Štrāzena algoritms

2.izcilības (desmitnieka) uzdevums

Sastādīt programmu, kas realizē matricu ātro reizināšanas (Štrāzena) algoritmu.

Kods:

```
# Programmas nosaukums: Štrāzena algoritms

# 2.izcilības (desmitnieka) uzdevums

# Uzdevuma formulējums: Sastādīt programmu, kas realizē matricu ātro reizināšanas (Štrāzena) algoritmu.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



"""

Kods realizē Štrāzena algoritmu kvadrātisku matricu reizināšanai.

Šis algoritms sadala matricas reizināšanu mazākās matricās un izmanto rekursiju.

"""



import numpy as np



def pad_matrix(M):

    # Funkcija aprēķina ar kādiem izmēriem jābut jaunai matricai, lai izmērs būtu pakāpē 2.

    # Atgriež sakotnēju matricu kreisā augšēja stūri un pēdejas rindas/kolonnas varētu būt nulles, ja bija nepieciešāms matricu pielāgot.



    (a, b) = M.shape



    # Uzzinām kurš ir garāks. Matricas garums vai platums ir garāks.
```

```
    size = a
    if b > size:
        size = b

    # Pārbaudam, vai izmērs ir pakāpē 2 (vai to var sadalīt, vai būs nepieciešāms likt liekas nulles, lai algoritms strādātu).
    is_power_of_two = (size - 1)
    if size == 0 and is_power_of_two == 0:
        is_power_of_two = True
    else:
        is_power_of_two = False

    # Ja nepieciešams, tad izrēķinām cik ir nepieciešām likt liekas rindiņas/kolonnas (size), lai izmērs ir pakāpē 2.
    if not is_power_of_two:
        power_of_two = 1
        while power_of_two < size:
            power_of_two = power_of_two * 2
        size = power_of_two

    # Izveidojam matricu ar nepieciešamam liekam nullem, lai matricas izmērs ir pakāpē 2.
    padded = np.zeros((size, size))  # Izveidojam jaunu kvadrātveida matricu ar nullēm ar nepieciešamiem izmēriem (izmērs kura pakāpē ir kāda 2).
    padded[:a, :b] = M  # Ievietojam ievadīto sakotnēju matricu M jaunās matricas augšējā kreisajā stūrī.

    return padded


def strassen(A, B):
    # Funkcija, lai veiktu matricas reizināšanu, izmantojot Štrāzena algoritmu.
    # Atgriež matricu C = A*B pēc Štrāzena algoritma. Varētu rasties liekas rindas/kolonnas, tāpēc pēc tam noņemam tos, ja ir vajadzīgi.
```

```
    # Pievienojam nulles matricam, ja ir vajadzīgi, lai matricas izmērs ir pakāpē 2 (izmērs kura pakāpē ir kāda 2).
    A = pad_matrix(A)
    B = pad_matrix(B)

    # Ja matricas izmērs ir 1x1, atgriežam reizinājumu (rekursijas visdziļāka iespējama situācija).
    # Rekursijas tā sauksim "pamatgadījums".
    n = A.shape[0]
    if n == 1:
        return A * B

    half = n // 2

    # Sadalam matricu "kvadrantos".

    # Matricai A
    # Paņemam pusi "pirmās" rindas un pusi "pirmās" kolonnas.
    a = A[:half, :half]
    # Paņemam pusi "pirmās" rindas un pusi "otrās" kolonnas.
    b = A[:half, half:]
    # Paņemam pusi "otrās" rindas un pusi "pirmās" kolonnas.
    c = A[half:, :half]
    # Paņemam pusi "otrās" rindas un pusi "otras" kolonnas.
    d = A[half:, half:]

    # Matricai B
    # Paņemam pusi "pirmās" rindas un pusi "pirmās" kolonnas.
    e = B[:half, :half]
    # Paņemam pusi "pirmās" rindas un pusi "otrās" kolonnas.
    f = B[:half, half:]
```

```
    # Paņemam pusi "otrās" rindas un pusi "pirmās" kolonnas.
    g = B[half:, :half]
    # Paņemam pusi "otrās" rindas un pusi "otras" kolonnas.
    h = B[half:, half:]

    # Veicam Štrāzena reizināšanu uz mazākām matricām rekursīvi.
    p1 = strassen(a, f - h)
    p2 = strassen(a + b, h)
    p3 = strassen(c + d, e)
    p4 = strassen(d, g - e)
    p5 = strassen(a + d, e + h)
    p6 = strassen(b - d, g + h)
    p7 = strassen(a - c, e + f)

    r = p5 + p4 - p2 + p6
    s = p1 + p2
    t = p3 + p4
    u = p5 + p1 - p3 - p7

    C = np.empty((n, n))
    C[:half, :half] = r
    C[:half, half:] = s
    C[half:, :half] = t
    C[half:, half:] = u

    return C  # Atgriež matricu C = A*B pēc Strasena algoritma. Varētu rasties liekas rindas/kolonnas, tāpēc pēc tam noņemam tos, ja ir vajadzīgi.



def unpad_matrix(M, row_count, col_count):
    # Atgriež matricu M bet ar nodzēstam pēdējam rindam row_count skaitā un ar nodzēstam pēdējam kolonnam col_count skaitā.
```

```
    # M - matrica, kurai gribam noņemt row_count rindas skaitu un col_count kolonnas skaitu.
    # row_count - cik rindas gribam noņemt no matricas M (pēdējas rindas).
    # col_count - cik kolonnas gribam noņemt no matricas M (pēdējas kolonnas).

    if row_count == 0 and col_count == 0:  # Ja rindas un kolonnu skaits, kuru gribam nodzēst ir 0, tad atgriežām sakotnēju matricu.
        return M

    # Nosakam pēdējo rindu un kolonnu skaitu, cik vajadzīgi "unpad" (nodzēst liekas nulles rindas/kolonnas), pamatojoties uz ievadītiem skaitliem row_count un col_count.
    last_row = M.shape[0] - row_count  # Nosakam pēdējo rindu skaitu, atņemot ievadītu rindu skaitu (nonēmam tik, cik tika ievadīts row_count'ā).
    last_col = M.shape[1] - col_count  # Nosakam pēdējo kolonnu skaitu, atņemot ievadītu kolonnu skaitu (nonēmam tik, cik tika ievadīts col_count'ā).

    # Noņemam norādīto rindu un kolonnu skaitu
    unpadded = M[:last_row, :last_col]

    return unpadded


def convert_matrix_elements_to_real(matrix):
    # Konvertējam matricas elementus reālos skaitļos (decimal ar punktu).
    # Atgriež matricu, kurai visi elementi ir float.
    # matrix - matrica, kuras visas vērtības būs konvērtētas float skaitļos.

    return np.array(matrix, dtype=float)


def convert_matrix_elements_to_int(matrix):
    # Konvertējam matricas elementus veselos skaitļos (int).
    # Atgriež matricu, kurai visi elementi ir int.
```

```
    # matrix - matrica, kuras visas vērtības būs konvērtētas int skaitļos.

    return np.array(matrix, dtype=int)


def create_matrix():
    # Metode, kas prasa lietotājam ievadīt kvadrātiskas matricas izmēru un prasa ievadīt katru matricas elementu.
    # Atgriež kvadrātisku matricu ar visām vertībam, kuru ievadīja lietotājs.

    # Prasa lietotājam ievadīt kvadrātiskas matricas izmēru.
    n = int(input("Ievadiet kvadrātiskas matricas izmēru ==> "))

    # jo vajadzīgi pielāgot ar nulles rindam/kolonnam un pēc tam viņus noņemt, tātad liekas darbības būs.
    # Arī nav ieteicams izmantot algoritmu ja matricas ir vajadzīgi pielagot (labāk uzreiz ievadīt, kā pielāgotus).
    # Jo tas varētu aizņēmt laiku ļoti lielām matricam.

    matrix = np.zeros((n, n))  # Tukša matrica ar ievadītiem izmēriem

    # Prasa lietotājam ievadīt katru vērtību matricā.
    for i in range(n):
        for j in range(m):
            matrix[i][j] = float(input(f"Ievadiet skaitlisko vērtību elementam ({i+1},{j+1}) ==> "))  # varam arī prasīt int ievādīt.

    return matrix


def row_number(matrix):
    # Atgriež kopējo rindu skaitu matrica matrix.
```

```
    # matrix - divdimensijas numpy masīvs.

    # Izveidojam mainīgo, lai saskaitītu rindas skaitu.
    row_count = 0
    for row in matrix:  # Eterejam cauri rindam un skaitam tos.
        row_count += 1

    # Atgriež kopējo rindu skaitu matricā.
    return row_count



def column_number(matrix):
    # Atgriež kopējo kolonnu skaitu matrica matrix.
    # matrix - divdimensijas numpy masīvs.

    # Izveidojam mainīgo, lai saskaitītu kolonnu skaitu.
    column_count = 0
    # Izvēlāmies pirmo rindu
    row = matrix[0]
    # Ejam cauri katras rindas elementam
    for column in row:
        # Katrs rindas elements apzīmē vienu kolonnu, tāpēc palielinam skaitu par vienu
        column_count += 1

    # Atgriež kopējo kolonnu skaitu matricā.
    return column_count



def result_with_padding(A, B, rinda_sk_A, kolonnu_sk_A, rinda_sk_B, kolonnu_sk_B):
    # Atgriež matricu bez liekam rindam un kolonnam, kura tika izveidota ar Štrāzena
algoritmu reizinot A ar B.
```

```
    # Atgriež arī, ka visi elementi ir int, to var noņemt pēc vajadzības. (!)
    # rinda_sk_A - rindas skaits matrica A
    # kolonnu_sk_A - kolonnu skaits matrica A
    # rinda_sk_B - rindas skaits matrica B
    # kolonnu_sk_B - kolonnu skaits matrica B
    # A - matrica A
    # B - matrica B
    # Atgriež matricu C = A*B (pēc Štrāzena algoritma) bez liekam nullem.

    if kolonnu_sk_A == rinda_sk_B:  # Tikai tad varam sareizināt divas matricas (ja rindas sk 1. matrica ir vienāds ar kolonnu sk 2. matrica).
        strasen = strassen(A, B)  # Izmantojam Štrāzena algoritmu.

        rinda_sk_strasen = row_number(strasen)  # Noteicam cik rindas ir matricai, kas bija iegūta ar Štrāzena algoritmu (tur var rastīties liekas rindas, lai varētu izmantot Štrāzena algoritmu).
        kolonnu_sk_strasen = column_number(strasen)  # Noteicam cik kolonnas ir matricai, kas bija iegūta ar Štrāzena algoritmu (tur var rastīties liekas kolonnas, lai varētu izmantot Štrāzena algoritmu).

        pad_row = 0  # par cik vienībam vajag unpad (nodzēst liekas) rindas
        pad_column = 0  # par cik vienībam vajag unpad (nodzēst liekas) kolonnas

        if rinda_sk_strasen != rinda_sk_A:  # Ja nesakrīt matricas izmērs (rindu skaits) ar to, kuru būtu nepieciešāms dabūt, pēc matricas reizināšanas likumiem, tad nosakam cik ir tā starpība.
            pad_row = rinda_sk_strasen - rinda_sk_A  # Ja rindas skaits pēc Štrāzena algoritma palielinājies, tad noņēmam tik rindas.

        if kolonnu_sk_strasen != kolonnu_sk_B:  # Ja nesakrīt matricas izmērs (kolonnu skaits) ar to, kuru būtu nepieciešāms dabūt, pēc matricas reizināšanas likumiem, tad nosakam cik ir tā starpība.
            pad_column = kolonnu_sk_strasen - kolonnu_sk_B  # Ja kolonnu skaits pēc Štrāzena algoritma palielinājies, tad noņēmam tik kolonnas.
```

```
        res_strasen = unpad_matrix(strasen, pad_row, pad_column)  # Noņēmam liekas rindas pad_row skaitā un noņēmam liekas kolonnas pad_column skaitā.


        res_strasen = convert_matrix_elements_to_int(res_strasen)  # (!) Lai butu visi skaitļi veseli. Vajag noņemt, ja gribam float vērtības.


        return res_strasen  # Atgriež matricu C = A*B (pēc Štrāzena algoritma) bez liekam nullem.


    else:

        return None

        #"Nevar sareizināt! Matricas A kolonnu skaits nav vienāds ar matricas B rindu skaitu!"



# ----------------------------------------------------------

# Galvenā programmas daļa

# ----------------------------------------------------------



# jo vajadzīgi pielāgot ar nulles rindam/kolonnam un pēc tam viņus noņemt, tātad liekas darbības.

print("Pirmās kvadrātiskas matricas izveidošana.")

A = create_matrix()

print("\nOtrās kvadrātiskas matricas izveidošana.")

B = create_matrix()


rinda_sk_A = row_number(A)  # Skaitam rindas skaitu matricā A

kolonnu_sk_A = column_number(A)  # Skaitam kolonnas skaitu matricā A


rinda_sk_B = row_number(B)  # Skaitam rindas skaitu matricā B

kolonnu_sk_B = column_number(B)  # Skaitam kolonnas skaitu matricā B
```

```
# Izmantojam Štrāzena algoritmu un pielagojam matricu un atgriežām bez liekām nullem.

strasen_result = result_with_padding(A, B, rinda_sk_A, kolonnu_sk_A, rinda_sk_B, kolonnu_sk_B)


if strasen_result is None:  # Ja nevaram sareizināt matricas.

   print("Matricas A rindu skaitam jāsakrīt ar matricas B kolonnu skaitu, lai varētu sareizināt matricas!")

else:

   print()

   print(A)

   print("*")

   print(B)

   print("=")

   print(strasen_result)  # Parāda matricas reizinājuma rezultātu.

   # Visi elementi tiek pārverti par int! Tāpēc ja gribat reizināt arī float skaitļus, tad vajag noņemt")

   # no result_with_padding(rinda_sk_A, kolonnu_sk_A, rinda_sk_B, kolonnu_sk_B) vajag noņemt res_strasen = convert_matrix_elements_to_int(res_strasen)

   # šeit tas ir realizēts, lai testa piemēros būtu vieglāk pārbaudīt.



'''

# TESTĒŠANAI (ērtak un ātrak šeit vērtības rakstīt).

#A = np.array([[1, 2], [3, 4]])

#B = np.array([[5, 6], [7, 8]])


A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

B = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])


rinda_sk_A = row_number(A)

kolonnu_sk_A = column_number(A)

rinda_sk_B = row_number(B)
```

```
kolonnu_sk_B = column_number(B)


strasen_result = result_with_padding(rinda_sk_A, kolonnu_sk_A, rinda_sk_B, kolonnu_sk_B)


if strasen_result is None:  # == False
    print("Matricas A rindu skaitam jāsakrīt ar matricas B kolonnu skaitu, lai varētu sareizināt matricas!")
else:
    print()
    print(A)
    print("*")
    print(B)
    print("=")
    print(strasen_result)
    # Visi elementi tiek pārverti par int! Tāpēc ja gribat reizināt arī float skaitļus, tad vajag noņemt")
    # no result_with_padding(rinda_sk_A, kolonnu_sk_A, rinda_sk_B, kolonnu_sk_B) vajag noņemt res_strasen = convert_matrix_elements_to_int(res_strasen)
    # šeit tas ir realizēts, lai testa piemēros būtu vieglāk pārbaudīt.
'''


'''
# TESTĒŠANAI
# Izveidojam divas nejaušas kvadrātveida matricas ar veseliem skaitļiem (var arī ar float, tikai būtu grūtak salidzināt, jo float nav precīzs, tāpēc būtu vajadzīgi ņemt kādu precizitāti).
# bet funkcija strāda arī ar float.
matrix_a = np.random.randint(1, 10, (10, 10))  # Matricas izmēri ir 10x10.
matrix_b = np.random.randint(1, 10, (10, 10))  # Matricas izmēri ir 10x10.


rinda_sk_a = row_number(matrix_a)
kolonnu_sk_a = column_number(matrix_a)
rinda_sk_b = row_number(matrix_b)
```

```
kolonnu_sk_b = column_number(matrix_b)


print(matrix_a)

print("*")


print(matrix_b)


# Veicam matricas reizināšanu, izmantojot Strassena algoritmu.

result_strassen = strassen(matrix_a, matrix_b)  # Izmantojot Štrāzena algoritmu

stras = result_with_padding(matrix_a, matrix_b, rinda_sk_a, kolonnu_sk_a, rinda_sk_b, kolonnu_sk_b)

# stras = unpad_matrix(result_strassen, )  # Noņemam liekas rindiņas un kolonnas ar 0, kuri varētu izveidoties.

# stras = convert_matrix_elements_to_int(stras)  # Konvertējam matricas visus elementus float -> int

print("\nRezultāts izmantojot Štrāzena algoritmu:")

print(stras)

print("\nRezultāts izmantojot iebūvētās numpy funkcijas:")


# Veicam matricas reizināšanu, izmantojot np.matmul (lai pārbaudītu Štrāzena funkciju, vai pareizi strāda).

result_matmul = np.matmul(matrix_a, matrix_b)  # Iebuvēta reizināšana pārbaudei.

result_matmul = convert_matrix_elements_to_int(result_matmul)  # Konvertējam matricas visus elementus float -> int

# (To daram, lai salidzinātu matricas, jo float elementus nevar salidzināt precīzi (tur vajag ņemt kādu precizitāti, tāpēc pārbaudam tikai int skaitļiem)

# Bet algoritms strāda ari kad float.

print(result_matmul)


# Pārbauda,, vai matricas ir vienādas.

if np.array_equal(stras, result_matmul):  # Varētu izveidot pašu definētu funkciju, bet tas ir tikai testēšanai, tāpēc izmantoju gatavu no numpy bibliotēkas

    print("\nMatricas ir vienādas.")
```

```
else:
    print("\nMatricas nav vienādas")
'''
```

Testa piemēri:

1) Matrica 2x2 reizinot ar matricu 2x2 (lietotājs ievada)

```
Pirmās kvadrātiskas matricas izveidošana.
Ievadiet kvadrātiskas matricas izmēru ==> 2
Ievadiet skaitlisko vērtību elementam (1,1) ==> 1
Ievadiet skaitlisko vērtību elementam (1,2) ==> 2
Ievadiet skaitlisko vērtību elementam (2,1) ==> 3
Ievadiet skaitlisko vērtību elementam (2,2) ==> 4

Otrās kvadrātiskas matricas izveidošana.
Ievadiet kvadrātiskas matricas izmēru ==> 2
Ievadiet skaitlisko vērtību elementam (1,1) ==> 1
Ievadiet skaitlisko vērtību elementam (1,2) ==> 2
Ievadiet skaitlisko vērtību elementam (2,1) ==> 3
Ievadiet skaitlisko vērtību elementam (2,2) ==> 4

[[1. 2.]
 [3. 4.]]
*
[[1. 2.]
 [3. 4.]]
=
[[ 7 10]
 [15 22]]
```

2) Matrica 10x10 reizinot ar matricu 10x10

```
[[5 6 6 3 7 9 3 8 8 8]
 [4 3 7 7 3 2 2 6 5 9]
 [7 5 7 4 8 6 4 6 9 2]
 [2 8 7 4 9 9 1 8 6 6]
 [3 4 4 1 5 8 9 5 4 6]
 [7 2 8 4 8 1 4 8 5 3]
 [5 6 5 1 5 9 3 6 3 9]
 [8 3 8 3 6 5 6 3 1 7]
 [7 3 3 1 4 4 2 9 7 1]
 [7 7 7 3 3 4 9 8 1 9]]
*
[[2 5 1 1 2 1 6 9 2 7]
 [6 4 4 4 6 3 4 1 3 3]
 [1 4 6 1 4 3 2 8 6 8]
 [4 9 6 7 4 3 3 9 8 3]
 [7 3 8 2 1 2 8 6 4 6]
 [5 8 6 5 9 4 7 5 3 9]
 [5 4 4 9 8 6 9 3 5 5]
 [8 1 4 8 9 5 2 9 9 6]
 [1 5 2 7 6 7 1 3 3 3]
 [5 4 9 6 6 3 4 7 1 3]]

Rezultāts izmantojot Štrāzena algoritmu:
[[285 285 325 310 362 238 277 374 262 344]
 [200 223 259 243 257 173 180 324 221 238]
 [240 266 269 267 302 218 259 345 256 323]
 [288 271 328 283 338 219 260 352 266 330]
 [232 220 255 265 304 197 260 268 202 275]
 [215 204 242 225 243 177 218 338 245 280]
 [246 234 282 246 306 183 248 310 200 294]
 [211 229 265 213 252 161 257 324 203 291]
 [181 168 166 202 233 161 170 258 190 233]
 [271 244 294 295 339 208 279 356 254 312]]

Rezultāts izmantojot iebūvētās numpy funkcijas:
[[285 285 325 310 362 238 277 374 262 344]
 [200 223 259 243 257 173 180 324 221 238]
 [240 266 269 267 302 218 259 345 256 323]
 [288 271 328 283 338 219 260 352 266 330]
 [232 220 255 265 304 197 260 268 202 275]
 [215 204 242 225 243 177 218 338 245 280]
 [246 234 282 246 306 183 248 310 200 294]
 [211 229 265 213 252 161 257 324 203 291]
 [181 168 166 202 233 161 170 258 190 233]
 [271 244 294 295 339 208 279 356 254 312]]

Matricas ir vienādas.
```

3) Matrica 6x6 reizinot ar matricu 6x6

```
[[2 7 9 5 6 2]
 [7 6 2 4 6 2]
 [7 4 2 1 8 3]
 [2 5 9 9 1 6]
 [3 5 2 2 3 6]
 [5 1 5 3 4 9]]
*
[[2 2 8 1 9 4]
 [6 5 5 4 5 9]
 [6 7 8 9 7 6]
 [6 3 3 4 4 6]
 [6 8 1 5 5 6]
 [6 5 3 8 5 6]]

Rezultāts izmantojot Štrāzena algoritmu:
[[178 175 150 177 176 203]
 [134 128 126 111 163 166]
 [122 130 112 109 156 148]
 [184 157 159 192 177 203]
 [114 105  92 112 119 135]
 [142 136 125 158 162 155]]

Rezultāts izmantojot iebūvētās numpy funkcijas:
[[178 175 150 177 176 203]
 [134 128 126 111 163 166]
 [122 130 112 109 156 148]
 [184 157 159 192 177 203]
 [114 105  92 112 119 135]
 [142 136 125 158 162 155]]

Matricas ir vienādas.
```

4) Matrica 100x100 reizinot ar matricu 100x100

```
[[9 8 7 ... 1 9 8]
 [3 2 6 ... 2 2 2]
 [8 4 2 ... 4 4 9]
 ...
 [9 8 2 ... 2 5 3]
 [6 7 9 ... 5 5 1]
 [7 4 4 ... 8 4 3]]
*
[[3 9 4 ... 1 7 1]
 [6 4 5 ... 7 6 6]
 [8 3 3 ... 6 9 7]
 ...
 [2 1 5 ... 9 6 7]
 [2 3 1 ... 8 5 9]
 [3 9 8 ... 3 7 5]]

Rezultāts izmantojot Štrāzena algoritmu:
[[2595 2507 2451 ... 2825 2636 2691]
 [2713 2674 2485 ... 2767 2826 2628]
 [2154 2251 2289 ... 2386 2468 2342]
 ...
 [2387 2344 2317 ... 2544 2519 2391]
 [2536 2346 2299 ... 2630 2473 2635]
 [2485 2471 2331 ... 2590 2623 2512]]

Rezultāts izmantojot iebūvētās numpy funkcijas:
[[2595 2507 2451 ... 2825 2636 2691]
 [2713 2674 2485 ... 2767 2826 2628]
 [2154 2251 2289 ... 2386 2468 2342]
 ...
 [2387 2344 2317 ... 2544 2519 2391]
 [2536 2346 2299 ... 2630 2473 2635]
 [2485 2471 2331 ... 2590 2623 2512]]

Matricas ir vienādas.
```

5) Matrica 100x100 reizinot ar matricu 100x100

```
[[6 7 6 ... 1 4 6]
 [5 2 9 ... 9 4 8]
 [6 6 7 ... 6 9 9]
 ...
 [4 4 5 ... 3 8 9]
 [9 1 3 ... 6 5 3]
 [7 9 8 ... 8 5 4]]
*
[[4 3 7 ... 1 4 2]
 [5 7 1 ... 3 6 7]
 [1 1 4 ... 7 7 2]
 ...
 [1 4 8 ... 2 3 4]
 [4 4 4 ... 4 1 2]
 [2 6 7 ... 4 5 9]]

Rezultāts izmantojot Štrāzena algoritmu:
[[2572 2277 2405 ... 2524 2309 2569]
 [2632 2568 2607 ... 2786 2665 2745]
 [2593 2452 2479 ... 2602 2476 2688]
 ...
 [2619 2396 2511 ... 2554 2458 2644]
 [2597 2312 2460 ... 2538 2602 2593]
 [2565 2207 2450 ... 2560 2539 2758]]

Rezultāts izmantojot iebūvētās numpy funkcijas:
[[2572 2277 2405 ... 2524 2309 2569]
 [2632 2568 2607 ... 2786 2665 2745]
 [2593 2452 2479 ... 2602 2476 2688]
 ...
 [2619 2396 2511 ... 2554 2458 2644]
 [2597 2312 2460 ... 2538 2602 2593]
 [2565 2207 2450 ... 2560 2539 2758]]

Matricas ir vienādas.
```