_evenly_spaced, color='red', label='Evenly Spaced Points')
plt.scatter(x_chebyshev, y_chebyshev, color='blue', label='Chebyshev Nodes (in Interval)')
plt.plot(x_plot, y_plot_evenly_spaced, label='Interpolated (Evenly Spaced)')
plt.plot(x_plot, y_plot_chebyshev, label='Interpolated (Chebyshev Nodes)')
plt.legend()
plt.title('Lagrange Polynomial Interpolation')
plt.xlabel('x')
plt.ylabel('y')
plt.show()

# Spausdinamos interpoliavimo funkcijų išraiškos
print("Interpoliavimo Funkcijos išraiška (Vienodai Išsidėstę Taškai):")
```

```
print(np.poly1d(poly_evenly_spaced))
print("\nInterpoliavimo Funkcijos išraiška (Chebyshevo Taškai):")
print(np.poly1d(poly_chebyshev))
```

Rezultatai:



Interpoliavimo Funkcijos išraiška (Vienodai Išsidėstę Taškai):

```
        19          18          17         16         15
1.563e-05 x  - 0.0001862 x  + 0.0004984 x  + 0.002164 x  - 0.01125 x
         14        13         12        11         10
 - 0.009463 x  + 0.1049 x  + 0.01916 x  - 0.6373 x  - 0.01715 x
       9        8       7       6       5       4
 + 2.73 x + 0.01124 x - 7.778 x - 0.1276 x + 13.03 x + 0.9972 x
       3       2
 - 10.5 x - 2.98 x + 3 x + 0.5
```

Interpoliavimo Funkcijos išraiška (Chebyshevo Taškai):

```
        19          18          17         16         15
8.947e-06 x  - 0.0001049 x  + 0.0002113 x  + 0.001877 x  - 0.007439 x
        14         13         12        11         10
 - 0.014 x  + 0.08869 x  + 0.05644 x  - 0.6117 x  - 0.1334 x
        9       8       7       6       5       4       3
 + 2.735 x + 0.1948 x - 7.84 x - 0.2799 x + 13.1 x + 1.059 x - 10.53 x
        2
 - 2.989 x + 3.003 x + 0.5003
```

II užduotis. Interpoliavimas splainu per duotus taškus: Antrąją dalį sudarė šiltnamio dujų emisijos interpoliavimas splainu, remiantis šalies metiniais duomenimis nuo 1998 iki 2018 metų. Pateiktas rezultatų grafikas su interpoliavimo mazgais ir gautomis kreivėmis.

Variantas:

Var. Nr.	Šalis	Splainas
10	Zambija	Ermito (Akima)

Akima interpoliacijos algoritmas, skirtas interpoliuoti duomenų taškus. Akima interpoliacija yra segmentinė metodika, kuri kiekvienam intervalui tarp duomenų taškų naudoja vietinį polinomą. Kiekviename intervalo intervale interpoliuotos funkcijos formulė yra kubinio polinomo formos.

Splainų pagrindinio koeficiento skaičiavimo formulė:

$$m[i] = \frac{y[i+1] - y[i]}{x[i+1] - x[i]}$$

Kraštinių koeficientų skaičiavimo formulės:

$m[0] = 2m[1] - m[2]$, kairysis koeficientas.
$m[n-1] = 2m[n-2] - m[n-3]$, dešinas koeficientas.

Interpoliacijai naudojama formulė:

$$f(x) = \frac{(x_{i+1} - x)(y_i) + (x - x_i)(y_{i+1})}{x_{i+1} - x_i} + \frac{(x_{i+1} - x)^2(x - x_i)m_i + (x - x_i)^2(x_{i+1} - x)m_{i+1}}{6(x_{i+1} - x_i)}$$

Duomenys:

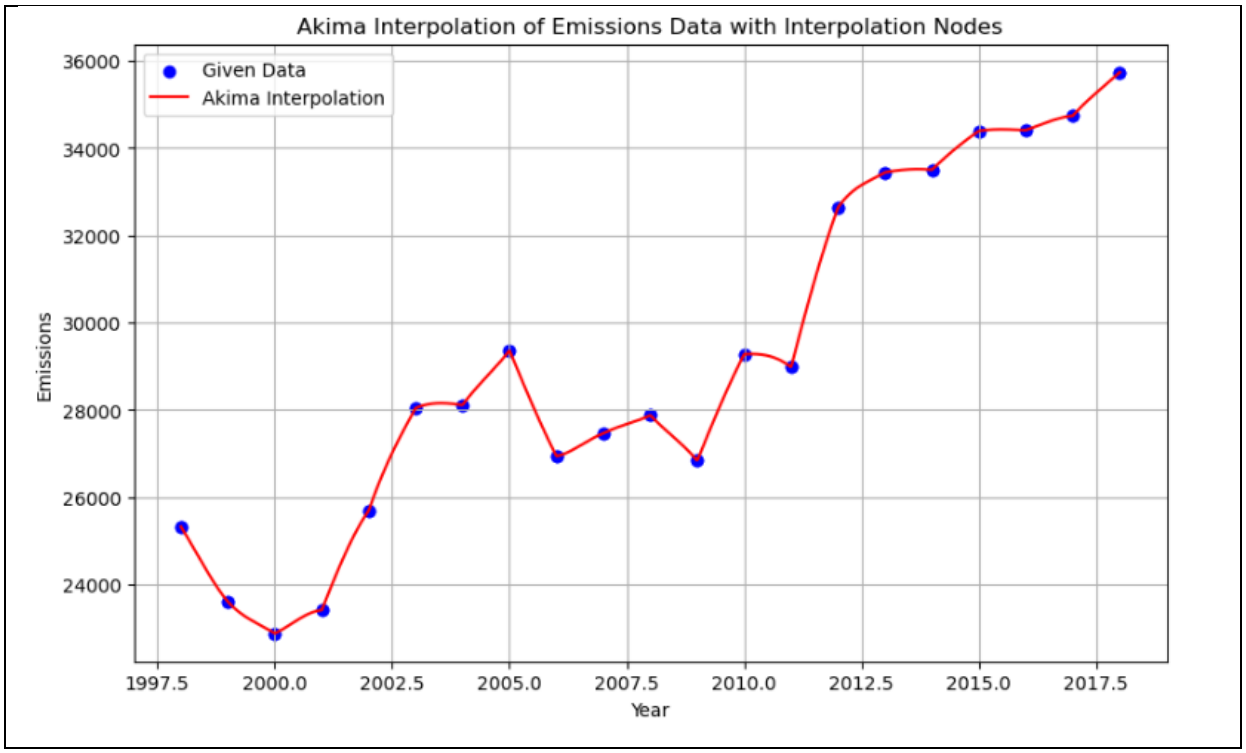
```
years = np.array([1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018])
emissions = np.array([25327.9246, 23608.99686, 22880.14222, 23411.84704, 25691.74375, 28044.39397, 28115.25888, 29370.2779, 26942.28254, 27461.05758, 27880.72203, 26839.27337, 29281.10911, 28979.05696, 32642.49531, 33445.97255, 33504.6955, 34395.35995, 34413.25661, 34743.01806, 35737.70702])
```

Kodas:

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

def akima_interpolation(x, y, x_new):
    n = len(x)
    m = np.zeros(n)
    slope = np.zeros(n)

    for i in range(1, n - 1):
        m[i] = (y[i + 1] - y[i]) / (x[i + 1] - x[i])
```

```
    m[0] = 2 * m[1] - m[2]
    m[n - 1] = 2 * m[n - 2] - m[n - 3]

    for i in range(1, n - 1):
        if np.abs(m[i] - m[i - 1]) < 1e-8:
            slope[i] = slope[i - 1] = m[i]
        else:
            slope[i] = m[i - 1] * (x[i + 1] - x[i]) / (x[i] - x[i - 1]) + m[i] * (x[i] - x[i - 1]) / (x[i + 1] - x[i])

    result = np.zeros_like(x_new)

    for i in range(n - 1):
        mask = np.logical_and(x[i] <= x_new, x_new <= x[i + 1])
        result[mask] = ((x[i + 1] - x_new[mask]) * y[i] + (x_new[mask] - x[i]) * y[i + 1]) / (x[i + 1] - x[i]) \
                       + ((x[i + 1] - x_new[mask]) ** 2 * (x_new[mask] - x[i]) * slope[i] + (x_new[mask] - x[i]) ** 2 * (x[i + 1] - x_new[mask]) * slope[i + 1]) / (6 * (x[i + 1] - x[i]))

    return result

# Given data
years = np.array([1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018])
emissions = np.array([25327.9246, 23608.99686, 22880.14222, 23411.84704, 25691.74375, 28044.39397, 28115.25888, 29370.2779, 26942.28254, 27461.05758, 27880.72203, 26839.27337, 29281.10911, 28979.05696, 32642.49531, 33445.97255, 33504.6955, 34395.35995, 34413.25661, 34743.01806, 35737.70702])

# Akima interpolation
interpolated_years = np.linspace(min(years), max(years), 1000)
interpolated_emissions = akima_interpolation(years, emissions, interpolated_years)

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(years, emissions, label='Given Data', color='blue')
plt.scatter(years, emissions, color='blue')  # Plotting interpolation nodes
plt.plot(interpolated_years, interpolated_emissions, label='Akima Interpolation', color='red')
plt.title('Akima Interpolation of Emissions Data with Interpolation Nodes')
plt.xlabel('Year')
plt.ylabel('Emissions')
plt.legend()
plt.grid(True)
plt.show()
```

Rezultatai:



III užduotis. Aproksimavimas: Trečiojoje dalyje buvo taikomas mažiausių kvadratų metodas šiltnamio dujų emisijos aproksimavimui daugianariais (pirmos, antros, trečios ir penktos eilės). Pateikti gauti daugianarių išraiškos ir jų grafiniai rezultatai.

Pagrindinė formulė, kurią naudoja šis kodas, yra polinomo regresijos formulė:

$$Y = a_0 + a_1X + a_2X^2 + \cdots + a_nX^n; +$$

Čia Y yra priklausoma kintamoji (šiltnamio dujų emisija), X yra nepriklausoma kintamoji (metai), o $a_0, a_1, \ldots, a_n$ yra polinomo koeficientai, kurie yra nustatomi naudojant polinominės regresijos metodą.

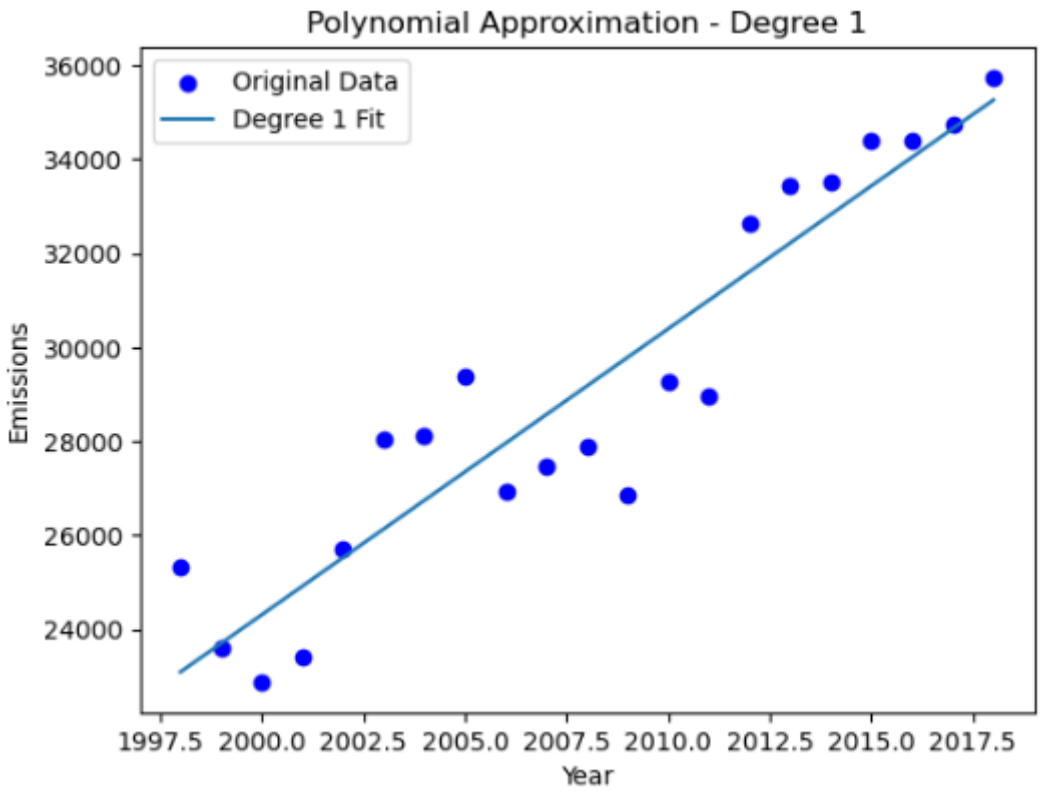
Visas kodas kartu atvaizduoja originalius duomenis (mėlyna spalva) ir skirtingų polinomų prognozes (skirtingomis spalvomis). Tai leidžia įvertinti, kaip polinominė regresija prisitaiko prie duomenų ir kaip skiriasi prognozės su skirtingais polinomo laipsniais.

Kodas**:**

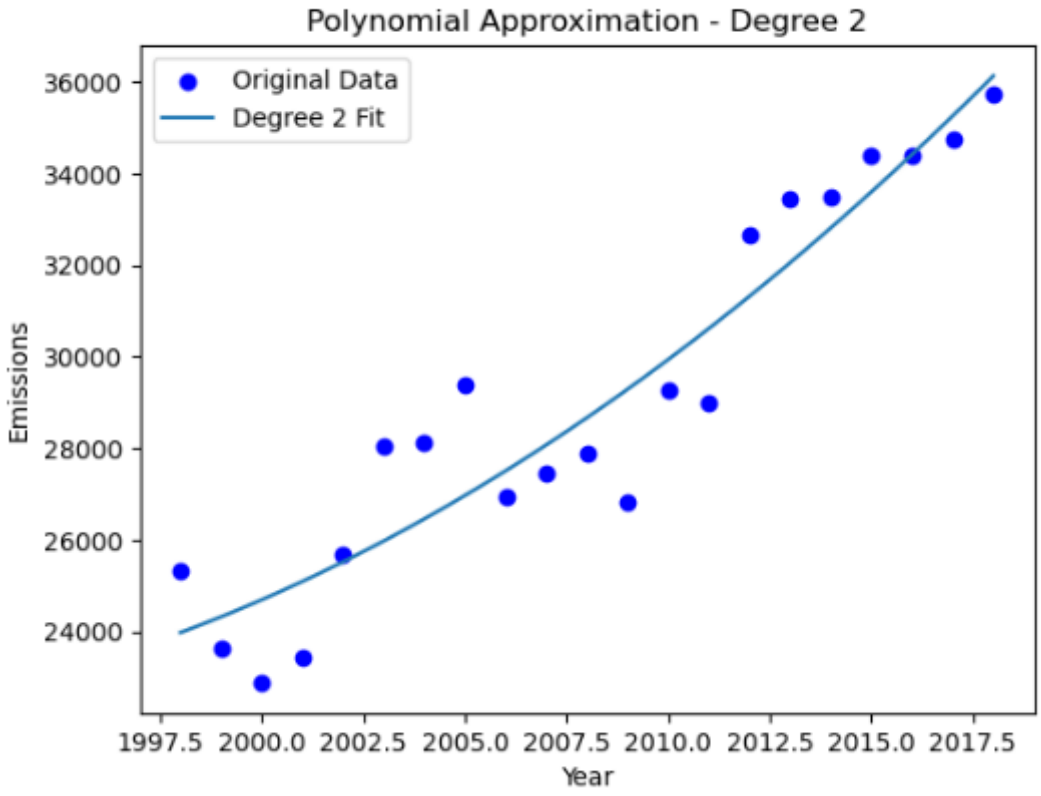
```
import numpy as np
import matplotlib.pyplot as plt
import warnings

# Ignore RankWarning
warnings.filterwarnings("ignore", category=np.RankWarning)
```

```
# Given data
years = np.array([1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018])
emissions = np.array([25327.9246, 23608.99686, 22880.14222, 23411.84704, 25691.74375, 28044.39397, 28115.25888, 29370.2779, 26942.28254, 27461.05758, 27880.72203, 26839.27337, 29281.10911, 28979.05696, 32642.49531, 33445.97255, 33504.6955, 34395.35995, 34413.25661, 34743.01806, 35737.70702])

# Polynomial approximation function
def polyfit_and_plot(x, y, degree):
    # Fit the polynomial
    coefficients = np.polyfit(x, y, degree, full=True)

    # Create the polynomial function
    poly_func = np.poly1d(coefficients[0])

    # Generate x values for the plot
    x_values = np.linspace(min(x), max(x), 100)

    # Plot the original data
    plt.scatter(x, y, label='Original Data', color='blue')

    # Plot the fitted curve
    plt.plot(x_values, poly_func(x_values), label=f'Degree {degree} Fit')

    # Set plot labels and legend
    plt.xlabel('Year')
    plt.ylabel('Emissions')
    plt.legend()
    plt.title(f'Polynomial Approximation - Degree {degree}')

    # Show the plot
    plt.show()

    # Print the polynomial expression
    print(f'Degree {degree} Polynomial: {poly_func}')

# Perform polynomial approximation for different degrees
for degree in [1, 2, 3, 4, 5]:
    polyfit_and_plot(years, emissions, degree)
```

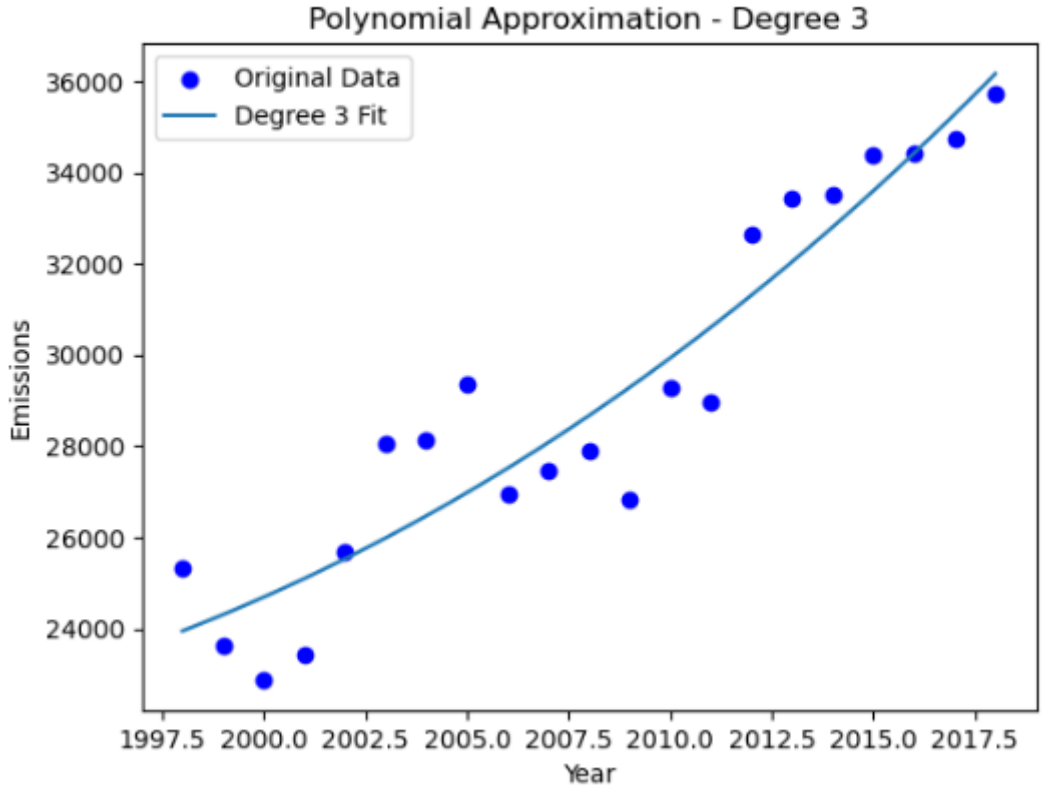
Rezultatai:



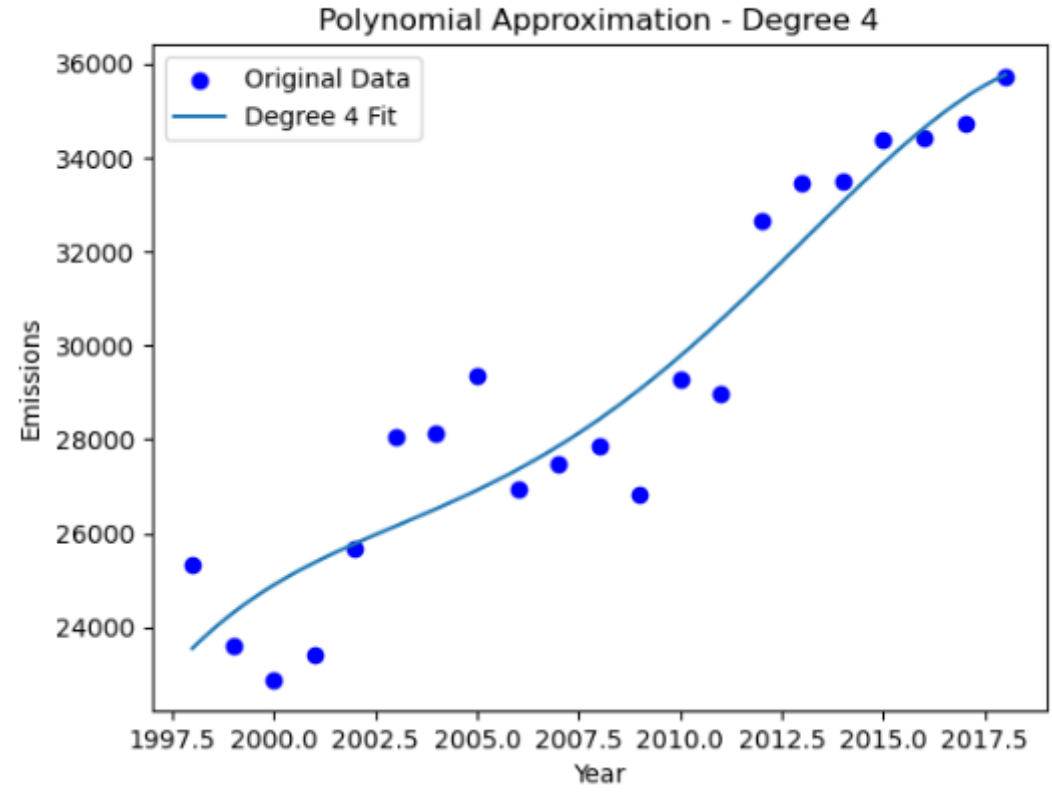
Degree 1 Polynomial:
608.2 x - 1.192e+06



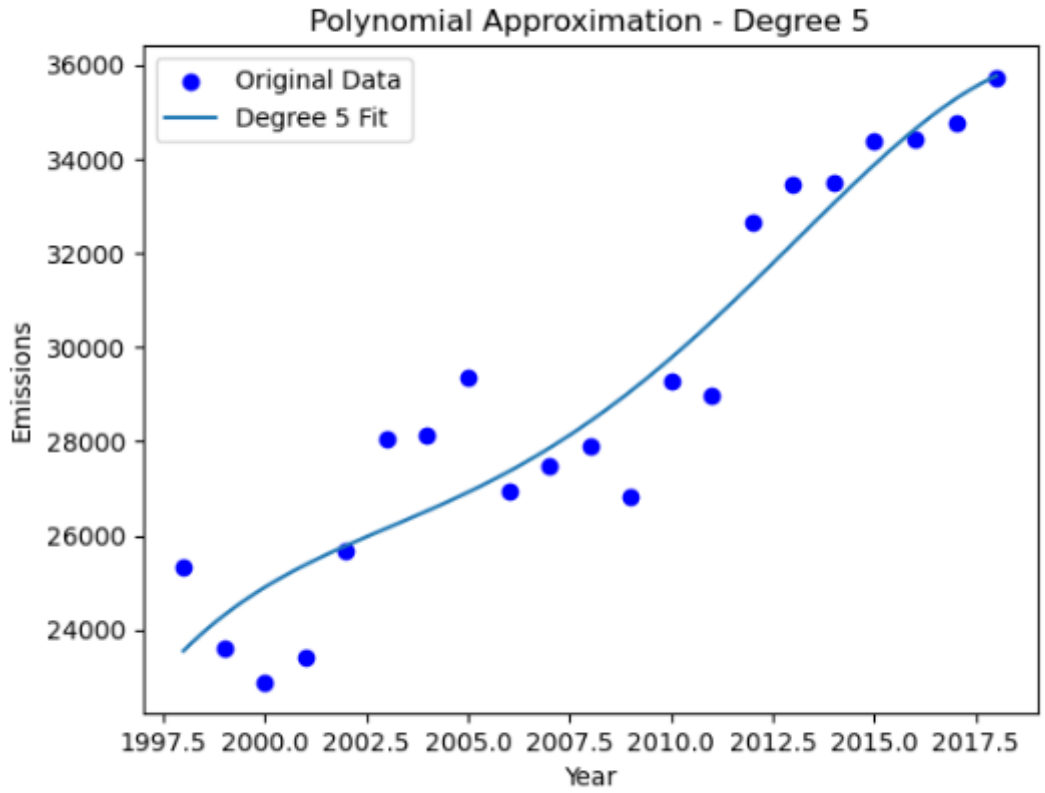
Degree 2 Polynomial:
$$13.86 x^2 - 5.506e+04 x + 5.469e+07$$



Degree 3 Polynomial:
$$0.06857 x^3 - 399.2 x^2 + 7.743e+05 x - 5.004e+08$$



Degree 4 Polynomial:
$$-0.239x^4 + 1920x^3 - 5.782e{+}06x^2 + 7.74e{+}09x - 3.886e{+}12$$



Degree 5 Polynomial:
$$-5.977e{-}05x^5 + 0.36x^4 - 482x^3 - 9.68e{+}05x^2 + 2.915e{+}09x - 1.951e{+}12$$

IV užduotis. Parametrinis aproksimavimas: Paskutinėje dalyje buvo formuojama šalies kontūro aproksimuojanti kreivė, taikant parametrinį aproksimavimą Haro bangelėmis. Analizuotos ir pateiktos bent 10 detalumo lygių aproksimacijos.

Šis kodas atlieka Haaro bangų transformaciją (Haar wavelet transform) su duotu signalu. Haaro bangų transformacija yra pirminė bangų transformacija, kurios metu signalas skaidomas į aproksimuotus signalus ir detalę. Tai leidžia atskleisti signalo savybes skirtinguose lygmenyse.

Ši programa turi kelis etapus:

- Nuskaitomas šalies kontūro failas (shapefile), randamas ID, kuris atitinka šalį "Zambia".
- Nuskaitomas kontūras ir randamas didžiausias plotas (jei tai daugiakampis).
- Aproksimuojama duoto signalo funkcija naudojant Haaro skalės funkcijas ir vaizduojamas rezultatas lygiuose nuo didžiausio iki mažiausio.

Kodas:

```
import matplotlib.pyplot as plt
import shapefile
import numpy as np
```

```
from shapely import geometry

# Hario Bangelių kodas (Pavyzdinis Kodas) ----------------------------------------
def Haar_Wavelet_Reconstruction(x,j,k,a,b): # bangeles funkcija
# vaizdavimo taskai x, lygis j, postumis/koeficientas k, intervalas [a,b]
    eps=1e-9
    xtld=(x-a)/(b-a)
    xx = np.power(2,j)*xtld-k
    h = np.power(2, j/2) * (np.sign(xx+eps) + np.sign(xx - 1 - eps) - 2*np.sign(xx-0.5)) / (2 * (b-
a))
    return h
# Smooth - mastelis
# details - detales
# mastelio reziai: [a, b]
def Haar_Pyramid_Algorithm(ValueArray, NL, a, b):
    details = []
    smooth = (b - a) * ValueArray * 2 **(-NL/2)
    for i in range(NL):
        # smooth[::2] - pasirenkame kas antrą elementą nuo pirmo
        # smooth[1::2] - pasirenkame kas antrą elementą nuo antro
        smooth_temp = (smooth[::2] + smooth[1::2]) / np.sqrt(2) # Skaičiuojame naują mastelį
        details.append((smooth[::2] - smooth[1::2]) / np.sqrt(2)) # Skaičiuojame naujas detales
        smooth = smooth_temp
    return smooth, details
# Nuskaitome pradinius taškus (Pavyzdinis kodas) ----------------------------------
shape = shapefile.Reader("ne_10m_admin_0_countries.shp")
id = -1
for i in range(len(shape)):
    feature = shape.shapeRecords()[i]
    if feature.record.NAME_EN == "Zambia":
        id = i
        break
if id == -1:
    print("Tokios šalies nėra")
else:
    print("id: " + str(id) )

#id = 5
feature = shape.shapeRecords()[id]
print(feature.record.NAME_EN)
largestAreaID = 0
if feature.shape.__geo_interface__['type'] == 'MultiPolygon':
    print(len(feature.shape.__geo_interface__['coordinates']))
    area = 0
    for i in range(len(feature.shape.__geo_interface__['coordinates'])):
```

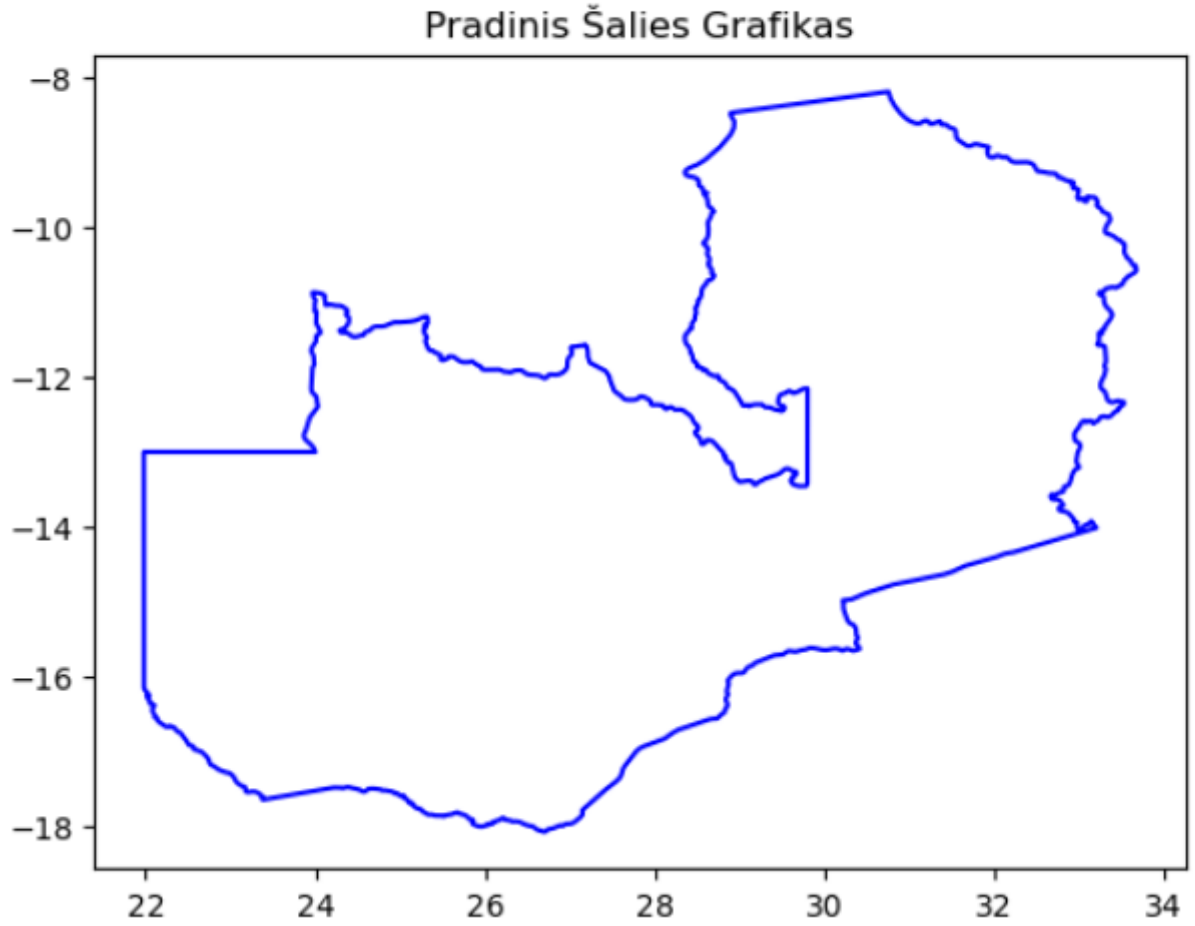
```
        points = feature.shape.__geo_interface__['coordinates'][i][0]
        polygon = geometry.Polygon(points)
        if polygon.area > area:
            area = polygon.area
            largestAreaID = i
    xxyy = feature.shape.__geo_interface__['coordinates'][largestAreaID][0]
else:
    xxyy = feature.shape.__geo_interface__['coordinates'][0]
# Nubražome pradinį grafiką ----------------------------------------------------------

xy = list(zip(*xxyy))
X = xy[0]
Y = xy[1]
print("Pradinis duomenų Grafikas")
print(f"Nuskaitytas narių kiekis: {len(X)}")

plt.title("Pradinis Šalies Grafikas")
plt.plot(X,Y, 'b')
plt.show()
# Haro Bangelių Aproksimacija ---------------------------------------------------------
# Kelintame lygyje vyks hario bangelių aproksimacija
NL = 10

starter_point_count = len(X)
interpolated_points_count = 2 ** NL
# Parametrizuojame taškus. Hario bangelės yra ištiesinė funkcija, tačiau mes turime konturą.
t = np.zeros(starter_point_count)
for i in range(1, starter_point_count):
    diff = np.array([X[i] - X[i-1], Y[i] - Y[i-1]]) # Vektoriaus x, y ašių reikšmės
    t[i] = t[i-1] + np.linalg.norm(diff) # Prie buvusio vektoriaus dalies pridedame naują vektoriaus ilgį
print("parametrizuoti taškai:")
print(t)

# Suskaičiuoajme Hario bangelių rėžius
# min_t formulėse - a; max_t formulėse - b
min_t = min(t)
max_t = max(t)
print(f"Hario Bangelių rėžiai: [{min_t} - {max_t}]")

# Interpoliuojame naujas X, Y reikšmes, nes taškų kiekis turi būti 2 ^ NL
t1 = np.linspace(min_t, max_t, interpolated_points_count)
interpolated_X = np.interp(t1, t, X) # Interpoliuojami taškai kaip x = t1; naudojant t (kryptis) ir X (Pradines reikšmes)
interpolated_Y = np.interp(t1, t, Y)
```

```
# Persiršome parametrizavimą į interpoliuotas reikšmes, nes dirbame su 2**NL taškų
t = t1

plt.plot(interpolated_X, interpolated_Y)
plt.title("Interpoliuoti nauji taškai")
plt.show()

# Išskaidome Grubiausiu masteliu (j = 0) ir surenkame visas detalęs
smooth_X, details_X = Haar_Pyramid_Algorithm(interpolated_X, NL, min_t, max_t)
smooth_Y, details_Y = Haar_Pyramid_Algorithm(interpolated_Y, NL, min_t, max_t)

# Apsirašome vaizdavimo taškus
hx = np.zeros(interpolated_points_count)
hy = np.zeros(interpolated_points_count)

# Jeigu Norime atkonstruoti pradines koordinates, bet neskaičiuoti nuo 0, 0
# for k in range(2 ** (NL)):
# hx += smooth_X[k] * haar_scaling(t, NL, k, min_t, max_t) # Atstatome mastelį pagal parametrizavimą.
# hy += smooth_Y[k] * haar_scaling(t, NL, k, min_t, max_t)

# Iš detalių ir grubiausio grafiko atstatome grafiką:
for i in range(NL):
    h1x, h1y = np.zeros(interpolated_points_count), np.zeros(interpolated_points_count)
    for k in range(2 ** i):
        # Apskaičiuojame kievieno kiekvieno rėžio pokyčius pagal haro bangelių formule.
        # Praeiname pro kiekviena detalės dalį
        # t - parametrizuoti kintamieji
        # i - kuriame lygyje
        # l - koeficientai
        # min_t - a
        # max_t - b
        h1x += details_X[NL - i - 1][k] * Haar_Wavelet_Reconstruction(t, i, k, min_t, max_t)
        h1y += details_Y[NL - i - 1][k] * Haar_Wavelet_Reconstruction(t, i, k, min_t, max_t)

    # Sudedame Detalių dalis prie dabartinio mastelio
    hx += h1x
    hy += h1y
    plt.title(f'Hario Bangelių funkcija Detalumo Lygyje: {i + 1}')
    plt.plot(hx, hy, color="blue")
    plt.plot([hx[0], hx[-1]], [hy[0], hy[-1]], color="blue") # sujungiame paskutinį su pirmu tašku
    #print(f"hx length:\n{len(hx)}")
    #print(f"hx:\n{hx}")
    #print(f"hy length:\n{len(hy)}")
```

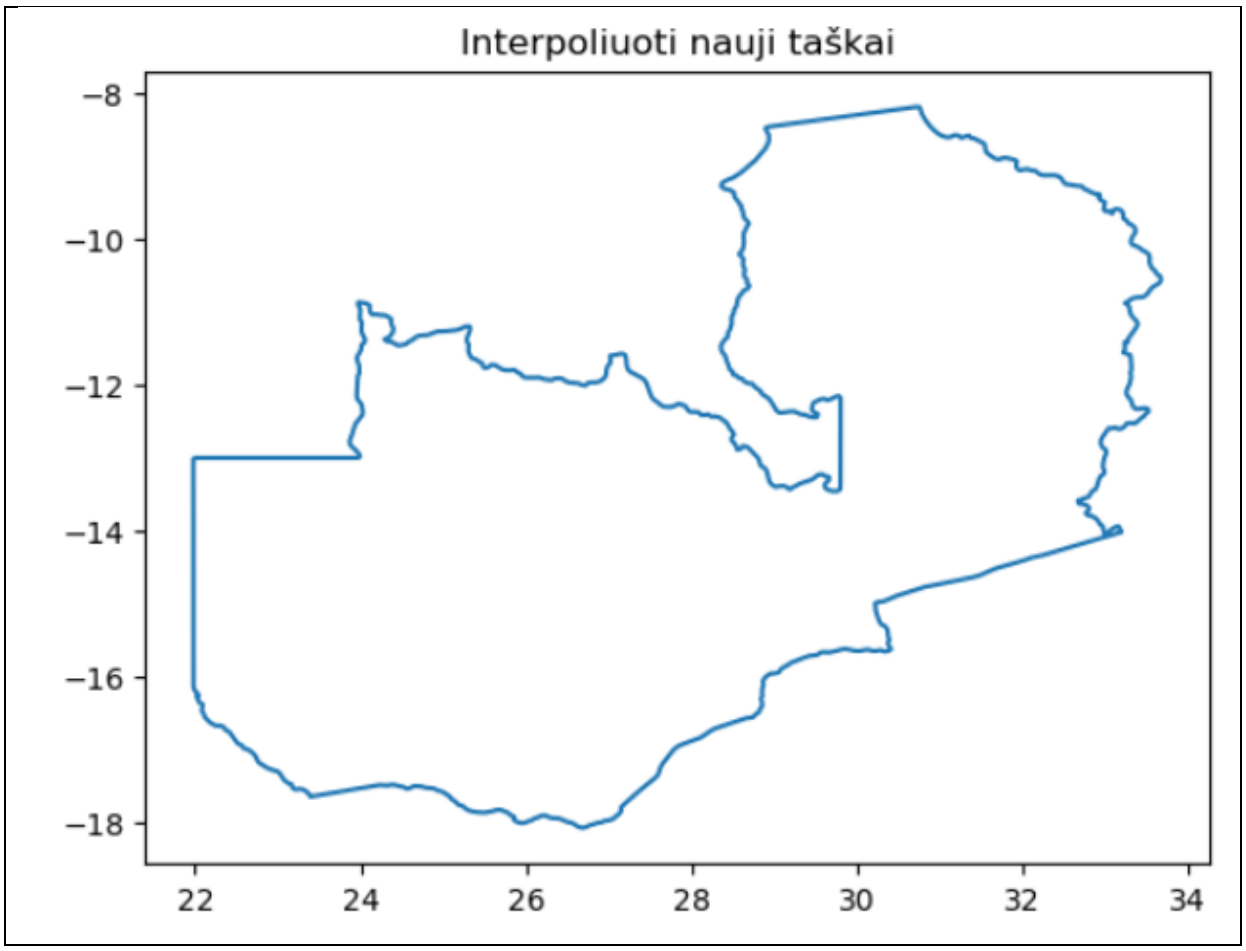
```
    #print(f"hy:\n{hy}")
    plt.ylim(min(hy) - 0.5, max(hy) + 0.5)
    plt.xlim(min(hx) - 0.5, max(hx) + 0.5)
    plt.show()
```

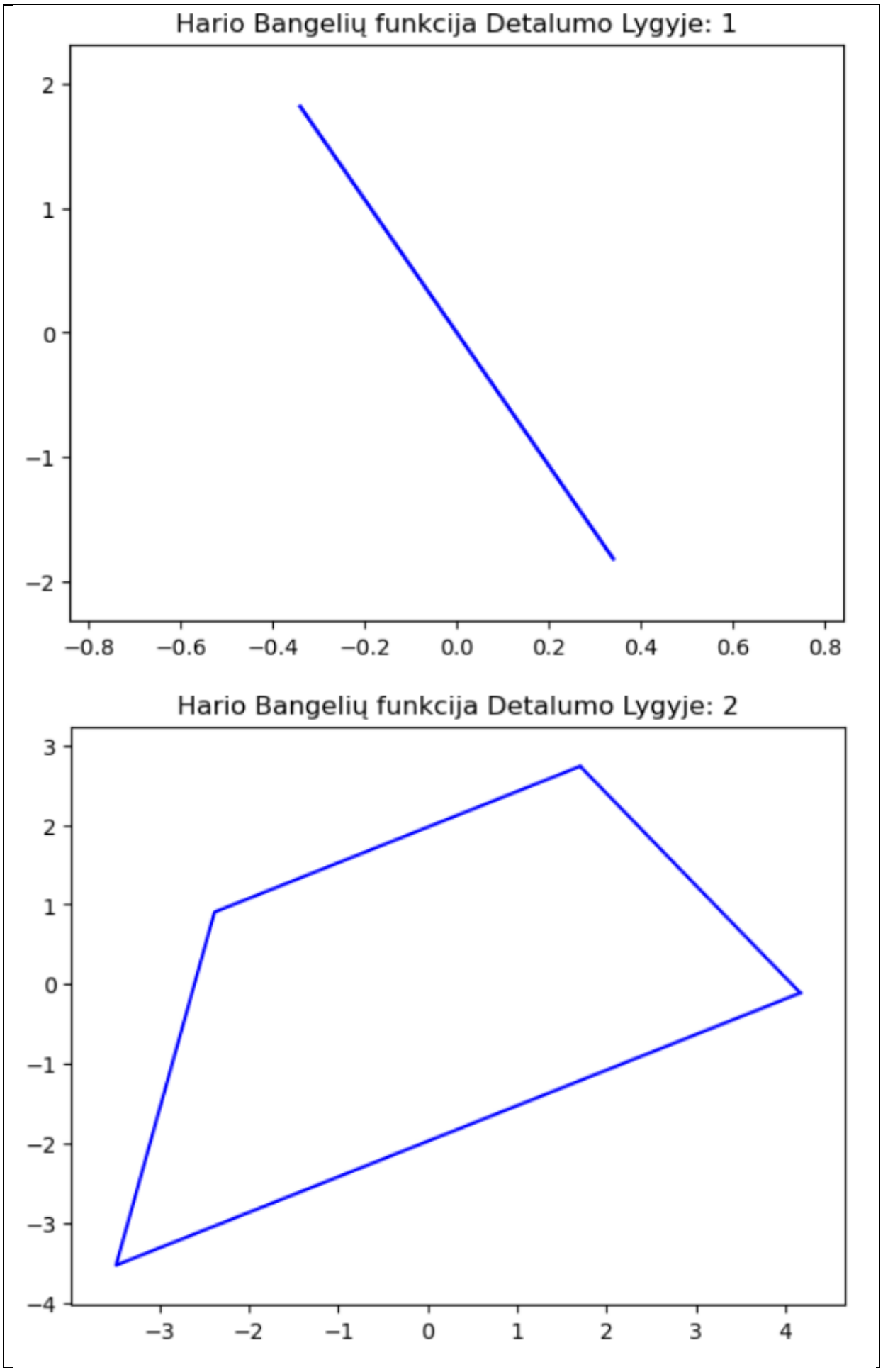
Rezultatai:

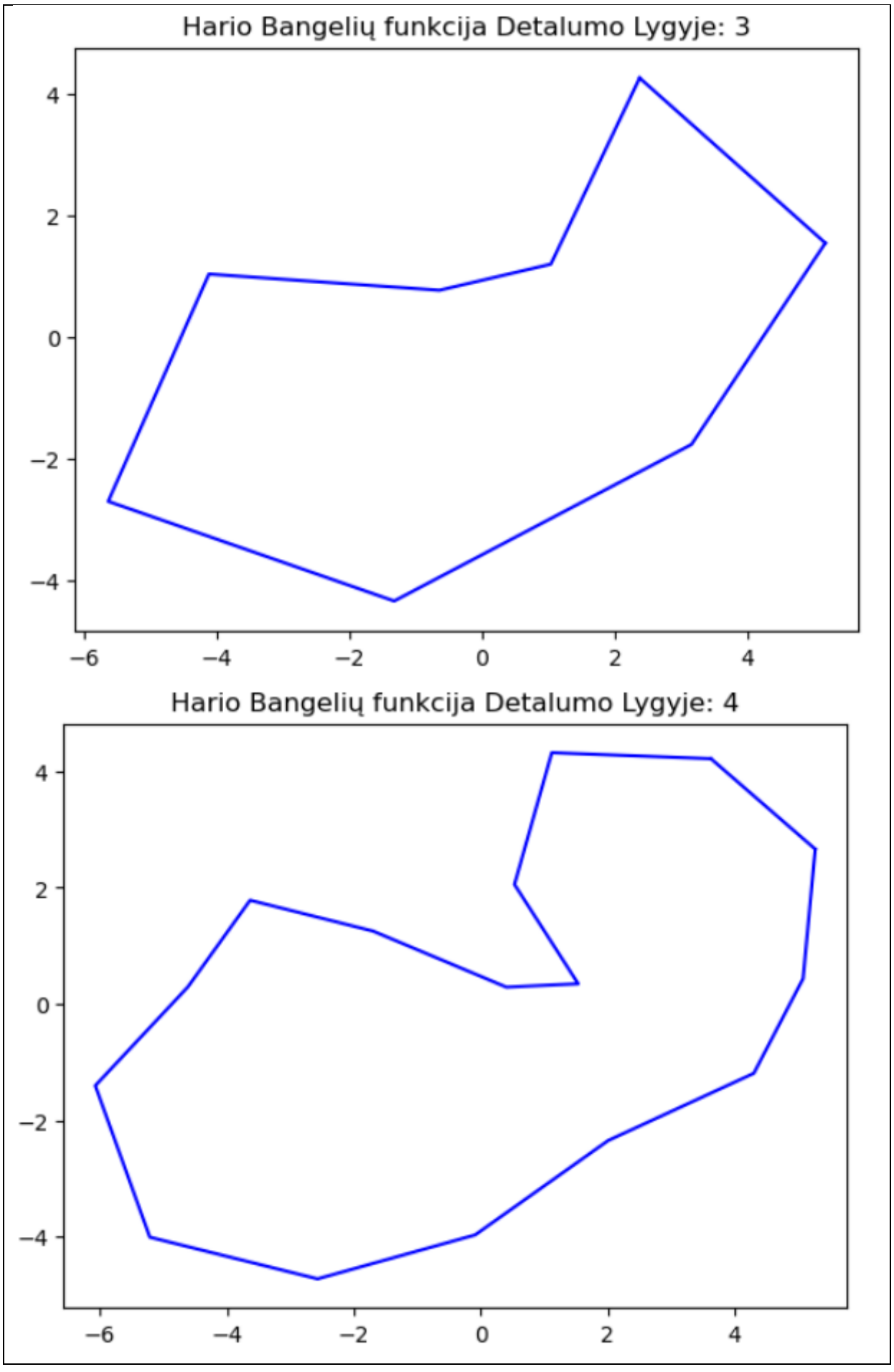
```
id: 81
Zambia
Pradinis duomenų Grafikas
Nuskaitytas narių kiekis: 1508
```

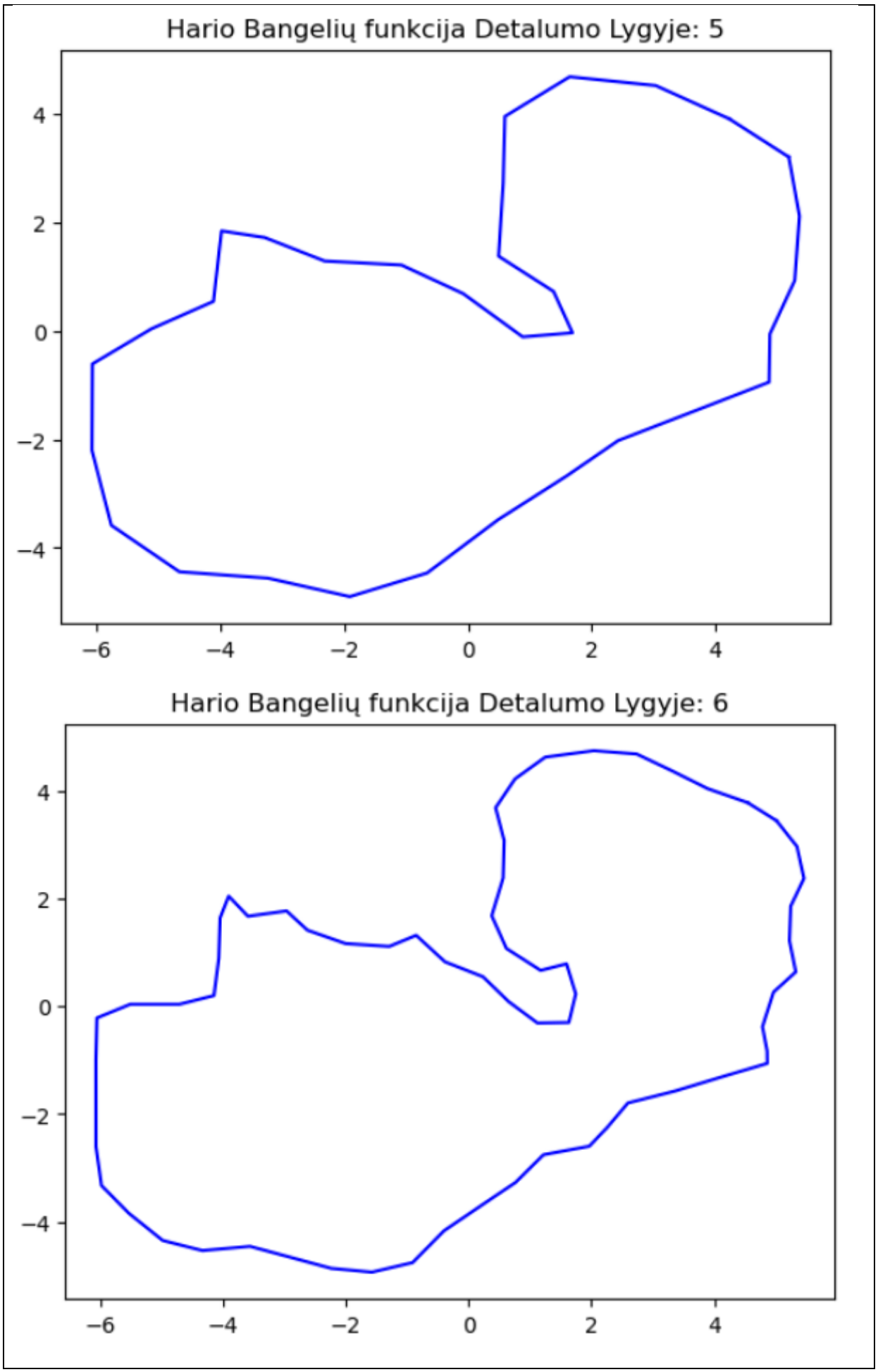


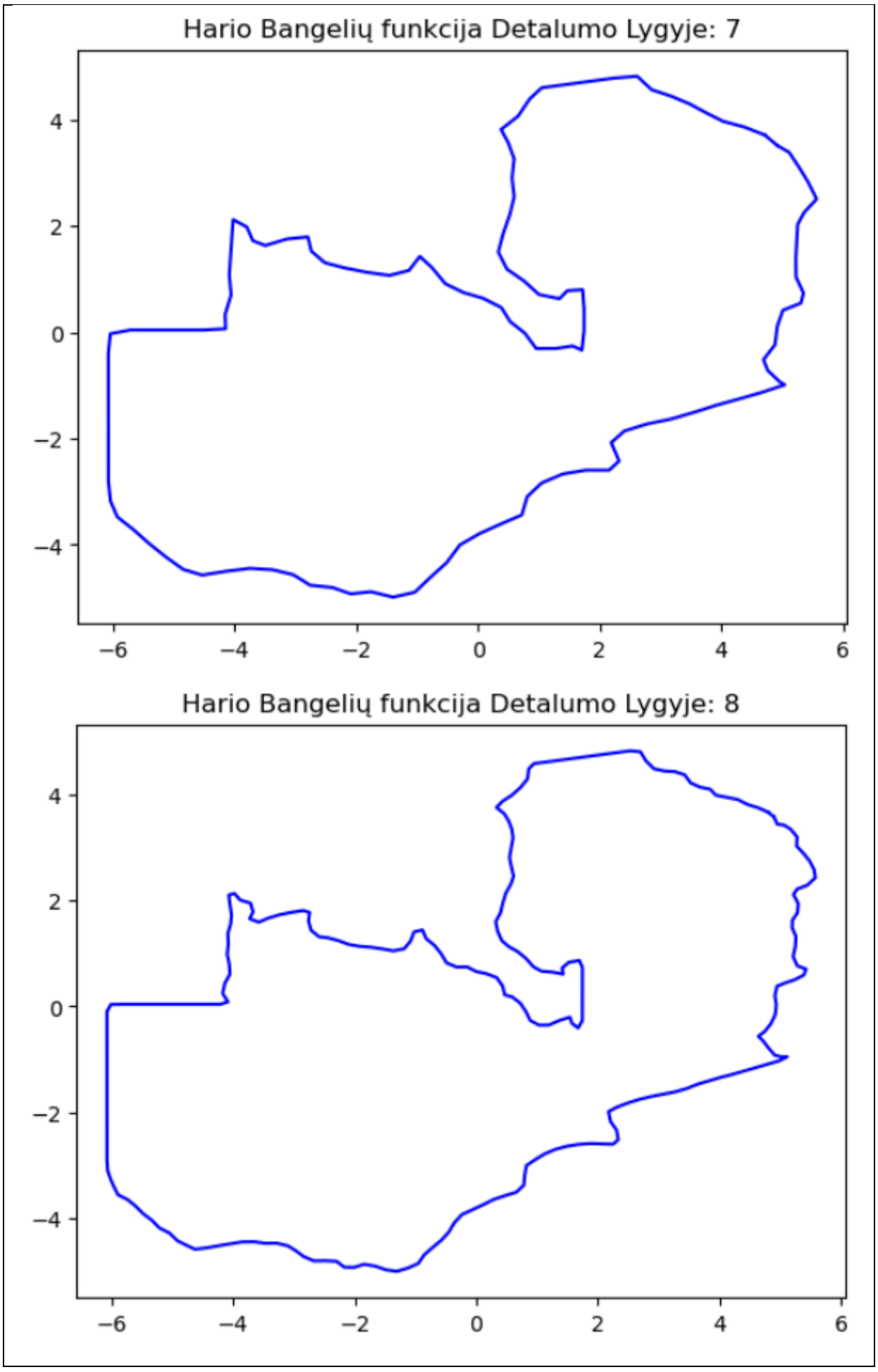
parametrizuoti taškai:
[0. 0.05843469 0.07278437 ... 51.12260381 51.20240467
 51.22061268]
Hario Bangelių rėžiai: [0.0 - 51.22061267950703]

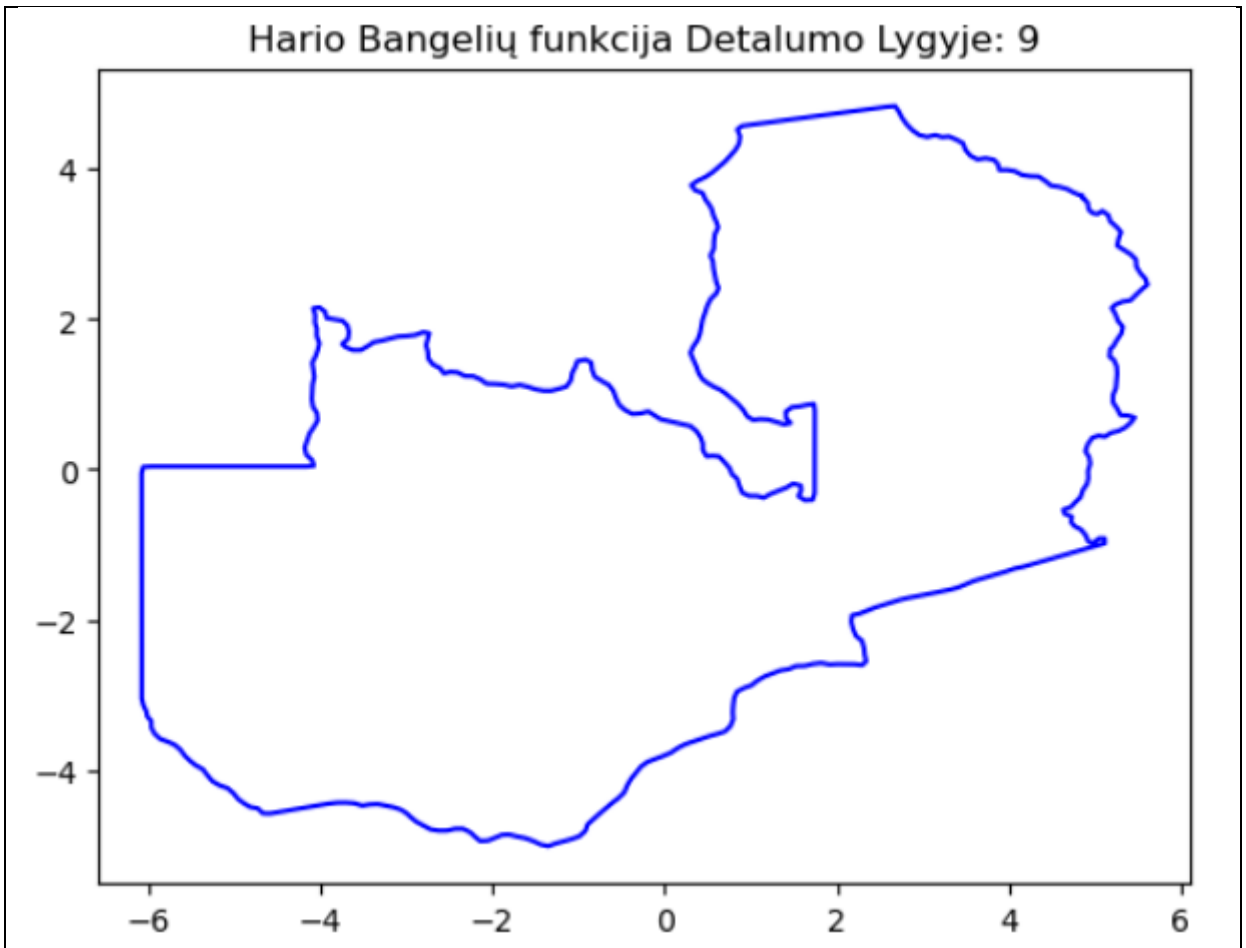

Interpoliuoti nauji taškai
−8
−10
−12
−14
−16
−18
22
24
26
28
30
32
34

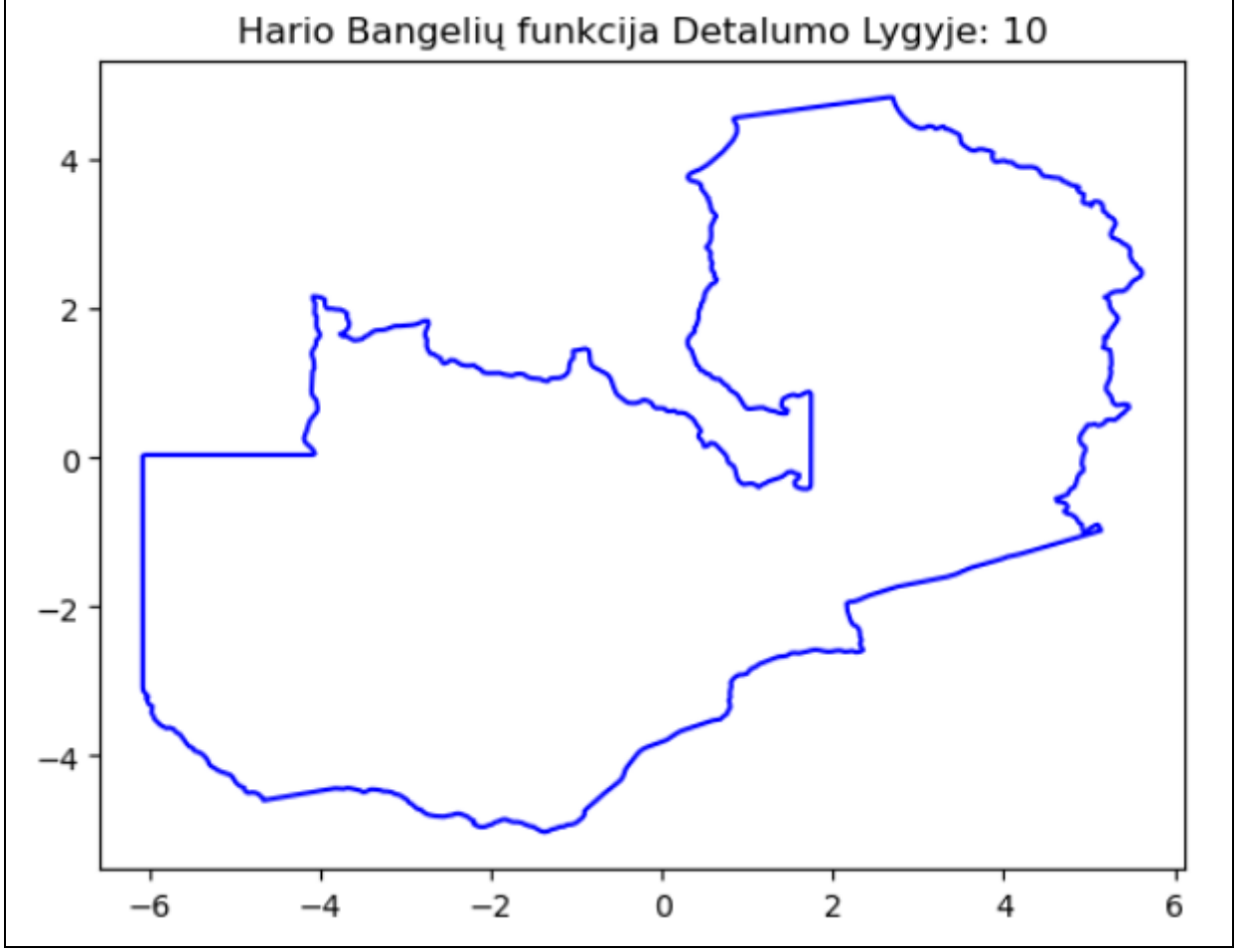

Hario Bangelių funkcija Detalumo Lygyje: 1
2
1
0
−1
−2
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
Hario Bangelių funkcija Detalumo Lygyje: 2
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4


Hario Bangelių funkcija Detalumo Lygyje: 3
4
2
0
−2
−4
−6
−4
−2
0
2
4
Hario Bangelių funkcija Detalumo Lygyje: 4
4
2
0
−2
−4
−6
−4
−2
0
2
4


Hario Bangelių funkcija Detalumo Lygyje: 5
4
2
0
−2
−4
−6
−4
−2
0
2
4
Hario Bangelių funkcija Detalumo Lygyje: 6
4
2
0
−2
−4
−6
−4
−2
0
2
4


Hario Bangelių funkcija Detalumo Lygyje: 7
4
2
0
−2
−4
−6
−4
−2
0
2
4
6
Hario Bangelių funkcija Detalumo Lygyje: 8
4
2
0
−2
−4
−6
−4
−2
0
2
4
6


Hario Bangelių funkcija Detalumo Lygyje: 9
4
2
0
−2
−4
−6
−4
−2
0
2
4
6


Hario Bangelių funkcija Detalumo Lygyje: 10
4
2
0
−2
−4
−6
−4
−2
0
2
4
6