**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**Informatikos fakultetas**

**Skaitiniai metodai ir algoritmai (P170B115)**

***3 laboratorinis darbas***

17 variantas

**Dėstytojas**:

lekt. Andrius Kriščiūnas

**Darbą atliko**:

IFF – 8/13 Mykolas Paulauskas

KAUNAS, 2020

Turinys

[Pirmoji užduotis 3](#_Toc57293659)

[Pirmos užduoties pirma dalis 3](#_Toc57293660)

[Pirmos užduoties antra dalis 6](#_Toc57293661)

[Pirmos užduoties išvados 10](#_Toc57293662)

[Antra užduotis 11](#_Toc57293663)

[Antros užduoties pirma dalis 11](#_Toc57293664)

[Antros užduoties antra dalis 14](#_Toc57293665)

[Antros užduoties išvados 17](#_Toc57293666)

[Trečia užduotis 17](#_Toc57293667)

[Trečios užduoties išvados 24](#_Toc57293668)

[Ketvirta užduotis 25](#_Toc57293669)

[Ketvirtos užduoties išvados 29](#_Toc57293670)

# Pirmoji užduotis

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

1. Taškai pasiskirstę tolygiai.
2. Taškai apskaičiuojami naudojant Čiobyševo abscises.

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



pav. 1 Užduoties variantas

## Pirmos užduoties pirma dalis

**Programos kodas:**

from matplotlib import pyplot as plt

import numpy as np

def f(x):

    return np.cos(2 \* x) / (np.sin(2 \* x) + 1.5) - x / 5

def chebyshev\_interpolation(x, coef):

    T = np.zeros(len(coef))

    xAmendent = (2 \* x) / (b - a) - (b + a) / (b - a)

    T[0] = 1

    T[1] = xAmendent

    # Čiobyševo interpoliavimo matricos sudarymas

    for i in range(2, n):

        T[i] = 2 \* xAmendent \* T[i - 1] - T[i - 2]

    # Dauginame koeficientus su Čiobyševo interpoliavimo matrica, gauname atsakymus

    y = np.dot(T, coef)

    return y

def target\_fx\_points(start, finnish, step):

    x = []

    y = []

    it = start

    while it <= finnish:

        x.append(it)

        y.append(f(it))

        it = it + step

    return x, y

def chebyshev\_interpolation\_points(start, finnish, step, coef):

    x = []

    y = []

    it = start

    while it <= finnish:

        x.append(it)

        y.append(chebyshev\_interpolation(it, coef))

        it = it + step

    return x, y

if \_\_name\_\_ == "\_\_main\_\_":

    # pasirenkamem, kurią užduoties dalį norime vykdyti

    first = True

    # laisvai pasitinktas interpoliavimo taškų kiekis

    n = 15

    # funkcijos intervalas kuria interpoliuosime

    a = -2

    b = 3

    # x taškai

    x = []

    # taškai pasiskirstę tolygiai

    if first:

        x = np.linspace(a, b, n, endpoint=True)

    # taškai apskaičiuojami naudojant Čiobyševo abscises

    else:

        for i in range(n):

            # Kadangi skaičiuojame ne intervale  [-1, 1] tai reikalinga x'ui apskaičiuoti keitinį

            xAmendment = ((b - a) / 2) \* np.cos((np.pi \* (2 \* i + 1)) / (2 \* n)) + (b + a) / 2

            x.append(xAmendment)

    # y taškai

    y = []

    for i in range(n):

        y.append(f(x[i]))

    T = np.zeros((n, n))

    # randa funkcijos x reikšmes

    for i in range(n):

        T[i, 0] = 1

        xAmendment = (2 \* x[i]) / (b - a) - (b + a) / (b - a)

        T[i, 1] = xAmendment

        for j in range(2, n):

            T[i, j] = 2 \* xAmendment \* T[i, j - 1] - T[i, j - 2]

    # Kadangi žimone kokias vertes y'as turi turėti, galime apsiskaičiuoti Čiobyševo koeficientus

    coef = np.linalg.solve(T, y)

    print("Koeficientai:\n", coef)

    # randa funkcijos analitinius taškus

    fx, fy = target\_fx\_points(a, b, 0.01)

    # randa interpoliuotos funkcijos taškus

    ix, iy = chebyshev\_interpolation\_points(a, b, 0.01, coef)

    # piešia grafiką

    plt.title('Duota funkcija f(x): cos(2\*x)/(sin(2\*x)+1.5) - x/5')

    plt.plot(fx, fy, label='Funkcija f(x)')

    plt.plot(ix, iy, label='Interpoliuota funkcija f(x)', color="green", linestyle='dashed')

    plt.fill\_between(fx, fy, iy, color="orange", label="Netiktis")

    plt.scatter(x, y, label='Funkcijos f(x) taškai')

    plt.xlabel('x')

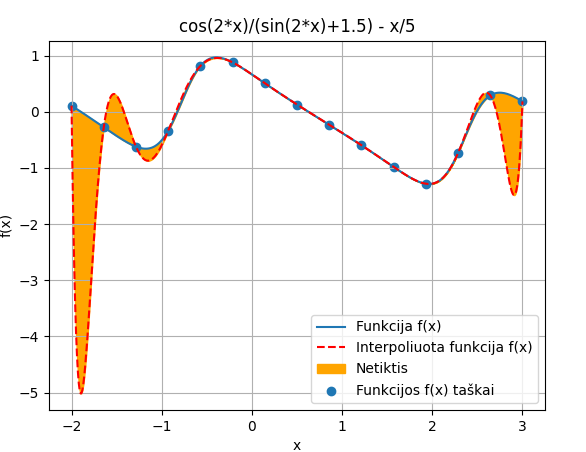
    plt.ylabel('f(x)')

    plt.legend()

    plt.grid(0.5)

    plt.show()

**Rezultatai:**



pav. 2 Pirmos užduoties pirmos dalies rezultatai

## Pirmos užduoties antra dalis

**Programos kodas:**

from matplotlib import pyplot as plt

import numpy as np

def f(x):

    return np.cos(2 \* x) / (np.sin(2 \* x) + 1.5) - x / 5

def chebyshev\_interpolation(x, coef):

    T = np.zeros(len(coef))

    xAmendent = (2 \* x) / (b - a) - (b + a) / (b - a)

    T[0] = 1

    T[1] = xAmendent

    # Čiobyševo interpoliavimo matricos sudarymas

    for i in range(2, n):

        T[i] = 2 \* xAmendent \* T[i - 1] - T[i - 2]

    # Dauginame koeficientus su Čiobyševo interpoliavimo matrica, gauname atsakymus

    y = np.dot(T, coef)

    return y

def target\_fx\_points(start, finnish, step):

    x = []

    y = []

    it = start

    while it <= finnish:

        x.append(it)

        y.append(f(it))

        it = it + step

    return x, y

def chebyshev\_interpolation\_points(start, finnish, step, coef):

    x = []

    y = []

    it = start

    while it <= finnish:

        x.append(it)

        y.append(chebyshev\_interpolation(it, coef))

        it = it + step

    return x, y

if \_\_name\_\_ == "\_\_main\_\_":

    # pasirenkamem, kurią užduoties dalį norime vykdyti

    first = False

    # laisvai pasitinktas interpoliavimo taškų kiekis

    n = 15

    # funkcijos intervalas kuria interpoliuosime

    a = -2

    b = 3

    # x taškai

    x = []

    # taškai pasiskirstę tolygiai

    if first:

        x = np.linspace(a, b, n, endpoint=True)

    # taškai apskaičiuojami naudojant Čiobyševo abscises

    else:

        for i in range(n):

            # Kadangi skaičiuojame ne intervale  [-1, 1] tai reikalinga x'ui apskaičiuoti keitinį

            xAmendment = ((b - a) / 2) \* np.cos((np.pi \* (2 \* i + 1)) / (2 \* n)) + (b + a) / 2

            x.append(xAmendment)

    # y taškai

    y = []

    for i in range(n):

        y.append(f(x[i]))

    T = np.zeros((n, n))

    # randa funkcijos x reikšmes

    for i in range(n):

        T[i, 0] = 1

        xAmendment = (2 \* x[i]) / (b - a) - (b + a) / (b - a)

        T[i, 1] = xAmendment

        for j in range(2, n):

            T[i, j] = 2 \* xAmendment \* T[i, j - 1] - T[i, j - 2]

    # Kadangi žimone kokias vertes y'as turi turėti, galime apsiskaičiuoti Čiobyševo koeficientus

    coef = np.linalg.solve(T, y)

    print("Koeficientai:\n", coef)

    # randa funkcijos analitinius taškus

    fx, fy = target\_fx\_points(a, b, 0.01)

    # randa interpoliuotos funkcijos taškus

    ix, iy = chebyshev\_interpolation\_points(a, b, 0.01, coef)

    # piešia grafiką

    plt.title('Duota funkcija f(x): cos(2\*x)/(sin(2\*x)+1.5) - x/5')

    plt.plot(fx, fy, label='Funkcija f(x)')

    plt.plot(ix, iy, label='Interpoliuota funkcija f(x)', color="green", linestyle='dashed')

    plt.fill\_between(fx, fy, iy, color="orange", label="Netiktis")

    plt.scatter(x, y, label='Funkcijos f(x) taškai')

    plt.xlabel('x')

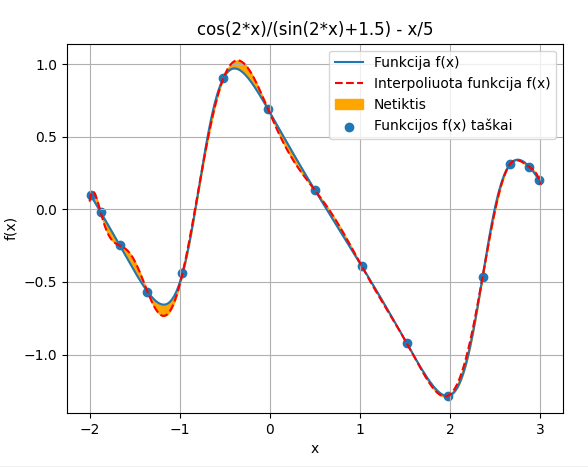
    plt.ylabel('f(x)')

    plt.legend()

    plt.grid(0.5)

    plt.show()

**Rezultatai:**



pav. 3 Pirmos užduoties antros dalies rezultatai

## Pirmos užduoties išvados

Pagal gautus grafikus matome jog kai taškus parenkame pagal Čiobyševo abscises gauta interpoliuojanti funkcija yra artimesnė funkcijai, kurios rezultato mes norime pasiekti. Šią išvaizdą pagrindžia antros dalies grafike matoma žymiai mažesnė netektis. Nors Čiobyševo taškai atrodo beveik lygūs, jie yra suskirstyti taip, kad interpoliuojančios funkcijos vingiai būtų kuo mažesni.

# Antra užduotis

**II užduotis. Interpoliavimas daugianariu ir splainu per duotus taškus**

Pagal 2 lentelėje pateiktą šalį ir metus, sudaryti interpoliuojančią kreivę 12 mėnesių temperatūroms atvaizduotinurodytais metodais:

1. Daugianariu, sudarytu naudojant 1 lentelėje nurodytas bazines funkcijas.
2. 2 lentelėje nurodyto tipo splainu

pav. 4 Antra užduotis



pav. 5 Antros užduoties pirmos dalies variantas



pav. 6 Antros užduoties antros dalies variantas

## Antros užduoties pirma dalis

**Programos kodas:**

from matplotlib import pyplot as plt

import numpy as np

import calendar

def f(x):

    return np.cos(2 \* x) / (np.sin(2 \* x) + 1.5) - x / 5

def chebyshev\_interpolation(x, coef):

    T = np.zeros(len(coef))

    xAmendent = (2 \* x) / (b - a) - (b + a) / (b - a)

    T[0] = 1

    T[1] = xAmendent

    # Čiobyševo interpoliavimo matricos sudarymas

    for i in range(2, n):

        T[i] = 2 \* xAmendent \* T[i - 1] - T[i - 2]

    # Dauginame koeficientus su Čiobyševo interpoliavimo matrica, gauname atsakymus

    y = np.dot(T, coef)

    return y

def chebyshev\_interpolation\_points(start, finnish, step, coef):

    x = []

    y = []

    it = start

    while it <= finnish:

        x.append(it)

        y.append(chebyshev\_interpolation(it, coef))

        it = it + step

    return x, y

if \_\_name\_\_ == "\_\_main\_\_":

    y = np.array([

        0.12371,

        1.3831,

        5.29799,

        8.2207,

        10.1892,

        14.8951,

        16.617,

        14.5986,

        12.6457,

        9.69687,

        5.61371,

        0.02563

    ])

    # interpoliavimo taškų skaičius

    n = len(y)

    # x taškai

    x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

    a = 1

    b = 12

    T = np.zeros((n, n))

    # randa funkcijos x reikšmes

    for i in range(n):

        T[i, 0] = 1

        xAmendment = (2 \* x[i]) / (b - a) - (b + a) / (b - a)

        T[i, 1] = xAmendment

        for j in range(2, n):

            T[i, j] = 2 \* xAmendment \* T[i, j - 1] - T[i, j - 2]

    # randa koeficientus

    coef = np.linalg.solve(T, y)

    print("Koeficientai:\n", coef)

    # randa interpoliuotos funkcijos taškus

    ix, iy = chebyshev\_interpolation\_points(1, 12, 0.01, coef)

    # piešia grafiką

    plt.plot(x, y, label='Funkcija f(x)')

    plt.plot(ix, iy, label='Interpoliuota funkcija f(x)', color="red", linestyle='dashed')

    plt.scatter(x, y, label='Funkcijos f(x) taškai')

    plt.title("Austria 2014 monthly temperatures\nPolynomial interpolation")

    plt.xlabel('Month')

    plt.ylabel('Average month temperature in C')

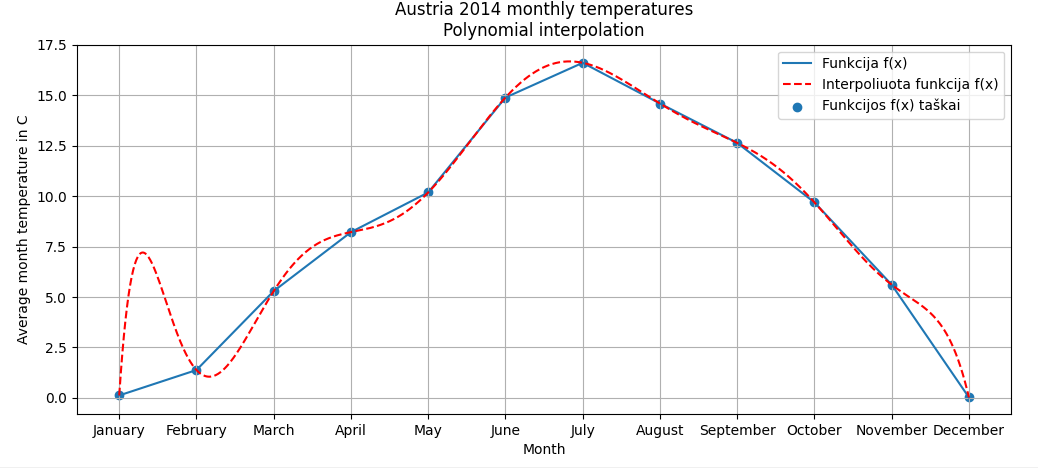
    plt.xticks(x, calendar.month\_name[1:13])

    plt.legend()

    plt.grid(0.5)

    plt.show()

**Rezultatai:**



## Antros užduoties antra dalis

**Programos kodas:**

import numpy as np

import calendar

from matplotlib import pyplot as plt

# Apskaičiuojame greta esančių taškų atstumus

def calculateDistance(x):

    n = len(x)

    d = np.zeros(n - 1)

    for i in range(n - 1):

        d[i] = x[i + 1] - x[i]

    return d

# Apskaičiuojame globalaus spline'o matricą

def calculateF(x, y):

    n = len(x)

    # x reikšmės

    T = np.zeros((n, n))

    # koeficientai

    b = np.zeros(n)

    # greta esančių taškų atstumai

    d = calculateDistance(x)

    # 40 skaidrė

    for i in range(n - 2):

        T[i, i] = d[i] / 6

        T[i, i + 1] = (d[i] + d[i + 1]) / 3

        T[i, i + 2] = d[i + 1] / 6

    # 40 skaidrė

    for i in range(n - 2):

        b[i] = ((y[i + 2] - y[i + 1]) / d[i + 1]) - ((y[i + 1] - y[i]) / d[i])

    # 43 skaidrė

    T[n - 2, 0] = d[0] / 3

    T[n - 2, 1] = d[0] / 6

    T[n - 2, n - 2] = d[n - 2] / 6

    T[n - 2, n - 1] = d[n - 2] / 3

    T[n - 1, 0] = 1

    T[n - 1, n - 1] = -1

    b[n - 2] = ((y[1] - y[0]) / d[0]) - ((y[n - 1] - y[n - 2]) / d[n - 2])

    yy = np.linalg.solve(T, b)

    return yy

# skaiciuoja funkcijos reiksme taske, 44 skaidre

def spline(ff1, ff2, s, d, y1, y2):

    a = (ff1 \* (s\*\*2 / 2)) - (ff1 \* (s\*\*3 / (6\*d)))

    b = (ff2 \* (s\*\*3 / (6\*d))) + (((y2 - y1) / d) \* s)

    c = (ff1 \* ((d/3) \* s)) + (ff2 \* ((d/6) \* s))

    return a + b - c + y1

# Taškų skaičius

n = 12

aa = 1

x = np.arange(aa, aa+n)

# temperatures

y = np.array([

        0.12371,

        1.3831,

        5.29799,

        8.2207,

        10.1892,

        14.8951,

        16.617,

        14.5986,

        12.6457,

        9.69687,

        5.61371,

        0.02563

    ])

ff = calculateF(x, y)

# 41 skaidrė

ff[0] = 0

ff[n - 1] = 0

# atstumai tarp gretimų taškų

d = calculateDistance(x)

# grafiko paišymo žingsnis -1 laipsnyje

step = 100

xx = []

yy = []

# pradedame nuo sausio mėnesio

x1 = 1

# pradedame ciklą nuo antro taško

for i in range(1, n):

    # x2 šiuo atveju bus pradžios taškas (kairysis taškas)

    x2 = x1

    for j in range(step):

        # s yra ilgis, kuris nurodo per kiek esamas taškas yra nutolęs nuo kairiojo (pradinio) taško

        s = x2 - i

        xx.append(x2)

        value = spline(ff[i - 1], ff[i], s, d[i - 1], y[i - 1], y[i])

        yy.append(value)

        # didiname x kas 0.01

        x2 += 1 / step

    x1 += 1

plt.plot(xx, yy, label="Funkcija f(x)")

plt.scatter(x, y, label="Funkcijos f(x) taškai")

plt.xlabel("Month")

plt.ylabel("Average month temperature in C")

plt.legend()

plt.grid()

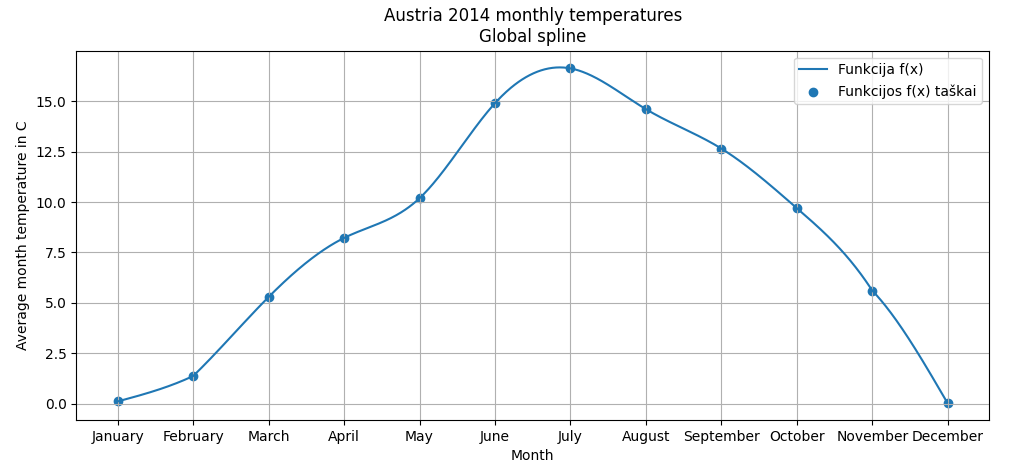
plt.xticks(x, calendar.month\_name[1:13])

plt.title("Austria 2014 monthly temperatures\nGlobal spline")

plt.grid(0.5)

plt.show()

**Rezultatai:**



## Antros užduoties išvados

Rezulatuose matome jog globalaus splaino interpoliuojanti funkcija kinta pastoviau negu nei interpoliuojančio daugianario funkcija. Kadangi interpoliuojančio daugianario funkcija vingiuoja tarp taškų, tai šis grafikas pateikia vaizdą, kuris tikėtinai neatspindi realaus temperatūrų kitimo grafiko. Skaičiuojant splainą mes jungiame taškus atskirai, dėl to vaizdas yra daug tolydesnis ir toks rezultatas teisingesnis mūsų duotiems duomenims

# Trečia užduotis

**III užduotis. Parametrinis interpoliavimas.**

Naudodami parametrinio interpoliavimo metodą 2 lentelėje nurodytu splainu suformuokite 2 lentelėje nurodytos šalies kontūrą. Pateikite pradinius duomenis ir rezultatus, gautus naudojant 10, 20, 50, 100 interpoliavimo taškų.



pav. 7 Trečios užduoties variantas

**Programos kodas:**

import numpy as np

from matplotlib import pyplot as plt

import math

# Hiperbolinė sinuso ir kosinuso funkcija

def sinh(a):

    return (math.exp(a) + math.exp(-a)) / 2

def cosh(a):

    return (math.exp(a) - math.exp(-a)) / 2

# randa delta x masyva

def calculateD(x):

    n = len(x)

    d = np.zeros(n - 1)

    for i in range(n - 1):

        d[i] = x[i + 1] - x[i]

    return d

# f'' masyvas

def CalculateF(x, y, q):

    n = len(x)

    # x reikšmės

    T = np.zeros((n, n))

    # koeficientai

    b = np.zeros(n)

    # greta esančių taškų atstumai

    d = calculateD(x)

    # 53 skaidrė

    for i in range(n - 2):

        T[i, i] = 1 / (d[i] \* q[i] \*\* 2) - 1 / (q[i] \* sinh(q[i] \* d[i]))

        f1 = cosh(sinh(q[i] \* d[i])) / (q[i] \* sinh(q[i] \* d[i]))

        f2 = cosh(sinh(q[i + 1] \* d[i + 1])) / (q[i + 1] \* sinh(q[i + 1] \* d[i + 1]))

        f3 = 1 / (d[i] \* q[i] \*\* 2) - 1 / (d[i + 1] \* q[i + 1] \*\* 2)

        T[i, i + 1] = f1 + f2 - f3

        T[i, i + 2] = 1 / (d[i + 1] \* q[i + 1] \*\* 2) - 1 / (q[i + 1] \* sinh(q[i + 1] \* d[i + 1]))

    # 53 skaidrė

    for i in range(n - 2):

        b[i] = ((y[i + 2] - y[i + 1]) / d[i + 1]) - ((y[i + 1] - y[i]) / d[i])

    # 53 skaidrė

    T[n - 2, 0] = 1 / (d[0] \* q[0] \*\* 2) - (cosh(q[0] \* d[0])) / (q[0] \* sinh(q[0] \* d[0]))

    T[n - 2, 1] = 1 / (q[0] \* sinh(q[0] \* d[0])) - 1 / (d[0] \* q[0] \*\* 2)

    T[n - 2, n - 2] = 1 / (q[n - 2] \* sinh(q[n - 2] \* d[n - 2])) - 1 / (d[n - 2] \* q[n - 2] \*\* 2)

    T[n - 2, n - 1] = 1 / (d[n - 2] \* q[n - 2] \*\* 2) - (cosh(q[n - 2] \* d[n - 2])) / (q[n - 2] \* sinh(q[n - 2] \* d[n - 2]))

    T[n - 1, 0] = 1

    T[n - 1, n - 1] = -1

    b[n - 2] = ((y[0] - y[1]) / d[0]) - ((y[n - 2] - y[n - 1]) / d[n - 2])

    yy = np.linalg.solve(T, b)

    return yy

def parametric\_spline(x1, x2, y1, y2, ff1, ff2, q):

        d = x2 - x1

        sss = 1

        a = ff2 / q \*\* 2 \* (sinh(q \* (d - sss))) / (sinh(q \* d))

        b = (y1 - ff1 / q \*\* 2) \* (d - sss) / d + ff2 / q \*\* 2 \* (sinh(q \* sss)) / (sinh(q \* d))

        c = (y2 - ff2 / q \*\* 2) \* sss / d

        return a + b + c

# program starts here

arr = np.array([…])

point = 100

n = arr.shape[0]

step = n / point

x = []

y = []

for i in range(point):

    index = int(step \* i)

    x.append(arr[index, 0])

    y.append(arr[index, 1])

n = len(x)

sigma = []

for i in range(n - 1):

    sigma.append(1)

t = np.arange(0, n, 1)

DDFX = CalculateF(t, x, sigma)

DDFY = CalculateF(t, y, sigma)

xx = []

yy = []

nnn = 100

for i in range(n - 1):

    nnn = 100

    for j in range(nnn):

        SX = parametric\_spline(t[i], t[i + 1], x[i], x[i + 1], DDFX[i], DDFX[i + 1], sigma[i])

        xx.append(SX)

        SY = parametric\_spline(t[i], t[i + 1], y[i], y[i + 1], DDFY[i], DDFY[i + 1], sigma[i])

        yy.append(SY)

plt.scatter(x, y, label="Taškai", color="red")

plt.plot(xx, yy, label="Interpoliacija")

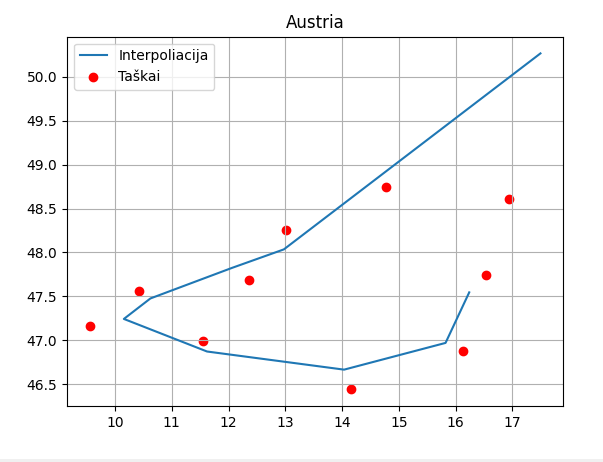
plt.title("Austria (not Australia ;))")

plt.legend()

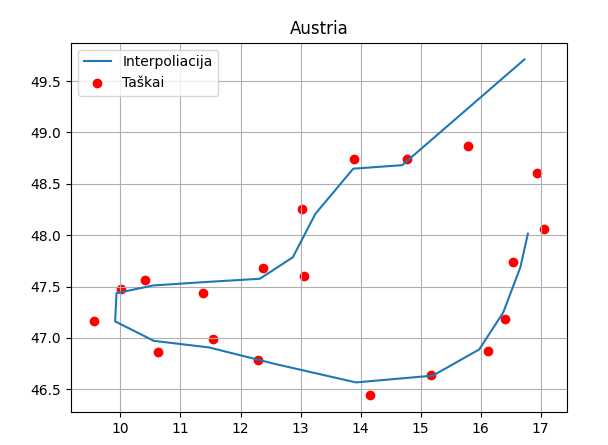
plt.grid(0.5)

plt.show()

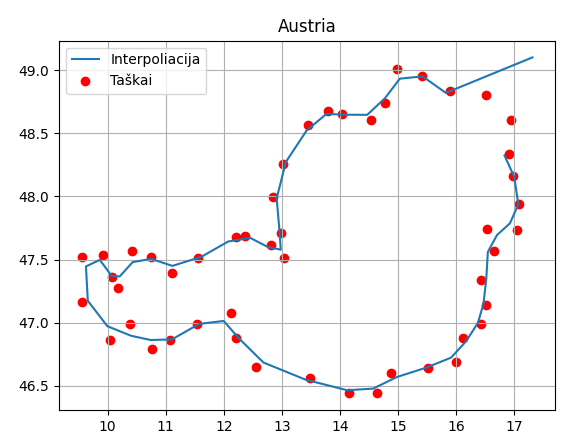
**Rezultatai:**



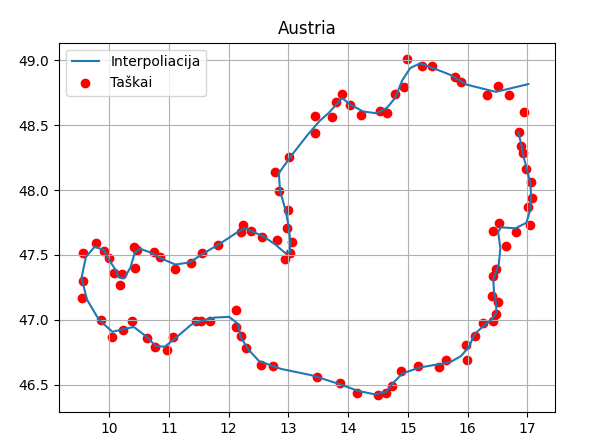
pav. 8 Globalaus splaino interpoliacija su 10 taškų



pav. 9 Globalaus splaino interpoliacija su 20 taškų



pav. 10 Globalaus splaino interpoliacija su 50 taškų



pav. 11 Globalaus splaino interpoliacija su 100 taškų

## Trečios užduoties išvados

Užduotį pavyko įgyvendinti dalinai. Preliminarus šalies sienų grafikas programą nupiešia, tačiau kadangi yra naudojamas parametrinio splaino metodas metodas tikimasis grafikus turėtų būti glotnus, taipogi kreivės galai turėtų būti nukreipti vienas į kitą. Tačiau mūsų matome rezultate funkcijos galai yra nukreipti į skirtingas puses ir tarp taškų splainas turi smailų kampa.

# Ketvirta užduotis

**IV užduotis. Aproksimavimas**

Pagal 2 lentelėje nurodytą šalį ir metus mažiausių kvadratų metodu sudarykite aproksimuojančią kreivę 12 mėnesių vidutinėms temperatūroms atvaizduoti naudojant antros, trečios, ketvirtos ir penktos eilės daugianarius. Pateikite gautas daugianarių išraiškas.



pav. 12 Ketvirtos užduoties variantas

**Programos kodas:**

import numpy as np

from matplotlib import pyplot as plt

import calendar

# finds base function matrix

def calculate\_base(X, m):

    n = len(X)

    T = np.zeros([n, m])

    for i in range(n):

        for j in range(m):

            T[i, j] = X[i] \*\* j

    return T

# G^T\*G . G^T\*y

def calculate\_coefficients(G, y):

    a = np.matmul(np.transpose(G), G)

    b = np.matmul(np.transpose(G), y)

    c = np.linalg.solve(a, b)

    return c

if \_\_name\_\_ == "\_\_main\_\_":

    # order of polynomial approximation

    m = 3

    X = np.arange(1, 13)

    # temperatures

    Y = np.array([

        0.12371,

        1.3831,

        5.29799,

        8.2207,

        10.1892,

        14.8951,

        16.617,

        14.5986,

        12.6457,

        9.69687,

        5.61371,

        0.02563

    ])

    # get base function matrix

    baz = calculate\_base(X, m)

    # calculate base coefficients

    c = calculate\_coefficients(baz, Y)

    nn = 50 # render points

    # spline x range

    xx = np.linspace(1, 12, nn)

    # set x range for Gc

    Gc = calculate\_base(xx, m)

    # calculate Gc

    fff = np.matmul(Gc, c)

    print(c)

    plt.title('%dth order polynomial' % m)

    plt.plot(xx, fff, label='f(x) temp. aproximated')

    plt.scatter(X, Y, label='month temp. points')

    plt.xlabel("month")

    plt.xticks(X, calendar.month\_name[1:13])

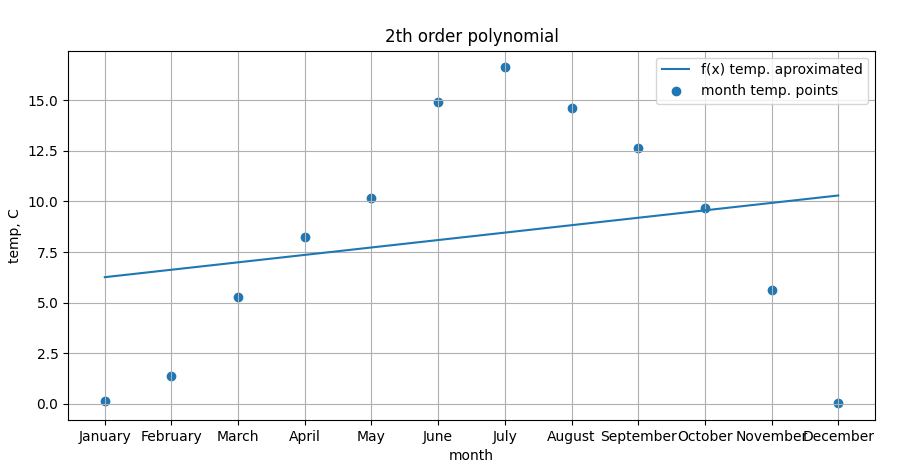
    plt.ylabel('temp, C')

    plt.legend()

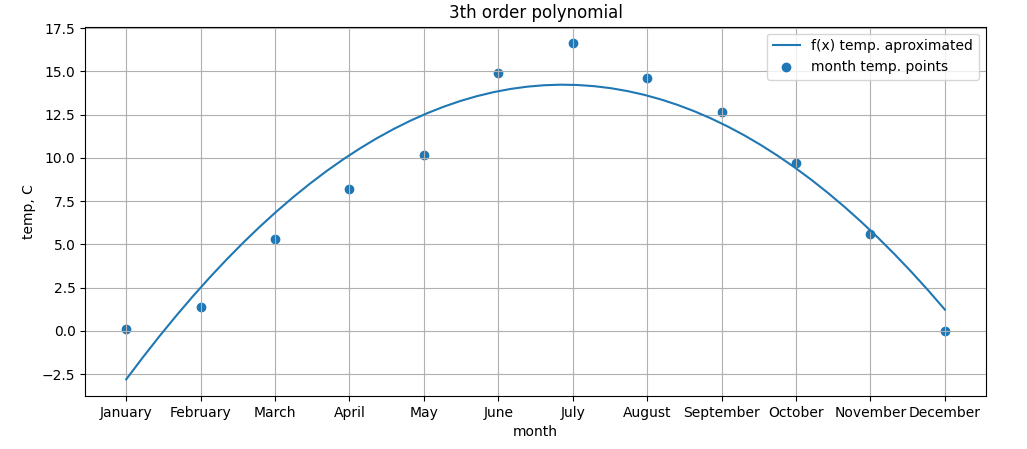
    plt.grid(1)

    plt.show()

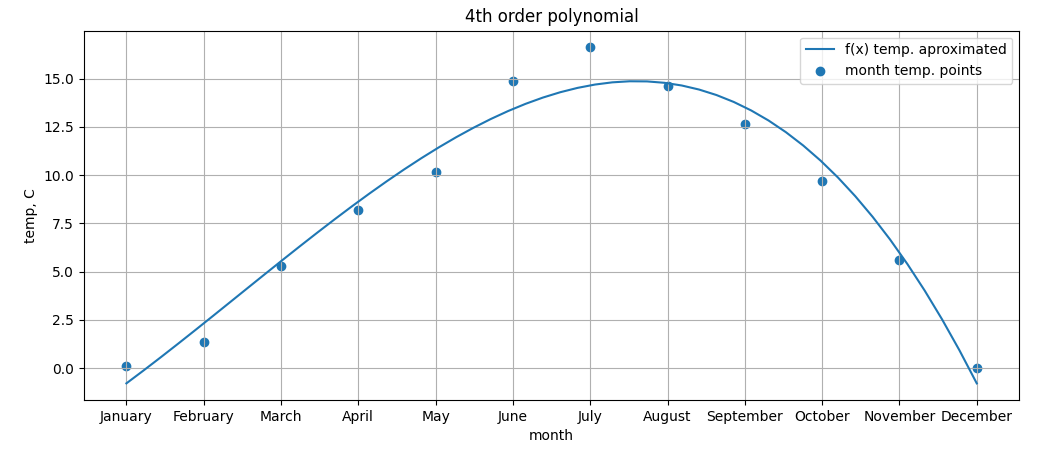
**Rezultatai:**



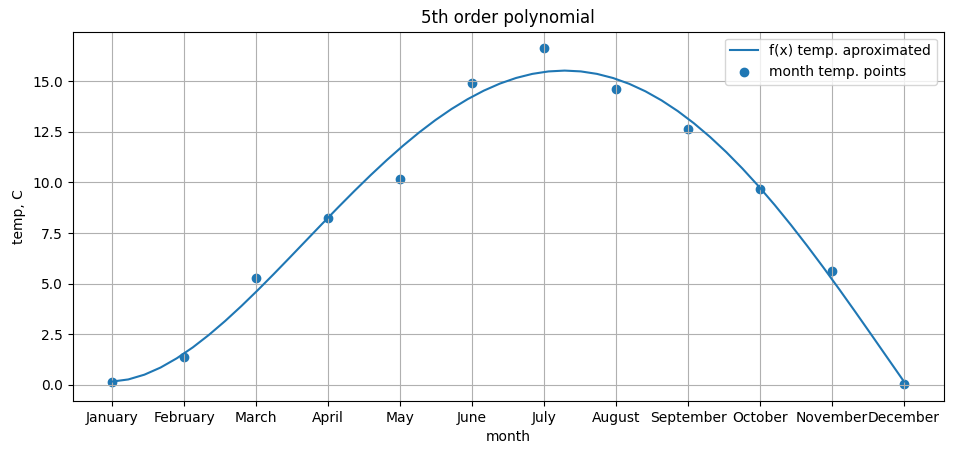
pav. 13 Antros eilės daugianario aproksimavimas



pav. 14 Trečios eilės daugianario aproksimavimas



pav. 15 Ketvirtos eilės daugianario aproksimavimas



pav. 16 Penktos eilės daugianario aproksimavimas

## Ketvirtos užduoties išvados

Pagal gautus rezultatus matome, kad didinant daugianario laipsnį aproksimuojanti funkcija vis tiksliau išsidėsto pateiktų taškų aibėje, tačiau kadangi aproksimaciją bando neatitikti tikslios taškų pasiskirstymo funkcijos, o bando rezultatą gauti kuo artimesnį jos, matomas vaizdas skiriasi nuo vaizdo, kurį matėme antroje užduotyje.