**KAUNO TECHNOLOGIJOS UNIVERSITETAS**
**INFORMATIKOS FAKULTETAS**

# Modulio P175B015 „Skaitiniai metodai ir algoritmai “

Laboratorinio darbo ataskaita

**Trečias laboratorinis darbas**

**Dėstytojas**
Doc. Andrius Kriščiūnas
Doc. Dalia Čalnerytė

**Studentas**
Rokas Puzonas IF-1/1

**KAUNAS, 2023**

# Turinys

# 1. I Užduotis

**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 interpoliuojančiosfunkcijos apskaičiuojamos naudojant skirtingas abscises ir gautas interpoliuojančių funkcijų išraiškas. Tame pačiame grafike vaizduokite duotąją funkciją, interpoliuojančią funkciją ir netiktį.

1 Lentelė:

| Var. Nr. | Funkcijos išraiška | Bazinė funkcija |
|---|---|---|
| 10 | $\cos(2x)\,(\sin(3x)+1.5)-cos\frac{x}{5}; -2 \le x \le 3$ | Vienanarių |

Vienanarių interpoliavimo metodu sprendžiama sukuriant N lygčių kurios pasakys kuriuose taškuose kreivė turės praeiti. Naudoti taškais lygtyse yra vadinami mazgai, nes jie apriboja kreivės formą. Visos sudarytos lygtys apjungiamos į lygčių sistemą ir išsprendžiama. Turint lygčių sistemą su N lygčių ir N nežinomųjų, ją galima išspresti naudoti įvarius metodus, aš naudosiu atspindžio metodą.

Python funkcija kuri išsprendžia lygčių sistemą atspindžio metodu:

```
def gauso_atispindzio_metodas(lygciu_sistema: LygciuSistema, epsilon=1e-7):
    koeficientai = list(lygtis.koeficientai for lygtis in lygciu_sistema.lygtys)
    A = np.matrix(koeficientai).astype(float)
    rezultatai = list(lygtis.rezultatas for lygtis in lygciu_sistema.lygtys)
    b = np.matrix(rezultatai).astype(float)

    n = (np.shape(A))[0]
    nb = (np.shape(b))[1]
    A1 = np.hstack((A, b))

    # tiesioginis etapas(atspindziai):
    for i in range(0, n - 1):
        z = A1[i:n, i]
        zp = np.zeros(np.shape(z))
        zp[0] = np.linalg.norm(z)
        omega = z - zp
        omega = omega / np.linalg.norm(omega)
        Q = np.identity(n - i) - 2 * omega * omega.transpose()
        A1[i:n, :] = Q.dot(A1[i:n, :])

    if np.sum(np.abs(A1[n-1, 0:n-nb+1])) < epsilon:
        return None

    # atgalinis etapas:
    x = np.zeros(shape=(n, 1))
    for i in range(n - 1, -1, -1):
        x[i, :] = (A1[i, n:n + nb] - A1[i, i + 1:n] * x[i + 1:n, :]) / A1[i, i]

    return list(x.flat)
```

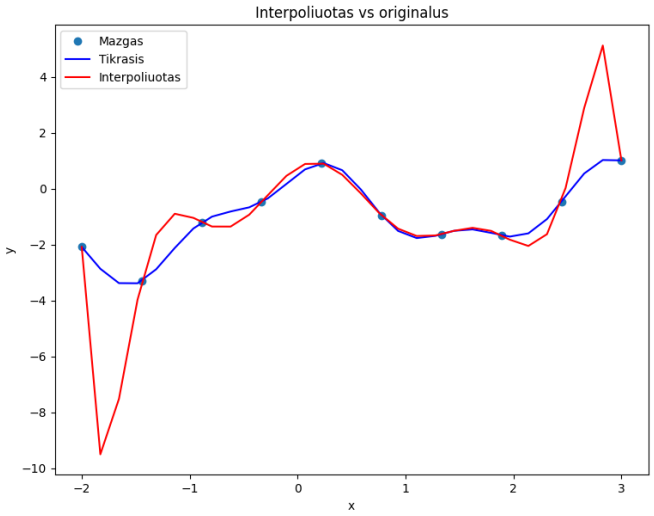
### 1.1. Tolygus mazgų paskirstymas

Sprendžiant vienanarių metodu labai svarbu kokie mazgai yra pasirenkami. Paprasčiausias metodas yra pasirinktis taškus tolygiai nutolią nutolę nuo vienas kito kažkokiame intervale.

Python kodas kuris apskaičiuoja vienanarius naudojant tolygų mazgų paskirstymą:

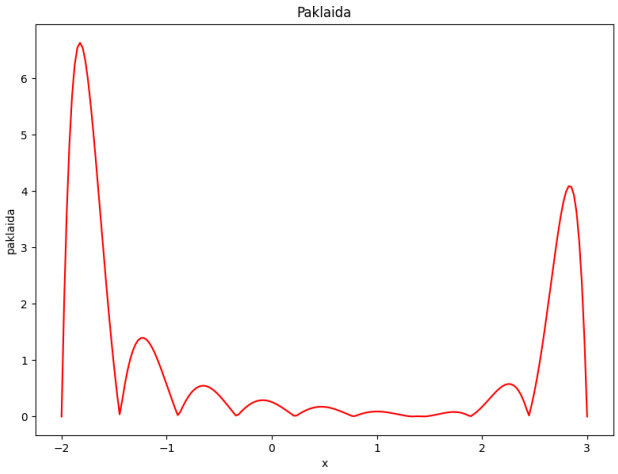
```python
def lerp(x, min_x, max_x):
    return min_x + x * (max_x - min_x)

def gauti_xy_taskus(F, from_x, to_x, density) -> tuple[list[float], list[float]]:
    x_s = gauti_x_taskus(from_x, to_x, density)
    y_s = []
    for x in x_s:
        y_s.append(F(x))
    return (x_s, y_s)

def gauti_x_taskus(from_x, to_x, density) -> list[float]:
    x_s = []
    for i in range(density):
        x_s.append(lerp(float(i) / (density-1), from_x, to_x))

    return x_s

def gauti_vienanario_sprendinius(x_mazgai: list[float], y_mazgai: list[float]) -> list[float]:
    lygtys = []

    for i, y in enumerate(y_mazgai):
        koeficientai = []
        x = x_mazgai[i]
        for j in range(len(x_mazgai)):
            koeficientai.append(x ** j)
        lygtys.append(Lygtis(koeficientai, [y]))

    return gauso_atispindzio_metodas(LygciuSistema(lygtys))

F = lambda x: cos(2*x) * (sin(3*x) + 1.5) - cos(x/5)
from_x = -2
to_x = 3
mazgu_kiekis = 10
tarpiniai_taskai = 30

mazgai = gauti_xy_taskus(F, from_x, to_x, mazgu_kiekis)
vienanariai = gauti_vienanario_sprendinius(mazgai[0], mazgai[1])
```

Kaip matome iš grafiko, pavyko sukurti kreivę kuri praeina visus mazgus, bet taip galima matyti kad prie intervalo galų kreivė pradeda stipriaia diverguoti. Šita problema dingsta jeigu turime daugiau mazgų kuo maženesniame intervale. Jeigu norime, kad mazgų skaičius išliktų toks pats, bet kreivė būtų artesnė orignalios, reikia naudoti Čiobyševo abscises.




*Grafikas 1. Lygaus mazgų paskirstymo grafikas*

Paklaidos suma grafike yra: 181.638




*Grafikas 2. Lygaus mazgų paskirstymo paklaida*

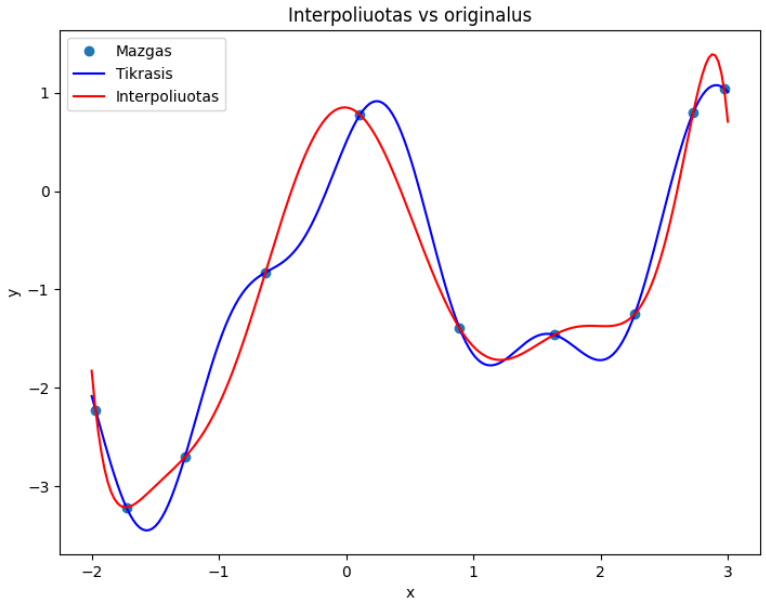
## 1.2. Mazgų paskirstymas pagal Čiobyševo abscises

Naudojant Čiobyševo abscises mazgai yra pasikristomi taip, kad daugiau jų bus intervalo galuose negu vidurį. Toks paskirstymas padeda sumažinti divergavimą sutelkiant daugiau taškų, ten kur diverguoja daugiau.

Python kodas kuris apskaičiuoja vienanarius naudojant Čiobyševo abscises:

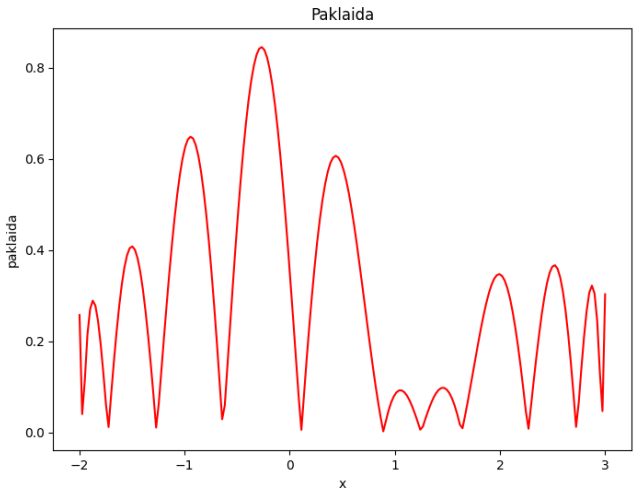
```
def gauti_ciobysevo_abscises(n: int) -> list[float]:
    x_s = []
    for i in range(n):
        x = cos(pi*(2*i + 1) / (2*n))
        x_s.append(x)
    return x_s

def konvertuoti_is_ciobysevo(abscises: list[float], a: float, b: float) -> list[float]:
    X_s = []
    for x in abscises:
        X = (a+b)/2 + (b-a)/2*x
        X_s.append(X)
    return X_s

F = lambda x: cos(2*x) * (sin(3*x) + 1.5) - cos(x/5)
mazgu_kiekis = 10
from_x = -2
to_x = 3

ciobysevo_abscises = gauti_ciobysevo_abscises(mazgu_kiekis)
x_mazgai = konvertuoti_is_ciobysevo(ciobysevo_abscises, to_x, from_x)
y_mazgai = list(F(x) for x in x_mazgai)
vienanariai = gauti_vienanario_sprendinius(x_mazgai, y_mazgai)
```

Iš grafiko galima matyti, kad interpoliuota funkcija yra arčiau originalios visame intervale lyginant su tolydžiu mazgų pasikirstymu.




*Grafikas 3. Mazgų paskirstymas pagal Čiobyševo abscises.*

Paklaidos suma grafike yra: 58.251
Lyginant paklaidą su tolygiu paskirstymu jinai yra 3 kartus mažėsnė, bet šiuo metodu didžiausia paklaida yra vidurį intervalo.




*Grafikas 4. Paklaida iš mazgų pagal Čiobyševo abscises.*

## 1.3. Python programinis kodas

```
from math import cos, sin, pi
import matplotlib.pyplot as plt
from dataclasses import dataclass
import numpy as np

@dataclass
class Lygtis:
    koeficientai: list[float]
    rezultatas: list[float]

@dataclass
class LygciuSistema:
    lygtys: list[Lygtis]

def lerp(x, min_x, max_x):
    return min_x + x * (max_x - min_x)

def gauti_sklaida(lygciu_sistema: LygciuSistema, x, rezultato_idx=0):
    n = lygciu_sistema.lygtys[0].koeficientai
    sklaida = 0
    for lygtis in lygciu_sistema.lygtys:
        issistatyta = sum(x[i]*lygtis.koeficientai[i] for i in range(len(n)))
        sklaida += abs(issistatyta - lygtis.rezultatas[rezultato_idx])
    return sklaida

def gauso_atispindzio_metodas(lygciu_sistema: LygciuSistema, epsilon=1e-7):
    koeficientai = list(lygtis.koeficientai for lygtis in lygciu_sistema.lygtys)
    A = np.matrix(koeficientai).astype(float)
    rezultatai = list(lygtis.rezultatas for lygtis in lygciu_sistema.lygtys)
    b = np.matrix(rezultatai).astype(float)

    n = (np.shape(A))[0]
    nb = (np.shape(b))[1]
    A1 = np.hstack((A, b))

    # tiesioginis etapas(atspindziai):
    for i in range(0, n - 1):
```

```
            z = A1[i:n, i]
            zp = np.zeros(np.shape(z))
            zp[0] = np.linalg.norm(z)
            omega = z - zp
            omega = omega / np.linalg.norm(omega)
            Q = np.identity(n - i) - 2 * omega * omega.transpose()
            A1[i:n, :] = Q.dot(A1[i:n, :])

        if np.sum(np.abs(A1[n-1, 0:n-nb+1])) < epsilon:
            return None

        # atgalinis etapas:
        x = np.zeros(shape=(n, 1))
        for i in range(n - 1, -1, -1):
            x[i, :] = (A1[i, n:n + nb] - A1[i, i + 1:n] * x[i + 1:n, :]) / A1[i, i]

        return list(x.flat)

    def gauti_x_taskus(from_x, to_x, density) -> list[float]:
        x_s = []
        for i in range(density):
            x_s.append(lerp(float(i) / (density-1), from_x, to_x))

        return x_s

    def gauti_xy_taskus(F, from_x, to_x, density) -> tuple[list[float], list[float]]:
        x_s = gauti_x_taskus(from_x, to_x, density)
        y_s = []
        for x in x_s:
            y_s.append(F(x))
        return (x_s, y_s)

    def plot_function(F, from_x, to_x, density, **kvargs):
        x_s, y_s = gauti_xy_taskus(F, from_x, to_x, density)

        plt.plot(x_s, y_s, **kvargs)

    def gauti_vienanario_sprendinius(x_mazgai: list[float], y_mazgai: list[float]) -> list[float]:
        lygtys = []

        for i, y in enumerate(y_mazgai):
            koeficientai = []
            x = x_mazgai[i]
            for j in range(len(x_mazgai)):
                koeficientai.append(x ** j)
            lygtys.append(Lygtis(koeficientai, [y]))

        return gauso_atispindzio_metodas(LygciuSistema(lygtys))

    def gauti_y_is_vienanariu(vienanariai: list[float], x: float):
        y = 0
        for i, a in enumerate(vienanariai):
            y += a * x ** i
        return y

    def gauti_ciobysevo_abscises(n: int) -> list[float]:
        x_s = []
        for i in range(n):
            x = cos(pi*(2*i + 1) / (2*n))
            x_s.append(x)
        return x_s

    def konvertuoti_is_ciobysevo(abscises: list[float], a: float, b: float) -> list[float]:
        X_s = []
        for x in abscises:
            X = (a+b)/2 + (b-a)/2*x
            X_s.append(X)
        return X_s

    def gauti_vienanariu_paklaidas(F, x_s: list[float], vienanariai):
        paklaidos = []
        for x in x_s:
            interpoliuotas_y = gauti_y_is_vienanariu(vienanariai, x)
            tikras_y = F(x)
            paklaidos.append(abs(interpoliuotas_y - tikras_y))
```

```
    return paklaidos

def plot_vienanariu_palyginima(F, x_s: list[float], vienanariai):
    plt.plot(x_s, list(F(x) for x in x_s), color="blue", label="Tikrasis")

    y_interpoliuotas = list(gauti_y_is_vienanariu(vienanariai, x) for x in x_s)
    plt.plot(x_s, y_interpoliuotas, color="red", label="Interpoliuotas")
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Interpoliuotas vs originalus")

def plot_vienanariu_paklaida(F, x_s: list[float], vienanariai):
    y_paklaida = gauti_vienanariu_paklaidas(F, x_s, vienanariai)
    plt.plot(x_s, y_paklaida, color="red")
    plt.xlabel("x")
    plt.ylabel("paklaida")
    plt.title("Paklaida")

def main(F, x_range, mazgu_kiekis, tarpiniai_taskai):
    from_x, to_x = x_range

    if False:
        mazgai = gauti_xy_taskus(F, from_x, to_x, mazgu_kiekis)
        vienanariai = gauti_vienanario_sprendinius(mazgai[0], mazgai[1])
        x_s = gauti_x_taskus(from_x, to_x, tarpiniai_taskai)

        #plt.plot(mazgai[0], mazgai[1], "o", label="Mazgas")
        #plot_vienanariu_palyginima(F, x_s, vienanariai)
        plot_vienanariu_paklaida(F, x_s, vienanariai)
        print("Paklaida:", sum(gauti_vienanariu_paklaidas(F, x_s, vienanariai)))

    if True:
        ciobysevo_abscises = gauti_ciobysevo_abscises(mazgu_kiekis)
        x_mazgai = konvertuoti_is_ciobysevo(ciobysevo_abscises, to_x, from_x)
        y_mazgai = list(F(x) for x in x_mazgai)
        vienanariai = gauti_vienanario_sprendinius(x_mazgai, y_mazgai)

        x_s = gauti_x_taskus(from_x, to_x, tarpiniai_taskai)
        plt.plot(x_mazgai, y_mazgai, "o", label="Mazgas")
        plot_vienanariu_palyginima(F, x_s, vienanariai)
        #plot_vienanariu_paklaida(F, x_s, vienanariai)
        print("Paklaida:", sum(gauti_vienanariu_paklaidas(F, x_s, vienanariai)))

    plt.show()

main(
    F = lambda x: cos(2*x) * (sin(3*x) + 1.5) - cos(x/5),
    x_range = (-2, 3),
    mazgu_kiekis = 10,
    tarpiniai_taskai = 200
)
```

## 2. II Užduotis

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)).

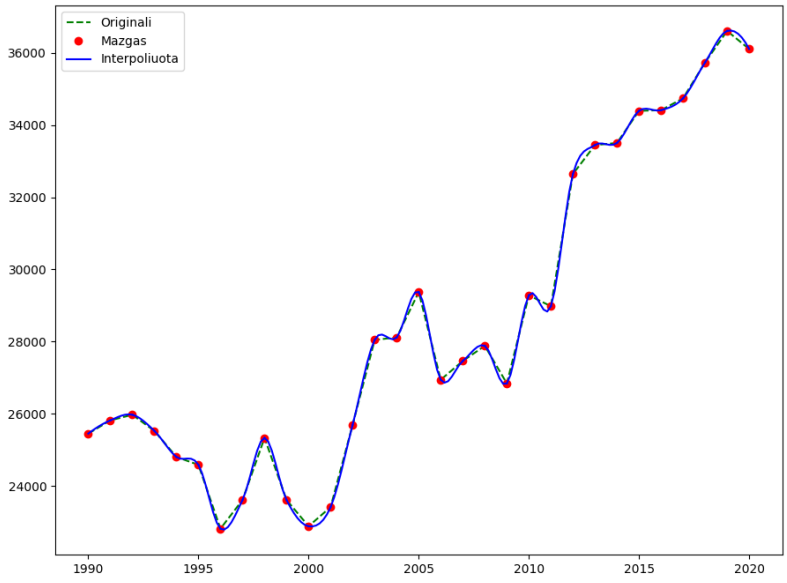
2 lentelė:

| Var. Nr. | Šalis | Splainas |
|---|---|---|

| | | |
|---|---|---|
| 10 | Zambija | Ermito (Akima) |

Ermito splainas kuria interpoliavimo kreivę tarp 2 mazgų, norint turėti kreivę per N mazgų reikia panaudoti N-1 atskirų ermito splainų. Norint apskaičiuoti erminto splainą būtina žinoti ne tik x,y kordinates, bet ir išvestinės reikšmę tame taške. Jeigu neturime išvestinių galima grubią jos reikšmią apskaičiuoti naudojant 3 šalia vienas kito esančius taškus, kad apskaičiuoti išvestinę. Toks splaino interpoliavimas naudojant grubias išvestines vadinamas Ermito (Akima) metodu.

## 2.1. Interpoliuotas splainas




*Grafikas 5. Emito (Akima) splainas*

## 2.2. Splaino Python programinis kodas

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

def lerp(x, min_x, max_x):
    return min_x + x * (max_x - min_x)

def remove_empty_years(emissions):
    empty_emissions = []
    for (year, emission) in emissions:
        if emission == None:
```

```
            empty_emissions.append((year, emission))

    for empty_entry in empty_emissions:
        emissions.remove(empty_entry)

def get_country_emissions(country: str):
    emissions_path = "API_EN.ATM.GHGT.KT.CE_DS2_en_csv_v2_5995567.csv"

    with open(emissions_path, "r", newline="", encoding="utf-8") as f:
        # Skip first 4 lines
        for _ in range(4):
            f.readline()

        reader = csv.reader(f, delimiter=",", quotechar='"')
        header_row = next(reader)
        years = list(int(header) for header in header_row[4:] if header != '')

        for row in reader:
            if row[0] == country:
                emissions_str = row[4:4+len(years)]
                emissions = list(float(emission) if emission != "" else None for emission in emissions_str)
                emission_points = list(zip(years, emissions))
                remove_empty_years(emission_points)

                years     = list(row[0] for row in emission_points)
                emissions = list(row[1] for row in emission_points)
                return years, emissions

def lagrange_dx_2d(
        x,
        x_prev, y_prev,
        x_curr, y_curr,
        x_next, y_next):
    return (((x - x_curr) + (x - x_next)) / ((x_prev - x_curr) * (x_prev - x_next))) * y_prev + \
           (((x - x_prev) + (x - x_next)) / ((x_curr - x_prev) * (x_curr - x_next))) * y_curr + \
           (((x - x_prev) + (x - x_curr)) / ((x_next - x_prev) * (x_next - x_curr))) * y_next

def akima_derivatives(Xs: list[float], Ys: list[float]):
    assert len(Xs) == len(Ys)

    result = []
    N = len(Xs)
    for i in range(N):
        pivot = i
        if pivot == 0:
            pivot += 1
        elif pivot == N-1:
            pivot -= 1

        x = Xs[i]

        x_prev = Xs[pivot-1]
        y_prev = Ys[pivot-1]
        x_curr = Xs[pivot  ]
        y_curr = Ys[pivot  ]
        x_next = Xs[pivot+1]
        y_next = Ys[pivot+1]

        result.append(lagrange_dx_2d(x, x_prev, y_prev, x_curr, y_curr, x_next, y_next))
    return result

def draw_hermite_curve(X: list[float], Y: list[float], dY: list[float], scalar = 1):
    assert len(X) == len(Y) == len(dY)
    N = len(X)

    plt.plot(X, Y, '--g', label="Originali")
    plt.plot(X[0], Y[0], 'ro', label="Mazgas")
    for i in range(N - 1):
        plot_x = np.linspace(X[i], X[i+1], scalar)
        plot_y = []

        plt.plot(X[i+1], Y[i+1], 'ro')
        d = X[i+1] - X[i]
        for k in range(scalar):
            s = plot_x[k] - X[i]
            U1 = 1 - 3 * (s**2 / d**2) + 2 * (s**3 / d**3)
            V1 = s - 2 * (s**2 / d) + (s**3 / d**2)
            U2 = 3 * (s ** 2 / d **2) - 2 * (s**3 / d ** 3)
            V2 = -1 * (s ** 2 / d) + (s ** 3 / d ** 2)

            f  = U1*Y[i  ] + V1*dY[i  ]
            f += U2*Y[i+1] + V2*dY[i+1]
            plot_y.append(f)
        plt.plot(plot_x, plot_y, 'b', label="Interpoliuota" if i == 0 else None)
        plt.draw()
    plt.legend()
    plt.show()
```

```
def main(country, interpolation):
    years, emissions = get_country_emissions(country)

    X = np.array(years)
    Y = emissions
    dY = akima_derivatives(years, emissions)

    draw_hermite_curve(X, Y, dY, 2 + interpolation)

main(
    country = "Zambia",
    interpolation = 5
)
```

# 3. III Užduotis

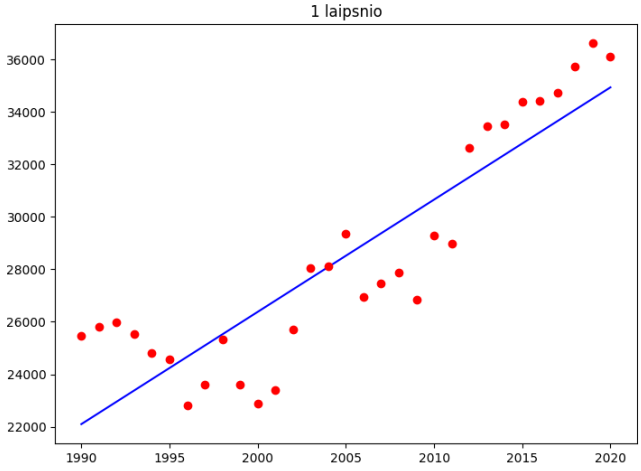
Mažiausių kvadratų metodu sudarykite **2 lentelėje** nurodytos šalies 1998-2018 metų šiltnamio dujų emisiją (galimo duomenų šaltinio nuoroda apačioje) aproksimuojančias kreives (**pirmos**, **antros**, **trečios** ir **penktos** eilės daugianarius). Pateikite gautas daugianarių išraiškas ir grafinius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

2 lentelė:

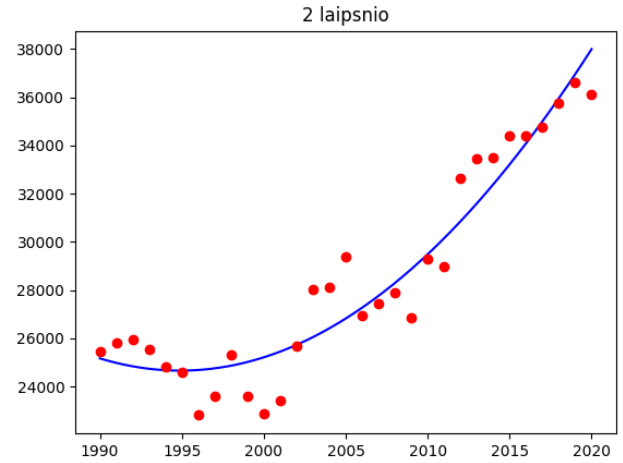
| Var. Nr. | Šalis | Splainas |
|---|---|---|
| 10 | Zambija | Ermito (Akima) |

Mažiausių kvadratų metodo esmė yra mažinimas kvadratinės paklaidos kiekviename duotame taške. Naudojant šį metodą galutinį dažniausiai nepraeis per visus taškus, bet bus panaši į formą kurią sudaro taškai. Toks metodas geriausiai tinka, kai taškuose yra triukšmo ir norima gauti kreivę per „vidurį“ visų taškų. Taikant šį metodą apskaičiuojama G matrica, ši matrica susideda iš taškų ir bazinių funkcijų. Kiekvienai bazinei funkcijai duodama kiekviena turima reikšmė. Šiuo atveju bazinės funkcijos yra n-tūjų eilių daugianariai. Toliau, naudojant išraišką $G^TGc = G^Ty$, išvedamas koeficientų vektorius c, kuris ir yra sprendinys užduoties.

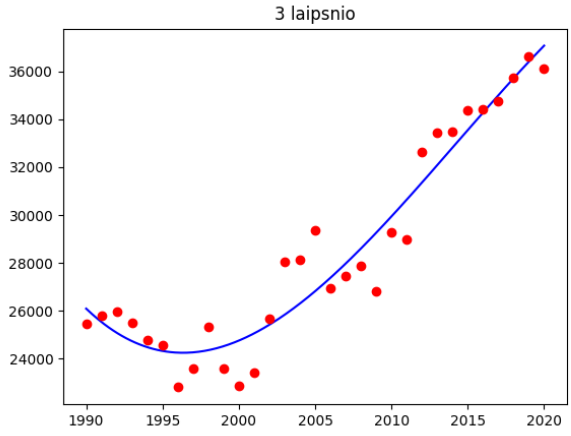
## 3.1. Aprosimuotos kreivės



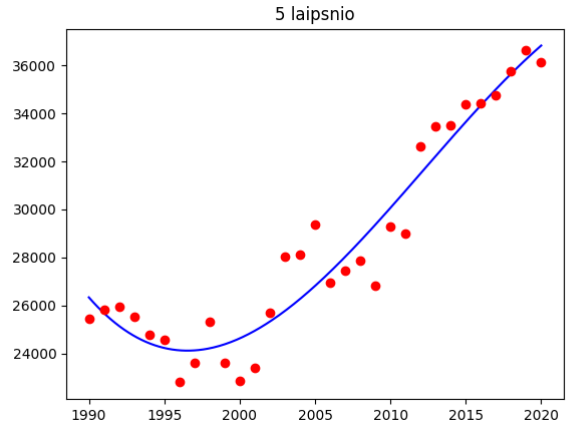

*Grafikas 6. Mažiausių kvadratų 1 eilės Zambijos kreivė*




*Grafikas 7. Mažiausių kvadratų 2 eilės Zambijos kreivė*




*Grafikas 8. Mažiausių kvadratų 3 eilės Zambijos kreivė*




*Grafikas 9. Mažiausių kvadratų 5 eilės Zambijos kreivė*

## 3.2. Python programinis kodas

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

def remove_empty_years(emissions):
    empty_emissions = []
    for (year, emission) in emissions:
        if emission == None:
            empty_emissions.append((year, emission))

    for empty_entry in empty_emissions:
        emissions.remove(empty_entry)

def get_country_emissions(country: str):
    emissions_path = "API_EN.ATM.GHGT.KT.CE_DS2_en_csv_v2_5995567.csv"

    with open(emissions_path, "r", newline="", encoding="utf-8") as f:
        # Skip first 4 lines
        for _ in range(4):
```

```
            f.readline()

        reader = csv.reader(f, delimiter=",", quotechar='"')
        header_row = next(reader)
        years = list(int(header) for header in header_row[4:] if header != '')

        for row in reader:
            if row[0] == country:
                emissions_str = row[4:4+len(years)]
                emissions = list(float(emission) if emission != "" else None for emission in emissions_str)
                emission_points = list(zip(years, emissions))
                remove_empty_years(emission_points)

                years     = list(row[0] for row in emission_points)
                emissions = list(row[1] for row in emission_points)
                return years, emissions

def approximate(X, Y, degree, scalar = 1):
    degree += 1

    G = np.zeros((len(X), degree), dtype=float)
    for i in range(degree):
        G[:, i] = np.power(X, i)

    coefficients = np.linalg.solve(
        np.dot(np.transpose(G), G),
        np.dot(np.transpose(G), Y)
    )

    approx_x = np.linspace(X[0], X[-1], len(X) * scalar);
    approx_y = np.zeros(approx_x.size, dtype=float)
    for i in range(degree):
        approx_y += np.power(approx_x, i) * coefficients[i];

    return approx_x, approx_y

def main(country, degrees, scale):
    years, emissions = get_country_emissions(country)

    X = np.array(years, dtype=float)
    Y = np.array(emissions)

    for degree in degrees:
        approx_x, approx_y = approximate(X, Y, degree, scale)
        plt.plot(approx_x, approx_y, 'b')

        for i in range(len(X)):
            plt.plot(X[i], Y[i], 'ro')

        plt.title(f"{degree} laipsnio")
        plt.show()

main(
    country = "Zambia",
    degrees = [1, 2, 3, 5],
    scale = 10
)
```

## 4. IV Užduotis

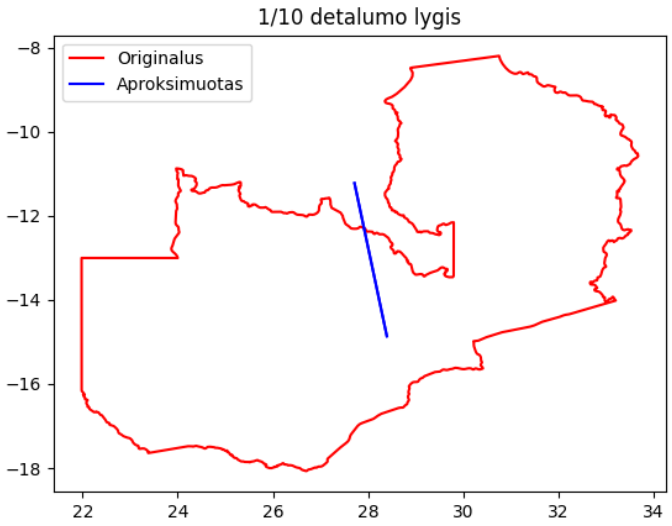
Naudodami parametrinį aproksimavimą Haro bangelėmis suformuokite **2 lentelėje** nurodytos šalies kontūrą. Analizuokite bent 10 detalumo lygių. Pateikite aproksimavimo rezultatus (aproksimuotą kontūro kreivę) ne mažiau kaip 4 skirtinguose lygmenyse. Jei šalis turi keletą atskirų teritorijų (pvz., salų),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

2 lentelė:

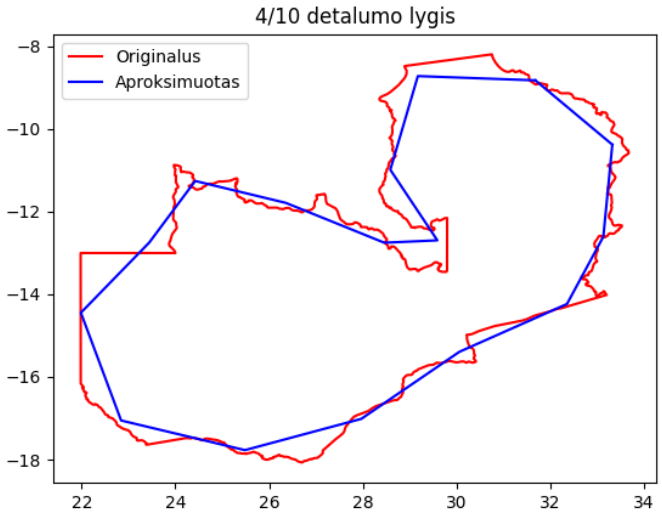
| Var. Nr. | Šalis | Splainas |
|---|---|---|
| 10 | Zambija | Ermito (Akima) |

Haro bangelių metodas naudoja masteliavimo ir transformacijos funkcijas kurios suteiks galimybę apibūdinti sudėtingos formos kontūrus kaip šalies kontūrą. Kad surasti geriausias koeficientų kombinacijas masteliavimo ir transformacijos funkcijoms naudojama mažiausių kvadratų metodas.Prieš taikant šį metodą reikia reikia pakeisti kontūrą į dvi parametris funkcijas x(t) ir y(t) kurios bus atskirai skaičiuojamos ir gale sujungiamos atgal, kad gauti kontūro aproksimaciją.

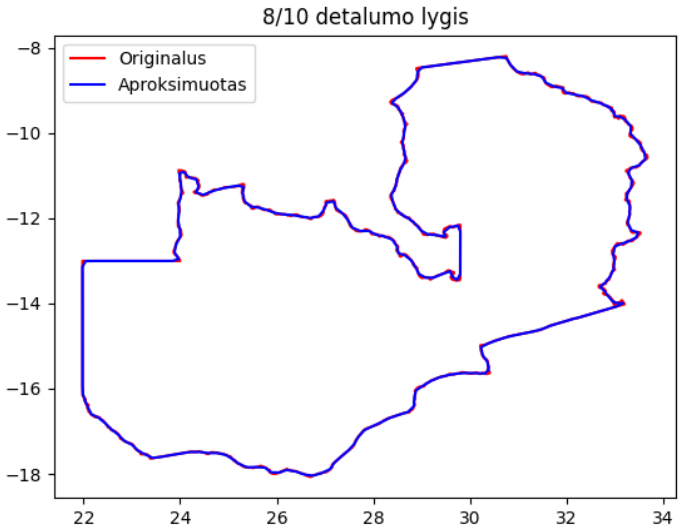
### 4.1. Detalumo lygių analizavimas



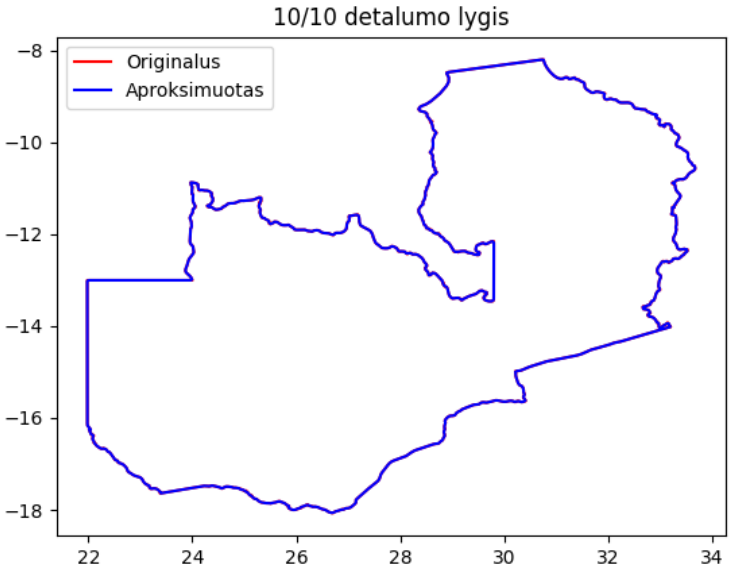

*Grafikas 10. Zambijos kontūro aproksimacija 1/10 detalumo lygis*




*Grafikas 11. Zambijos kontūro aproksimacija 4/10 detalumo lygis*

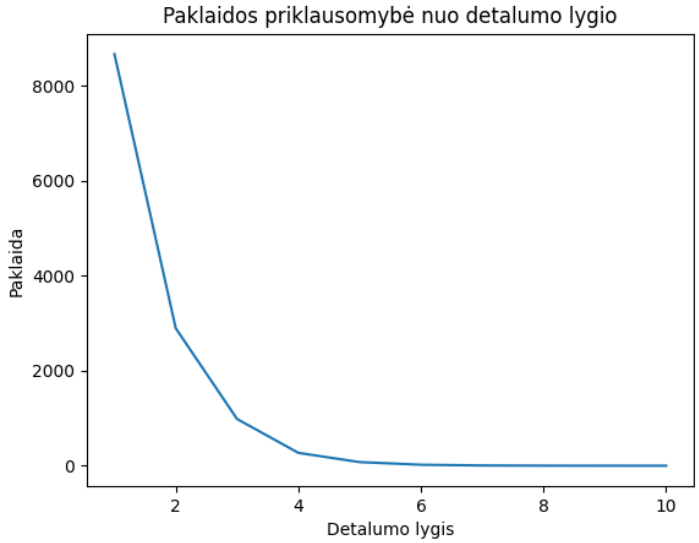



*Grafikas 12. Zambijos kontūro aproksimacija 8/10 detalumo lygis*




*Grafikas 13. Zambijos kontūro aproksimacija 10/10 detalumo lygis*

## 4.2. Paklaidos priklausomybė nuo detalumo lygio




## 4.3. Python programinis kodas

```
from typing import Optional
import shapefile
from shapely import geometry
```

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

shape = shapefile.Reader("ne_10m_admin_0_countries.shp")
shape_records = shape.shapeRecords()

def find_country_id(name: str) -> Optional[int]:
    for i in range(len(shape_records)):
        feature = shape_records[i]
        if feature.record.NAME_EN == name:
            return i

def get_country_points(country_id: int):
    feature = shape_records[country_id]
    geo_interface = feature.shape.__geo_interface__
    coordinates = geo_interface['coordinates']

    if geo_interface['type'] == 'MultiPolygon':
        largest_area_idx = 0
        area = 0
        for idx in range(len(coordinates)):
            points = coordinates[idx][0]
            polygon_area = geometry.Polygon(points).area
            if polygon_area > area:
                area = polygon_area
                largest_area_idx = idx

        return coordinates[largest_area_idx][0]
    else:
        return coordinates[0]

def approximate(t, s, NL, m):
    sqrt_2 = np.sqrt(2)
    a, b = min(t), max(t)
    smooth = (b - a) * s * 2 ** (-NL / 2)

    details = []
    for _ in range(m):
        smooth_evens = smooth[0::2]
        smooth_odds  = smooth[1::2]

        details.append((smooth_evens - smooth_odds) / sqrt_2)
        smooth = (smooth_evens + smooth_odds) / sqrt_2

    return smooth, details

def haro_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 haro_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 get_accuracy(new_x, new_y, original_x, original_y):
    x_err = 1/2*sum((new_x - original_x)**2)
    y_err = 1/2*sum((new_y - original_y)**2)
    return x_err + y_err

def main(country: str, levels: list[int], show_accuracy_plot):
    country_id = find_country_id(country)
    country_points = get_country_points(country_id)

    X, Y = [], []
    for (x, y) in country_points:
        X.append(x)
        Y.append(y)

    N = max(levels)
    nN = 2 ** N

    t = np.zeros(len(X))
    for i in range(1, len(X)):
        dx = X[i] - X[i-1]
        dy = Y[i] - Y[i-1]
        t[i] = t[i-1] + np.linalg.norm((dx, dy))

    t_min = t[0]
    t_max = t[-1]
    t_interp = np.linspace(t_min, t_max, nN)
    X_interp = np.interp(t_interp, t, X)
```

```
    Y_interp = np.interp(t_interp, t, Y)

    m = N

    smooth_x, det_x = approximate(t_interp, X_interp, N, m)
    smooth_y, det_y = approximate(t_interp, Y_interp, N, m)

    hx = np.zeros(nN)
    hy = np.zeros(nN)
    for k in range(2 ** (N - m)):
        scalar = haro_scaling(t_interp, N - m, k, t_min, t_max)
        hx += smooth_x[k] * scalar
        hy += smooth_y[k] * scalar

    accuracies = []
    for i in range(m):
        show_plot = (i+1 in levels)

        if show_plot: plt.plot(X, Y, 'r', label="Originalus")
        h1x = np.zeros(nN)
        h1y = np.zeros(nN)

        for k in range(2 ** (N - m + i)):
            wavelet = haro_wavelet(t_interp, N - m + i, k, t_min, t_max)
            h1x += det_x[m-i-1][k] * wavelet
            h1y += det_y[m-i-1][k] * wavelet

        hx += h1x
        hy += h1y

        accuracy = get_accuracy(hx, hy, X_interp, Y_interp)
        accuracies.append(accuracy)

        if show_plot:
            plt.plot(hx, hy, 'b')
            plt.plot((hx[0], hx[-1]), (hy[0], hy[-1]), 'b', label="Aproksimuotas")
            plt.title(f"{i+1}/{N} detalumo lygis")
            plt.legend()
            plt.show()

    if show_accuracy_plot:
        plt.plot(range(1,m+1), accuracies)
        plt.xlabel("Detalumo lygis")
        plt.ylabel("Paklaida")
        plt.title("Paklaidos priklausomybė nuo detalumo lygio")
        plt.show()

main(
    country = "Zambia",
    levels = [1, 4, 8, 10],
    show_accuracy_plot = False,
)
```