Kauno technologijos universitetas
Informatikos fakultetas

Skaitiniai metodai ir algoritmai

Interpoliavimas ir aproksimavimas

Vytenis Kriščiūnas IFF-1/1

Studentas

doc. Kriščiūnas Andrius

Dėstytojas

Kaunas 2023

TURINYS

Contents

1. Interpoliavimas daugianariu

1.1. 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.

9	$\cos(2 \cdot x) \cdot (\sin(2 \cdot x) + 1{,}5) + \cos x\,; -2 \le x \le 3;$	Čiobyševo

1.2. Teorinė dalis

Taškai pasiskirstę tolygiai:

```
# Intervalas
a, b = -2, 3
```

```
def f(x):
    return np.cos(2*x) * (np.sin(2*x) + 1.5) + np.cos(x)
```

```
# Interpoliavimo taškai pasiskirstę tolygiai
n = 10
x = np.linspace(a, b, n)
y = f(x)
```

```
T1 = vien(x, n)
y_T=(np.matrix(y)).transpose()
A1=np.hstack((T1,y_T))
coeff1, ar = Gous(A1, n)
```

```
def vien(x, n):
    A=np.zeros((n,n),dtype=float)
```

```
    for i in range(n):
        A[:,i]=np.power(x,i)
    return A
```

```
def Gous(A1, n):
    ar = "Viena" #Skirtas singuliarumo salygai
    for i in range (0,n-1):
        a, iii = np.max(np.abs(A1[i:n, i])), np.argmax(np.abs(A1[i:n, i])) + i

        if a == 0:
            continue

        if iii > i:
            A1[[i, iii], :] = A1[[iii, i], :]

        for j in range (i+1,n):
            A1[j,i:n+1]=A1[j,i:n+1]-A1[i,i:n+1]*A1[j,i]/A1[i,i]
            A1[j,i]=0


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

    return x, ar
```

```
xxx=np.linspace(a,b,100)
yyy=interp(xxx, n, coeff1)
fig=plt.figure(0)
ax1=fig.add_subplot(1,1,1)
ax1.plot(x, y, label='Duotoji funkcija', color='black')
ax1.plot(x, y,'bo')
ax1.plot(xxx,yyy, label='Interpoliacinė funkcija (tolygiai)', color='blue')
ax1.set_title("Tolygūs taškai")
plt.grid()
plt.legend()
plt.show()
```

```
def interp(x, n, coeff):
    yyy=np.zeros(x.size,dtype=float)
    for i in range (n):
        yyy+=np.power(x,i)*coeff[i]
```

```
    return yyy
```

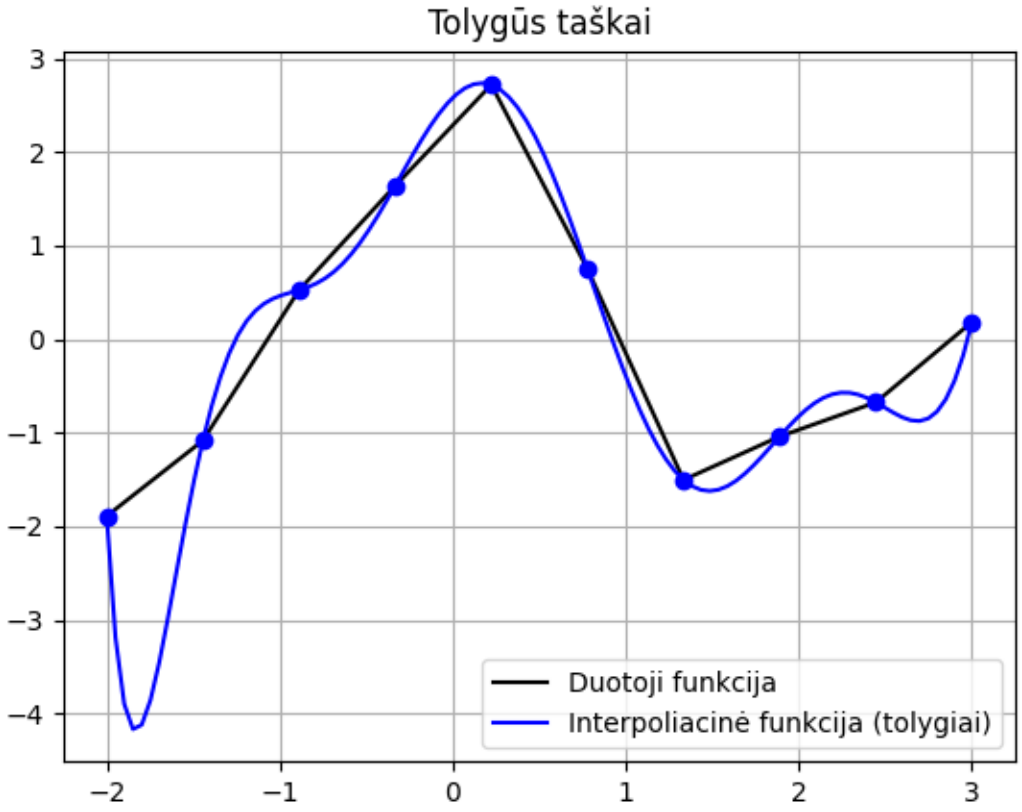
Taškai apskaičiuojami naudojant Čiobyševo abscises:

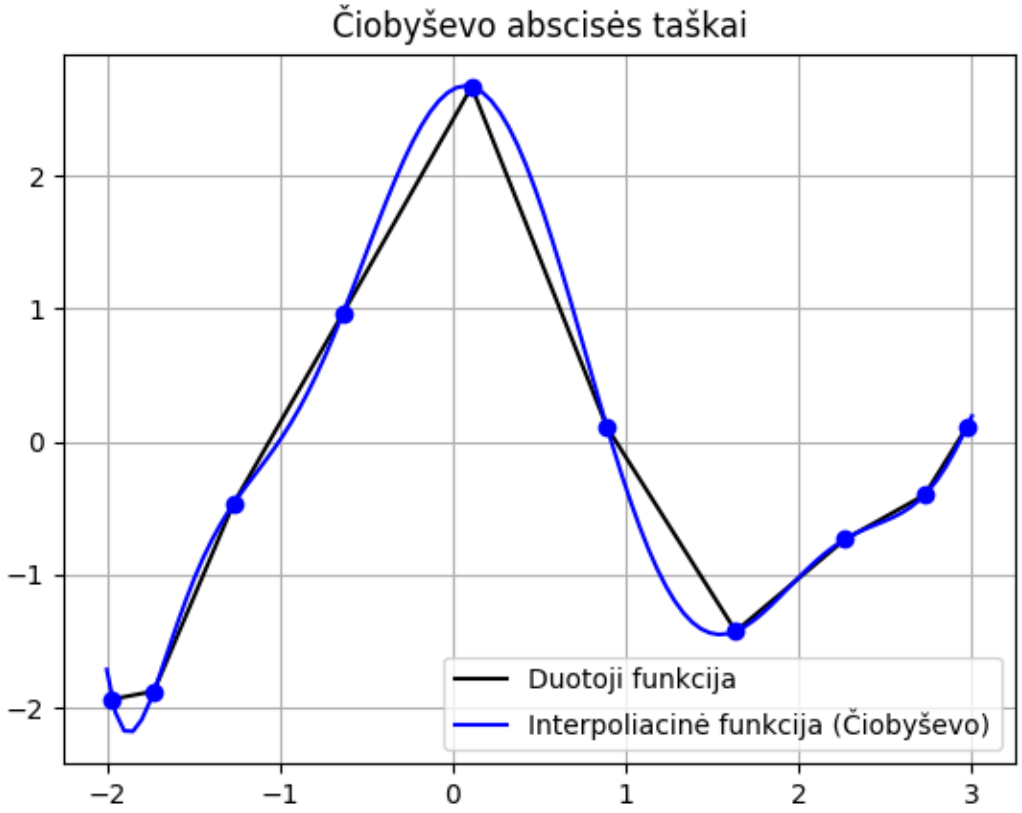
```
# Interpoliavimo taškai apskaičiuoti naudojant Čiobyševo abscises
n_chebyshev = 10
x_chebyshev = 0.5 * (a + b) + 0.5 * (b - a) * np.cos((2 * np.arange(0,
n_chebyshev) + 1) * np.pi / (2 * n_chebyshev))
y_chebyshev = f(x_chebyshev)
```

```
T2 = vien(x_chebyshev, n_chebyshev)
y_chebyshev_T=(np.matrix(y_chebyshev)).transpose()
A2=np.hstack((T2,y_chebyshev_T))
coeff2, ar = Gous(A2, n_chebyshev)
```

```
xxx=np.linspace(a,b,100)
yyy=interp(xxx, n_chebyshev, coeff2)
fig=plt.figure(1)
ax2=fig.add_subplot(1,1,1)
ax2.plot(x_chebyshev, y_chebyshev, label='Duotoji funkcija', color='black')
ax2.plot(x_chebyshev, y_chebyshev,'bo')
ax2.plot(xxx,yyy, label='Interpoliacinė funkcija (Čiobyševo)', color='blue')
ax2.set_title("Čiobyševo abscisės taškai")
plt.grid()
plt.legend()
plt.show()
```

1.3. Rezultatai





1.4. Programos aprašymas

Naudojant vienanarių bazę yra bandoma gauti interpoliuojančią funkciją, einančią per duotus taškus. Interpoliuojantys taškai gali būti pasiskirstę tolygiai arba pagal Čiobyševo abscises. Iš pradžių reikia sudaryti tiesinę lygčių sistema, kurioje lygtys yra nepriklausomos ir gauti koeficientus. Galiausiai yra gaunama funkcija sudaryta iš bazinės funkcijos ir koeficientų sandaugos.

2. Interpoliavimas splainu per duotus taškus

2.1. 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)).

9	Panama	Globalus

2.2. Teorinė dalis

```
Splainu_interpoliacija()
```

```
def Splainu_interpoliacija():
    plt.figure(1), plt.grid(True), plt.axis('equal')
    nP = 21
    xrange = [1998, 2018]
    X = np.linspace(xrange[0], xrange[1], nP)
    Y = [16.53, 15.67, 15.77, 15.54, 14.82, 15.13, 15.11, 16.76, 17.29, 17.30,
17.41, 19.00, 19.62, 20.72, 21.42, 21.40, 21.83, 21.73, 21.94, 21.51, 21.53]
    plt.plot(X, Y, 'ko')

    DDF = splaino_koeficientai(X, Y)
    for iii in range(nP-1):
        nnn = 100
        sss = np.linspace(X[iii], X[iii+1], nnn)
        S = splainas(X[iii:iii+2], Y[iii:iii+2], DDF[iii:iii+2], nnn)
        plt.plot(sss, S, 'r-')

    plt.legend(['duoti taškai', 'Splainai {} intervaluose'.format(nP-1)])
    plt.show()

    return
```

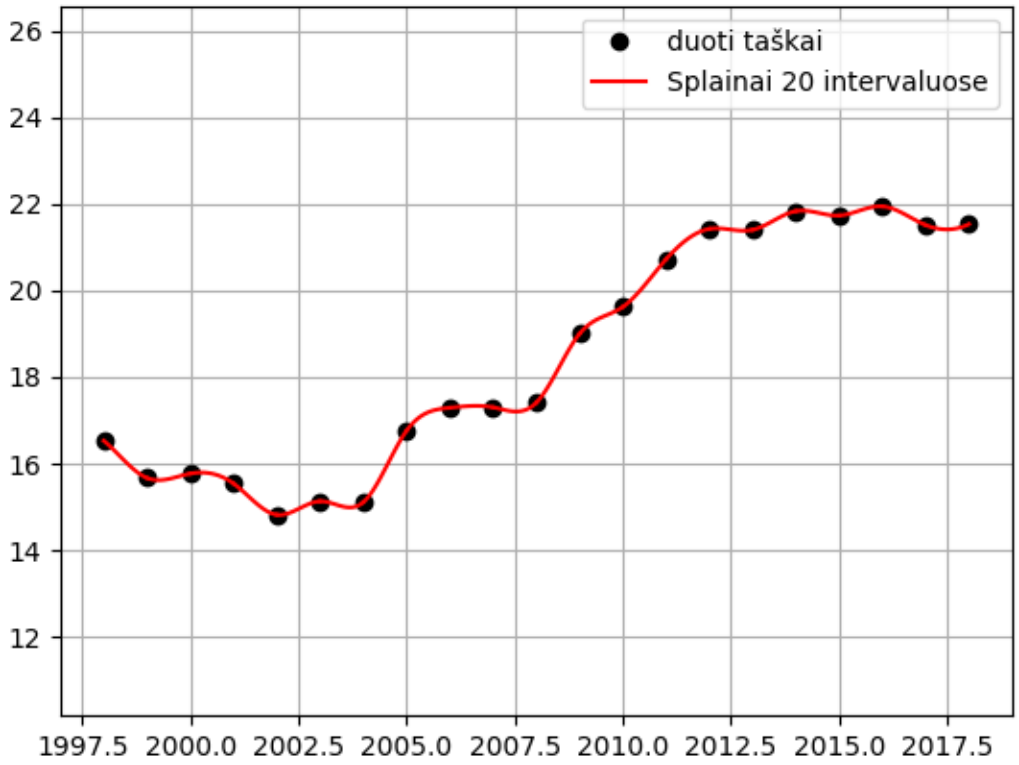
```
def splaino_koeficientai(X, Y):
    n = len(X)
    A = np.zeros((n, n))
    b = np.zeros(n)
    d = X[1:] - X[:-1]
    for i in range(n-2):
        A[i, i:i+3] = [d[i]/6, (d[i]+d[i+1])/3, d[i+1]/6]
        b[i] = (Y[i+2]-Y[i+1])/d[i+1] - (Y[i+1]-Y[i])/d[i]

    A[n-1, 0] = 1
    A[n-1, n-1] = 1

    y_T=(np.matrix(b)).transpose()
    A1=np.hstack((A,y_T))
    DDF = Gous(A1, n)
```

```
    return DDF
```

```
def Gous(A1, n):
    for i in range (0,n-1):
        a, iii = np.max(np.abs(A1[i:n, i])), np.argmax(np.abs(A1[i:n, i])) + i

        if a == 0:
            continue

        if iii > i:
            A1[[i, iii], :] = A1[[iii, i], :]

        for j in range (i+1,n):
            A1[j,i:n+1]=A1[j,i:n+1]-A1[i,i:n+1]*A1[j,i]/A1[i,i]
            A1[j,i]=0


    #Grizimas atgal
    x=np.zeros(shape=(n,1))
    for i in range (n-1,-1,-1):
        if (A1[i,i] == 0 and A1[i,n:n+1] == 0):
            x[i,:] = 1
        elif (A1[i,i] == 0 and A1[i,n:n+1] != 0):
            return None
        else:
            x[i,:]=(A1[i,n:n+1]-A1[i,i+1:n]*x[i+1:n,:])/A1[i,i]

    return x
```

```
def splainas(X, Y, DDF, nnn):
    d = X[1] - X[0]
    sss = np.linspace(X[0], X[1], nnn)
    S = DDF[0]/2*(sss-X[0])**2 + (DDF[1]-DDF[0])/(6*d)*(sss-X[0])**3 + (sss-
X[0])*((Y[1]-Y[0])/d-DDF[0]*d/3-DDF[1]*d/6) + Y[0]

    return S
```

2.3. Rezultatai



2.4. Programos aprašymas

Interpoliuojant globaliu splainu per duotus nurodytos šalies šiltnamio dujų emisijos interpoliavimo mazgus yra bandoma sudaryti interpoliuojančias kreives. Nėra žinomos funkcijų išvestinės, o tik duotos mazgų koordinatės. Yra turima globalaus splaino funkcijos formulė, pagal ją yra sudaroma trijų įstrižainių matrica, tada reikia rasti antros eilės išvestnių reikšmes Gauso metodu. Kai šie kofiecientai yra randami, galima pasinaudoti glogalaus splaino funkcijos išraiška ir apskaičiuoti interpoliavimo kreives.

3. Aproksimavimas

3.1. 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.

3.2. Teorinė dalis

```
Calcutalte(1)
Calcutalte(2)
Calcutalte(3)
Calcutalte(5)
```

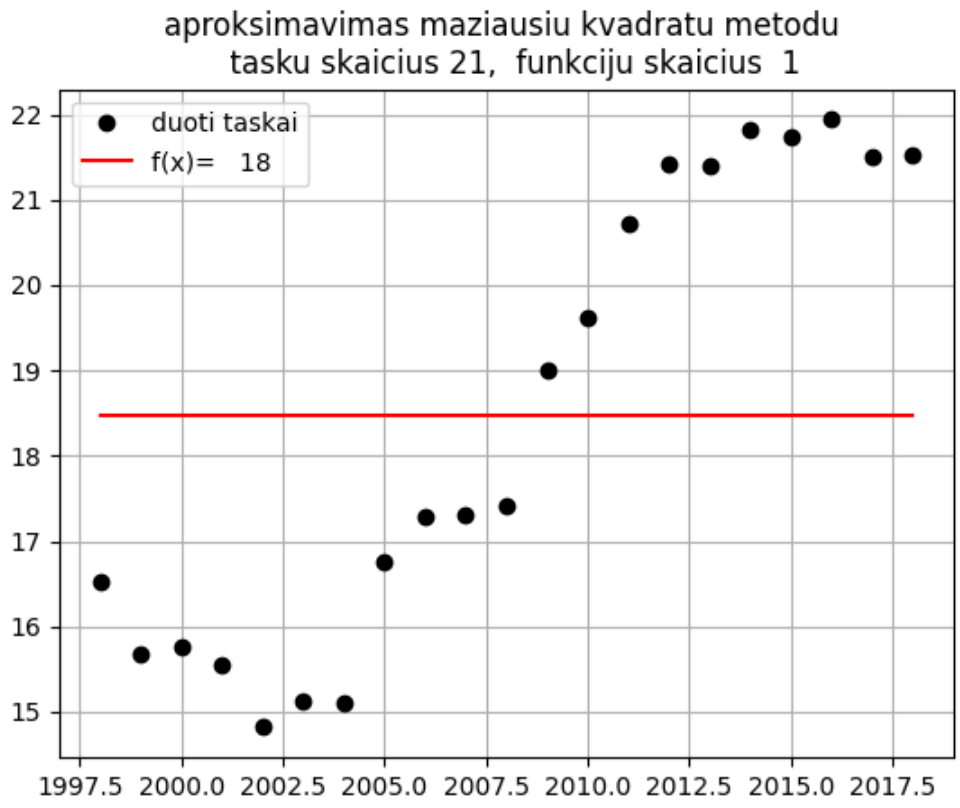
```
def Calcutalte(m):
    plt.figure(1)
    plt.grid(True)

    nP = 21
    xrange = [1998, 2018]
    X = np.linspace(xrange[0], xrange[1], nP)
    Y = [16.53, 15.67, 15.77, 15.54, 14.82, 15.13, 15.11, 16.76, 17.29, 17.30,
17.41, 19.00, 19.62, 20.72, 21.42, 21.40, 21.83, 21.73, 21.94, 21.51, 21.53]
    plt.plot(X, Y, 'ko', label='Duotoji funkcija', color='black')
    n = len(X)  # tasku skaicius

    # Maziausiu kvadratu metodo lygciu sistema:
    G = base(m, X)
    A_transpose_A = np.dot(G.T, G)
    A_transpose_Y = np.dot(G.T, Y)
    y_T=(np.matrix(A_transpose_Y)).transpose()
    A1=np.hstack((A_transpose_A,y_T))
    c = Gous(A1, m)

    sss = '{:5.2g}'.format(float(c[0]))
    for i in range(1, m):
        sss += '+{:5.2g}x^{}'.format(float(c[i]), i)
    sss = sss.replace('+  -', '-')

    # Aproksimuojanti funkcija:
    nnn = 200  # vaizdavimo tasku skaicius
    xxx = np.linspace(xrange[0], xrange[1], nnn)  # vaizdavimo taskai
    Gv = base(m, xxx)
    fff = np.dot(Gv, c)
```

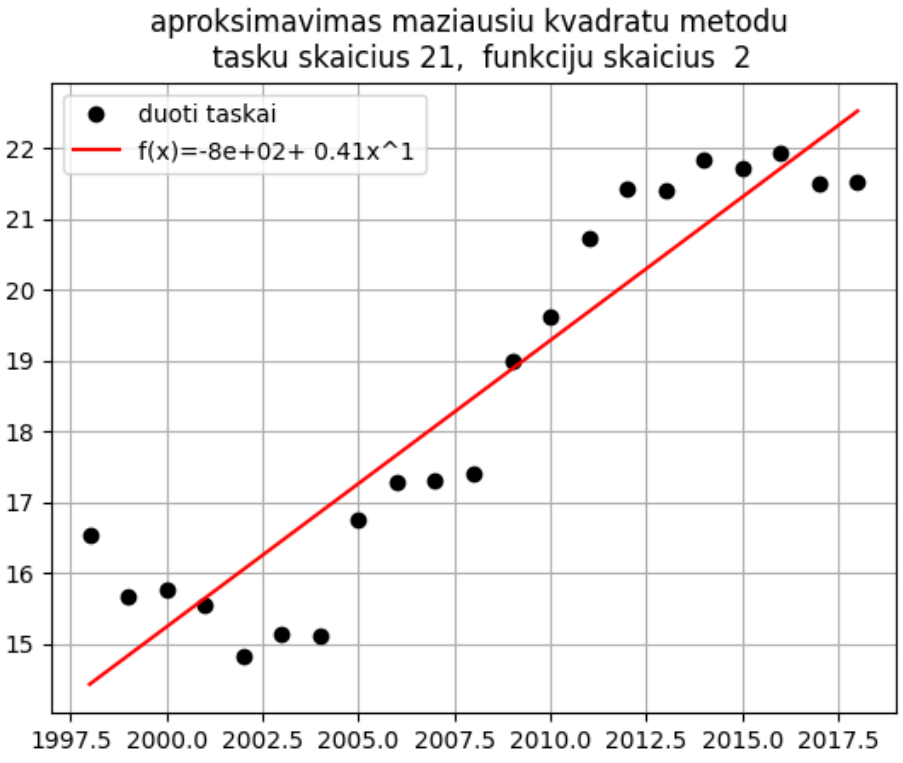
```
    plt.plot(xxx, fff, 'r-')
    plt.legend(['duoti taskai', 'f(x)={}'.format(sss)])
    plt.title(f'aproksimavimas maziausiu kvadratu metodu \n  tasku skaicius
{n},  funkciju skaicius  {m}')
    plt.show()
```

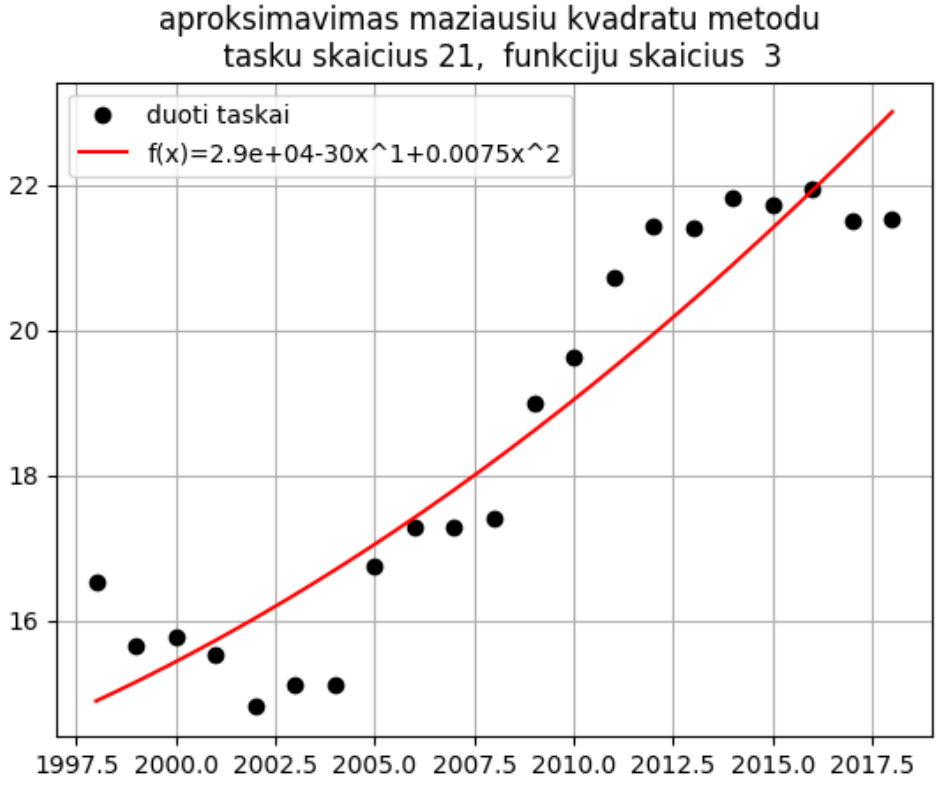
```
def base(m, x):
    G=np.zeros((len(x),m),dtype=float)
    for i in range(m):
        G[:,i]=np.power(x,i)
    return G
```

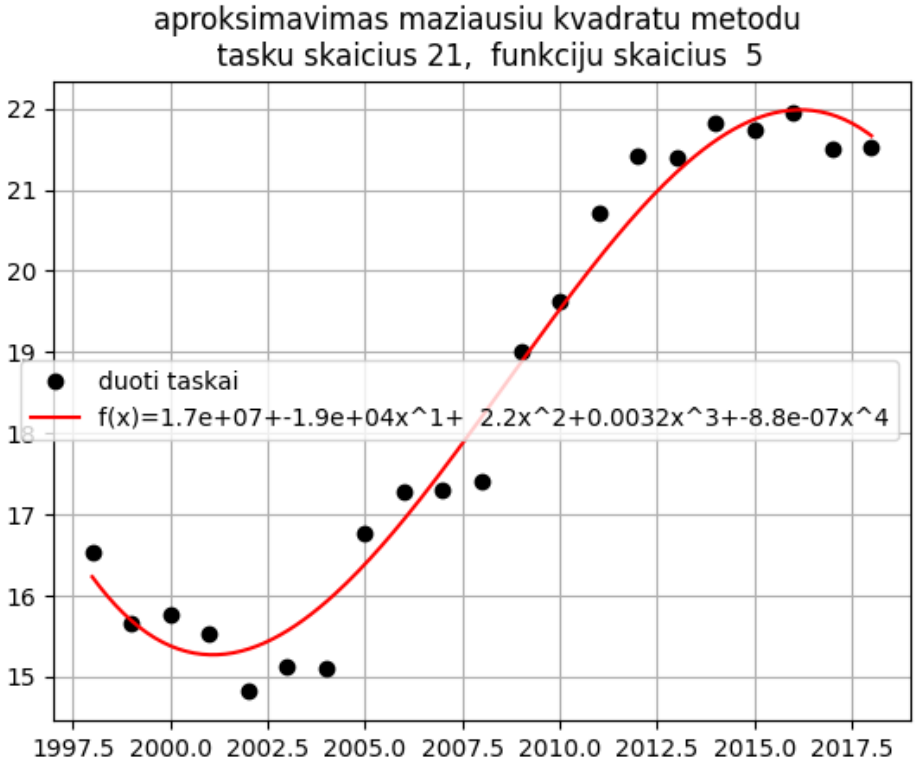
```
def Gous(A1, n):
    for i in range (0,n-1):
        a, iii = np.max(np.abs(A1[i:n, i])), np.argmax(np.abs(A1[i:n, i])) + i

        if a == 0:
            continue

        if iii > i:
            A1[[i, iii], :] = A1[[iii, i], :]

        for j in range (i+1,n):
            A1[j,i:n+1]=A1[j,i:n+1]-A1[i,i:n+1]*A1[j,i]/A1[i,i]
            A1[j,i]=0


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

    return x
```

3.3. Rezultatai


aproksimavimas maziausiu kvadratu metodu
tasku skaicius 21, funkciju skaicius 1
duoti taskai
f(x)= 18
22
21
20
19
18
17
16
15
1997.5
2000.0
2002.5
2005.0
2007.5
2010.0
2012.5
2015.0
2017.5


aproksimavimas maziausiu kvadratu metodu
tasku skaicius 21, funkciju skaicius 2
duoti taskai
f(x)=-8e+02+ 0.41x^1
22
21
20
19
18
17
16
15
1997.5
2000.0
2002.5
2005.0
2007.5
2010.0
2012.5
2015.0
2017.5


aproksimavimas maziausiu kvadratu metodu
tasku skaicius 21, funkciju skaicius 3
duoti taskai
f(x)=2.9e+04-30x^1+0.0075x^2
22
20
18
16
1997.5 2000.0 2002.5 2005.0 2007.5 2010.0 2012.5 2015.0 2017.5


aproksimavimas maziausiu kvadratu metodu
tasku skaicius 21, funkciju skaicius 5
duoti taskai
f(x)=1.7e+07+-1.9e+04x^1+ 2.2x^2+0.0032x^3+-8.8e-07x^4
22
21
20
19
17
16
15
1997.5 2000.0 2002.5 2005.0 2007.5 2010.0 2012.5 2015.0 2017.5

3.4. Programos aprašymas

Naudojant mažiausių kvadratų metodą yra bandoma rasti aproksimuojančias funkcijas (pirmos, antros, trečios ir penktos eilės daugianarius). Pasiremiant gradientu ir Jakobino matrica yra sudaroma bendroji mažiausių kvadratų metodo formulė. Iš pradžių reikia rasti bazinę funkciją, kurios apskaičiavimas yra labai panašus į vienanarių bazę. Ši bazė yra panaudojama šio naudojamo metodo formulėje sienkiant rasti koeficientus. Jie yra randami Gauso metodu. Galiausiai yra dauginami koeficinetai iš tam tikrų x reikšmių ir brėžiama norimos eilės daugianarių aproksimuojanti funkcija. Tam tikras taškų išsidėstymo tendencijas galima pastebėti iki 3 eilės, nes tolesnės eilės nieko reikšmingo neparodo.

4. Parametrinis aproksimavimas

4.1. 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.

4.2. Teorinė dalis

```
main()
```

```
def main():
    plt.figure(1)
    plt.axis('equal')
    plt.grid(True)

    # Reading data from files
    n = 10
    nnn = 2**n
    SX, SY = coordinates()

    # Parameterization
    t = np.cumsum(np.linalg.norm(np.diff(np.column_stack((SX, SY)), axis=0),
axis=1))
    t = np.concatenate(([0], t))
    t1 = np.linspace(min(t), max(t), nnn)
    SX = interp1d(t, SX, kind='linear', fill_value='extrapolate')(t1)
```

```
    SY = interp1d(t, SY, kind='linear', fill_value='extrapolate')(t1)
    t = t1

    plt.plot(SX, SY, 'c')
    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 = 6
    smoothx, detailsx = haar_wavelet_approximation(t, SX, n, m)
    smoothy, detailsy = haar_wavelet_approximation(t, SY, n, m)

    smoothx, smoothy

    # Function reconstruction
    hx = np.zeros(nnn)
    hy = np.zeros(nnn)

    for k in range(2**(n-m)):
        hx += smoothx[k] * haar_scaling(t, n-m, k, min(t), max(t))
        hy += smoothy[k] * haar_scaling(t, n-m, k, min(t), max(t))

    leg = [f'Suglodinta funkcija, detalumo lygmuo {n-m}']
    plt.figure(2)
    plt.subplot(m+1, 1, 1)
    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')

    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-1][k] * haar_wavelet(t, n-m+i, k, min(t), max(t))
            h1y += detailsy[m-i-1][k] * haar_wavelet(t, n-m+i, k, min(t), max(t))

        hx += h1x
        hy += h1y

        plt.figure(2)
```

```
        plt.subplot(m+1, 1, i+2)
        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.show()
```

```
def coordinates():
  shape = shapefile.Reader("ne_10m_admin_0_countries.shp")

  #id = 157
  feature = shape.shapeRecords()[157]

  largestAreaID = 0
  if feature.shape.__geo_interface__['type'] == 'MultiPolygon':
    area = 0
    for i in range(len(feature.shape.__geo_interface__['coordinates'])):
      points = feature.shape.__geo_interface__['coordinates'][i][0]
      polygon = geometry.Polygon(points)
      if polygon.area > area:
        area = polygon.area
        largestAreaID = i

    xxyy = feature.shape.__geo_interface__['coordinates'][largestAreaID][0]
  else:
    xxyy = feature.shape.__geo_interface__['coordinates'][0]


  xy = list(zip(*xxyy))
  X = xy[0]
  Y = xy[1]
  return X, Y
```

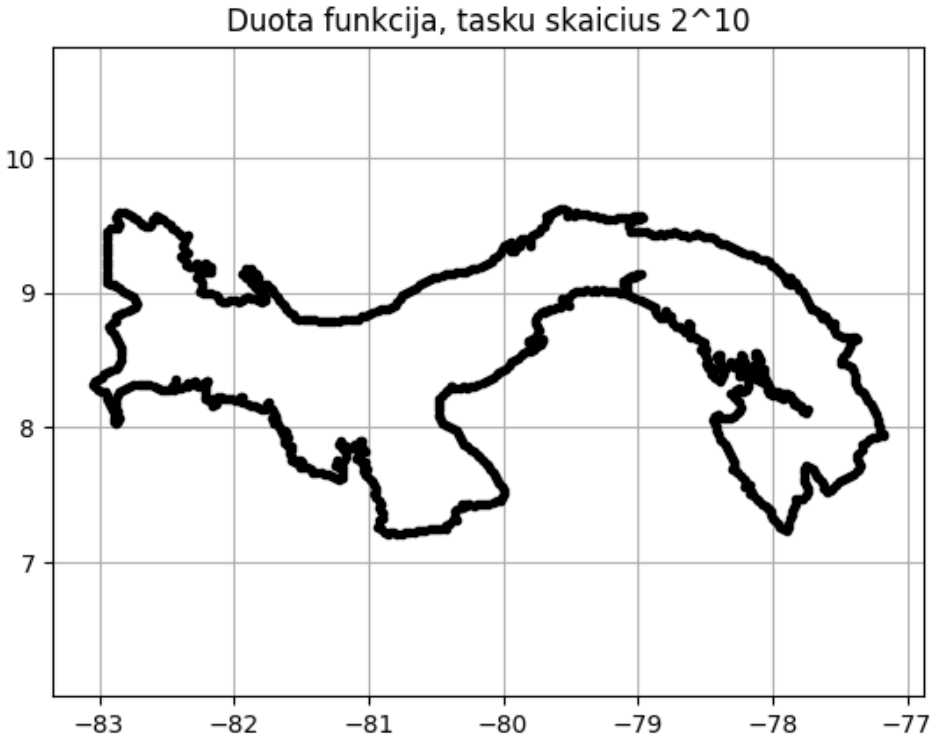
```
def haar_wavelet_approximation(SX, SY, n, m):
    a, b = min(SX), max(SX)
    nnn = 2**n
    smooth = (b - a) * SY * 2**(-n/2)
```

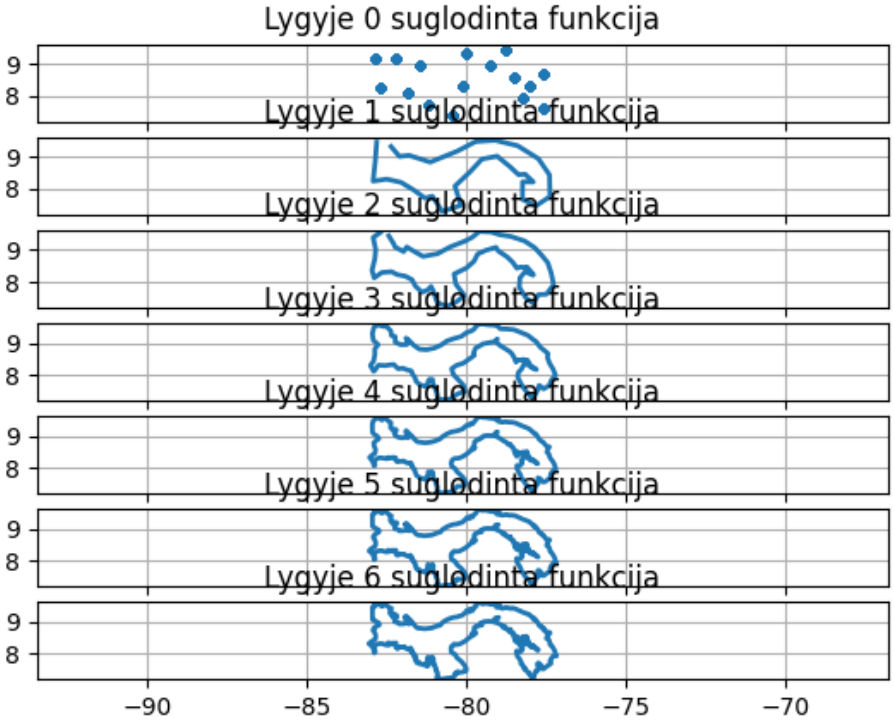
```
    details = []
    for i in range(1, m + 1):
        smooth1 = (smooth[0::2] + smooth[1::2]) / np.sqrt(2)
        detail = (smooth[0::2] - smooth[1::2]) / np.sqrt(2)
        details.append(detail)
        smooth = smooth1

    return smooth, details
```

```
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
```

4.3. Rezultatai


Duota funkcija, tasku skaicius 2^10
10
9
8
7
−83
−82
−81
−80
−79
−78
−77


Lygyje 0 suglodinta funkcija
Lygyje 1 suglodinta funkcija
Lygyje 2 suglodinta funkcija
Lygyje 3 suglodinta funkcija
Lygyje 4 suglodinta funkcija
Lygyje 5 suglodinta funkcija
Lygyje 6 suglodinta funkcija
9
8
−90
−85
−80
−75
−70

4.4. Programos aprašymas

Pagal duotą taškų seką, kuri atspindi šalies kontūrą, naudojant parametrinį aproksimavimą Haro bangelėmis reikia suformuoti šalies kontūrą. Remiantis Euklido sumos algoritmu ir interpoliaciją yra gaunamos X ir Y reikšmės reikalingos nubraižyti šalies kontūrą. Galiausiai reikia apskaičiuoti aproksimavimo funkcijas skirtinguose lygmenyse. Tai yra daroma apskaičiuojant Haro bangelių ir Haro fukcijų reikšmes. Jos yra sudedamos atitinkamose vietose, kad būtų galima gauti tolimesnio lygmens funkciją.