

Kauno technologijos universitetas

Informatikos fakultetas

Skaitiniai metodai ir algoritmai

Netiesinių lygčių sprendimas

Vytenis Kriščiūnas IFF-1/1

Studentas

**doc. Kriščiūnas Andrius**

Dėstytojas

Kaunas 2023

**TURINYS**

[1. Pirma dalis 4](#_Toc146740580)

[1.1. Nustatykite daugianario 𝑓(𝑥) šaknų intervalą. 4](#_Toc146740581)

[1.1.1. Užduotis 4](#_Toc146740582)

[1.1.2. Grubus įvertis 4](#_Toc146740583)

[1.1.3. Tikslesnis įvertis 4](#_Toc146740584)

[1.1.4. Grafinis vaizdavimas f(x) funkcijos 5](#_Toc146740585)

[1.1.5. Grafinis vaizdavimas g(x) funkcijos 6](#_Toc146740586)

[1.2. Skenavimo algoritmo panaudojimas 7](#_Toc146740587)

[1.2.1. Užduotis 7](#_Toc146740588)

[1.2.2. Funkcijos f(x) skenavimo intervalų radimas 8](#_Toc146740589)

[1.2.3. Funkcijos g(x) skenavimo intervalų radimas 9](#_Toc146740590)

[1.3. Funkcijų šaknų tikslinimas 10](#_Toc146740591)

[1.3.1. Užduotis 10](#_Toc146740592)

[1.3.2. Funkcijos f(x) šaknų radimas pusiaukirtos metodu 10](#_Toc146740593)

[1.3.3. Funkcijos f(x) šaknų radimas Niutono (liestinių) metodu 12](#_Toc146740594)

[1.3.4. Funkcijos g(x) šaknų radimas pusiaukirtos metodu 13](#_Toc146740595)

[1.3.5. Funkcijos g(x) šaknų radimas Niutono (liestinių) metodu 15](#_Toc146740596)

[1.3.6. Iteracijų palyginimas 17](#_Toc146740597)

[1.4. Gautų šaknų tikrinimas wolframalpha.com tinklapyje 17](#_Toc146740598)

[1.4.1. Užduotis 17](#_Toc146740599)

[1.4.2. Funkcijos f(x) šaknų patikrinimo rezultatai 17](#_Toc146740600)

[1.4.3. Funkcijos g(x) šaknų patikrinimo rezultatai 17](#_Toc146740601)

[2. Antra dalis 18](#_Toc146740602)

[2.1. Tarpinių grafikų sudarymas (3, 4 ir 5 TE narių skaičius) 18](#_Toc146740603)

[2.1.1. Užduotis 18](#_Toc146740604)

[2.1.2. Funkcijos h(x) ir TE 3, 4 ir 5 narių atvaizdavimas grafiškai 18](#_Toc146740605)

[2.2. Reikalaujamą tikslumą užtikrinantis daugianaris 22](#_Toc146740606)

[2.2.1. Užduotis 22](#_Toc146740607)

[2.2.2. Gautas TE narių skaičius 22](#_Toc146740608)

[2.2.3. Gautas grafikas 23](#_Toc146740609)

[2.2.4. Skaičiavimams ir grafiko braižymui naudotas Python kodas 23](#_Toc146740610)

[2.3. TE analitinė išraiška daugianario pavidalu 25](#_Toc146740611)

[2.3.1. Užduotis 25](#_Toc146740612)

[2.3.2. Gauta išraiška 25](#_Toc146740613)

[2.3.3. Programos kodas reikalingas skaičiavimams 25](#_Toc146740614)

[2.4. Sprendinių gerėjimo grafikai 27](#_Toc146740615)

[2.4.1. Užduotis 27](#_Toc146740616)

[2.4.2. Randamų šaknų skaičius nagrinėjamame intervale 27](#_Toc146740617)

[2.4.2.1. Gautas grafikas 28](#_Toc146740618)

[2.4.2.2. Naudotas kodas 28](#_Toc146740619)

[2.4.3. Tikslumo įverčių ir TE narių ieškojimas kiekvienai šakniai 29](#_Toc146740620)

[2.4.3.1. Gauti grafikai 29](#_Toc146740621)

[2.4.3.2. Naudotas kodas 32](#_Toc146740622)

# Pirma dalis

Išspręskite netiesines lygtis (1 ir 2 lentelės), kai lygties funkcija yra daugianaris 𝑓(𝑥) = 0 ir transcendentinė funkcija 𝑔(𝑥) = 0.

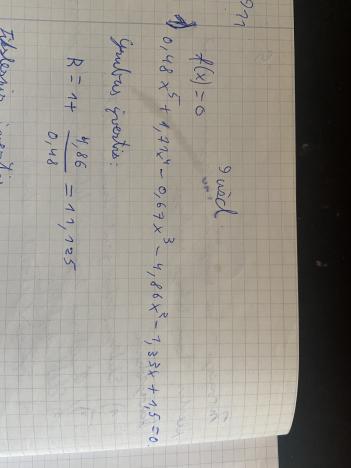


## Nustatykite daugianario 𝑓(𝑥) šaknų intervalą.

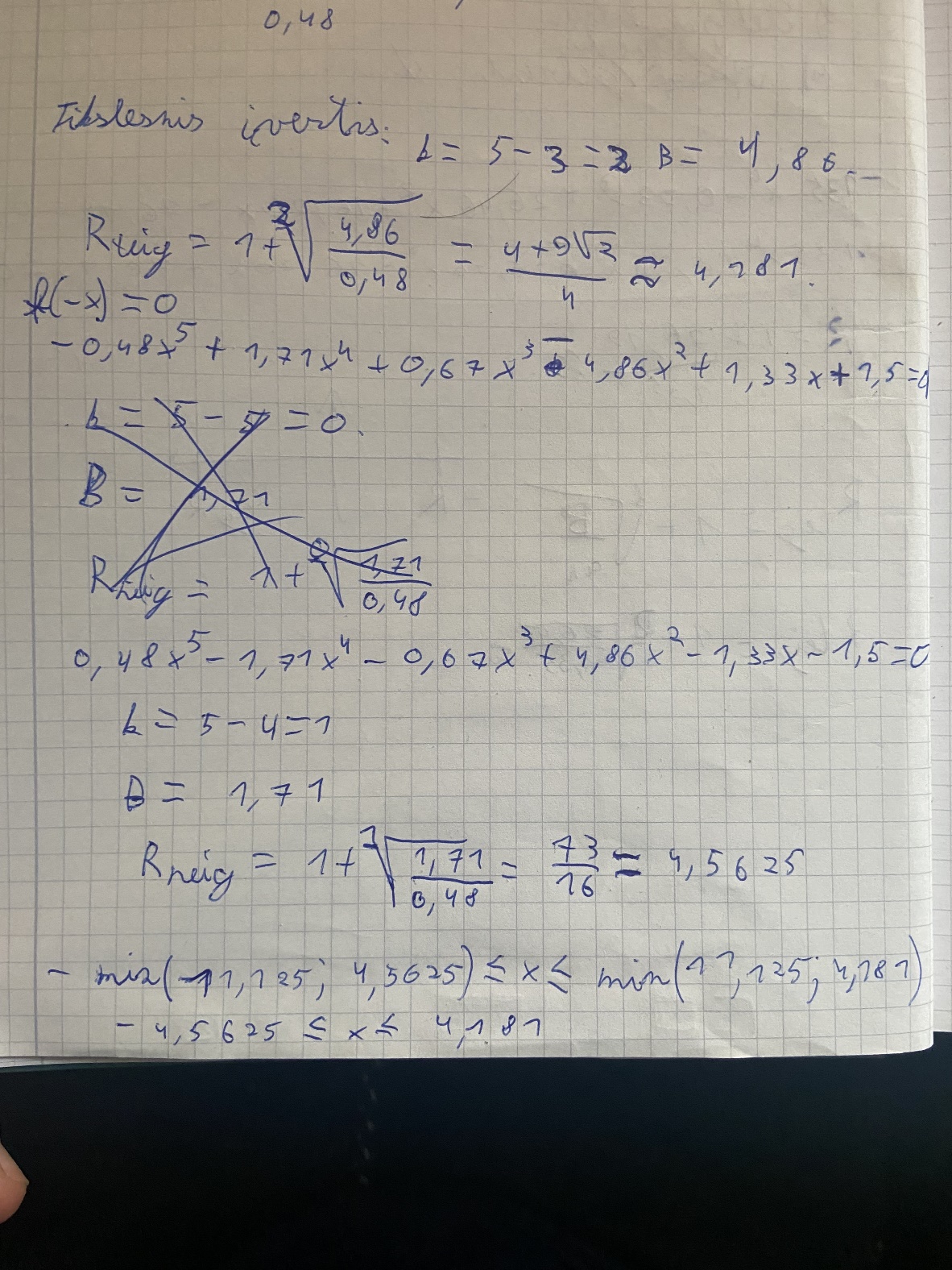
### Užduotis

Nustatykite daugianario 𝑓(𝑥) šaknų intervalą, taikydami „grubų“ ir „tikslesnį“ įverčius. Grafiškai pavaizduokite daugianarį tokiame intervale, kad matytųsi abu įverčiai. Funkciją 𝑔(𝑥) grafiškai pavaizduokite užduotyje nurodytame intervale. Esant poreikiui, grafikų ašis pakeiskite taip, kad būtų aiškiai matomos funkcijų šaknys.

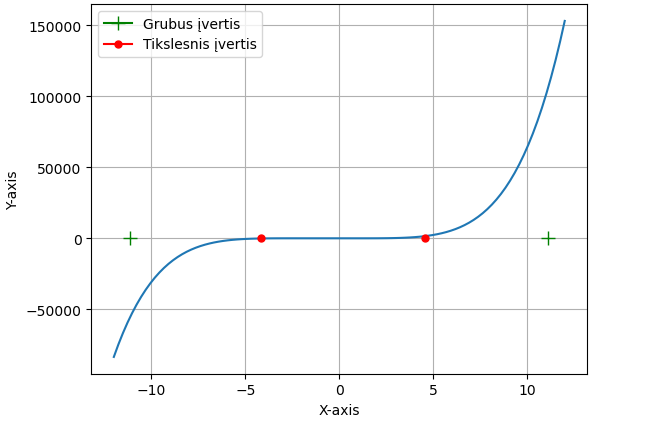
### Grubus įvertis



### Tikslesnis įvertis



### Grafinis vaizdavimas f(x) funkcijos



# funkcia, kuriai ieškome šaknų f(x)

def fx(x):

  return 0.48\*x\*\*5 + 1.71\*x\*\*4 - 0.67\*x\*\*3 - 4.86\*x\*\*2  - 1.33\*x + 1.5

x = np.linspace(-12, 12, 100)

y = fx(x)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.plot(x,y)

plt.grid()

#Grubus įvertis

plt.plot(11.125, 0, markersize=10, marker='+', color = 'green', label = "Grubus įvertis")

plt.plot(-11.125, 0, markersize=10, marker='+', color = 'green')

#Tikslesnis įvertis

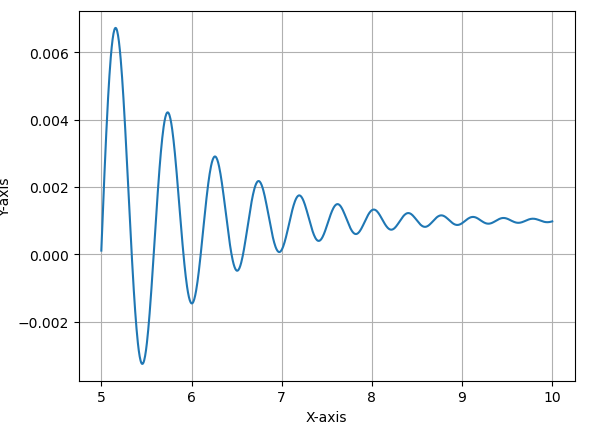
plt.plot(-4.181, 0, markersize=5, marker='o', color='red', label = "Tikslesnis įvertis")

plt.plot(4.5625, 0, markersize=5, marker='o', color='red')

plt.legend()

plt.show()

### Grafinis vaizdavimas g(x) funkcijos



#funkcija g(x)

def gx(x):

  return math.exp(-x) \* math.sin(x\*\*2) + 0.001

g = np.vectorize(gx)

x = np.linspace(5, 10, 1000)

y = g(x)

plt.grid()

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.plot(x,y)

plt.show()

## Skenavimo algoritmo panaudojimas

### Užduotis

Naudodami skenavimo algoritmą su nekintančiu skenavimo žingsniu raskite šaknų atskyrimo intervalus. Daugianariui skenavimo intervalas parenkamas pagal 1 užduoties punkte gautas įverčių reikšmes. Funkcija 𝑔(𝑥) skenuojama užduotyje nurodytame intervale.

### Funkcijos f(x) skenavimo intervalų radimas

#Skenavimo algoritmas

def scaning(func, From, To, arr1, arr2, postumis):

   li = 0

   while (From < To):

      zenkl1 = func(From)

      zenkl2 = func(From + postumis)

      if (np.sign(zenkl1) != np.sign(zenkl2)):

         plt.plot(From, 0, markersize=5, marker='o', color='red')

         plt.plot(From + postumis, 0, markersize=5, marker='o', color='blue')

         arr1.append(From)

         arr2.append(From + postumis)

         li += 1

      From = From + postumis

   print('Total number of intervals: = {0}'.format(li))

#Šaknų atskyrimo intervalai f(x) funkcijai (skenavimo žingsnis - 0.1)

x = np.linspace(-5, 5, 100)

y = fx(x)

plt.plot(x,y)

my\_listStart2 = []

my\_listEnd2 = []

scaning(fx, -4.181, 4.5625, my\_listStart2, my\_listEnd2, 0.1)

plt.ylim([-50000, 50000])

plt.xlim([-5, 5])

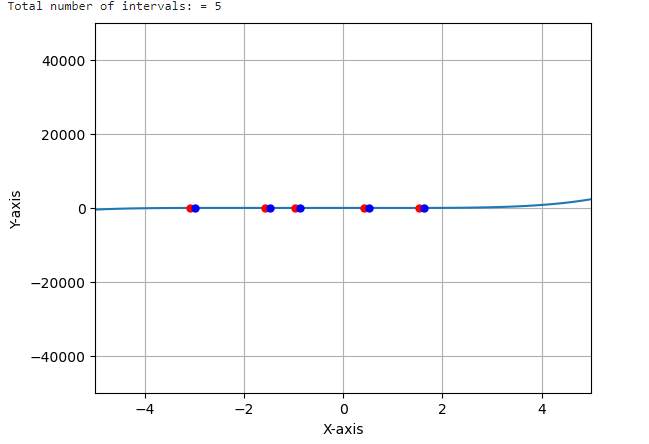
plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.grid()

plt.show()

print('Nuo: {0}  iki {1}'.format(my\_listStart2, my\_listEnd2))





### Funkcijos g(x) skenavimo intervalų radimas

#Šaknų atskyrimo intervalai g(x) funkcijai (skenavimo žingsnis - 0.01)

g = np.vectorize(gx)

x = np.linspace(5, 10, 1000)

y = g(x)

plt.plot(x,y)

my\_listStart3 = []

my\_listEnd3 = []

scaning(g, 5, 10, my\_listStart3, my\_listEnd3, 0.01)

plt.ylim([-0.011, 0.011])

plt.xlim([5, 7])

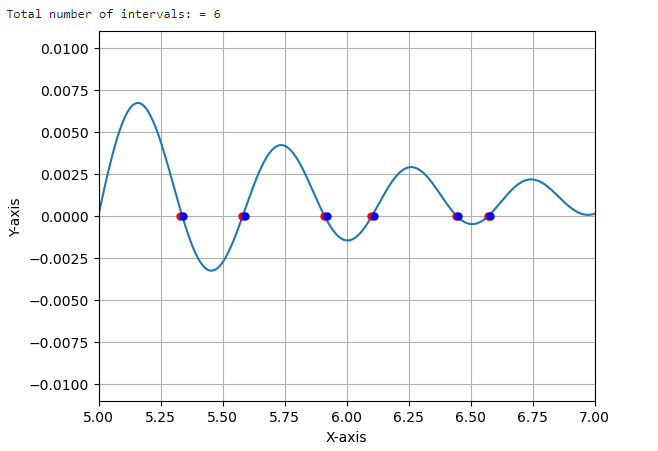
plt.grid()

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.show()

print('Nuo: {0}  iki {1}'.format(my\_listStart3, my\_listEnd3))





## Funkcijų šaknų tikslinimas

### Užduotis

Skenavimo metodu atskirtas daugianario ir funkcijos šaknis tikslinkite užduotyje nurodytais metodais. Skaičiavimo scenarijuje turi būti panaudotos skaičiavimų pabaigos sąlygos. Skaičiavimų rezultatus pateikite lentelėje, kurioje nurodykite šaknies tikslinimui naudojamą metodą, pradinį artinį arba atskyrimo intervalą, gautą sprendinį (šaknį), funkcijos reikšmę ties šaknimi, tikslumą, iteracijų skaičių. Palyginkite, kuriuo metodu sprendiniui rasti panaudota mažiau iteracijų.

### Funkcijos f(x) šaknų radimas pusiaukirtos metodu

# pusiaukirtos metodas

def bisection(func, xFrom, xTo):

    xmid = (xFrom + xTo) / 2

    iter = 0

    while (np.abs(func(xmid)) > 1e-10):

        iter += 1

        if (np.sign(func(xmid)) == np.sign(func(xFrom))):

            xFrom = xmid

        else:

            xTo = xmid

        xmid = (xFrom + xTo) / 2

    print('Total iteration: = {0}'.format(iter))

    return xmid

#Spausdina šaknis pusiaukirtos metodu

def printRootsPusiaukirtos(func, arr1, arr2):

   i=0

   while(i < len(arr1)):

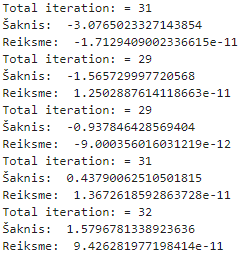
      root = bisection(func, arr1[i], arr2[i])

      print("Šaknis: ", root)

      print("Reiksme: ", func(root))

      i += 1

printRootsPusiaukirtos(fx, my\_listStart2, my\_listEnd2)



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Naudotas metodas** | **Atskirimo intervalas** | **Šaknis** | **Funkcijos reikšmė** | **Tikslumas** | **Iteracijų skaičius** |
| Pusiaukirtos | [-3.0765023327143854;  -2.9809999999999994] | -3.0765023327143854 | -1.7129409002336615e-11 | 1e-10 | 31 |
| Pusiaukirtos | [-1.5809999999999982; -1.480999999999998] | -1.565729997720568 | 1.2502887614118663e-11 | 1e-10 | 29 |
| Pusiaukirtos | [-0.9809999999999978; -0.8809999999999978] | -0.937846428569404 | -9.000356016031219e-12 | 1e-10 | 29 |
| Pusiaukirtos | [0.41900000000000215; 0.5190000000000021] | 0.43790062510501815 | 1.3672618592863728e-11 | 1e-10 | 31 |
| Pusiaukirtos | [1.5190000000000026; 1.6190000000000027] | 1.5796781338923636 | 9.426281977198414e-11 | 1e-10 | 32 |

### Funkcijos f(x) šaknų radimas Niutono (liestinių) metodu

def Niutono(func, dFunc, xFrom):

   iter = 0

   xi = xFrom

   eps = 1e-8

   while np.abs(func(xi)) > eps:

     iter += 1

     xi = xi - 0.5 \* (1 / dFunc(xi)) \* func(xi)

   print('Total iteration: = {0}'.format(iter))

   return xi

def printRootsNiutono(func, dFunc, arr1):

   i=0

   while(i < len(arr1)):

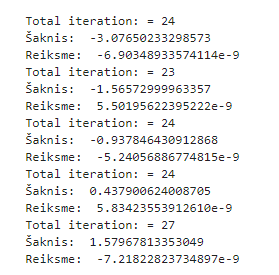
      root = Niutono(func, dFunc, arr1[i])

      print("Šaknis: ", root)

      print("Reiksme: ", func(root))

      i += 1

printRootsNiutono(fx, dfx, my\_listStart2)



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Naudotas metodas** | **Pradinis artinys** | **Šaknis** | **Funkcijos reikšmė** | **Tikslumas** | **Iteracijų skaičius** |
| Niutono(liesninių) | -3.0809999999999995 | -3.07650233298573 | -6.90348933574114e-9 | 1e-8 | 24 |
| Niutono(liesninių) | -1.5809999999999982 | -1.56572999963357 | 5.50195622395222e-9 | 1e-8 | 23 |
| Niutono(liesninių) | -0.9809999999999978 | -0.937846430912868 | -5.24056886774815e-9 | 1e-8 | 24 |
| Niutono(liesninių) | 0.41900000000000215 | 0.437900624008705 | 5.83423553912610e-9 | 1e-8 | 24 |
| Niutono(liesninių) | 1.5190000000000026 | 1.57967813353049 | -7.21822823734897e-9 | 1e-8 | 27 |

### Funkcijos g(x) šaknų radimas pusiaukirtos metodu

# pusiaukirtos metodas

def bisection(func, xFrom, xTo):

    xmid = (xFrom + xTo) / 2

    iter = 0

    while (np.abs(func(xmid)) > 1e-10):

        iter += 1

        if (np.sign(func(xmid)) == np.sign(func(xFrom))):

            xFrom = xmid

        else:

            xTo = xmid

        xmid = (xFrom + xTo) / 2

    print('Total iteration: = {0}'.format(iter))

    return xmid

#Spausdina šaknis pusiaukirtos metodu

def printRootsPusiaukirtos(func, arr1, arr2):

   i=0

   while(i < len(arr1)):

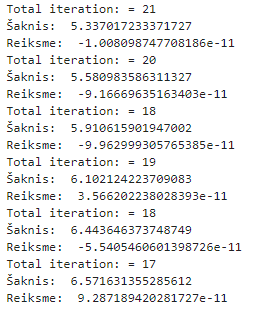
      root = bisection(func, arr1[i], arr2[i])

      print("Šaknis: ", root)

      print("Reiksme: ", func(root))

      i += 1

printRootsPusiaukirtos(g, my\_listStart3, my\_listEnd3)



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Naudotas metodas** | **Atskirimo intervalas** | **Šaknis** | **Funkcijos reikšmė** | **Tikslumas** | **Iteracijų skaičius** |
| Pusiaukirtos | [5.329999999999993; 5.339999999999993  ] | 5.337017233371727 | -1.008098747708186e-11 | 1e-10 | 21 |
| Pusiaukirtos | [5.579999999999988  ; 5.589999999999987  ] | 5.580983586311327 | -9.16669635163403e-11 | 1e-10 | 20 |
| Pusiaukirtos | [5.909999999999981  ; 5.91999999999998  ] | 5.910615901947002 | -9.962999305765385e-11 | 1e-10 | 18 |
| Pusiaukirtos | [6.0999999999999766  ; 6.109999999999976  ] | 6.102124223709083 | 3.566202238028393e-11 | 1e-10 | 19 |
| Pusiaukirtos | [6.439999999999969; 6.449999999999969  ] | 6.443646373748749 | -5.5405460601398726e-11 | 1e-10 | 18 |
| Pusiaukirtos | [6.5699999999999665  ; 6.579999999999966  ] | 6.571631355285612 | 9.287189420281727e-11 | 1e-10 | 17 |

### Funkcijos g(x) šaknų radimas Niutono (liestinių) metodu

def Niutono(func, dFunc, xFrom):

   iter = 0

   xi = xFrom

   eps = 1e-8

   while np.abs(func(xi)) > eps:

     iter += 1

     xi = xi - 0.5 \* (1 / dFunc(xi)) \* func(xi)

   print('Total iteration: = {0}'.format(iter))

   return xi

def printRootsNiutono(func, dFunc, arr1):

   i=0

   while(i < len(arr1)):

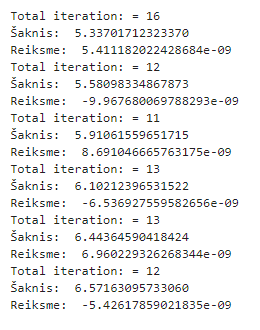
      root = Niutono(func, dFunc, arr1[i])

      print("Šaknis: ", root)

      print("Reiksme: ", func(root))

      i += 1

printRootsNiutono(g, dgx, my\_listStart3)



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Naudotas metodas** | **Pradinis artinys** | **Šaknis** | **Funkcijos reikšmė** | **Tikslumas** | **Iteracijų skaičius** |
| Niutono(liesninių) | 5.329999999999993 | 5.33701712323370 | 5.411182022428684e-09 | 1e-8 | 16 |
| Niutono(liesninių) | 5.579999999999988 | 5.58098334867873 | -9.967680069788293e-09 | 1e-8 | 12 |
| Niutono(liesninių) | 5.909999999999981 | 5.91061559651715 | 8.691046665763175e-09 | 1e-8 | 11 |
| Niutono(liesninių) | 6.0999999999999766 | 6.10212396531522 | -6.536927559582656e-09 | 1e-8 | 13 |
| Niutono(liesninių) | 6.439999999999969 | 6.44364590418424 | 6.960229326268344e-09 | 1e-8 | 13 |
| Niutono(liesninių) | 6.5699999999999665 | 6.57163095733060 | -5.42617859021835e-09 | 1e-8 | 12 |

### Iteracijų palyginimas

Funkcijos f(x) ir funkcijos g(x) iteracijų skaičius buna mažesnis naudojant Niutono (liestinių) metodą, nes naudodamas pusiaukirtos metodą pasirinkau didesnį tikslumą.

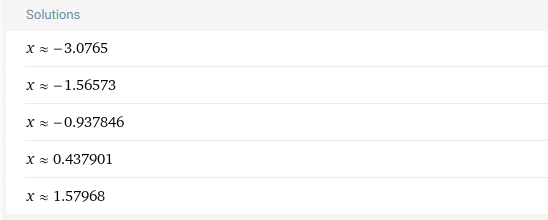
## Gautų šaknų tikrinimas wolframalpha.com tinklapyje

### Užduotis

Gautas šaknų reikšmes patikrinkite naudodami išorinius išteklius ( funkcijas roots arba fzero, tinklapį wolframalpha.com arba kitas priemones) ir pateikite patikrinimo rezultatus.

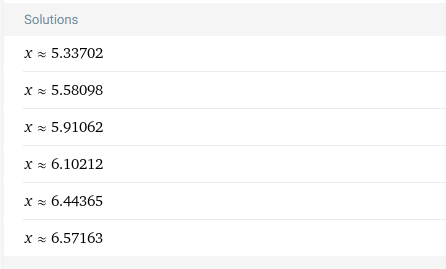
### Funkcijos f(x) šaknų patikrinimo rezultatai





### Funkcijos g(x) šaknų patikrinimo rezultatai





# Antra dalis

3 lentelėje pateiktą funkciją ℎ(𝑥) išskleiskite Teiloro eilute (TE) nurodyto intervalo vidurio taško aplinkoje. Nustatykite TE narių skaičių, su kuriuo visos TE šaknys esančios nurodytame intervale, skiriasi nuo funkcijos ℎ(𝑥) šaknų ne daugiau negu |1e-4|. Tiek pateiktos funkcijos ℎ(𝑥) šaknis, tiek TE šaknis raskite antru iš pirmoje dalyje realizuotų skaitinių metodų (Niutono arba Kvazi-Niutono, priklausomai nuo varianto). Darbo ataskaitoje pateikite:



## Tarpinių grafikų sudarymas (3, 4 ir 5 TE narių skaičius)

### Užduotis

Tarpinius grafikus, kai drauge su pateikta funkcija ℎ(𝑥) nurodytame intervale atvaizduojama TE, kai jos narių skaičius lygus 3, 4 ir 5.

### Funkcijos h(x) ir TE 3, 4 ir 5 narių atvaizdavimas grafiškai

def fx(x):

   return 2 \* np.cos(x) - 47 \* np.cos(2\*x) + 2 # -6 ir 0

def df1(x):

  return 2 \* (-np.sin(x)) + 94 \* np.sin(2\*x)

def df2(x):

  return 2 \* (-np.cos(x)) + 188 \* np.cos(2\*x)

def df3(x):

  return  2 \* (np.sin(x)) - 376 \* np.sin(2\*x)

def df4(x):

  return 2 \* (np.cos(x)) - 752 \* np.cos(2\*x)

def df5(x):

  return 2 \* (-np.sin(x)) + 1504 \* np.sin(2\*x)

def ts3(x, x0):

  return fx(x0) + (x-x0)\*df1(x0) + (np.power((x-x0), 2) / 2 ) \*df2(x0) + (np.power((x-x0), 3) / (3\*2)) \*df3(x0)

def ts4(x, x0):

  return fx(x0) + (x-x0)\*df1(x0) + (np.power((x-x0), 2) / 2 ) \*df2(x0) + (np.power((x-x0), 3) / (3\*2)) \*df3(x0) + (np.power((x-x0), 4) / (4\*3\*2)) \*df4(x0)

def ts5(x, x0):

  return fx(x0) + (x-x0)\*df1(x0) + (np.power((x-x0), 2) / 2 ) \*df2(x0) + (np.power((x-x0), 3) / (3\*2)) \*df3(x0) + (np.power((x-x0), 4) / (4\*3\*2)) \*df4(x0) + (np.power((x-x0), 5) / (5\*4\*3\*2)) \*df5(x0)

dx= 0.1

x=np.arange(-6, 0+dx, dx)

y = fx(x)

#3 Teiloro eilute

plt.xlim([-6, 0]); plt.ylim([-100, 100])

x0 = -3; # vidurio taskas

tsy3 = ts3(x, x0)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.plot(x, y, 'b',  linewidth=3.0) # pagrindine funkcija

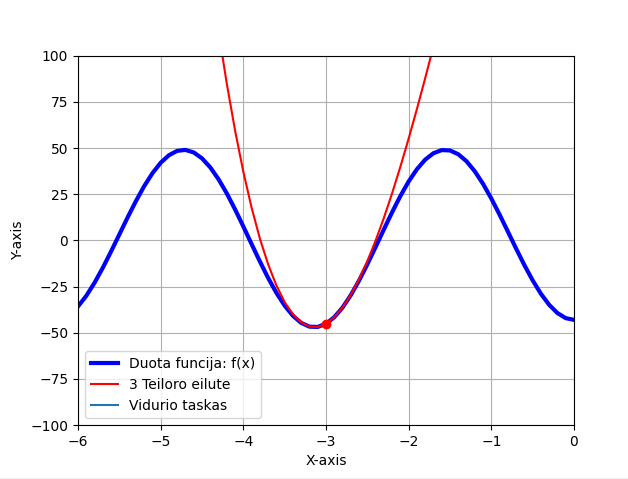
plt.plot(x, tsy3, 'r', 1) # 3 order

plt.plot(x0, fx(x0), 'or') # vidurio tasko atvaizdavimas

plt.legend(['Duota funcija: f(x)', '3 Teiloro eilute', 'Vidurio taskas'])

plt.grid()

plt.show()



#4 Teiloro eilute

plt.xlim([-6, 0]); plt.ylim([-100, 100])

x0 = -3; # vidurio taskas

tsy4 = ts4(x, x0)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.plot(x, y, 'b',  linewidth=3.0) # pagrindine funkcija

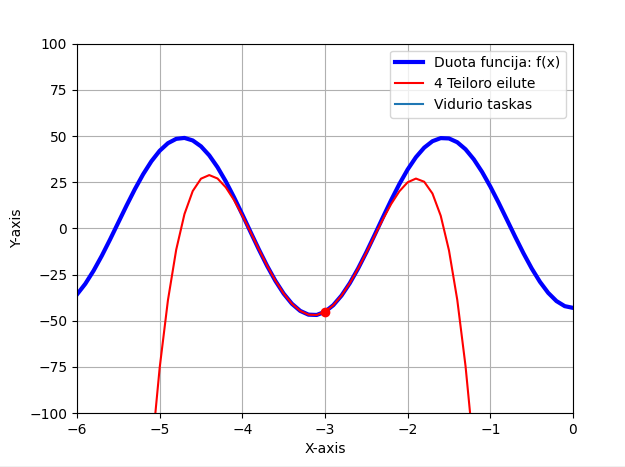
plt.plot(x, tsy4, 'r', 1) # 4 order

plt.plot(x0, fx(x0), 'or') # vidurio tasko atvaizdavimas

plt.legend(['Duota funcija: f(x)', '4 Teiloro eilute', 'Vidurio taskas'])

plt.grid()

plt.show()



#5 Teiloro eilute

plt.xlim([-6, 0]); plt.ylim([-100, 100])

x0 = -3; # vidurio taskas

tsy5 = ts5(x, x0)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.plot(x, y, 'b',  linewidth=3.0) # pagrindine funkcija

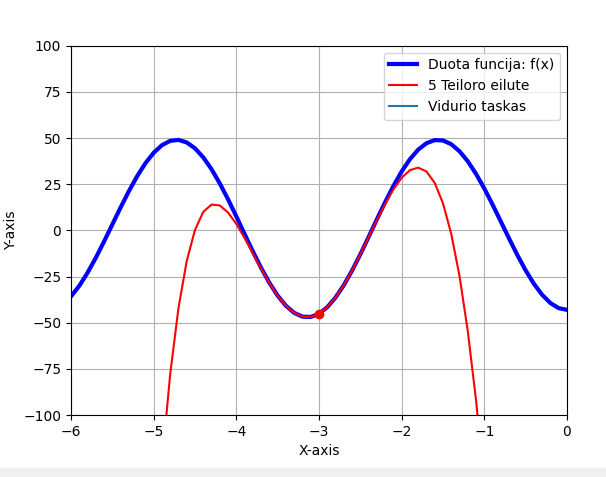
plt.plot(x, tsy5, 'r', 1) # 5 order

plt.plot(x0, fx(x0), 'or') # vidurio tasko atvaizdavimas

plt.legend(['Duota funcija: f(x)', '5 Teiloro eilute', 'Vidurio taskas'])

plt.grid()

plt.show()



## Reikalaujamą tikslumą užtikrinantis daugianaris

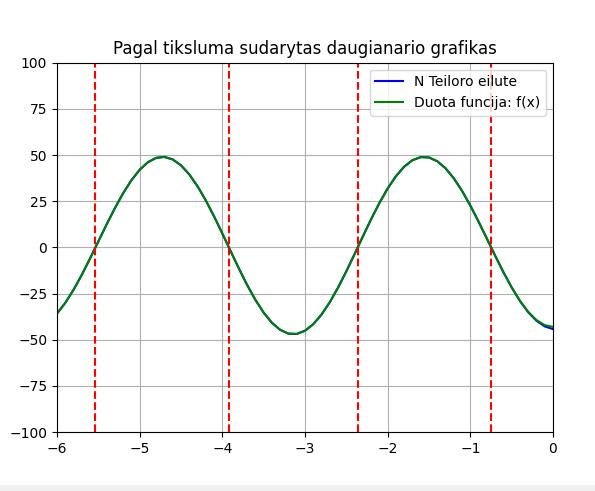
### Užduotis

Grafiką, kuriame pavaizduotas reikalaujamą tikslumą užtikrinantis pagal TE sudarytas daugianaris, drauge pateikiant ir funkcijos ℎ(𝑥) grafiką.

### Gautas TE narių skaičius

TE nariu skaicius: 17

### Gautas grafikas



### Skaičiavimams ir grafiko braižymui naudotas Python kodas

#Pradinės funkcijos šaknys

def hSaknys(h\_Saknys, Artiniai):

   i = 0

   while(i < len(Artiniai)):

      root = Niutono(fx, df1, Artiniai[i])

      h\_Saknys.append(root)

      print(root)

      i += 1

#Teiloro funkcija

def Taylor(f, N):

    x, f, fp = sp.symbols(('x','f','fp'))

    f = 2 \* sp.cos(x) - 47 \* sp.cos(2\*x) + 2 #Turima funkcija

    fp = f.subs(x, -3) #Pirmasis TE narys

    for i in range(1, N):

        f=f.diff(x)

        fp = fp + f.subs(x, -3)/math.factorial(i)\*(x+3)\*\*i

        i += 1

        N += 1

    return fp

#Šaknų lyginimas pagal 1e-4 tikslumą

def SaknuPanasumas(saknis, mas):

   x, fp = sp.symbols(('x','fp'))

   f\_prad = 2 \* sp.cos(x) - 47 \* sp.cos(2\*x) + 2 #Pradine funkcija, kuri nekis

   i = 2

   fp = Taylor(f\_prad, 1)

   mas.append(0)

   fp = Taylor(f\_prad, 2)

   mas.append(np.abs(saknis - Niutono(sp.lambdify(x, fp, modules=['numpy']), sp.lambdify(x, fp.diff(x), modules=['numpy']), saknis)))

   while (np.abs(saknis - Niutono(sp.lambdify(x, fp, modules=['numpy']), sp.lambdify(x, fp.diff(x), modules=['numpy']), saknis)) >= 1e-4):

      i += 1

      fp = Taylor(f\_prad, i)

      mas.append(np.abs(saknis - Niutono(sp.lambdify(x, fp, modules=['numpy']), sp.lambdify(x, fp.diff(x), modules=['numpy']), saknis)))

   return i, fp

VisiTESkaiciai = []

Artiniai = [-5.5, -4, -2.5, -0.5]

h\_Saknys = []

hSaknys(h\_Saknys, Artiniai)

mas = [] #Nereikalingas

maximum = 0

for saknis in h\_Saknys:

   SkaiciusTE, Daugianaris = SaknuPanasumas(saknis, mas)

   VisiTESkaiciai.append(SkaiciusTE)

   print(SkaiciusTE)

   if SkaiciusTE > maximum:

      maximum = SkaiciusTE

      DidziausiasDaugianaris = Daugianaris

print("TE nariu skaicius: ", maximum)

#Grafiko h(x) ir N-tosios Teiloro eilutes braizymas

x = sp.symbols('x')

dx= 0.1

f\_prad = 2 \* sp.cos(x) - 47 \* sp.cos(2\*x) + 2

xvalue=np.arange(-6, 0+dx, dx)

y\_vals = [DidziausiasDaugianaris.subs(x, val) for val in xvalue]

y\_vals\_f\_prad = [f\_prad.subs(x, val) for val in xvalue]

plt.plot(xvalue, y\_vals, 'b')

plt.plot(xvalue, y\_vals\_f\_prad,  'g', 1)

plt.legend(['N Teiloro eilute', 'Duota funcija: f(x)'])

plt.ylim(-100, 100)

plt.xlim(-6, 0)

for saknis in h\_Saknys:

      plt.axvline(x=saknis, color='r', linestyle='--')

plt.title('Pagal tiksluma sudarytas daugianario grafikas')

plt.grid()

plt.show()

## TE analitinė išraiška daugianario pavidalu

### Užduotis

Nustatytos reikalaujamą tikslumą užtikrinančios TE analitinę išraišką daugianario pavidalu.

### Gauta išraiška

TE daugianario pavidalas: (x + 3)\*\*16\*(-94\*cos(6)/638512875 + cos(3)/10461394944000) + (x + 3)\*\*15\*(752\*sin(6)/638512875 - sin(3)/653837184000) + (x + 3)\*\*14\*(-cos(3)/43589145600 + 376\*cos(6)/42567525) + (x + 3)\*\*13\*(sin(3)/3113510400 -

376\*sin(6)/6081075) + (x + 3)\*\*12\*(-188\*cos(6)/467775 + cos(3)/239500800) + (x + 3)\*\*11\*(376\*sin(6)/155925 - sin(3)/19958400) + (x + 3)\*\*10\*(-cos(3)/1814400 + 188\*cos(6)/14175) + (x + 3)\*\*9\*(sin(3)/181440 - 188\*sin(6)/2835) + (x + 3)\*\*8\*(-94\*cos(6)/315 + cos(3)/20160) + (x + 3)\*\*7\*(376\*sin(6)/315 - sin(3)/2520) + (x + 3)\*\*6\*(-cos(3)/360 + 188\*cos(6)/45) + (x + 3)\*\*5\*(sin(3)/60 - 188\*sin(6)/15) + (x + 3)\*\*4\*(-94\*cos(6)/3 + cos(3)/12) + (x + 3)\*\*3\*(188\*sin(6)/3 - sin(3)/3)

+ (x + 3)\*\*2\*(-cos(3) + 94\*cos(6)) + (x + 3)\*(2\*sin(3) - 94\*sin(6)) - 47\*cos(6) + 2\*cos(3) + 2

### Programos kodas reikalingas skaičiavimams

#Pradinės funkcijos šaknys

def hSaknys(h\_Saknys, Artiniai):

   i = 0

   while(i < len(Artiniai)):

      root = Niutono(fx, df1, Artiniai[i])

      h\_Saknys.append(root)

      print(root)

      i += 1

#Teiloro funkcija

def Taylor(f, N):

    x, f, fp = sp.symbols(('x','f','fp'))

    f = 2 \* sp.cos(x) - 47 \* sp.cos(2\*x) + 2 #Turima funkcija

    fp = f.subs(x, -3) #Pirmasis TE narys

    for i in range(1, N):

        f=f.diff(x)

        fp = fp + f.subs(x, -3)/math.factorial(i)\*(x+3)\*\*i

        i += 1

        N += 1

    return fp

#Šaknų lyginimas pagal 1e-4 tikslumą

def SaknuPanasumas(saknis, mas):

   x, fp = sp.symbols(('x','fp'))

   f\_prad = 2 \* sp.cos(x) - 47 \* sp.cos(2\*x) + 2 #Pradine funkcija, kuri nekis

   i = 2

   fp = Taylor(f\_prad, 1)

   mas.append(0)

   fp = Taylor(f\_prad, 2)

   mas.append(np.abs(saknis - Niutono(sp.lambdify(x, fp, modules=['numpy']), sp.lambdify(x, fp.diff(x), modules=['numpy']), saknis)))

   while (np.abs(saknis - Niutono(sp.lambdify(x, fp, modules=['numpy']), sp.lambdify(x, fp.diff(x), modules=['numpy']), saknis)) >= 1e-4):

      i += 1

      fp = Taylor(f\_prad, i)

      mas.append(np.abs(saknis - Niutono(sp.lambdify(x, fp, modules=['numpy']), sp.lambdify(x, fp.diff(x), modules=['numpy']), saknis)))

   return i, fp

VisiTESkaiciai = []

Artiniai = [-5.5, -4, -2.5, -0.5]

h\_Saknys = []

hSaknys(h\_Saknys, Artiniai)

mas = [] #Nereikalingas

maximum = 0

for saknis in h\_Saknys:

   SkaiciusTE, Daugianaris = SaknuPanasumas(saknis, mas)

   VisiTESkaiciai.append(SkaiciusTE)

   print(SkaiciusTE)

   if SkaiciusTE > maximum:

      maximum = SkaiciusTE

      DidziausiasDaugianaris = Daugianaris

print("TE daugianario pavidalas: ", DidziausiasDaugianaris)

## Sprendinių gerėjimo grafikai

### Užduotis

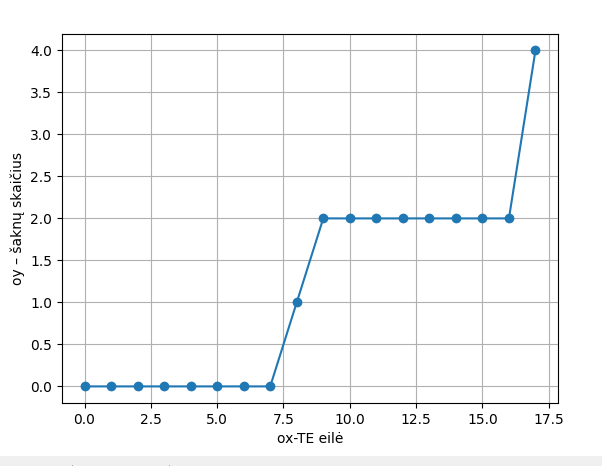
Grafikus, pagal kuriuos būtų galima įvertinti, kaip gerėjo sprendinys priklausomai nuo TE narių skaičiaus.

a) grafikas, kuris nurodo visą randamų šaknų skaičių nagrinėjamame intervale (ox-TE eilė, oy – šaknų skaičius);

b) atskiri grafikai kiekvienai šakniai, kuriuose oy ašyje pateikti tikslumo įverčiai tarp ℎ(𝑥) apskaičiuotos šaknies ir artimiausios TE šaknies, o ox ašyje TE narių skaičiai.

### Randamų šaknų skaičius nagrinėjamame intervale

#### Gautas grafikas



#### Naudotas kodas

#Gaunamas bendras šaknų kiekis prilausantis nuo TE eilių

def GetSaknysTEskaiciu(masTE, SaknuSk):

   i = 0

   z = 0

   while (i < len(masTE)):

      j = 0

      if (i + 1 < len(masTE)):

          while (j <= (masTE[i + 1] - 1) - masTE[i]):

            SaknuSk.append(i)

            z += 1

            j += 1

      i += 1

   SaknuSk.append(i - 1)

#4. a dalis

saknuSk = []

VisiTESkaiciai = VisiTESkaiciai + [0]

VisiTESkaiciai.sort()

GetSaknysTEskaiciu(VisiTESkaiciai, saknuSk) #Gražinamas saknuSk masyvas

y\_values = np.linspace(0, 4, 100)

plt.figure()

plt.plot(range(0, maximum + 1), saknuSk, marker='o', label='tikslumai')

plt.xlabel('ox-TE eilė')

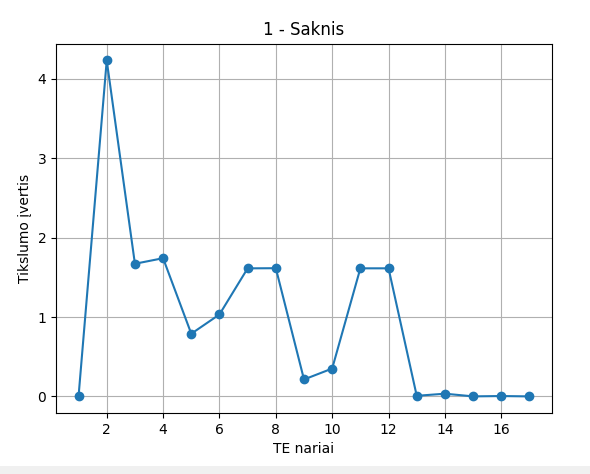
plt.ylabel('oy – šaknų skaičius')

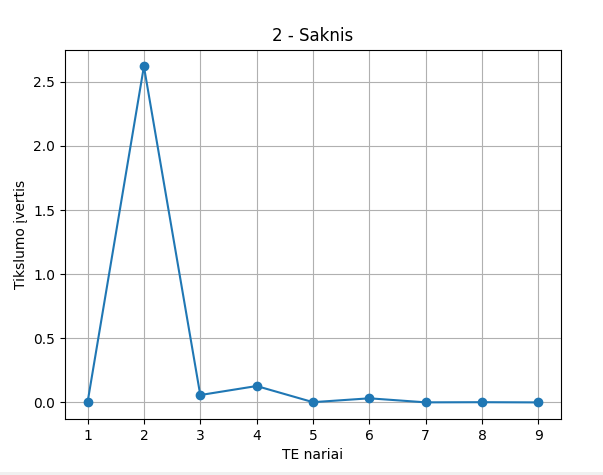
plt.grid(True)

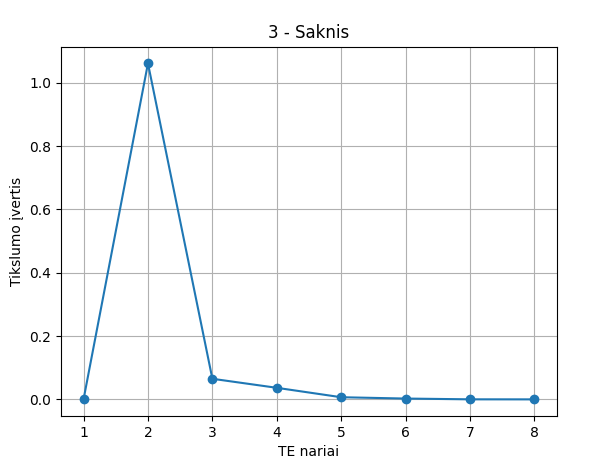
plt.show()

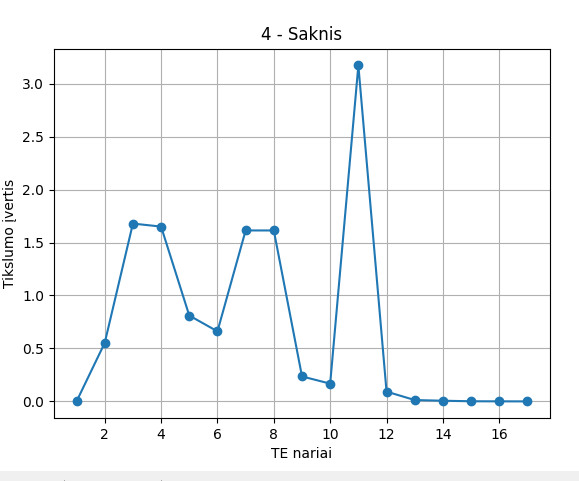
### Tikslumo įverčių ir TE narių ieškojimas kiekvienai šakniai

#### Gauti grafikai









#### Naudotas kodas

#4. b dalis

i = 1

for saknis in h\_Saknys:

   tikslumuMas = []

   SkaiciusTE, daugianaris = SaknuPanasumas(saknis, tikslumuMas)

   # Atskiras grafikas kiekvienai šakniai

   plt.figure()

   plt.plot(range(1, SkaiciusTE + 1), tikslumuMas, marker='o', label='tikslumai')

   plt.xlabel('TE nariai')

   plt.ylabel('Tikslumo įvertis')

   plt.title('{0} - Saknis'.format(i))

   plt.grid(True)

   i += 1

plt.show()