KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Skaitiniai metodai ir algoritmai (P170B115)

Laboratorinių darbų ataskaita

Atliko:
IFF-1/4 gr. studentas
Mildaras Karvelis
2023 m. gruodžio 29 d.
Priėmė:
Prof. Barauskas Rimantas

KAUNAS 2023

TURINYS

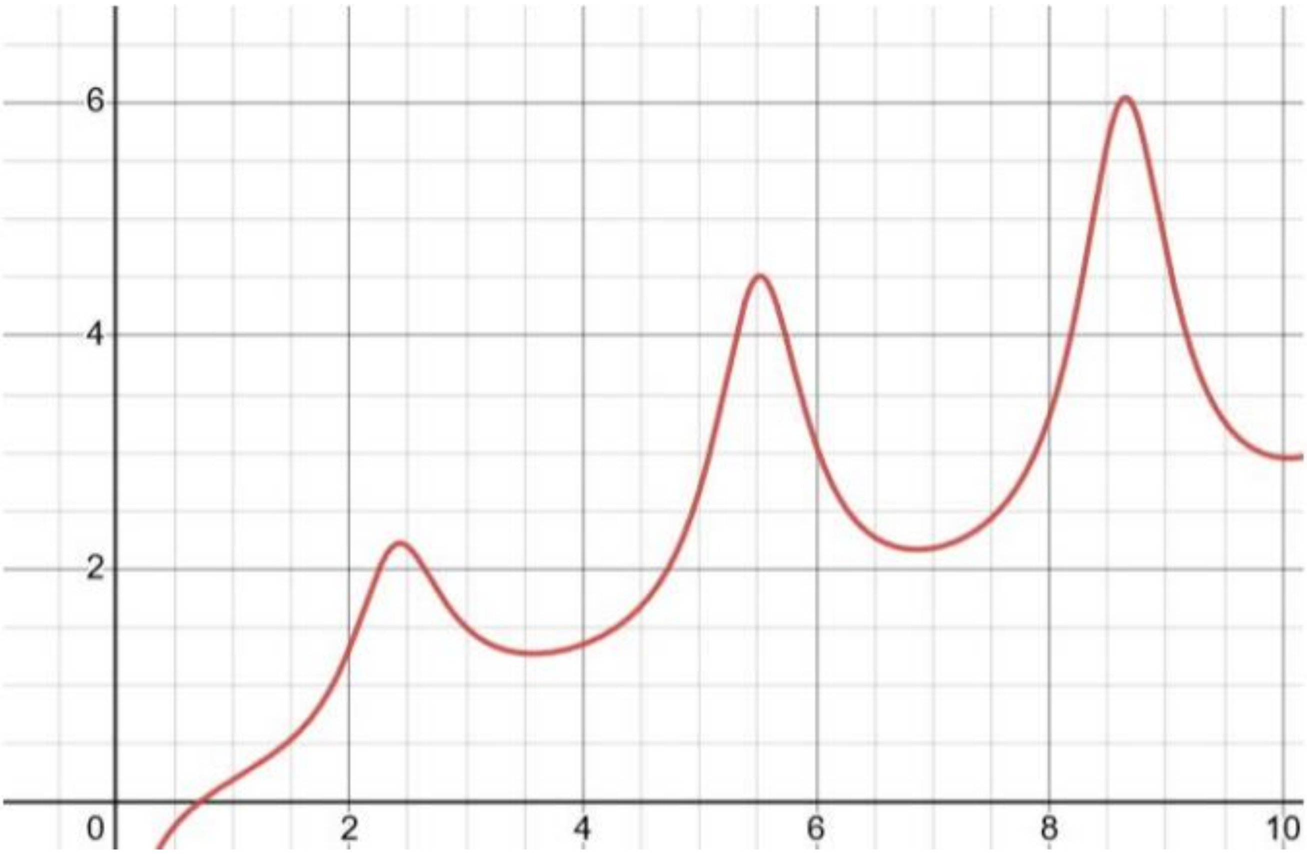
1. Pirma užduotis. Interpoliavimas daugianariu

1 lentelėje duota interpoliuojamos funkcijos analitinė išraiška. Pateikite interpoliacinės funkcijos išraišką naudodami ***1 lentelėje*** nurodytas bazines funkcijas, kai:

a. Taškai pasiskirstę tolygiai.
b. Taškai apskaičiuojami naudojant Čiobyševo abscises.

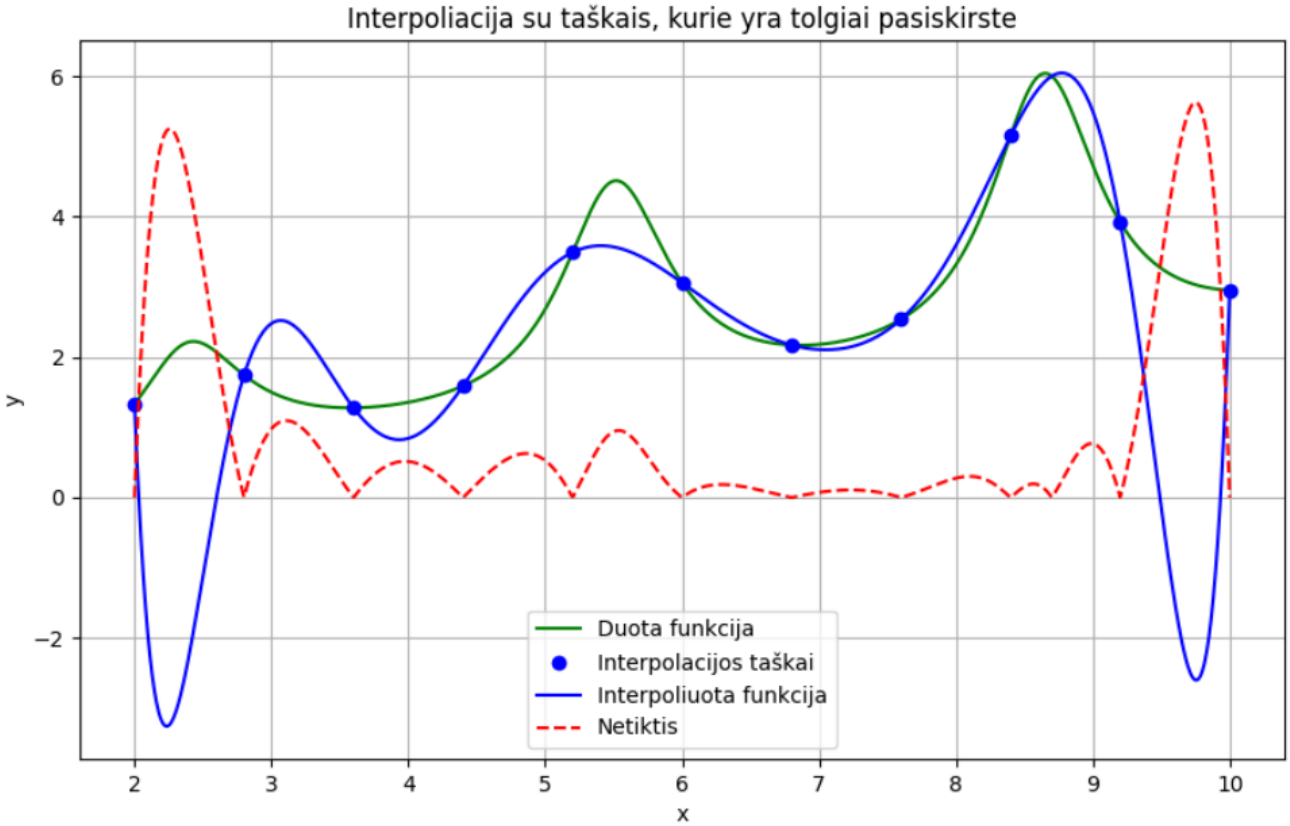
Interpoliavimo taškų skaičių parinkite laisvai, bet jis turėtų neviršyti 30. Pateikite du grafikus, kai interpoliuojančios funkcijos apskaičiuojamos naudojant skirtingas abscises ir gautas interpoliuojančių funkcijų išraiškas. Tame pačiame grafike vaizduokite duotąją funkciją, interpoliuojančią funkciją ir netiktį.

8	$\dfrac{\ln(x)}{(\sin(2 \cdot x) + 1{,}5)} + x/5; 2 \leq x \leq 10$	Vienanarių



1 pav. daugianario grafikas

1.1. a dalis. Taškai pasiskirstę tolygiai

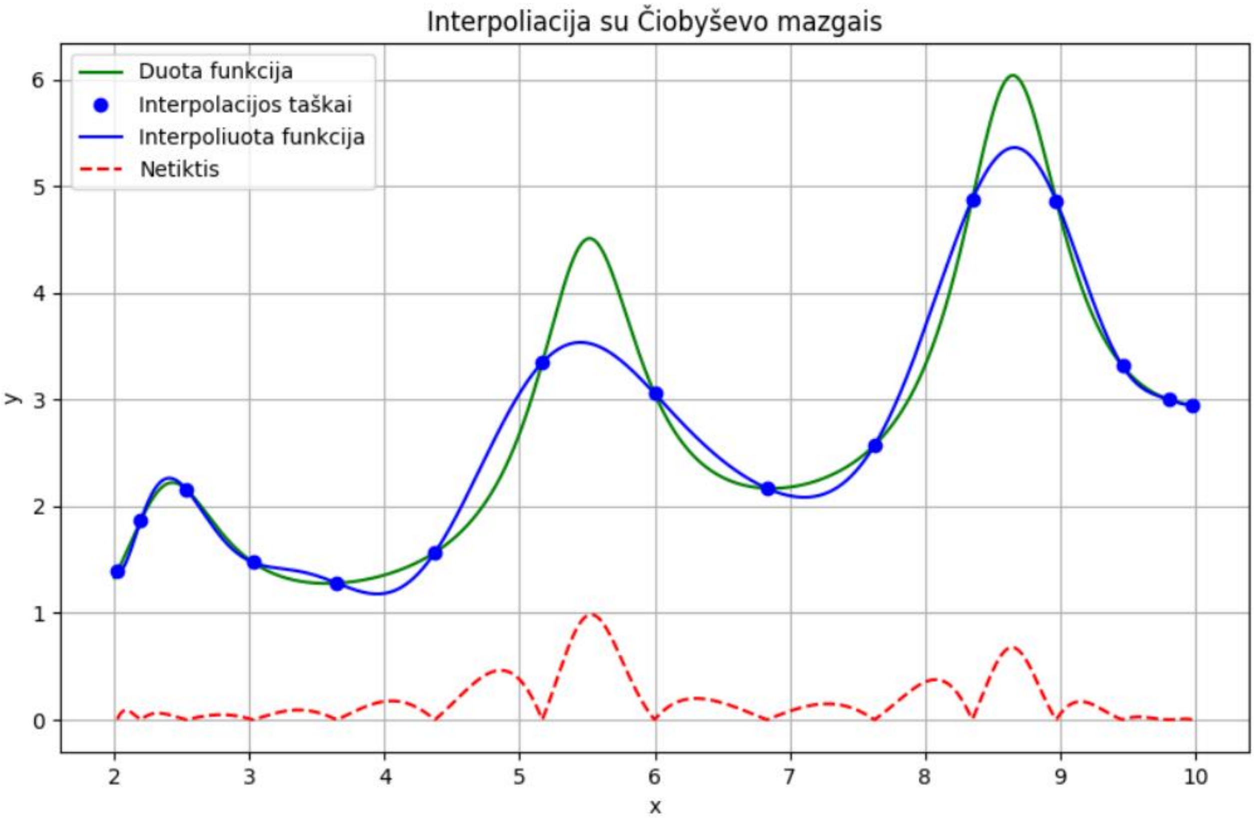


Gautos konstantos:
a0 =8.5656; a1 = -1.8281; a2 = 1.6897; a3 =-8.9297; a4 =2.9964; a5=-6.6899;
a6 = 1.0091; a7 = -1.0182; a8 = 6.5903; a9 = -2.4757; a10 = 4.1053

Interpoliuojančios funkcijos išraiška:

$$f(x) = a(0) + a(1) * x + a(2) * x^2 + a(3) * x^3 + a(4) * x^4 + a(5) * x^5 + a(6) * x^6 + a(7) * x^7 + a(8) * x^8 + a(9) * x^9 + a(10) * x^{10}$$

1.2. b dalis. Taškai apskaičiuojami naudojant Čiobyševo abscises



Gautos konstantos:
a0 = 5.26709; a1 = -1.64347; a2 = 2.31521; a3 = -1.95243; a4 = 1.10175; a5 = -4.40390;
a6 = 1.28691; a7 = -2.79547; a8 = 4.54047; a9 = -5.49316; a10 = 4.87710; a11 = -3.08639
a12 = 1.31665, a13 = -3.39313, a14 = 3.98924

Interpoliuojančios funkcijos išraiška:

$$f(x) = a(0) + a(1) * x + a(2) * x^2 + a(3) * x^3 + a(4) * x^4 + a(5) * x^5 + a(6) * x^6 + a(7) * x^7 + a(8) * x^8 + a(9) * x^9 + a(10) * x^{10} + a(11) * x^{11} + a(12) * x^{12} + a(13) * x^{13} + a(14) * x^{14}$$

1.3. Programos kodas

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

def LF(x):
    return np.log(x) / (np.sin(2 * x) + 1.5) + x / 5

def base(x, n):
    return np.power(x, n)


def chebyshevRange(count, start, end):
    xRange = []
    for i in range(1, count + 1):
        temp = (start + end) / 2 + (end - start) / 2 * np.cos((2 * i - 1) * np.pi / (2 * count))
        xRange.append(temp)

    return xRange

def InterpolationEvenlyXY():
    #n = 12
```

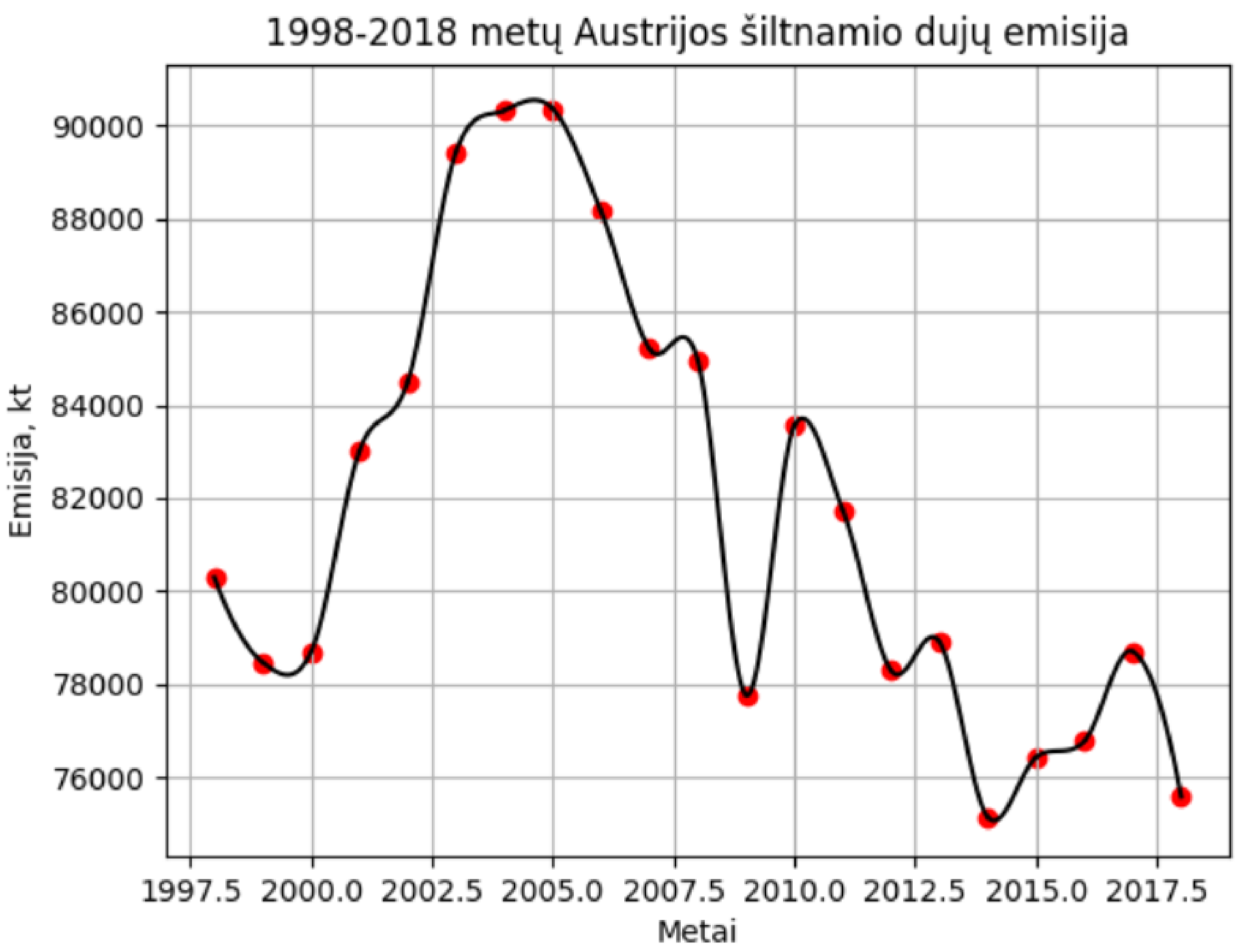
```
    n = 11
    rangeStart = 2
    rangeEnd = 10

    xRange = np.linspace(rangeStart, rangeEnd, n)
    yRange = [LF(x) for x in xRange]

    xP = np.linspace(rangeStart, rangeEnd, 1000)
    yP = LF(xP)

    # Sukuriama Vandermonde matricą
    N = len(xRange)
    A = np.zeros((N, N), dtype=float)
    for i in range(N):
        A[:, i] = base(xRange, i)

    # Gaunami koeficientai
    coeff = np.linalg.solve(A, yRange)
    print("Koeficientai:", coeff)  # Spausdinami koeficientus konsoleje

    # Įvertiname     interpoliuojantį daugianarį
    xxx = np.linspace(xRange[0], xRange[-1], 1000)
    yyy = np.zeros(xxx.size, dtype=float)
    for i in range(N):
        yyy += base(xxx, i) * coeff[i]

    # Skaičiuoja netiktis
    loss = np.abs(yyy - LF(xxx))

    plt.figure(figsize=(10, 6))

    plt.plot(xP, yP, 'g', label='Duota funkcija')
    plt.plot(xRange, yRange, 'bo', label='Interpolacijos taškai')
    plt.plot(xxx, yyy, 'b-', label='Interpoliuota funkcija')
    plt.plot(xxx, loss, 'r--', label='Netiktis')
    plt.legend()
    plt.title('Interpoliacija su taškais, kurie yra tolgiai pasiskirste')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid(True)

    plt.show()

def ChebushevRangeInterpolation():
    #pradiniai duomenys
    #15
    n = 11
    rangeStart = 2
    rangeEnd = 10



    xRange = chebyshevRange(n, rangeStart, rangeEnd)
    yRange = [LF(x) for x in xRange]



    xP = np.linspace(rangeStart, rangeEnd, 1000)
    yP = LF(xP)
```

```
    # Vandermonde matrix
    N = len(xRange)
    A = np.zeros((N, N), dtype=float)
    for i in range(N):
        A[:, i] = base(xRange, i)

    # koeficientu sprendimas
    coeff = np.linalg.solve(A, yRange)
    print("Koeficientai:", coeff)


    xxx = np.linspace(xRange[0], xRange[-1], 1000)
    yyy = np.zeros(xxx.size, dtype=float)
    for i in range(N):
        yyy += base(xxx, i) * coeff[i]

    # netiktis
    loss = np.abs(yyy - LF(xxx))

    plt.figure(figsize=(10, 6))

    plt.plot(xP, yP, 'g', label='Duota funkcija')
    plt.plot(xRange, yRange, 'bo', label='Interpolacijos taškai')
    plt.plot(xxx, yyy, 'b-', label='Interpoliuota funkcija')
    plt.plot(xxx, loss, 'r--', label='Netiktis')
    plt.legend()
    plt.title('Interpoliacija su Čiobyševo mazgais')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid(True)

    plt.show()

# Funckijų vykdymas

InterpolationEvenlyXY()
ChebushevRangeInterpolation()
```

2. Antra užduotis. Interpoliavimas splainu per duotus taškus

Sudarykite ***2 lentelėje*** nurodytos šalies 1998-2018 metų šiltnamio dujų emisiją (galimo duomenų šaltinio nuoroda apačioje) interpoliuojančias kreives, kai interpoliuojama ***2 lentelėje*** nurodyto tipo splainu. Pateikite rezultatų grafiką (interpoliavimo mazgus ir gautą kreivę (vaizdavimo taškų privalo būti daugiau nei interpoliavimo mazgų)).

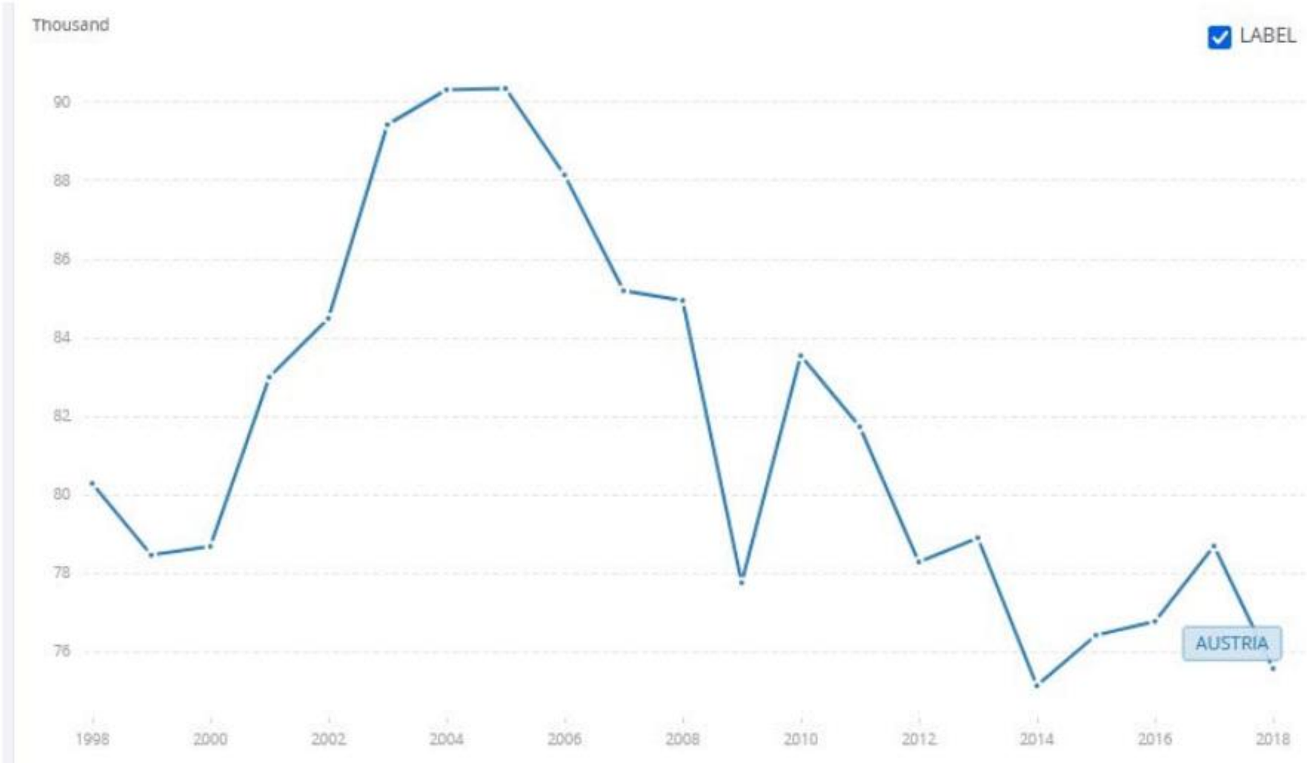
8	Austrija	Ermito (Akima)



Gautas grafikas su panaudotais duomenimis:

- Metai: [1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018]
- Šiltnamio dujų emisijos duomenys (kiekvienų metų): [80295, 78470, 78694, 83002, 84500, 89432, 90323, 90357, 88153, 85204, 84954, 77767, 83551, 81739, 78293, 78909, 75143, 76430, 76781, 78699, 75582]

Grafiko patikrinimas su interneto šaltiniu:



2 pav. https://data.worldbank.org/indicator/EN.ATM.GHGT.KT.CE?end=2018&start=1998

2.1. Programos kodas

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


def spline(x_array, y_array):
    print("Metai")
    print(x_array)
    print("Ismestos dujos, kt")
    print(y_array)
    dy_array = akima(x_array, y_array)

    # Patikrinimas ar isvestines yra tikslios
    dy = np.gradient(y_array, x_array)
    for i in range(len(x_array)):
        print(f"x = {x_array[i]}: derivative = {dy[i]}, Akima derivative = {dy_array[i]}")

    plt.scatter(x_array, y_array, color="red")
    count = len(x_array)
    print("Mazgu skaicius N")
    print(count)
    for i in range(count - 1):
        nnn = 100
        xxx = np.linspace(x_array[i], x_array[i + 1], nnn)
        fff = 0
        for j in range(2):
            U, V = hermite(np.array([x_array[i], x_array[i + 1]]), j, xxx)
            fff = fff + U * y_array[i + j] + V * dy_array[i + j]
        plt.plot(xxx, fff, color="black")
    plt.title("1998-2018 metu Austrijos siltnamio duju emisija")
    plt.xlabel("Metai")
    plt.ylabel("Emisija, kt")
    plt.grid()
```

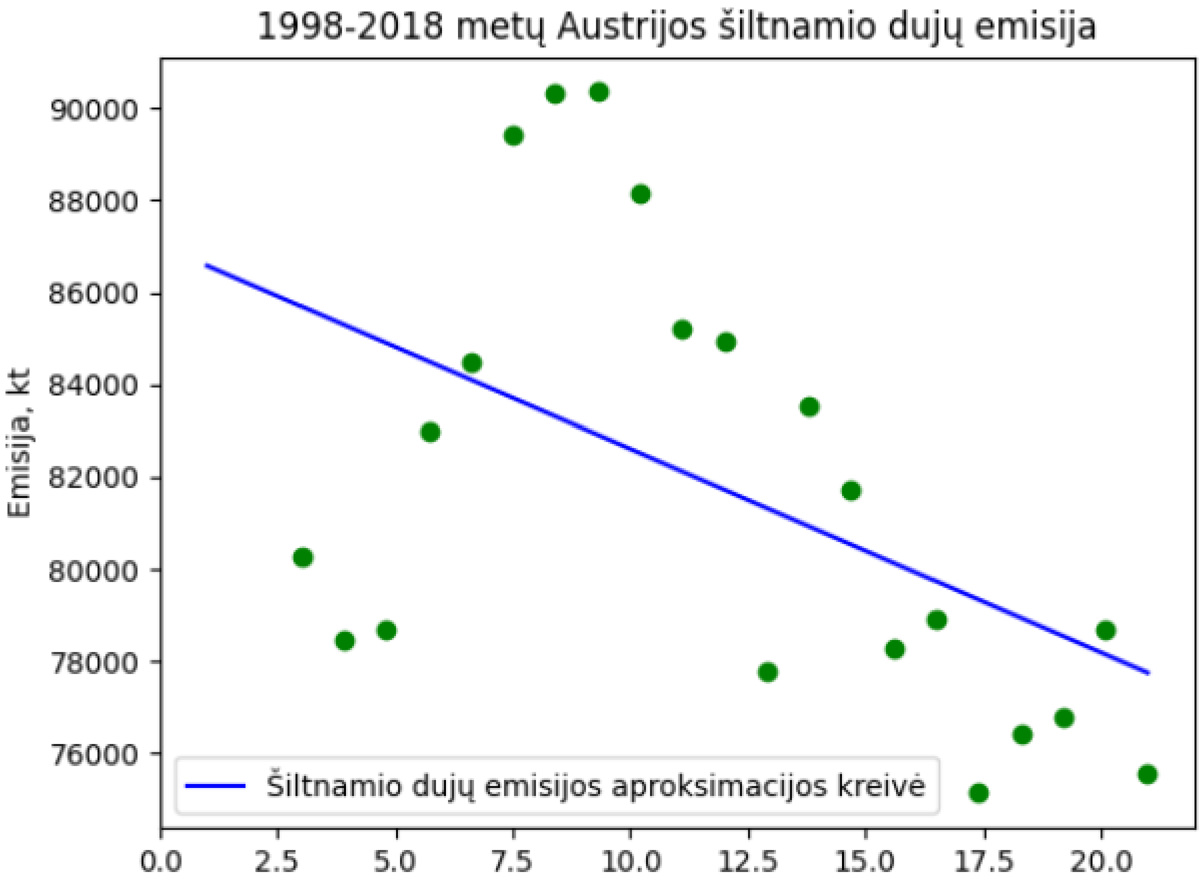
```
    plt.show()


# skaiciuoja lagranzo daugianario isvestine pagal x
def d_lagrange(x_array, j, x):
    count = len(x_array)
    dl = np.zeros(x.shape, dtype=np.double)  # dl israiskos skaitiklis
    for i in range(0, count):  # ciklas per atmetamus narius
        if i == j:
            continue
        lds = np.ones(x.shape, dtype=np.double)
        for k in range(0, count):
            if k != j and k != i:
                lds = lds * (x - x_array[k])
        dl = dl + lds
    ldv = np.ones(x.shape, dtype=np.double)  # dl israiskos vardiklis
    for k in range(0, count):
        if k != j:
            ldv = ldv * (x_array[j] - x_array[k])
    dl = dl / ldv
    return dl


# sumuoja visas daugianario koeficientu vertes
def lagrange(x_array, j, x):
    n = len(x_array)
    lagrange_val = np.ones(x.shape, dtype=np.double)
    for k in range(0, n):
        if j != k:
            lagrange_val = lagrange_val * ((x - x_array[k]) / (x_array[j] - x_array[k]))
    return lagrange_val


def f_dy(x, xi, x1, x_, _y, y, y_):
    return (2 * x - xi - x_) / ((x1 - xi) * (x1 - x_)) * _y + (2 * x - x1 - x_) / ((xi - x1) * (xi - x_)) * y + (
            2 * x - x1 - xi) / ((x_ - x1) * (x_ - xi)) * y_


def hermite(X, j, x):
    lagr_val = lagrange(X, j, x)
    lagr_deriv = d_lagrange(X, j, X[j])
    U = (1 - 2 * lagr_deriv * (x - X[j])) * np.square(lagr_val)
    V = (x - X[j]) * np.square(lagr_val)
    return U, V


def akima(x, y):
    dy = []
    n = len(x)
    for i in range(n):
        if i == 0:
            x1 = x[0]
            xi = x[1]
            x_ = x[2]
            _y = y[0]
            _y_ = y[1]
            y_ = y[2]
```

```
            dy.append(f_dy(x1, xi, x1, x_, _y, _y_, y_))
        elif i == n - 1:
            x1 = x[n - 3]
            xi = x[n - 2]
            x_ = x[n - 1]
            _y = y[n - 3]
            _y_ = y[n - 2]
            y_ = y[n - 1]
            dy.append(f_dy(x_, xi, x1, x_, _y, _y_, y_))
        else:
            x1 = x[i - 1]
            xi = x[i]
            x_ = x[i + 1]
            _y = y[i - 1]
            _y_ = y[i]
            y_ = y[i + 1]
            dy.append(f_dy(xi, xi, x1, x_, _y, _y_, y_))
    return dy


years = np.array(
    [1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018])
emission = np.array([80295, 78470, 78694, 83002, 84500, 89432, 90323, 90357, 88153, 85204, 84954, 77767, 83551, 81739, 78293, 78909, 75143, 76430, 76781, 78699, 75582])

spline(years, emission)
```

3. Trečia užduotis. Aproksimavimas

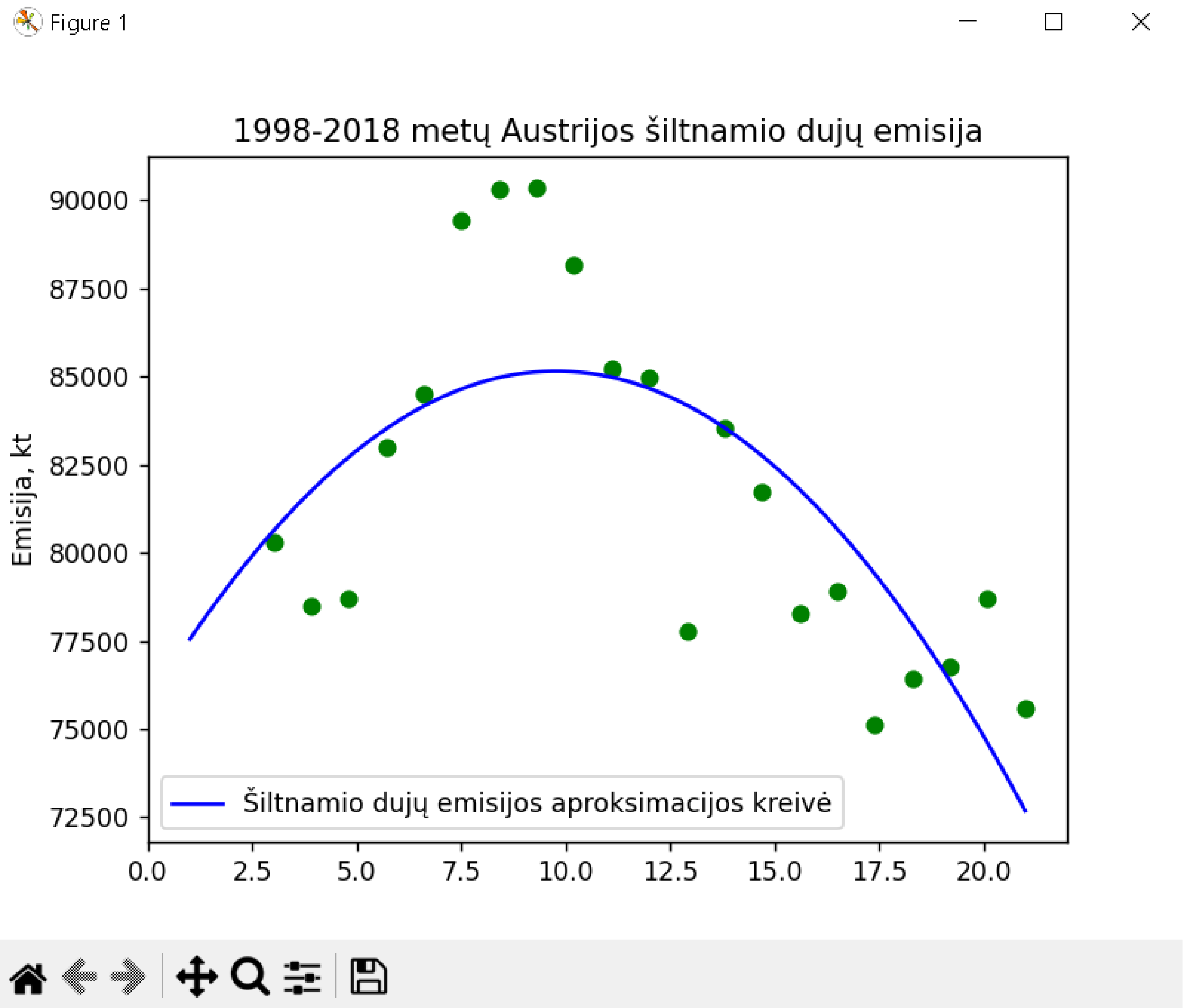
Mažiausių kvadratų metodu sudarykite ***2 lentelėje*** nurodytos šalies 1998-2018 metų šiltnamio dujų emisiją (galimo duomenų šaltinio nuoroda apačioje) aproksimuojančias kreives (**pirmos, antros**, **trečios** ir **penktos** eilės daugianarius). Pateikite gautas daugianarių išraiškas ir grafinius rezultatus.

8	Austrija	Ermito (Akima)



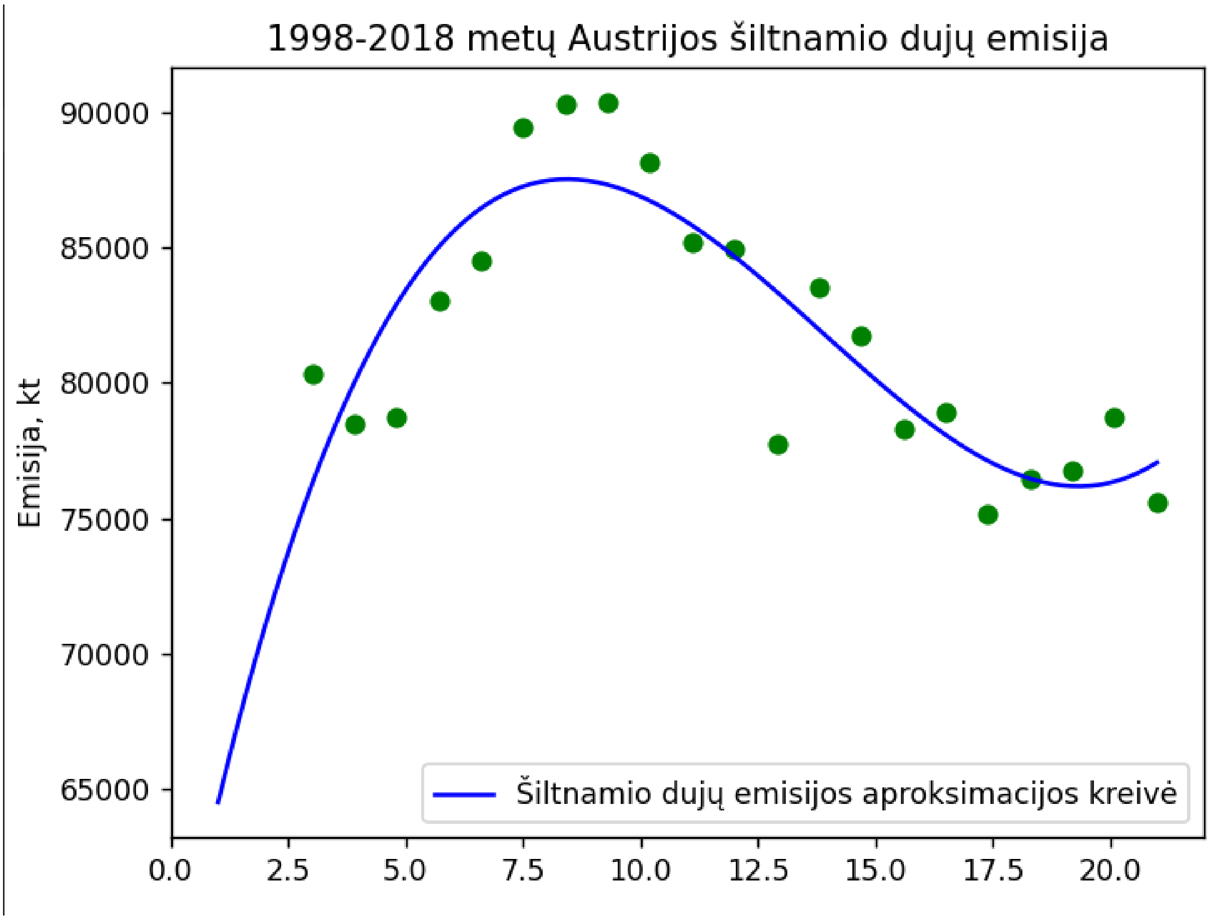
3 pav. Pirmos eiles daugianaris

Aproksimacijos funkcijos koeficientai: [79525.28138528, 401.38528139]



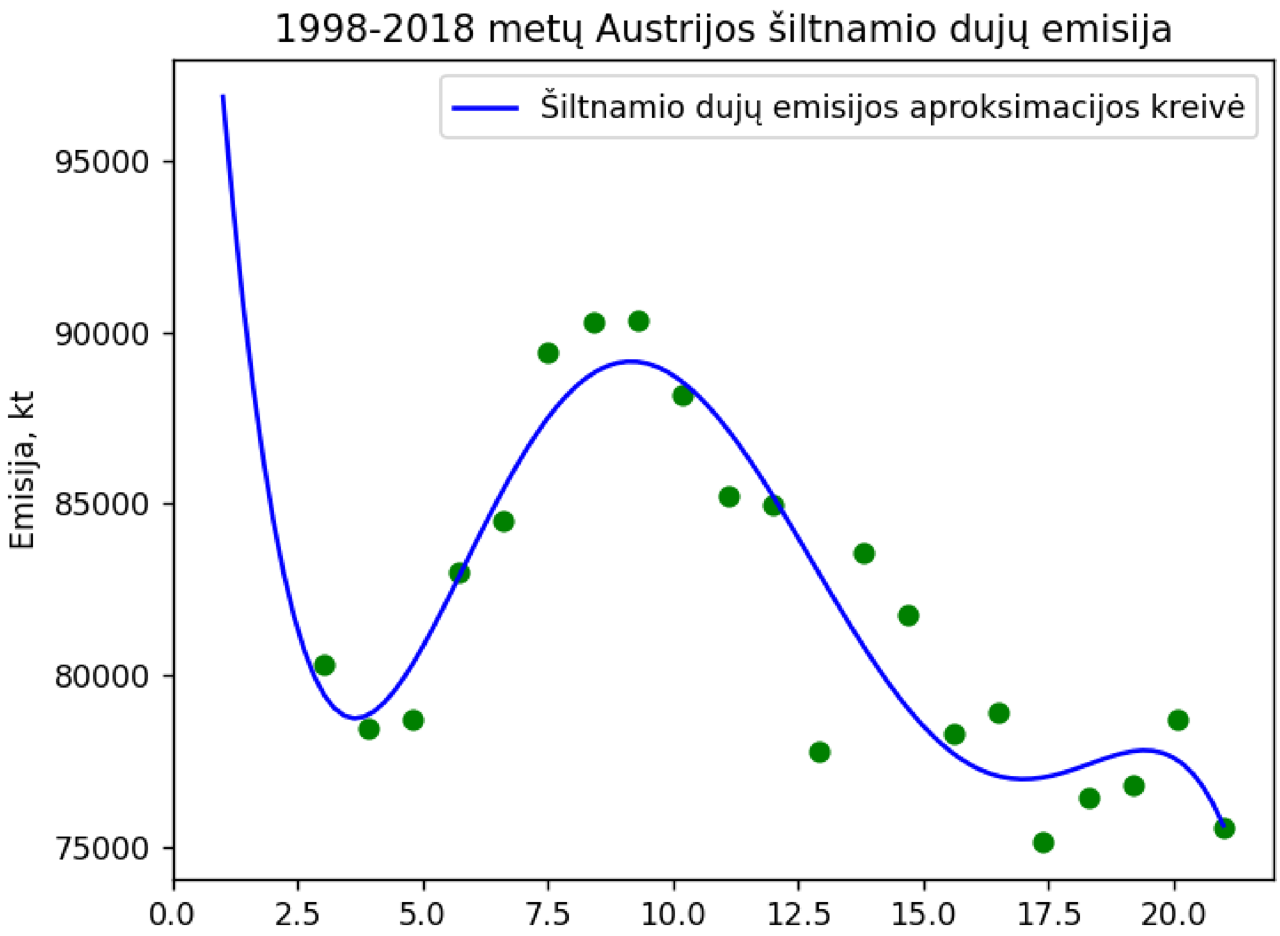
4 pav. Antros eilės daugianaris

Aproksimacijos funkcijos koeficientai:
[[75717.6223794   1932.40089794   -98.89988109]]



5 pav. Trečios eilės daugianaris

Aproksimacijos funkcijos koeficientai:
[[5.66523047e+04 8.56591313e+03 -7.29491851e+02 1.75164436e+01]]



6 pav. Penktos eilės daugianaris

Aproksimacijos funkcijos koeficientai:
[[1.19700123e+05 -2.93836667e+04 7.25741176e+03 -7.37694628e+02
  3.28162977e+01 -5.33262589e-01]]

3.1. Programos kodas

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

# suranda daugianario reikšmes mažiausių kvadratų metodu
def base(degree, x_array):
    count = len(x_array)
    g_array = np.zeros((count, degree + 1))
    for index in range(0, count):
        for j in range(0, degree + 1):
            g_array[index, j] = x_array[index] ** j
    return g_array

# suranda daugianares funkcijos koeficientus
def coefficients(g_array, y_array):
    tmp1 = (g_array.transpose()).dot(g_array)
    tmp2 = (g_array.transpose()).dot((np.matrix(y_array)).transpose())
    coefficient_array = np.linalg.solve(tmp1, tmp2)
    return coefficient_array

# funkcijos Y atsakymo reikšmės
def answers(c_arr, x):
    y = 0
```

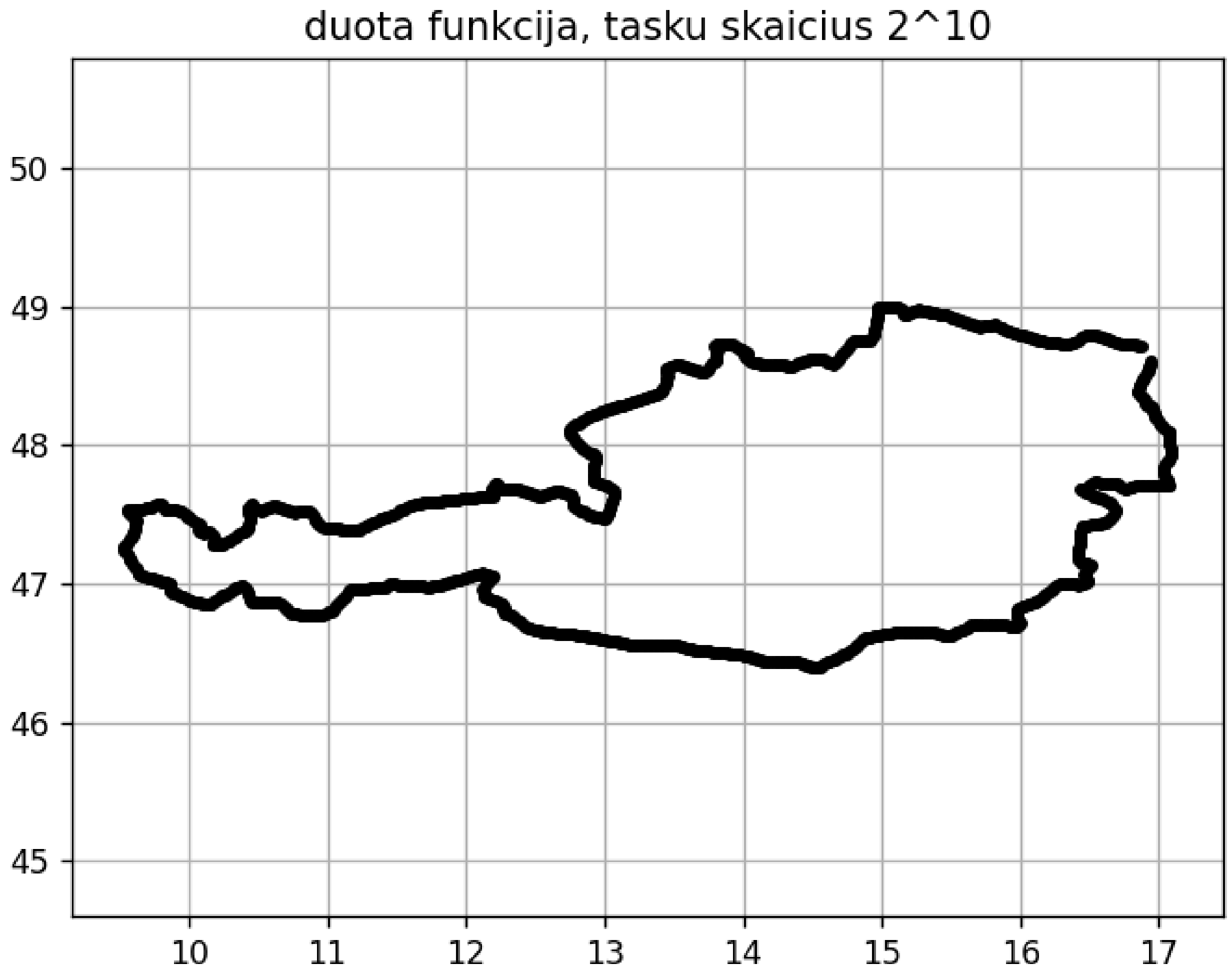
```
    for index in range(0, len(c_arr)):
        y = y + c_arr[index] * (x ** index)
    return y

Y = np.array([80295, 78470, 78694, 83002, 84500, 89432, 90323, 90357, 88153, 85204, 84954, 77767, 83551, 81739, 78293,
78909, 75143, 76430, 76781, 78699, 75582])
n = len(Y)

X = np.linspace(3, n, n)
# aproksimuojancios kreivės eilė
deg = 7
draw_points = 100

G = base(deg, X)
c = coefficients(G, Y)

print("Aproksimacijos funkcijos koeficientai:")
print(c.transpose())

draw_x = np.linspace(1, n, draw_points)
draw_y = np.zeros(draw_points)
for i in range(0, draw_points):
    draw_y[i] = answers(c, draw_x[i])


fig = plt.figure()
ax = fig.add_subplot()
ax.plot(X, Y, 'go')
plt.title("1998-2018 met\u0173 Austrijos \u0161iltnamio duj\u0173 emisija")
plt.ylabel("Emisija, kt")
ax.plot(draw_x, draw_y, 'b-', label='Šiltnamio dujų emisijos aproksimacijos kreivė')
plt.legend()
plt.draw()
plt.show()
```

4. Ketvirta užduotis. Parametrinis aproksimavimas

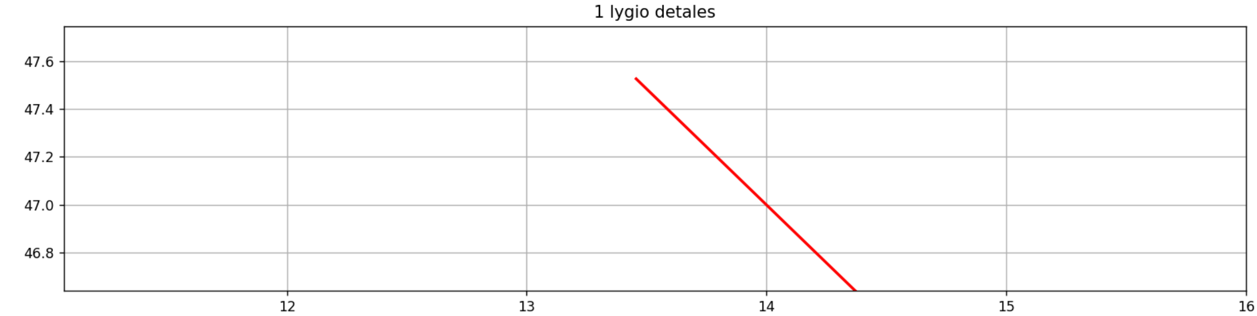
Naudodami **parametrinį aproksimavimą Haro bangelėmis** suformuokite ***2 lentelėje*** nurodytos šalies kontūrą. Analizuokite bent 10 detalumo lygių. Pateikite aproksimavimo rezultatus (aproksimuotą kontūro kreivę) ne mažiau kaip 4 skirtinguose lygmenyse. Jei šalis turi keletą atskirų teritorijų (pvz., salų), pakanka analizuoti didžiausią iš jų.

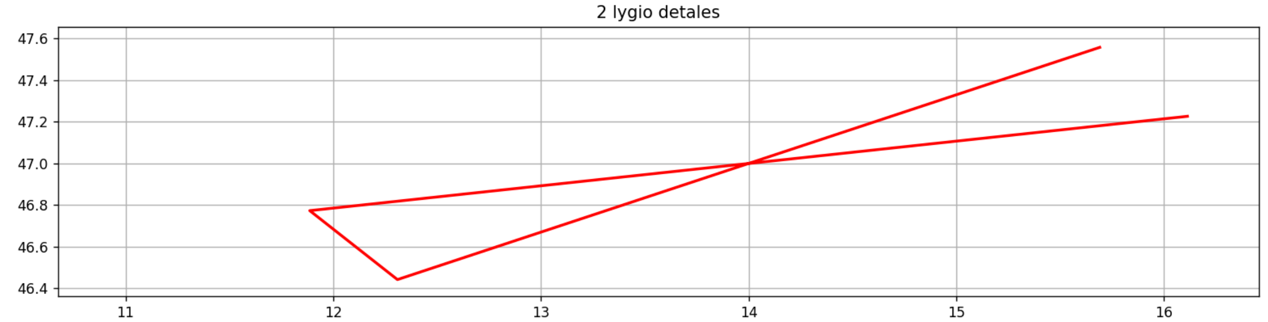
8	Austrija	Ermito (Akima)

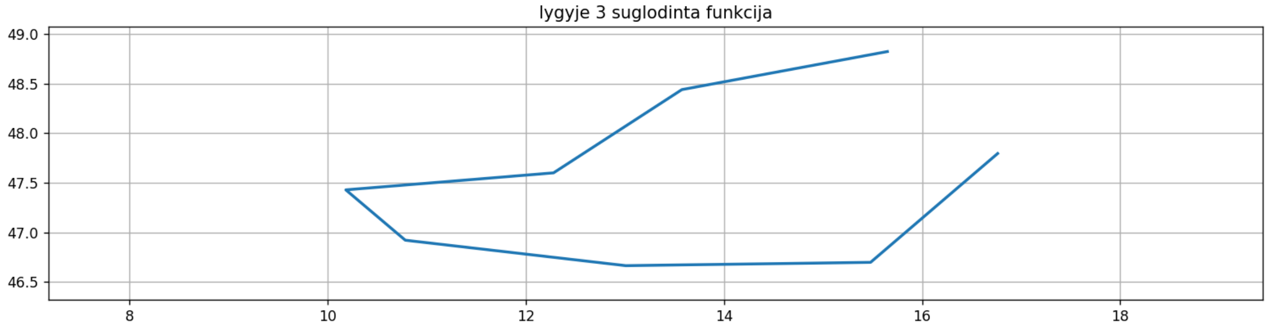
4.1. Pradiniai duomenys

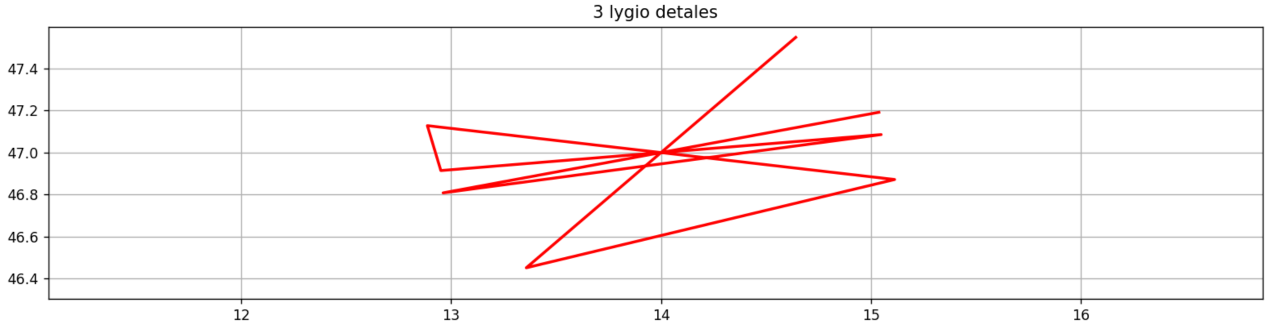


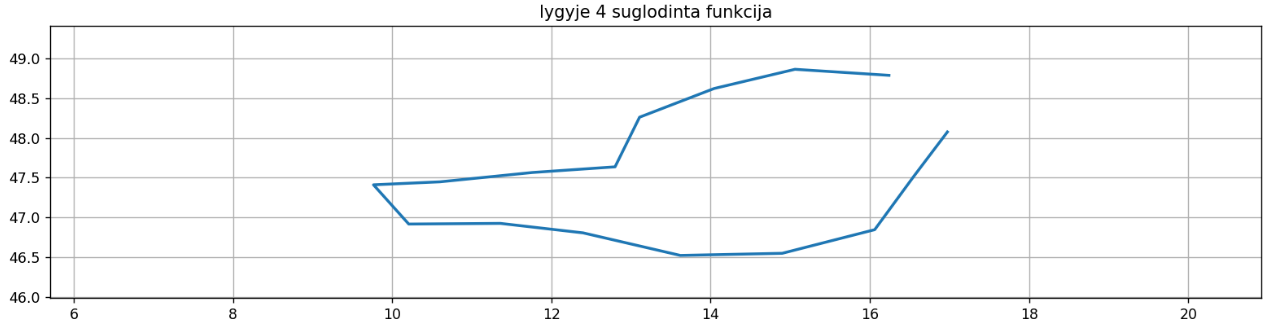
4.2. Aproksimavimo rezultatai

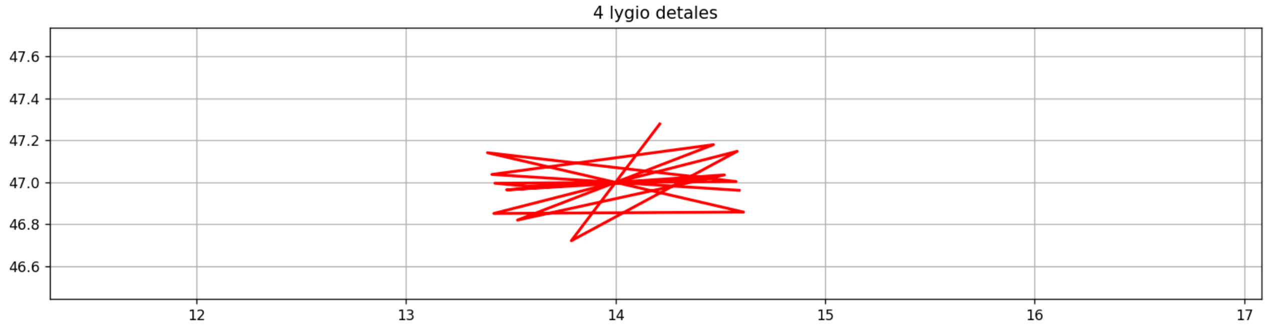


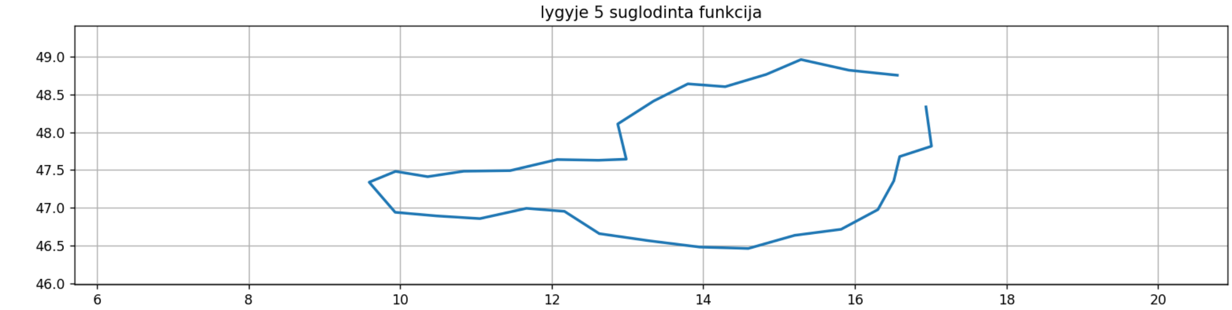


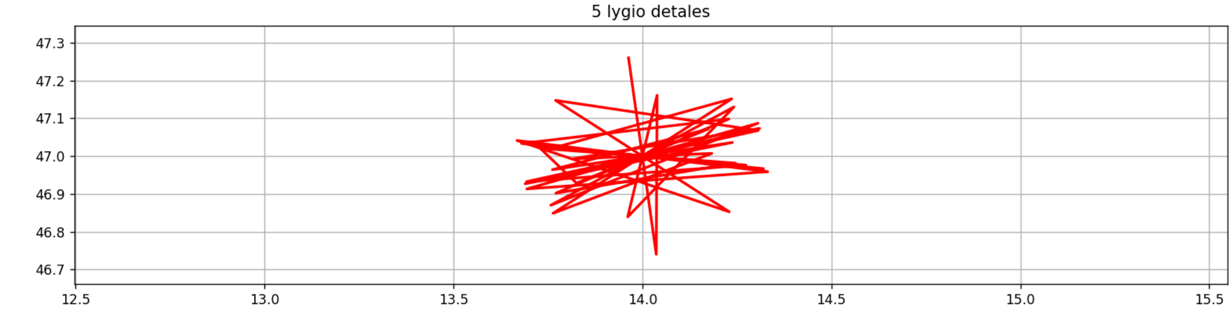

lygyje 3 suglodinta funkcija
49.0
48.5
48.0
47.5
47.0
46.5
8
10
12
14
16
18

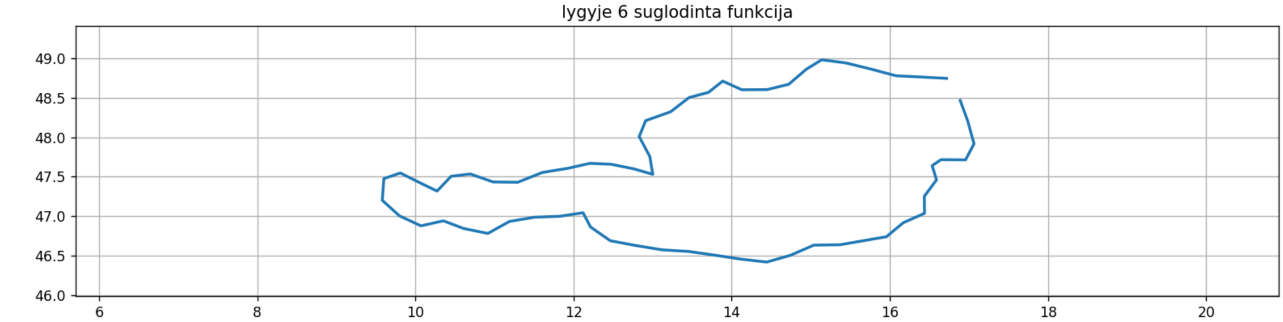

3 lygio detales
47.4
47.2
47.0
46.8
46.6
46.4
12
13
14
15
16

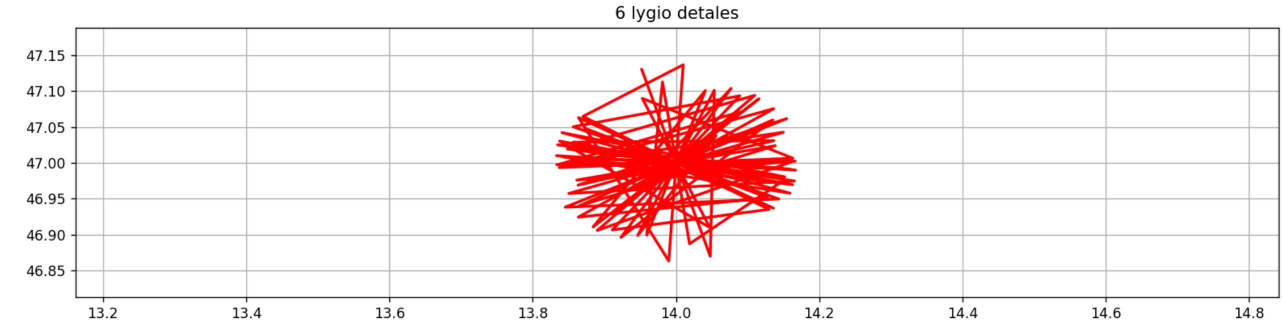

lygyje 4 suglodinta funkcija
49.0
48.5
48.0
47.5
47.0
46.5
46.0
6
8
10
12
14
16
18
20

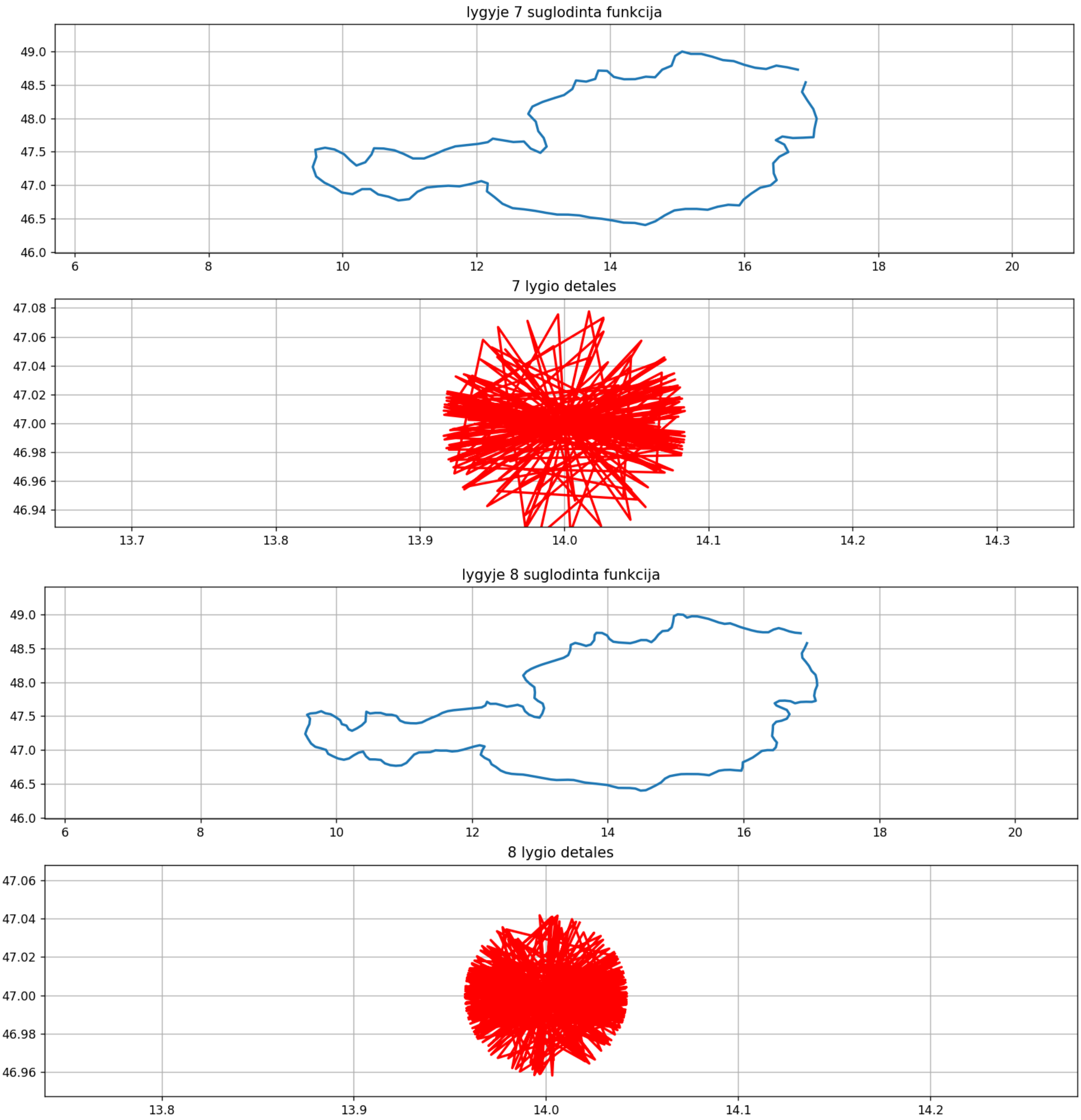

4 lygio detales
47.6
47.4
47.2
47.0
46.8
46.6
12
13
14
15
16
17

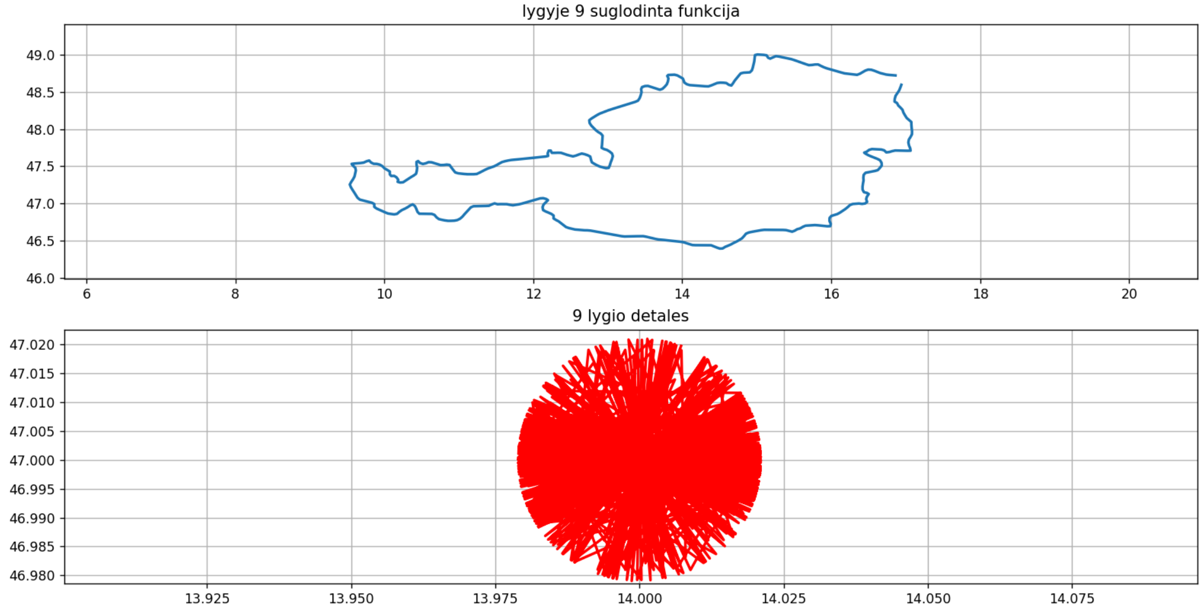

lygyje 5 suglodinta funkcija
49.0
48.5
48.0
47.5
47.0
46.5
46.0
6
8
10
12
14
16
18
20

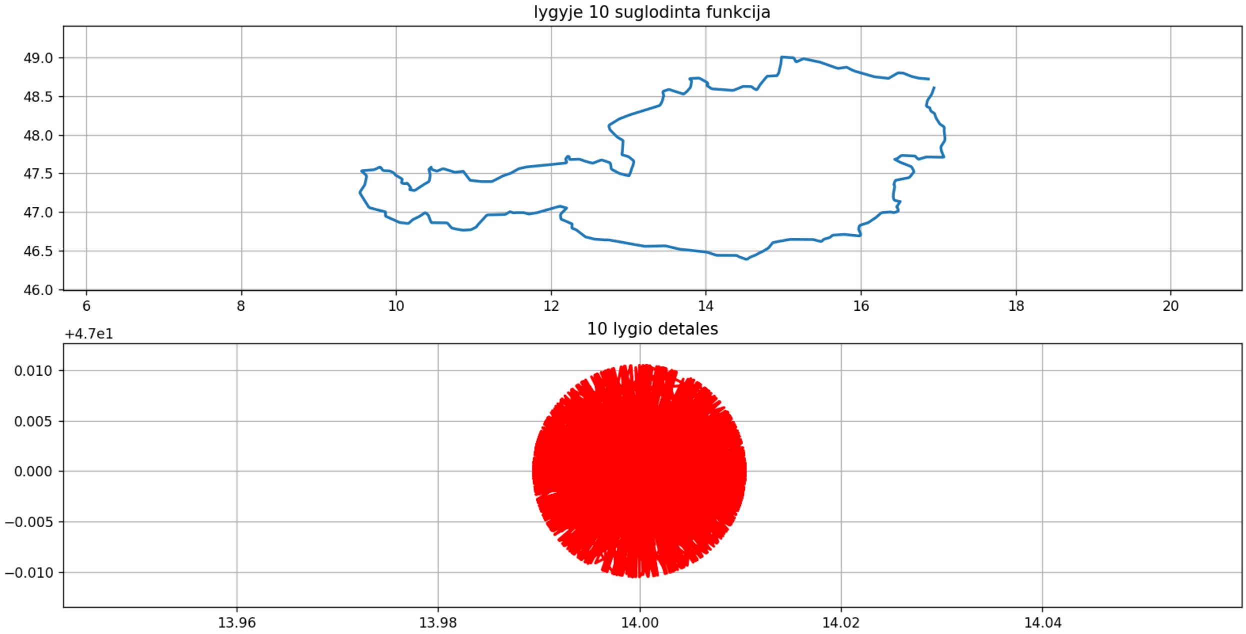

5 lygio detales
47.3
47.2
47.1
47.0
46.9
46.8
46.7
12.5
13.0
13.5
14.0
14.5
15.0
15.5


lygyje 6 suglodinta funkcija
49.0
48.5
48.0
47.5
47.0
46.5
46.0
6
8
10
12
14
16
18
20


6 lygio detales
47.15
47.10
47.05
47.00
46.95
46.90
46.85
13.2
13.4
13.6
13.8
14.0
14.2
14.4
14.6
14.8


lygyje 7 suglodinta funkcija
49.0 48.5 48.0 47.5 47.0 46.5 46.0
6 8 10 12 14 16 18 20
7 lygio detales
47.08 47.06 47.04 47.02 47.00 46.98 46.96 46.94
13.7 13.8 13.9 14.0 14.1 14.2 14.3
lygyje 8 suglodinta funkcija
49.0 48.5 48.0 47.5 47.0 46.5 46.0
6 8 10 12 14 16 18 20
8 lygio detales
47.06 47.04 47.02 47.00 46.98 46.96
13.8 13.9 14.0 14.1 14.2





4.3. *Programos kodas*

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

def Haar_scaling(x, j, k, a, b):
    eps = 1e-9
    xtld = (x - a) / (b - a)
    xx = 2**j * xtld - k
    h = 2**(j/2) * (np.sign(xx + eps) - np.sign(xx - 1 - eps)) / (2 * (b - a))
    return h

def Haar_wavelet(x, j, k, a, b):
    eps = 1e-9
    xtld = (x - a) / (b - a)
    xx = 2**j * xtld - k
```

```
    h = 2**(j/2) * (np.sign(xx + eps) - 2 * np.sign(xx - 0.5) + np.sign(xx - 1 - eps)) / (2 * (b - a))
    return h

def Haar_wavelet_approximation(SX, SY, n, m):
    a = min(SX)
    b = max(SX)
    nnn = 2**n
    smooth = (b - a) * SY * 2**(-n/2)

    details = {}
    for i in range(1, m + 1):
        smooth1 = (smooth[::2] + smooth[1::2]) / np.sqrt(2)
        details[i] = (smooth[::2] - smooth[1::2]) / np.sqrt(2)
        print(f'\n details {i} : ', details[i])
        smooth = smooth1

    print(f'\n smooth {i} : ', smooth)
    return smooth, details

def main():
    plt.close('all')

    n = 10
    nnn = 2**n
    fhx = open(r'austriax.txt', 'r')
    fhy = open(r'austriay.txt', 'r')
    plt.figure(1)
    plt.axis('equal')
    plt.grid(True)

    SX = np.array(fhx.read().split(), dtype=float)
    SY = np.array(fhy.read().split(), dtype=float)

    t = np.zeros_like(SX)
    t[1:] = np.cumsum(np.linalg.norm(np.column_stack((SX[1:], SY[1:])) - np.column_stack((SX[:-1], SY[:-1])), axis=1))

    a, b = min(t), max(t)
    t1 = np.linspace(a, b, nnn)
    tsx = np.interp(t1, t, SX)
    tsy = np.interp(t1, t, SY)
    SX, SY, t = tsx, tsy, t1

    plt.plot(SX, SY, 'k.')
    plt.title(f'duota funkcija, tasku skaicius 2^{n}')
    xmin, xmax = min(SX), max(SX)
    ymin, ymax = min(SY), max(SY)

    m = 10
    smoothx, detailsx = Haar_wavelet_approximation(t, SX, n, m)
    smoothy, detailsy = Haar_wavelet_approximation(t, SY, n, m)

    print("smoothx:", smoothx)
    print("smoothy:", smoothy)

    hx = np.zeros(nnn)
    hy = np.zeros(nnn)
```

```
    for k in range(2**(n-m)):
        hx += smoothx[k] * Haar_scaling(t, n-m, k, a, b)
        hy += smoothy[k] * Haar_scaling(t, n-m, k, a, b)

    plt.figure(2)




    plt.figure(2)
    plt.axis('equal')
    plt.axis([xmin, xmax, ymin, ymax])
    plt.grid(True)
    plt.plot(hx, hy, '.', linewidth=2)
    plt.title(f'lygyje {0} suglodinta funkcija')
    leg = [f'suglodinta funkcija, detalumo lygmuo {n-m}']


    for i in range(m):
        h1x = np.zeros(nnn)
        h1y = np.zeros(nnn)

        for k in range(2**(n-m+i)):
            h1x += detailsx[m-i][k] * Haar_wavelet(t, n-m+i, k, a, b)
            h1y += detailsy[m-i][k] * Haar_wavelet(t, n-m+i, k, a, b)


        hx += h1x
        hy += h1y

        plt.figure(i + 3, figsize=(10, 10))

        plt.subplot(2, 1, 1)  # First subplot
        plt.axis('equal')
        plt.axis([xmin, xmax, ymin, ymax])
        plt.grid(True)
        plt.plot(hx, hy, linewidth=2)
        plt.title(f'lygyje {i+1} suglodinta funkcija')

        plt.subplot(2, 1, 2)  # Second subplot
        plt.axis('equal')
        plt.axis([xmin, xmax, ymin, ymax])
        plt.grid(True)
        plt.plot(h1x + 14, h1y + 47, 'r-', linewidth=2)
        plt.title(f'{i+1} lygio detales')

        leg.append(f'lygmens {n-m+i} detales')




    plt.show()

if __name__ == "__main__":
    main()
```