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 3

1.1. a dalis. Taškai pasiskirstę tolygiai 4

1.2. b dalis. Taškai apskaičiuojami naudojant Čiobyševo abscises 5

1.3. Programos kodas 5

2. Antra užduotis. Interpoliavimas splainu per duotus taškus 8

2.1. Programos kodas 9

3. Trečia užduotis. Aproksimavimas 12

3.1. Programos kodas 15

4. Ketvirta užduotis. Parametrinis aproksimavimas 17

4.1. Pradiniai duomenys 17

4.2. Aproksimavimo rezultatai 18

4.3. Programos kodas 22

# Pirma užduotis. Interpoliavimas daugianariu

A close up of a text

Description automatically generated



A graph with a red line

Description automatically generated

pav. daugianario grafikas

## a dalis. Taškai pasiskirstę tolygiai

A graph with different colored lines

Description automatically generated

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:

## b dalis. Taškai apskaičiuojami naudojant Čiobyševo abscises

A graph with lines and dots

Description automatically generated

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

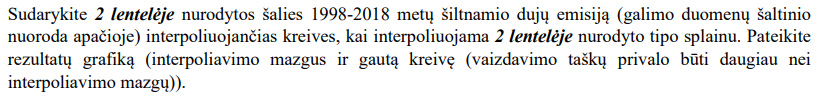
a12 = 1.31665, a13 = -3.39313, a14 = 3.98924

Interpoliuojančios funkcijos išraiška:

## 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() |

# Antra užduotis. Interpoliavimas splainu per duotus taškus





A graph with red dots and black lines

Description automatically generated

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:

A graph showing a line

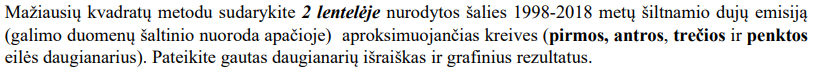
Description automatically generated

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

## 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) |

# Trečia užduotis. Aproksimavimas





A graph with green and blue dots

Description automatically generated

pav. Pirmos eiles daugianaris

Aproksimacijos funkcijos koeficientai: [79525.28138528, 401.38528139]

A screen shot of a graph

Description automatically generated

pav. Antros eilės daugianaris

Aproksimacijos funkcijos koeficientai:

[[75717.6223794 1932.40089794 -98.89988109]]

A graph with green and blue dots

Description automatically generated

pav. Trečios eilės daugianaris

Aproksimacijos funkcijos koeficientai:

[[ 5.66523047e+04 8.56591313e+03 -7.29491851e+02 1.75164436e+01]]

A graph with green and blue lines

Description automatically generated

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]]

## 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() |

# Ketvirta užduotis. Parametrinis aproksimavimas

A close up of text

Description automatically generated



## Pradiniai duomenys

A map of austria with black outline

Description automatically generated

## Aproksimavimo rezultatai

A graph of a line and a line

Description automatically generated

A graph of lines and a line

Description automatically generated with medium confidence

A graph of a car with red lines

Description automatically generated

A graph with lines and red lines

Description automatically generated

A graph with a red mark

Description automatically generated with medium confidence

A graph with a red marker

Description automatically generated

A graph with a red marker on it

Description automatically generated

A map of austria with a red circle

Description automatically generated

A graph with a red circle and a map

Description automatically generated

A map of austria with a red circle

Description automatically generated

## 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() |