Kauno technologijos universitetas

Informatikos fakultetas

III projektinio darbo ataskaita

Laboratorinio užduotis

Aistis Jakutonis

Studentas

Čalnerytė Dalia
Kriščiūnas Andrius

Dėstytojai

Kaunas, 2025

Turinys

1. Pirma užduotis – Interpoliavimas daugianariu 3

1.1. Interpoliacinės funkcijos išraiškos radimas (taškai pasiskirstę tolygiai) 3

1.2. Interpoliacinės funkcijos išraiškos radimas (naudojamos Čiobyševo abcisės) 5

2. Antra užduotis – Interpoliavimas splainu 8

2.1. Kreivės interpoliavimas splainais 8

3. Trečia užduotis – Aproksimavimas 11

3.1. Aproksimuojančių kreivių sudarymas 11

4. Ketvirta užduotis – Parametrinis aproksimavimas Haro bangelėmis 13

4.1. Šalies kontūro formavimas naudojant parametrinį aproksimavimą Haro bangelėmis 13

1. Pirma užduotis – Interpoliavimas daugianariu

Interpoliavimas – tai būdas nubrėžti glotnią kreivę, kuri eina per duotus taškus.

Pagrindinė Lagranžo metodo idėja: galima sukurti daugianarį, kuris eina tiksliai per visus taškus, imant sumą iš specialių bazinių funkcijų $L_j(x)$.

Pagrindinė Čiobyševo idėja: interpoliavimo taškus išdėstyti ne tolygiai, o labiau kraštuose – tada daugianaris mažiau svyruos.

Metodas: Lagranžo

Funkcijos išraiška:

$$e^{(-x^2)} * sin(x^2) * (x-3); -3 \leq x \leq 2$$

1.1. Interpoliacinės funkcijos išraiškos radimas (taškai pasiskirstę tolygiai)

Naudotos formulės:

Tolygiai išdėstyti mazgai: $x_i = a + \frac{b-a}{n-1}\, i, \qquad i = 0,1, \dots, n-1$

Klasikinė Lagranžo interpoliacija: $P_n(x) = \sum_{i=0}^{n-1} y_i \; L_i(x)$

Interpoliacijos paklaida: $\varepsilon(x) = f(x) - P_n(x)$

Rezultatai:

```
INTERVALAS: [a, b] = [-3.0, 2.0]
Mazgų skaičius n = 15

Tolygiai išdėstyti mazgai (x_i):
 [-3.      , -2.642857, -2.285714, -1.928571, -1.571429, -1.214286,
 -0.857143, -0.5     , -0.142857,  0.214286,  0.571429,  0.928571,
  1.285714,  1.642857,  2.      ]
Atitinkamos reikšmės (y_i = f(x_i)):
 [-3.051568e-04, -3.372053e-03,  2.480345e-02,  6.527485e-02, -2.409341e-01,
 -9.601610e-01, -1.240225e+00, -6.743744e-01, -6.283987e-02, -1.221317e-01,
 -5.619778e-01, -6.640766e-01, -3.271121e-01, -3.910425e-02,  1.386132e-02]
```

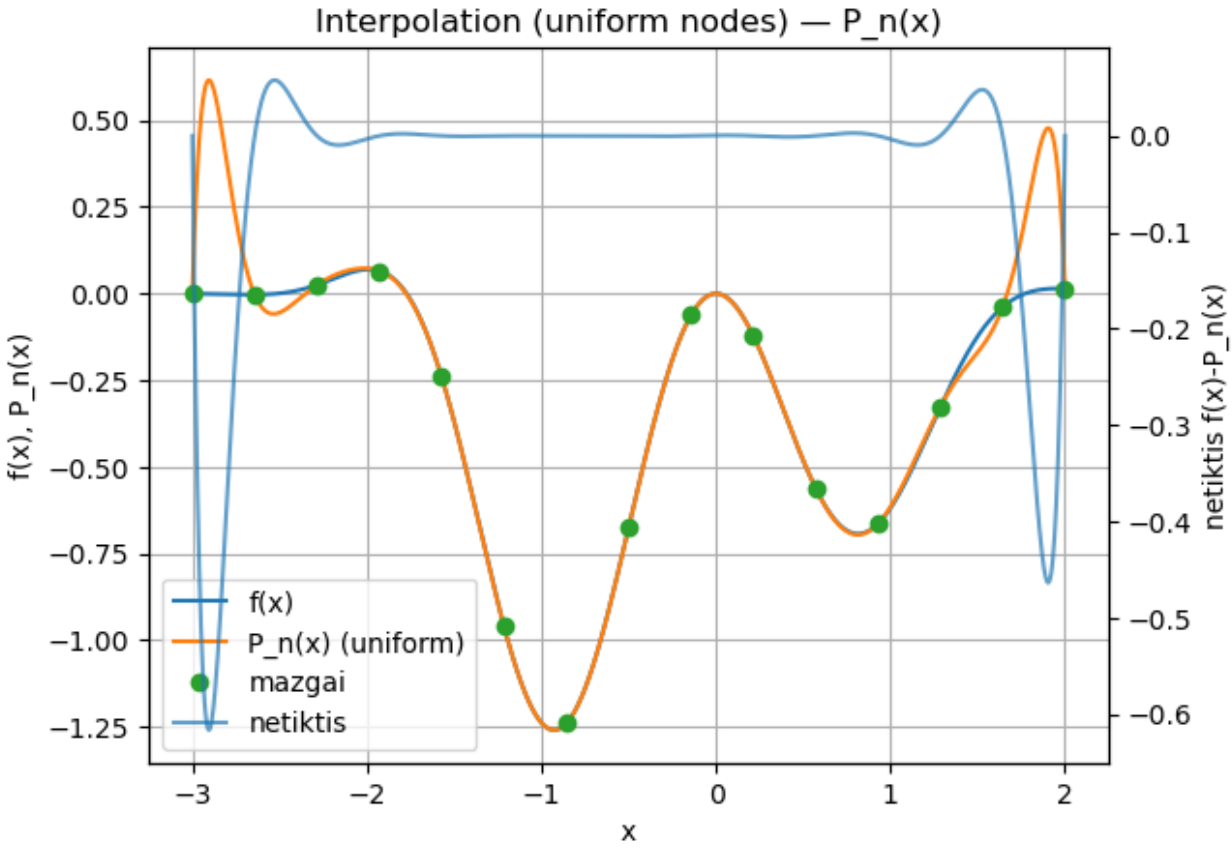
1 pav. Tolygiai išdėstyti mazgai

```
Maksimali absoliuti paklaida |f - P_uni|: 0.6164587561317897
```

2 pav. Paklaida

```
--- Interpolianto išraiška (Lagrange forma, tolygūs mazgai) ---
P(x) = Σ_{i=0}^{14} y_i · L_i(x), kur L_i(x) = ∏_{j≠i} (x − x_j)/(x_i − x_j)
Termas i=0: y_0= -0.000305157;  L_0(x) = (x − -2.64286) · (x − -2.28571) · (x − -1.92857) · (x − -1.571
43) · (x − -1.21429) · (x − -0.857143) · (x − -0.5) · (x − -0.142857) · (x − 0.214286) · (x − 0.571429)
· (x − 0.928571) · (x − 1.28571) · (x − 1.64286) · (x − 2) / (47884.6)
Termas i=1: y_1= -0.00337205;  L_1(x) = (x − -3) · (x − -2.28571) · (x − -1.92857) · (x − -1.57143) ·
(x − -1.21429) · (x − -0.857143) · (x − -0.5) · (x − -0.142857) · (x − 0.214286) · (x − 0.571429) · (x
− 0.928571) · (x − 1.28571) · (x − 1.64286) · (x − 2) / (-3420.33)
Termas i=2: y_2= 0.0248035;  L_2(x) = (x − -3) · (x − -2.64286) · (x − -1.92857) · (x − -1.57143) · (x
− -1.21429) · (x − -0.857143) · (x − -0.5) · (x − -0.142857) · (x − 0.214286) · (x − 0.571429) · (x −
0.928571) · (x − 1.28571) · (x − 1.64286) · (x − 2) / (526.204)
Termas i=3: y_3= 0.0652749;  L_3(x) = (x − -3) · (x − -2.64286) · (x − -2.28571) · (x − -1.57143) · (x
− -1.21429) · (x − -0.857143) · (x − -0.5) · (x − -0.142857) · (x − 0.214286) · (x − 0.571429) · (x −
0.928571) · (x − 1.28571) · (x − 1.64286) · (x − 2) / (-131.551)
Termas i=4: y_4= -0.240934;  L_4(x) = (x − -3) · (x − -2.64286) · (x − -2.28571) · (x − -1.92857) · (x
− -1.21429) · (x − -0.857143) · (x − -0.5) · (x − -0.142857) · (x − 0.214286) · (x − 0.571429) · (x −
0.928571) · (x − 1.28571) · (x − 1.64286) · (x − 2) / (47.8368)
... (rodyti tik pirmi 5 iš 15 narių)
```

3 pav. Interpoliavimo išraiška (tolygūs mazgai)



4 pav. Interpoliavimo išraiškos grafikas (tolygūs mazgai)

Tolygiai išdėstytų mazgų interpoliacijoje aiškiai matyti, kad daugianaris tiksliai praeina per visus duotus taškus, tačiau kraštuose atsiranda didesni svyravimai. Tai yra klasikinio Runge efekto pavyzdys: kuo aukštesnė interpolianto eilė ir kuo taškai toliau nuo centro, tuo labiau daugianaris linkęs „peršokti“ virš realios funkcijos. Dėl to paklaida kraštuose yra didžiausia, o viduryje – mažiausia. Tai rodo, kad tolygus mazgų išdėstymas nėra pats stabiliausias pasirinkimas interpoliacijai su didesniu mazgų skaičiumi.

1.2. Interpoliacinės funkcijos išraiškos radimas (naudojamos Čiobyševo abcisės)

Naudotos formulės:

Čiobyševo mazgai intervale [-1, 1]: $\xi_i = \cos\left(\frac{2i+1}{2n}\pi\right), \qquad i = 0,1,\dots,n-1$

Čiobyševo mazgų transformacija į reikiamą intervalą: $x_i = \frac{a+b}{2} + \frac{b-a}{2}\,\xi_i$

Klasikinė Lagranžo interpoliacija: $P_n(x) = \sum_{i=0}^{n-1} y_i\ L_i(x)$

Interpoliacijos paklaida: $\varepsilon(x) = f(x) - P_n(x)$

Rezultatai:

```
INTERVALAS: [a, b] = [-3.0, 2.0]
Mazgų skaičius n = 15
```

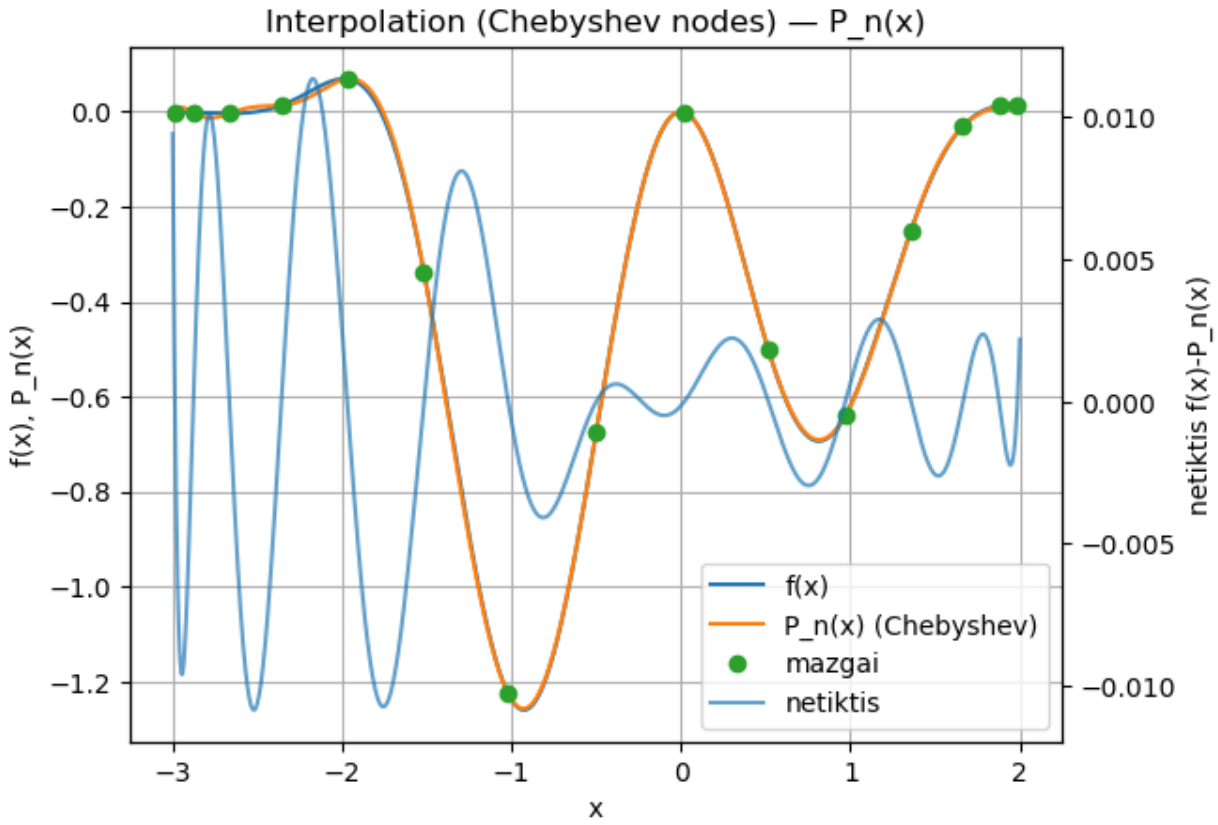
5 pav. Intervalas bei pasirinktas mazgų skaičius

```
Čiobyševo abscisės (perskaičiuotos į [a,b]) (x_i):
 [ 1.986305,  1.877641,  1.665064,  1.357862,  0.969463,  0.516842,
  0.019779, -0.5     , -1.019779, -1.516842, -1.969463, -2.357862,
 -2.665064, -2.877641, -2.986305]
Atitinkamos reikšmės (y_i = f(x_i)):
 [ 1.411811e-02,  1.237529e-02, -3.010978e-02, -2.501924e-01, -6.405667e-01,
 -5.018005e-01, -1.165459e-03, -6.743744e-01, -1.225338e+00, -3.371783e-01,
  6.906861e-02,  1.366079e-02, -3.406846e-03, -1.355388e-03, -3.891953e-04]
```

6 pav. Čiobyševo mazgai

```
--- Interpolanto išraiška (Lagrange forma, Čiobyševo mazgai) ---
P(x) = Σ_{i=0}^{14} y_i · L_i(x), kur L_i(x) = ∏_{j≠i} (x − x_j)/(x_i − x_j)
Termas i=0: y_0= 0.0141181;  L_0(x) = (x − 1.87764) · (x − 1.66506) · (x − 1.35786) · (x − 0.969463) ·
(x − 0.516842) · (x − 0.0197792) · (x − −0.5) · (x − −1.01978) · (x − −1.51684) · (x − −1.96946) · (x −
−2.35786) · (x − −2.66506) · (x − −2.87764) · (x − −2.9863) / (3262.85)
Termas i=1: y_1= 0.0123753;  L_1(x) = (x − 1.9863) · (x − 1.66506) · (x − 1.35786) · (x − 0.969463) ·
(x − 0.516842) · (x − 0.0197792) · (x − −0.5) · (x − −1.01978) · (x − −1.51684) · (x − −1.96946) · (x −
−2.35786) · (x − −2.66506) · (x − −2.87764) · (x − −2.9863) / (−1103.7)
Termas i=2: y_2= −0.0301098;  L_2(x) = (x − 1.9863) · (x − 1.87764) · (x − 1.35786) · (x − 0.969463) ·
(x − 0.516842) · (x − 0.0197792) · (x − −0.5) · (x − −1.01978) · (x − −1.51684) · (x − −1.96946) · (x −
−2.35786) · (x − −2.66506) · (x − −2.87764) · (x − −2.9863) / (682.121)
Termas i=3: y_3= −0.250192;  L_3(x) = (x − 1.9863) · (x − 1.87764) · (x − 1.66506) · (x − 0.969463) ·
(x − 0.516842) · (x − 0.0197792) · (x − −0.5) · (x − −1.01978) · (x − −1.51684) · (x − −1.96946) · (x −
−2.35786) · (x − −2.66506) · (x − −2.87764) · (x − −2.9863) / (−509.707)
Termas i=4: y_4= −0.640567;  L_4(x) = (x − 1.9863) · (x − 1.87764) · (x − 1.66506) · (x − 1.35786) · (x
− 0.516842) · (x − 0.0197792) · (x − −0.5) · (x − −1.01978) · (x − −1.51684) · (x − −1.96946) · (x − −
2.35786) · (x − −2.66506) · (x − −2.87764) · (x − −2.9863) / (421.574)
... (rodyti tik pirmi 5 iš 15 narių)
```

7 pav. Interpoliavimo išraiška (Čiobyševo mazgai)



8 pav. Interpoliavimo išraiškos grafikas (Čiobyševo mazgai)

Naudojant Čiobyševo mazgus, interpoliantas tampa kur kas stabilesnis: svyravimai kraštuose sumažėja, o maksimali paklaida tampa žymiai mažesnė nei tolygiai išdėstytų mazgų atveju. Taip yra todėl, kad Čiobyševo taškai tankiau išdėstomi intervalo galuose. Gauti rezultatai patvirtina teoriją – Čiobyševo mazgai leidžia pasiekti daug geresnį interpoliacijos tikslumą, ypač esant sudėtingoms funkcijoms.

Pagrindinės sprendimo dalies programos kodas:

```
def chebyshev_nodes_on_interval(a, b, n):
    i = np.arange(n)
    xi = np.cos((2*i + 1) * math.pi / (2*n))    # Čiobyševo mazgai intervale [-1,1]
    return (a + b)/2 + (b - a)/2 * xi           # linijinis pernešimas į [a,b]

def uniform_nodes(a, b, n):
    """Sugeneruoja n tolygiai išdėstytų mazgų intervale [a,b]."""
    return np.linspace(a, b, n)

def lagrange_classic(x_nodes, y_nodes, x_eval):
    x_nodes = np.asarray(x_nodes, dtype=float)
    y_nodes = np.asarray(y_nodes, dtype=float)
    x_eval  = np.atleast_1d(x_eval).astype(float)

    n = len(x_nodes)
    P = np.zeros_like(x_eval)
```

```
    # Einame per visus bazinius daugianarius L_i(x)
    for i in range(n):
        Li = np.ones_like(x_eval)
        xi = x_nodes[i]
        for j in range(n):          # skaičiuojame sandaugą per visus j != i
            if j == i:
                continue
            Li *= (x_eval - x_nodes[j]) / (xi - x_nodes[j])  # daugindami
sudarom L_i(x)
        P += y_nodes[i] * Li        # P(x) += y_i * L_i(x)
    return P if P.ndim > 0 else P.item()
```

2. Antra užduotis – Interpoliavimas splainu

Splainas – glotni kreivė, sudaryta iš kelių daugianarių gabaliukų, sujungtų taip, kad funkcijos reikšmės sutaptų sandūrose.

Pagrindinė globalaus splaino idėja: visus intervalus sieja viena bendra sąlygų sistema. Splaino forma priklauso nuo visų taškų kartu, pakeitus vieną mazgą, kreivė pasikeičia visame intervale.

Metodas: Globalus

Data, AMS:

2024-10-15; Klaipėdos AMS

2.1. Kreivės interpoliavimas splainais

Naudotos formulės:

Duomenų taškai: $x_i = i, \qquad i = 0,1, \dots, 23$

Tarpai tarp mazgų: $d_i = x_{i+1} - x_i, \qquad i = 0,1, \dots, n-1$

Antrosios išvestinės mazguose: $m_i = S''(x_i), \qquad i = 0,1, \dots, n$

Tridiagonalė lygtis vidiniams mazgams: $d_{i-1}\, m_{i-1} + 2(d_{i-1} + d_i)\, m_i + d_i\, m_{i+1} = 6\left(\frac{y_{i+1}-y_i}{d_i} - \frac{y_i-y_{i-1}}{d_{i-1}}\right)$

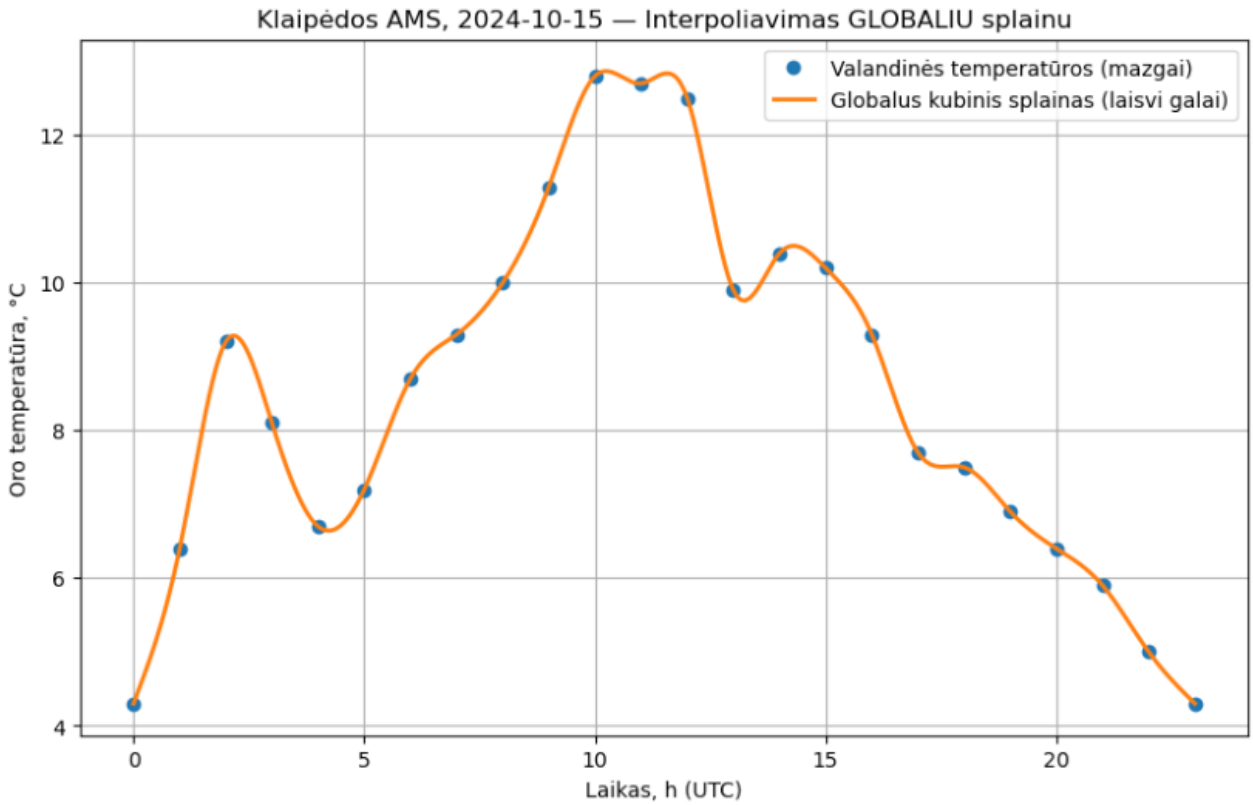
Kraštinės sąlygos (laisvi galai): $m_0 = 0, \qquad m_n = 0$

Kubinio splaino išraiška intervale $[x_i, x_{(i+1)}]$: $s = x - x_i, \qquad d_i = x_{i+1} - x_i.$

$$S_i(x) = y_i + \left(\frac{y_{i+1} - y_i}{d_i} - \frac{(2m_i + m_{i1})\, d_i}{6}\right) s + \frac{m_i}{2}\, s^2 + \frac{m_{i+1} m_i}{6 d_i}\, s^3$$

Bendras splaino apibrėžimas: $S(x) = S_i(x), \qquad x \in [x_i, x_{i+1}], \quad i = 0,1, \dots, n-1$

Rezultatai:



9 pav. Interpoliacijos globaliu splainu grafikas (Klaipėdos AMS)

Globalus kubinis splainas leido sudaryti glotnią kreivę, kuri tiksliai jungia visus matavimo taškus ir tuo pačiu išlieka lygi bei natūrali. Splainas skirtingai nei aukštos eilės daugianariai nesvyruoja ir neturi Runge efekto, nes kiekvienas jo gabalas priklauso tik nuo lokalių taškų. Laisvų galų sąlyga $m_0 = m_n = 0$ dar labiau sumažina kraštinius iškreipimus. Gautas grafikas realistiškai perteikia temperatūrų kitimą per parą – kreivė lygi, fiziškai prasminga ir tiksliai atitinka duomenų tendencijas.

Pagrindinės sprendimo dalies programos kodas:

```
# ----------------------------------------
# 2) Skaičiuojame globalų kubinį splainą
# ----------------------------------------
# intervalų skaičius
n = len(x) - 1

# d_i = atstumai tarp mazgų
d = np.diff(x)

# Matricą A ir dešine puse rhs vidinėms lygtims
A = np.zeros((n-1, n-1))
rhs = np.zeros(n-1)

# Formuojame tridiagonalę matricą pagal splaino lygtis
for i in range(1, n):
    im = i-1
    A[im, im] += 2*(d[i-1] + d[i])
```

```
    if i-2 >= 0:
        A[im, im-1] = d[i-1]

    if im+1 <= n-2:
        A[im, im+1] = d[i]

    # dešinioji pusė pagal formulę
    rhs[im] = 6 * ((y[i+1]-y[i]) / d[i] - (y[i]-y[i-1]) / d[i-1])

# antrosios išvestines mazguose
m = np.zeros(n+1)

# Sprendžiame sistemą tik vidiniams m_i
if n-1 > 0:
    m[1:n] = np.linalg.solve(A, rhs)

# ------------------------------------------
# 3) Splaino vertinimas bet kuriame taške xx
# ------------------------------------------
def eval_cubic_spline(xq):
    xq = np.atleast_1d(xq)
    yq = np.empty_like(xq, dtype=float)

    # Randame, į kurį intervalą patenka xq
    idx = np.clip(np.searchsorted(x, xq) - 1, 0, n-1)

    # s = atstumas nuo x_i iki xq
    s = xq - x[idx]

    di = d[idx]

    # Splaino formulės nariai
    term1 = y[idx]
    term2 = ((y[idx+1]-y[idx]) / di - (2*m[idx] + m[idx+1]) * di / 6.0) * s
    term3 = (m[idx] / 2.0) * s**2
    term4 = ((m[idx+1] - m[idx]) / (6.0 * di)) * s**3

    # Galutinė splaino reikšmė
    yq[:] = term1 + term2 + term3 + term4
    return yq
```

3. Trečia užduotis – Aproksimavimas

Aproksimavimas – kreivės, kuri kuo geriau atitinka taškų tendenciją, radimas. Funkcijos radimas, kurios reikšmės kuo mažiau nutolsta nuo duotųjų taškų reikšmių.

Pagrindinė mažiausių kvadratų metodo idėja: rasti tokią funkciją, kuri būtų arčiausiai visų duotų taškų, mažinant bendrą kvadratinių nuokrypių sumą. Tikslas - bendra paklaidų suma turi būti kuo mažesnė.

Metodas: Mažiausių kvadratų

Data, AMS:

2024-10-15; Klaipėdos AMS

3.1. Aproksimuojančių kreivių sudarymas

Naudotos formulės:

Duomenų taškai: $x_i = i, \qquad i = 0,1,\dots,23, \qquad N = 24$

Aproksimuojantis n-tos eilės daugianaris: $p_1(x) = a_0 + a_1 x$

$$p_2(x) = a_0 + a_1 x + a_2 x^2$$

$$p_3(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

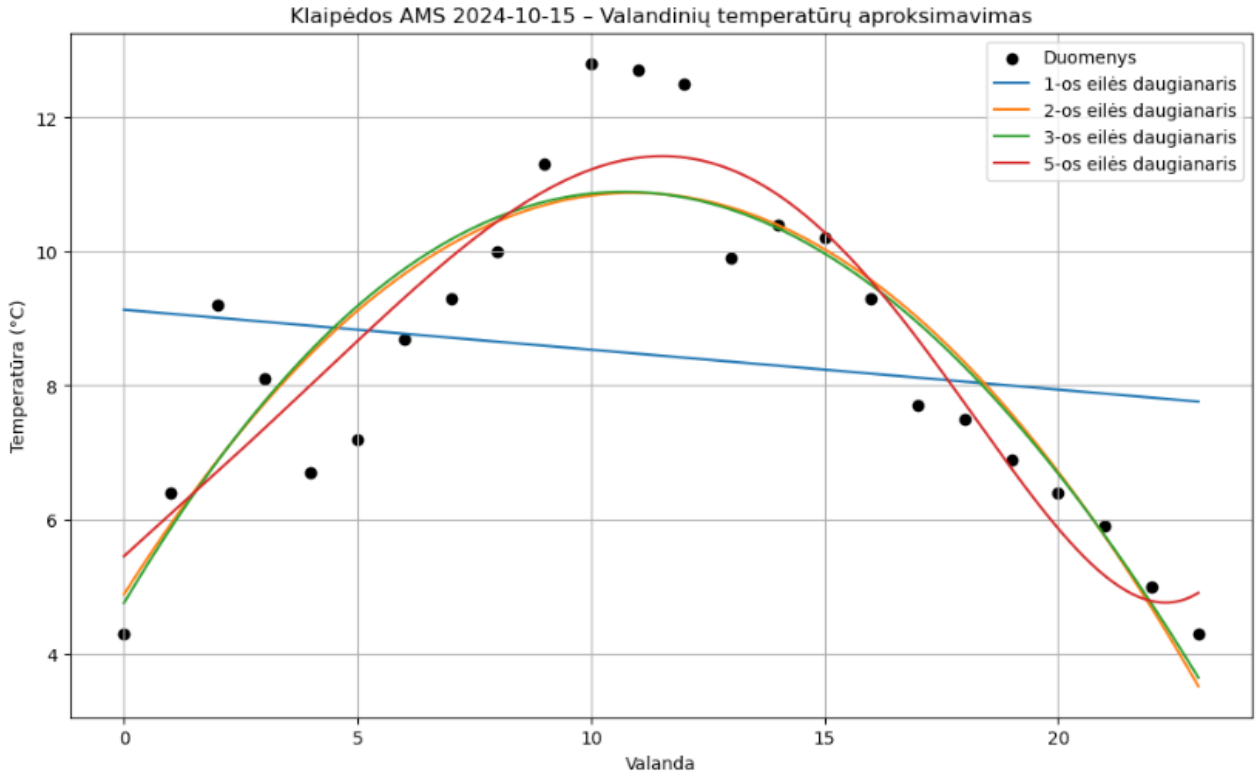
$$p_5(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5$$

Mažiausių kvadratų: $\min\limits_{a_0,\dots,a_n} \sum_{i=0}^{N-1} \big(y_i - p_n(x_i)\big)^2$

Rezultatai:

```
--- APROKSIMAVIMO DAUGIANARIŲ IŠRAIŠKOS ---

1-os eilės daugianaris:
f(x) =  - 0.05961·x + 9.13133

2-os eilės daugianaris:
f(x) =  - 0.05037·x^2 + 1.09887·x + 4.88358

3-os eilės daugianaris:
f(x) = 0.00024·x^3 - 0.05878·x^2 + 1.17464·x + 4.75402

5-os eilės daugianaris:
f(x) = 0.00002·x^5 - 0.00069·x^4 + 0.00599·x^3 - 0.01459·x^2 + 0.64206·x + 5.45164
```

10 pav. Aproksimavimo daugianarių išraiškos (1, 2, 3, 5 eilės)



11 pav. Aproksimavimo daugianarių išraiškų grafikas (Klaipėdos AMS)

Aproksimuojant duomenis skirtingo laipsnio daugianariais matyti, kad žemesnės eilės daugianariai (1–2 laipsnių) gerai atspindi bendrą duomenų trendą, tačiau nesugeba užfiksuoti lokalių svyravimų. Didėjant laipsniui (3–5 eilės), aproksimacija tikslėja – daugianaris geriau prisitaiko prie duotų taškų, ypač ten, kur temperatūra greitai kinta. Tačiau labai aukšti laipsniai pradėtų per daug „gaudyti“ triukšmą. Gauti rezultatai patvirtina mažiausių kvadratų metodo esmę: žemesnės eilės kreivės išlygina duomenis, o aukštesnės geriau prisitaiko prie realių matavimų.

Pagrindinės sprendimo dalies programos kodas:

```
# ------------------------------------------
# 5) Apskaičiuojame aproksimuojančius daugianarius
# ------------------------------------------
for n in orders:

    # Mažiausių kvadratų metodu apskaičiuojame n-to laipsnio daugianario
koeficientus
    coeffs = np.polyfit(x, y, n)

    polynomials[n] = coeffs

    yy = np.polyval(coeffs, xx)

    plt.plot(xx, yy, label=f"{n}-os eilės daugianaris")
```

4. Ketvirta užduotis – Parametrinis aproksimavimas Haro bangelėmis

Parametrinis aproksimavimas – būdas aproksimuoti kreivę plokštumoje, kai jos negalima aprašyti kaip $y = f(x)$.

Pagrindinė Haro bangelių metodo idėja: vietoje globalių funkcijų naudojamos trumpos, lokalios bangelės – labai tinka atskiriems ir trumpiems signalams. Aproksimuoja signalą lokaliai – tik ten, kur reikia, todėl puikiai tinka neperiodiniams, trumpiems signalams.

Metodas: Haro bangelės

Šalis:

Kroatija

4.1. Šalies kontūro formavimas naudojant parametrinį aproksimavimą Haro bangelėmis

Naudotos formulės:

Parametrizacija: $x = x(t), \qquad y = y(t), \qquad t \in [0,1]$

$$t_i = \frac{i}{n-1}, \qquad i = 0,1,\dots,n-1$$

Haro bangelės: $a_{k+1}[i] = \frac{1}{\sqrt{2}}(a_k[2i] + a_k[2i+1])$

$$d_{k+1}[i] = \frac{1}{\sqrt{2}}(a_k[2i] - a_k[2i+1])$$

Haro rekonstrukcija: $a_k[2i] = \frac{1}{\sqrt{2}}(a_{k+1}[i] + d_{k+1}[i])$

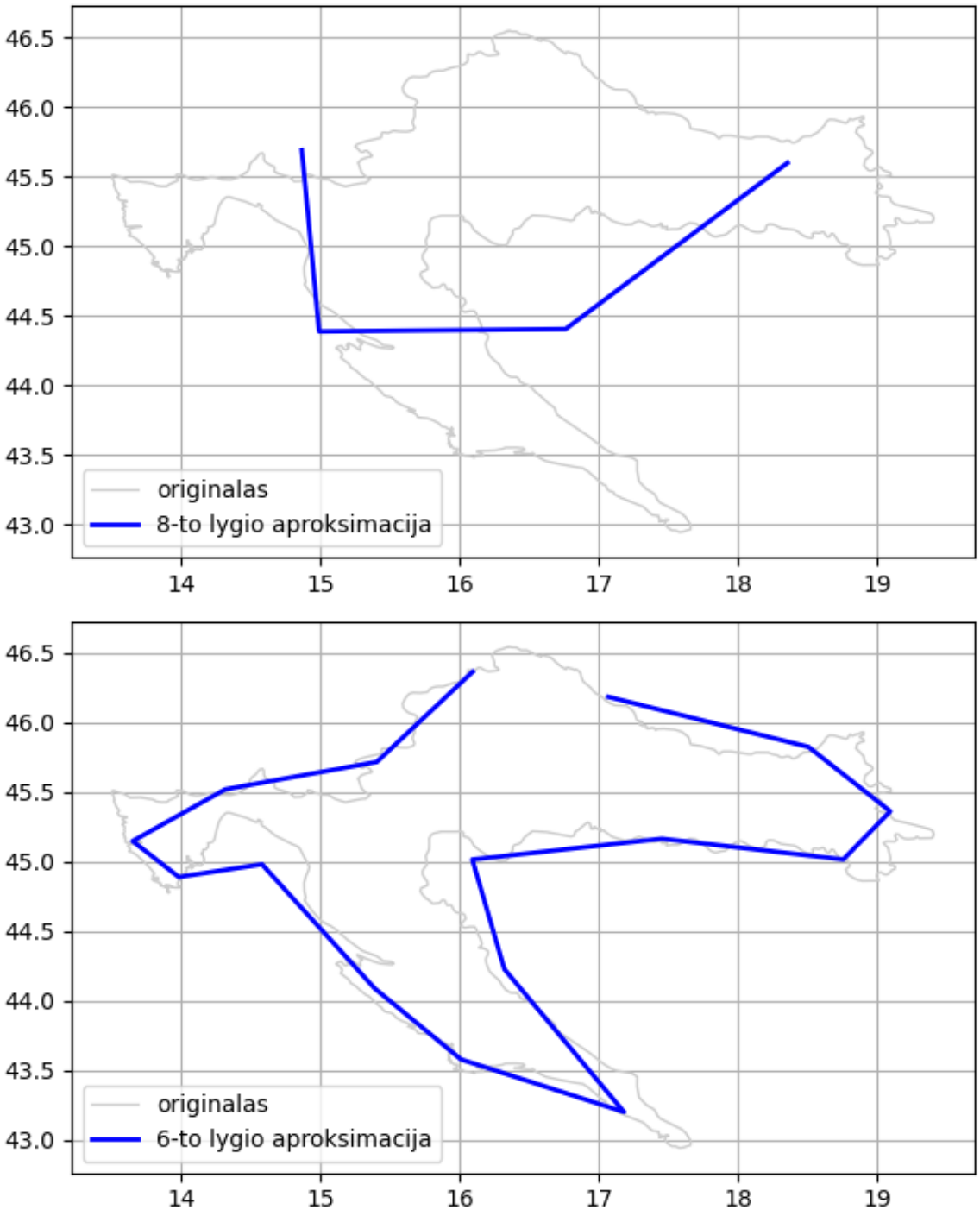
$$a_k[2i+1] = \frac{1}{\sqrt{2}}(a_{k+1}[i] - d_{k+1}[i])$$

Parametrinė rekonstrukcija: $x_L(t) = \mathrm{HaarReconstruct}(x, L)$

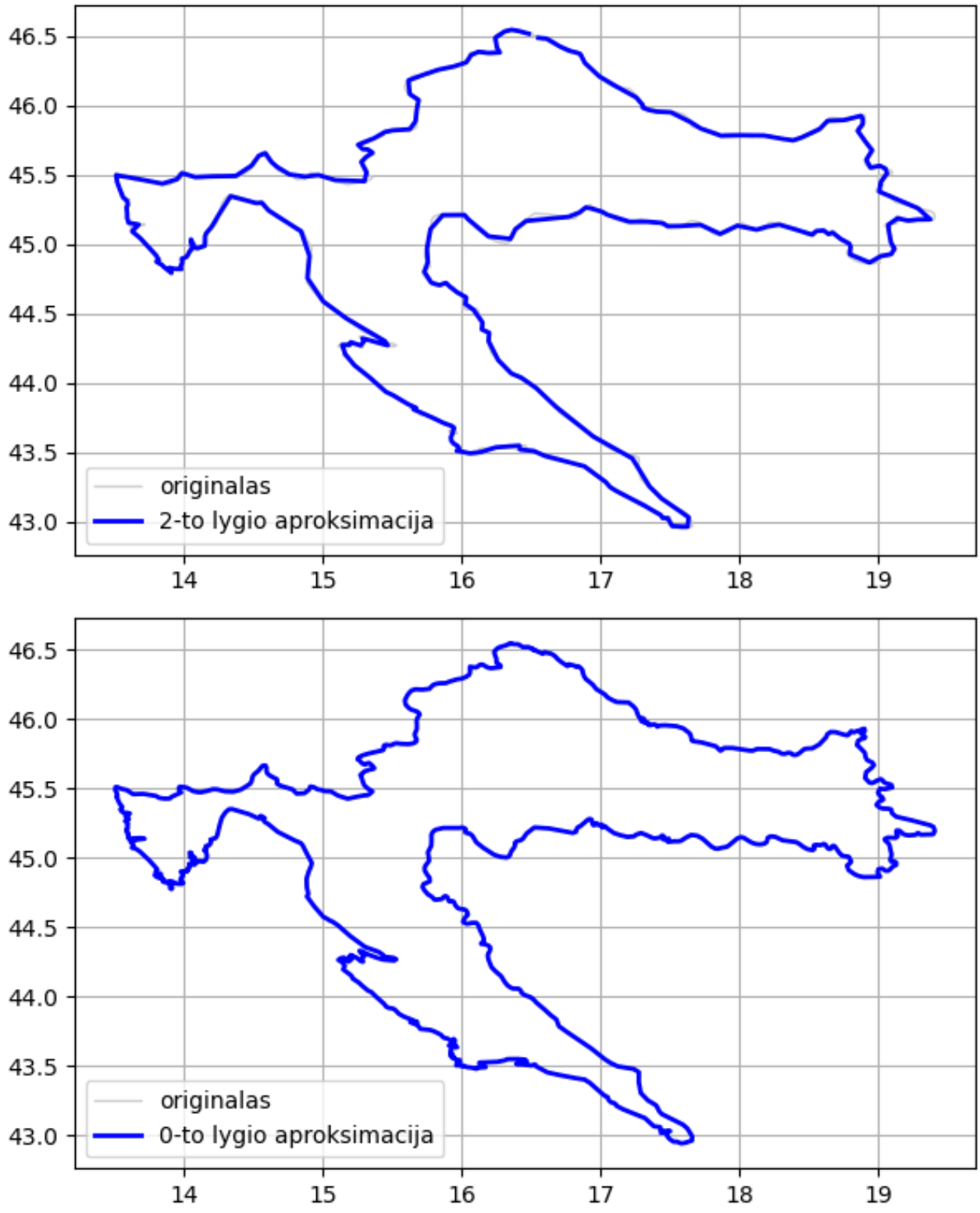
$$y_L(t) = \mathrm{HaarReconstruct}(y, L)$$

Aproksimavimo lygio interpretacija: $d_1 = d_2 = \cdots = d_L = 0$

Rezultatai:



12 pav. 8 ir 6 lygio aproksimacija (Kroatija)



13 pav. 2 ir 0 lygio aproksimacija (Kroatija)

Aproksimavus Kroatijos kontūrą Haro bangelėmis matyti aiški detalumo lygių įtaka rezultato kokybei. Mažiausiame lygyje (0) rekonstruojamas beveik tikslus kontūrų aprašas, nes naudojamos visos detalės. Vidutiniuose lygiuose (2–6) pradedamos šalinti smulkios kontūro nelygumos – kontūras glotnėja, o jo forma tampa paprastesnė, bet vis dar aiškiai atpažįstama. Aukštuose lygiuose (pvz., 8 ir 10) išlieka tik stambiausios šalies formos – kontūras supaprastėja iki esminių geometrinių

bruožų. Tai visiškai atitinka Haro bangelių teoriją: didėjant lygiui pašalinamos vis smulkesnės detalės, todėl signalas tampa vis glodesnis ir labiau apibendrintas. Gauti rezultatai aiškiai parodo, kaip bangelės leidžia kontroliuoti kontūro detalumo lygį.

Pagrindinės sprendimo dalies programos kodas:

```
# ------------------------------------------------------------
# 3. PIRAMIDINIS HARO WAVELET SKILIMAS
# ------------------------------------------------------------
def haar_decompose(sig):
    approximations = []
    details = []

    a = sig.copy()
    approximations.append(a)

    while len(a) > 1:
        # Dabartinio lygio ilgis
        n = len(a)
        # Skaičiuojame sekančio lygio aproksimacijas (vidurkius)
        a_next = (a[0:n:2] + a[1:n:2]) / np.sqrt(2)
        # Skaičiuojame sekančio lygio detales (skirtumus)
        d_next = (a[0:n:2] - a[1:n:2]) / np.sqrt(2)
        approximations.append(a_next)
        details.append(d_next)
        a = a_next

    return approximations, details


def haar_reconstruct(approximations, details, level):
    """
    Rekonstruoja signalą iš Haar koeficientų,
    išmetant smulkiausias detales.

    level = 0  -> pilna rekonstrukcija (visos detalės)
    level = 1  -> išmetame pačią smulkiausią detalę
    ...
    level = K  -> paliekame tik grubiausią aproksimaciją
    """

    # K - detalių lygių skaičius
    K = len(details)

    # Pradedam nuo grubiausios aproksimacijos (pats trumpiausias a_K)
    a = approximations[-1]

    # Einame nuo grubiausio lygio žemyn iki smulkiausio (0)
    for j in range(K-1, -1, -1):
        # Jei lygis mažesnis už nurodytą level - šią detalę anuliuojame
        if j < level:
            d = np.zeros_like(details[j])
        else:
            d = details[j]

        up = np.zeros(len(d)*2)
        # Į lyginius indeksus dedame (a + d)/sqrt(2)
```

```
        up[0::2] = (a + d)/np.sqrt(2)
        up[1::2] = (a - d)/np.sqrt(2)
        a = up

    return a

# ------------------------------------------------------------
# !! SVARBU: Haar reikalauja 2^k taškų
# ------------------------------------------------------------
def resample_to_power_of_two(x, y):
    n = len(x)
    # Randame didžiausią k, kad 2^k <= n
    k = int(np.floor(np.log2(n)))
    N = 2**k

    t_old = np.linspace(0, 1, n)
    t_new = np.linspace(0, 1, N)

    x_new = np.interp(t_new, t_old, x)
    y_new = np.interp(t_new, t_old, y)
    return x_new, y_new

x, y = resample_to_power_of_two(x, y)

# Skilimas
ax, dx = haar_decompose(x)
ay, dy = haar_decompose(y)
```