Ko‘p o‘lchovli massivlarda belgilangan funksiyalarning xususiyatlari va tavsiflari, ularning tuzilishi va tahlili

Gulbodom Oybek qizi Norqulova
BXU

Annotatsiya: Ushbu maqolada ko‘p o‘lchovli massivlarda belgilangan funksiyalarning tuzilishi, xususiyatlari va tavsiflarini chuqur tahlil qilish amalga oshirilgan. Massiv funksiyalarining ichki mexanizmlari, ularning matematik asoslari, algoritmik murakkabligi va amaliy dasturlashdagi qo‘llanilishi o‘rganilgan. Tadqiqot davomida massiv funksiyalarining klassifikatsiyasi yaratilgan, har bir funktsiya turining xususiyatlari batafsil tahlil qilingan va ularning samaradorligini oshirish yo‘llari ko‘rsatilgan. Maqolada zamonaviy dasturlash tillarida (Python, Java, C++) mavjud funksiyalarning qiyosiy tahlili berilgan hamda ta’lim jarayonida ushbu mavzuni o‘qitishning didaktik asoslari ishlab chiqilgan. Tadqiqot natijalari dasturchilar va o‘qituvchilar uchun amaliy qo‘llanma sifatida xizmat qiladi.

Kalit so‘zlar: massiv funksiyalari, funksiya tuzilishi, algoritmik tahlil, xotira boshqaruvi, funksiya murakkabligi, ma’lumotlar qayta ishlash, massiv operatsiyalari, funksiyalar tasnifi, dasturlash kutubxonalari, kod optimallashuvi, funksional dasturlash, massiv metodlari, hisoblash samaradorligi

Properties and characteristics of functions defined in multidimensional arrays, their structure and analysis

Gulbodom Oybek qizi Norqulova
BIU

Abstract: This article provides an in-depth analysis of the structure, properties and characteristics of functions defined in multidimensional arrays. The internal mechanisms of array functions, their mathematical foundations, algorithmic complexity and application in practical programming are studied. During the research, a classification of array functions was created, the characteristics of each type of function were analyzed in detail, and ways to increase their efficiency were shown. The article provides a comparative analysis of functions available in modern programming languages (Python, Java, C++) and develops didactic foundations for teaching this topic in the educational process. The results of the research serve as a practical guide for programmers and teachers.

(cc) BY

Keywords: array functions, function structure, algorithmic analysis, memory management, function complexity, data processing, array operations, function classification, programming libraries, code optimization, functional programming, array methods, computational efficiency

Kirish. Zamonaviy dasturlashda ko‘p o‘lchovli massivlar bilan ishlash funksiyalari muhim ahamiyatga ega bo‘lib, ma’lumotlarni samarali qayta ishlash va murakkab hisoblashlarni amalga oshirish imkonini beradi. Ko‘p o‘lchovli massivlar uchun belgilangan funksiyalar dasturlash tillarining standart kutubxonalarida mavjud bo‘lib, ular yillar davomida ishlab chiqilgan va optimallashtirilgan. Funksiyalar dasturlashning asosiy tushunchalari bo‘lib, kod takrorlanishini kamaytiradi, dastur tuzilmasini yaxshilaydi va xatolarni kamaytirishga yordam beradi.

Ko‘p o‘lchovli massivlar kontekstida funksiyalar maxsus ahamiyat kasb etadi, chunki ular murakkab ma’lumotlar strukturalari bilan ishlashni soddalashtiradi va abstraktsiya darajasini oshiradi. Hozirgi kunda turli dasturlash paradigmalari mavjud bo‘lib, ularning har biri massiv funksiyalarini o‘ziga xos tarzda amalga oshiradi. Python tilida NumPy kutubxonasi, Java tilida Arrays sinfi, C++ tilida STL kutubxonasi va MATLAB muhitida o‘rnatilgan funksiyalar keng qo‘llaniladi. Bu funksiyalarning tuzilishi, ishlash mexanizmi va samaradorligi turlicha xususiyatlarga ega.

Massiv funksiyalari turli sohalarda qo‘llaniladi. Ilmiy hisoblashlarda chiziqli algebra operatsiyalari, tasvirlarni qayta ishlashda piksellar bilan ishlash, ma’lumotlar tahlilida statistik hisoblashlar, sun’iy intellektda neyron tarmoqlari uchun tensorli operatsiyalar va boshqa ko‘plab sohalarda ushbu funksiyalar asosiy vosita hisoblanadi. Dastur samaradorligi ko‘p jihatdan massiv funksiyalarining to‘g‘ri tanlanishi va qo‘llanilishiga bog‘liq.

Ta’lim jarayonida talabalar ko‘pincha massiv funksiyalarini faqat qo‘llash darajasida o‘rganadilar, lekin ularning ichki mexanizmlarini, matematik asoslarini va optimallash usullarini yetarlicha tushunmaydilar. Bu esa murakkab dasturlarni ishlab chiqishda va samaradorlik masalalarini hal qilishda qiyinchiliklarga olib keladi. Funksiyalarning vaqt va xotira murakkabligini tushunish, to‘g‘ri algoritmni tanlash va kodni optimallash muhim ko‘nikmalar hisoblanadi.

Ushbu tadqiqotning maqsadi ko‘p o‘lchovli massivlarda belgilangan funksiyalarning to‘liq tavsifini berish, ularning tuzilishini tahlil qilish, xususiyatlarini sistemalashtirish va samaradorlikni oshirish usullarini ishlab chiqishdan iborat. Tadqiqot ob’ekti sifatida turli dasturlash tillarida amalga oshirilgan massiv funksiyalari va ularning taqqoslash tahlili tanlab olingan. Tadqiqot metodologiyasi nazariy tahlil, algoritmik murakkablikni baholash va amaliy dasturlash tajribasini o‘z ichiga oladi.

Asosiy qism

Massiv funksiyalarining umumiy klassifikatsiyasi

(cc) BY

Ko‘p o‘lchovli massivlar uchun belgilangan funksiyalarni tizimli tarzda o‘rganish uchun ularni toifalarga ajratish zarur. Funksiyalarning klassifikatsiyasi ularning vazifasi, ishlash mexanizmi va qo‘llanish sohasiga asoslangan.

Yaratish va initsializatsiya funksiyalari yangi massiv ob’ektlarini yaratish va ularni boshlang‘ich qiymatlar bilan to‘ldirish uchun xizmat qiladi. Bu funksiyalar dasturning boshida yoki kerak bo‘lganda massivlarni tashkil etish imkonini beradi. Kirish va o‘zgartirish funksiyalari massiv elementlariga murojaat qilish va ularni o‘zgartirish uchun mo‘ljallangan bo‘lib, indekslash, slicing va elementlarni yangilash operatsiyalarini o‘z ichiga oladi.

Matematik operatsiyalar funksiyalari arifmetik va algebraik amallarni bajaradi. Bu guruhga elementma-element operatsiyalar, matritsalarni ko‘paytirish, transpozitsiya va boshqa chiziqli algebra operatsiyalari kiradi. Qidiruv va filtratsiya funksiyalari massivda ma’lum shartlarni qanoatlantiruvchi elementlarni topish imkonini beradi. Agregatsiya funksiyalari massiv bo‘yicha statistik hisoblashlarni amalga oshiradi va yig‘indi, o‘rtacha, maksimum, minimum kabi qiymatlarni hisoblaydi.

Transformatsiya funksiyalari massiv tuzilmasini o‘zgartiradi va reshape, transpose, flatten kabi operatsiyalarni bajaradi. Saralash va tartiblash funksiyalari elementlarni ma’lum tartibda joylashtiradi. Mantiqiy operatsiyalar funksiyalari shartli tekshirish va mantiqiy amallarni bajaradi. Birlashtirish va bo‘lish funksiyalari massivlarni birlashtirilsh yoki kichik qismlarga ajratish imkonini beradi.

Yaratish va initsializatsiya funksiyalarining tuzilishi va mexanizmi

Massiv yaratish funksiyalari dasturlashning asosiy operatsiyalari hisoblanib, ular xotira bilan ishlash, ma’lumot turlarini aniqlash va boshlang‘ich qiymatlarni berish vazifalarini bajaradi.

Python tilida NumPy kutubxonasi massiv yaratish uchun keng imkoniyatlar taqdim etadi. numpy.zeros funksiyasi nollar bilan to‘ldirilgan massiv yaratadi va shape, dtype, order parametrlarini qabul qiladi. numpy.ones funksiyasi birlar bilan to‘ldirilgan massiv yaratadi. numpy.full funksiyasi berilgan qiymat bilan to‘ldirilgan massiv hosil qiladi. numpy.empty funksiyasi initsializatsiyasiz massiv yaratadi va tezroq ishlaydi, lekin xotira tozalanmagan bo‘ladi. numpy.eye funksiyasi birlik matritsasini yaratadi. numpy.arange funksiyasi berilgan diapazon va qadam bilan massiv yaratadi. numpy.linspace funksiyasi berilgan diapazonni teng qismlarga bo‘lib massiv yaratadi.

Massiv yaratish jarayoni bir necha bosqichlardan iborat. Birinchi bosqichda zarur xotira hajmi hisoblanadi va bu hajm massiv o‘lchamlari va ma’lumot turi ko‘paytmasiga teng. Xotira hajmi shape[0] × shape[1] × ... × shape[n] × dtype_size formulasi bilan aniqlanadi. Ikkinchi bosqichda operatsion tizimdan xotira so‘raladi va bu operatsiya ko‘pincha O(1) vaqt murakkabligiga ega, chunki faqat ko‘rsatgichlar bilan ishlash amalga oshiriladi.

(cc) BY

Uchinchi bosqichda ajratilgan xotira boshlang‘ich qiymatlar bilan to‘ldiriladi. zeros va ones funksiyalari uchun bu jarayon O(n) vaqt murakkabligiga ega, chunki barcha elementlarni to‘ldirish kerak. empty funksiyasi uchun bu bosqich bajarilmaydi va vaqt murakkabligi O(1) bo‘ladi. full funksiyasi berilgan qiymat bilan xotirani to‘ldiradi va O(n) vaqt talab qiladi. To‘rtinchi bosqichda massiv ob’ekti yaratiladi va metama’lumotlar saqlanadi. Metama’lumotlar o‘lchamlar, ma’lumot turi, xotira tartibi va boshqa xususiyatlarni o‘z ichiga oladi.

Java tilida massiv yaratish jarayoni o‘ziga xos xususiyatlarga ega. int[][] matrix = new int[rows][cols] sintaksisi bilan ikki o‘lchovli massiv yaratiladi. Java da massiv yaratilganda xotira avtomatik ravishda default qiymatlar bilan to‘ldiriladi. Raqamli turlar uchun 0, boolean uchun false, ob’ektlar uchun null qiymatlari beriladi. Massiv ob’ekti heap xotirada saqlanadi va garbage collection tizimiga avtomatik ro‘yxatdan o‘tkaziladi.

C++ tilida massiv yaratish statik yoki dinamik bo‘lishi mumkin. Statik massivlar int matrix[rows][cols] sintaksisi bilan yaratiladi va stack xotirada saqlanadi. Dinamik massivlar new operatori yordamida yaratiladi va heap xotirada joylashadi. C++ da qo‘lda xotirani tozalash delete operatori bilan amalga oshirilishi kerak. std::vector konteyneridan foydalanganda avtomatik xotira boshqaruvi ta’minlanadi va bu zamonaviy C++ dasturlashda tavsiya etiladigan yondashuv hisoblanadi.

Kirish va o‘zgartirish funksiyalarining tahlili

Massiv elementlariga murojaat qilish va ularni o‘zgartirish dasturlashning eng asosiy va tez-tez bajariladigan operatsiyalari hisoblanadi. Bu operatsiyalarning samaradorligi butun dastur ishlashiga sezilarli ta’sir ko‘rsatadi.

Indekslash operatsiyalari har bir dasturlash tilida o‘ziga xos sintaksisga ega. Python va NumPy da element = array[i, j] sintaksisi bilan ikki o‘lchovli massivning elementiga murojaat qilinadi. Ketma-ket indekslash ham qo‘llanilishi mumkin: element = array[i][j]. Qiymatni o‘zgartirish uchun array[i, j] = value sintaksisidan foydalaniladi. Indekslash operatsiyasining ichki mexanizmi o‘lchamlar bo‘yicha offset hisoblashga asoslangan. Offset qiymati i × cols + j formulasi bilan hisoblanadi. Elementning xotiradagi manzili base_address + offset × element_size formulasi yordamida aniqlanadi. Bu operatsiyaning vaqt murakkabligi O(1) bo‘lib, bu massivlarning asosiy afzalliklaridan biridir.

Slicing yoki kesish operatsiyalari massivning bir qismini ajratib olish imkonini beradi. Python va NumPy da subarray = array[start:end, start:end] sintaksisi bilan subregio olinadi. Bitta qatorni olish uchun row = array[i, :] va bitta ustunni olish uchun col = array[:, j] sintaksisidan foydalaniladi. Slicing operatsiyasi ikki xil tarzda amalga oshirilishi mumkin. View usulida yangi massiv yaratilmaydi va faqat ko‘rsatgich qaytariladi. Bu usulda xotira sarfi O(1) bo‘lib, yangi xotira ajratilmaydi. Vaqt

(cc) BY

murakkabligi ham O(1) bo‘lib, faqat metama’lumotlar yaratiladi. View da o‘zgartirish asl massivga ta’sir qiladi.

Copy usulida yangi mustaqil massiv yaratiladi. Bu usulda xotira sarfi O(k) bo‘lib, bu yerda k - ajratilgan elementlar soni. Vaqt murakkabligi O(k) bo‘lib, barcha elementlar ko‘chiriladi. Copy da o‘zgartirish asl massivga ta’sir qilmaydi. NumPy da default holatda view qaytariladi va bu xotira va vaqtni tejaydi. Agar yangi mustaqil massiv kerak bo‘lsa, .copy() metodidan foydalanish kerak.

Fancy indexing yoki murakkab indekslash NumPy ning kuchli xususiyatlaridan biridir. Boolean indekslash orqali filtered = array[array > 5] sintaksisi bilan shartga mos keladigan elementlar tanlanadi. Massiv bilan indekslash orqali indices = [0, 2, 4] va selected = array[indices] sintaksisi bilan kerakli indekslar tanlanadi. Ko‘p o‘lchovli indekslash orqali rows = [0, 2], cols = [1, 3] va selected = array[rows, cols] sintaksisi bilan murakkab tanlov amalga oshiriladi.

Fancy indexing mexanizmi bir necha bosqichlardan iborat. Birinchi bosqichda indeks massivi yoki boolean massiv tahlil qilinadi. Ikkinchi bosqichda mos keladigan elementlar aniqlanadi. Uchinchi bosqichda yangi massiv yaratiladi va elementlar ko‘chiriladi. Fancy indexing har doim copy yaratadi va vaqt murakkabligi O(n) bo‘lib, bu yerda n - tanlangan elementlar soni.

Matematik operatsiyalar funksiyalarining tuzilishi

Matematik operatsiyalar massivlar bilan ishlashning markaziy qismi bo‘lib, ular ilmiy hisoblashlar, ma’lumotlar tahlili va sun’iy intellekt sohalarida keng qo‘llaniladi.

Elementma-element operatsiyalar har bir elementga alohida qo‘llaniladi va massivlarning bir xil o‘lchamga ega bo‘lishini talab qiladi. C = A + B operatsiyasi ikki massivni elementma-element qo‘shadi. C = A - B ayirish operatsiyasini bajaradi. C = A * B elementma-element ko‘paytirishni amalga oshiradi. C = A / B bo‘lish operatsiyasini bajaradi. C = A ** 2 darajaga ko‘tarish operatsiyasini amalga oshiradi. Trigonometrik funksiyalar sin_array = np.sin(A), cos_array = np.cos(A), tan_array = np.tan(A) sintaksislari bilan qo‘llaniladi. Eksponensial funksiyalar exp_array = np.exp(A), log_array = np.log(A) orqali ishlatiladi.

Broadcasting mexanizmi turli o‘lchamdagi massivlarni operatsiyalarda ishlatish imkonini beradi. Broadcasting qoidalari quyidagicha: birinchi qoidada o‘lchamlar soni tenglanadi va qisqa massivga 1 qo‘shiladi. Ikkinchi qoidada har bir o‘lcham bo‘yicha tekshirish amalga oshiriladi va o‘lchamlar teng bo‘lishi yoki biri 1 ga teng bo‘lishi kerak. Uchinchi qoidada natija o‘lchami maksimal qiymatlar bilan aniqlanadi. Masalan, A.shape = (3, 4) va B.shape = (4,) bo‘lsa, B (1, 4) ga aylanadi va C = A + B natijasida C.shape = (3, 4) bo‘ladi. Broadcasting algoritmi ikkita ichma-ich sikl yordamida amalga oshiriladi va vaqt murakkabligi O(n×m) bo‘ladi.

Chiziqli algebra operatsiyalari matritsalar bilan ishlashning asosiy vositalari hisoblanadi. C = np.matmul(A, B) yoki C = A @ B sintaksisi bilan matritsalarni

(cc) BY

ko‘paytirish amalga oshiriladi. A_T = np.transpose(A) yoki A_T = A.T sintaksisi bilan transpozitsiya bajariladi. det = np.linalg.det(A) determinantni hisoblaydi. A_inv = np.linalg.inv(A) teskari matritsani topadi. eigenvalues, eigenvectors = np.linalg.eig(A) xos qiymatlar va xos vektorlarni hisoblaydi.

Matritsalarni ko‘paytirish algoritmi uch siklli tuzilmaga ega. Standart algoritmda C[i, j] += A[i, k] * B[k, j] operatsiyasi bajariladi va vaqt murakkabligi O(m×n×p) bo‘ladi. Optimallashtirilgan algoritmlar mavjud bo‘lib, Strassen algoritmi O(n^2.807) murakkablikka ega. Blok ko‘paytirish usulida matritsalar kichik bloklarga bo‘linadi va kesh xotiridan samarali foydalaniladi. BLAS kutubxonasi optimallashtirilgan past darajali funksiyalarni taqdim etadi va protsessor vektorlash imkoniyatlaridan foydalanadi.

Qidiruv va filtratsiya funksiyalarining tahlili

Qidiruv va filtratsiya operatsiyalari massivdan kerakli ma’lumotlarni topish va ajratib olish uchun muhim ahamiyatga ega. Bu operatsiyalar ma’lumotlar tahlili va qayta ishlashda keng qo‘llaniladi.

Element qidirish funksiyalari turli usullarda amalga oshirilishi mumkin. numpy.where funksiyasi shartga mos keladigan elementlarning indekslarini qaytaradi. indices = np.where(A == value) sintaksisi qiymat bo‘yicha qidiradi. indices = np.where(A > threshold) sintaksisi shart bo‘yicha qidiradi. Boolean indekslash orqali filtered = A[A > threshold] sintaksisi shartga mos elementlarni ajratib oladi. first_index = np.argmax(A > threshold) birinchi mos keladigan elementning indeksini topadi.

Chiziqli qidiruv algoritmi eng oddiy qidiruv usuli bo‘lib, barcha elementlarni ketma-ket tekshiradi. Algoritmning vaqt murakkabligi O(m×n) bo‘lib, xotira murakkabligi O(1) ga teng. Chiziqli qidiruvning afzalligi - massivning tartiblangan bo‘lishini talab qilmaydi. Kamchiligi - katta massivlarda sekin ishlaydi. Ikkilik qidiruv tartiblangan massivlar uchun samaraliroq usul hisoblanadi. Algoritmning vaqt murakkabligi O(m + n) bo‘lib, bu chiziqli qidiruvdan ancha tez.

Filtratsiya operatsiyalari ma’lum shartga mos keladigan elementlarni tanlash jarayoni hisoblanadi. Oddiy filtratsiya result = A[A > 0] sintaksisi bilan musbat elementlarni tanlaydi. Murakkab shartlar mantiqiy operatorlar bilan birlashtirilishi mumkin. result = A[(A > 0) & (A < 10)] sintaksisi mantiqiy VA operatsiyasini qo‘llaydi. result = A[(A < 0) | (A > 10)] sintaksisi mantiqiy YOKI operatsiyasini bajaradi. Funksiya orqali filtratsiya maxsus shartlar uchun ishlatilishi mumkin.

Filtratsiya mexanizmi uch bosqichdan iborat. Birinchi bosqichda boolean massiv yaratiladi va vaqt murakkabligi O(n) bo‘ladi. Ikkinchi bosqichda True elementlar soni sanab chiqiladi va bu ham O(n) vaqt talab qiladi. Uchinchi bosqichda natija massivi yaratiladi va shartga mos elementlar ko‘chiriladi, bu jarayon O(n) vaqtda amalga oshiriladi. Umumiy vaqt murakkabligi O(n) bo‘ladi, bu filtratsiyani samarali operatsiya qiladi.

(cc) BY

Transformatsiya funksiyalarining tuzilishi

Transformatsiya funksiyalari massiv shaklini va tuzilmasini o‘zgartirish uchun mo‘ljallangan bo‘lib, ular ma’lumotlarni turli ko‘rinishlarda taqdim etish imkonini beradi.

Reshape funksiyasi massiv shaklini o‘zgartiradi va B = A.reshape((new_shape)) sintaksisi bilan qo‘llaniladi. Avtomatik o‘lcham hisoblash B = A.reshape((3, -1)) sintaksisi orqali amalga oshiriladi va bu yerda -1 qiymati avtomatik hisoblanadi. Bir o‘lchovli qilish flat = A.flatten() sintaksisi bilan nusxa yaratadi yoki flat = A.ravel() sintaksisi bilan view qaytaradi.

NumPy da reshape operatsiyasi ko‘pincha view qaytaradi va xotira ko‘chirilmaydi. Jarayon quyidagi bosqichlardan iborat: birinchi bosqichda yangi o‘lchamlar tekshiriladi va umumiy razmer bir xil bo‘lishi kerak. Ikkinchi bosqichda yangi stride qiymatlari hisoblanadi. Stride - bu har bir o‘lcham bo‘yicha keyingi elementga o‘tish uchun zarur baytlar soni. Uchinchi bosqichda yangi massiv ob’ekti yaratiladi, lekin xotira bir xil qoladi. Bu jarayonning vaqt murakkabligi O(1) bo‘lib, faqat metama’lumotlar o‘zgaradi. Xotira murakkabligi ham O(1) bo‘lib, yangi xotira ajratilmaydi.

Transpose funksiyasi matritsani transpozitsiya qiladi. B = A.T yoki B = np.transpose(A) sintaksislari ishlatiladi. Ko‘p o‘lchovli transpozitsiya uchun B = np.transpose(A, axes=(2, 0, 1)) sintaksisi qo‘llaniladi va bu o‘qlarning tartibini o‘zgartiradi. Samarali amalga oshirish view yaratadi va xotiradagi ma’lumotlar o‘zgarmaydi. O‘lchamlar va stride’lar teskari tartibda qaytariladi. Vaqt murakkabligi O(1) va xotira murakkabligi ham O(1) bo‘ladi, chunki faqat metama’lumotlar o‘zgaradi. Agar nusxa kerak bo‘lsa, B = A.T.copy() sintaksisi ishlatiladi va bu holda vaqt murakkabligi O(m×n) va xotira murakkabligi O(m×n) bo‘ladi.

Concatenate funksiyasi massivlarni birlashtiradi. C = np.vstack([A, B]) vertikal birlashtirish amalga oshiradi. C = np.hstack([A, B]) gorizontal birlashtiradi. C = np.concatenate([A, B], axis=0) umumiy birlashtirish funksiyasi hisoblanadi. Concatenate algoritmi to‘rt bosqichdan iborat: birinchi bosqichda o‘lchamlar mos kelishini tekshirish, ikkinchi bosqichda natija massivi hajmini hisoblash, uchinchi bosqichda yangi massiv yaratish va to‘rtinchi bosqichda ma’lumotlarni ketma-ket ko‘chirish. Vaqt murakkabligi O(n) bo‘lib, bu yerda n - barcha elementlar soni. Xotira murakkabligi O(n) bo‘lib, chunki yangi massiv yaratiladi.

Saralash funksiyalarining tahlili

Saralash funksiyalari ma’lumotlarni tartiblash va tashkil etish uchun juda muhim bo‘lib, ular ko‘plab algoritmlarda asosiy operatsiya hisoblanadi.

NumPy da turli saralash funksiyalari mavjud. sorted_array = np.sort(A, axis=None) butun massivni saralaydi. sorted_rows = np.sort(A, axis=1) har bir qatorni alohida saralaydi. sorted_cols = np.sort(A, axis=0) har bir ustunni saralaydi. indices =

(cc) BY

np.argsort(A) saralangan elementlarning asl indekslarini qaytaradi. A.sort() joyida saralash amalga oshiradi va asl massivni o‘zgartiradi.

NumPy da uchta asosiy saralash algoritmi qo‘llaniladi. Quicksort default algoritm bo‘lib, o‘rtacha holatda O(n log n) vaqtda ishlaydi, lekin eng yomon holatda $O(n^2)$ vaqt talab qiladi. Xotira sarfi O(log n) bo‘lib, bu rekursiya steki uchun zarur. Quicksort barqaror emas, ya’ni teng elementlarning asl tartibi saqlanmasligi mumkin. Mergesort barqaror algoritm bo‘lib, har doim O(n log n) vaqtda ishlaydi. Xotira sarfi O(n) bo‘lib, qo‘shimcha massiv talab qilinadi. Heapsort algoritmi ham har doim O(n log n) vaqtda ishlaydi, xotira sarfi O(1) bo‘lib, joyida saralash amalga oshiriladi, lekin barqaror emas.

Ko‘p o‘lchovli massivlarni saralash murakkab operatsiya bo‘lib, bir nechta kalitlar bo‘yicha saralashni talab qilishi mumkin. numpy.lexsort funksiyasi leksikografik saralashni amalga oshiradi. sorted_indices = np.lexsort((A[:, 1], A[:, 0])) sintaksisi birinchi ustun, keyin ikkinchi ustun bo‘yicha saralaydi. Maxsus kalit funksiyasi bilan saralash uchun har bir qatorga kalit funksiyasi qo‘llaniladi va natijalar bo‘yicha saralash amalga oshiriladi.

Topk elementlarni topish ko‘pincha to‘liq saralashdan samaraliroq. K ta eng katta element topish uchun k_largest = np.partition(A, -k)[-k:] sintaksisi ishlatiladi. K ta eng kichik element topish uchun k_smallest = np.partition(A, k)[:k] sintaksisi qo‘llaniladi. Partition algoritmi Quickselect algoritmiga asoslangan bo‘lib, o‘rtacha holatda O(n) vaqtda ishlaydi. Bu to‘liq saralashdan ancha tez, chunki barcha elementlarni tartiblash shart emas.

Agregatsiya va statistik funksiyalar

Agregatsiya funksiyalari massiv bo‘yicha umumlashtiruvchi qiymatlarni hisoblash uchun xizmat qiladi va ma’lumotlar tahlilida muhim rol o‘ynaydi.

Asosiy agregatsiya funksiyalari turli statistik ko‘rsatkichlarni hisoblaydi. total = np.sum(A) barcha elementlarning yig‘indisini topadi. mean = np.mean(A) o‘rtacha qiymatni hisoblaydi. median = np.median(A) medianani topadi. std = np.std(A) standart og‘ishni hisoblaydi. var = np.var(A) dispersiyani topadi. minimum = np.min(A) minimal elementni aniqlaydi. maximum = np.max(A) maksimal elementni topadi. O‘q bo‘yicha agregatsiya ham mumkin: row_sums = np.sum(A, axis=1) har bir qator yig‘indisini hisoblaydi va col_means = np.mean(A, axis=0) har bir ustun o‘rtachasini topadi.

Agregatsiya funksiyalarining tuzilishi algoritmik murakkablikka bog‘liq. Bitta o‘tishli algoritmlar sum, min, max kabi funksiyalar uchun qo‘llaniladi va bir marta aylanish bilan hisoblanadi. Vaqt murakkabligi O(n) va xotira murakkabligi O(1) bo‘ladi. Ikki o‘tishli algoritmlar dispersiya va standart og‘ish uchun ishlatiladi. Birinchi o‘tishda o‘rtacha hisoblanadi, ikkinchi o‘tishda har bir elementdan o‘rtachaning ayirmasi kvadrati yig‘iladi. Vaqt murakkabligi O(2n) = O(n) bo‘ladi.

(cc) BY

Welford algoritmi onlayn dispersiyani hisoblash uchun ishlatiladi va bir o‘tishda dispersiyani hisoblashga imkon beradi. Algoritm har bir yangi element uchun o‘rtacha va M2 qiymatlarini yangilaydi. Bu algoritm raqamli jihatdan barqaror bo‘lib, katta sonlar bilan ishlashda xatoliklarni kamaytiradi. Vaqt murakkabligi O(n) va xotira murakkabligi O(1) bo‘ladi, bu esa katta massivlar uchun juda samarali.

Mantiqiy operatsiyalar va taqqoslash funksiyalari

Mantiqiy operatsiyalar shartli tekshirish va mantiqiy amallarni bajarish uchun mo‘ljallangan bo‘lib, ular ma’lumotlarni filtratsiya qilish va tahlil qilishda keng qo‘llaniladi.

Element bo‘yicha mantiqiy operatsiyalar ikki yoki bir massiv ustida mantiqiy amallarni bajaradi. result = np.logical_and(A, B) mantiqiy VA operatsiyasini amalga oshiradi. result = np.logical_or(A, B) mantiqiy YOKI operatsiyasini bajaradi. result = np.logical_not(A) mantiqiy EMAS operatsiyasini qo‘llaydi. result = np.logical_xor(A, B) mantiqiy XOR operatsiyasini bajaradi.

Agregat mantiqiy operatsiyalar massiv bo‘yicha umumlashtiruvchi mantiqiy qiymatlarni qaytaradi. all_true = np.all(A > 0) barcha elementlar shartni qanoatlantirishini tekshiradi. any_true = np.any(A > 0) hech bo‘lma bitta element shartni qanoatlantirishini tekshiradi. Bu funksiyalar mantiqiy qisqa tutashuvdan foydalanadi va zarur bo‘lganda to‘xtaydi.

Taqqoslash funksiyalari massivlarni solishtirish uchun ishlatiladi. equal = np.array_equal(A, B) ikki massivning mutlaq tengligini tekshiradi. close = np.allclose(A, B, rtol=1e-5) yaqin qiymatlarni solishtiradi va bu suzuvchi nuqta sonlari bilan ishlashda muhimdir. Elementma-element taqqoslash operatorlari >, <, ==, != ham qo‘llanilishi mumkin va boolean massiv qaytaradi.

Mantiqiy operatsiyalar mexanizmi bitwise operatsiyalar yordamida amalga oshiriladi. Zamonaviy protsessorlar SIMD ko‘rsatmalarini qo‘llab-quvvatlaydi va bir vaqtning o‘zida bir nechta mantiqiy operatsiyalarni bajarishi mumkin. Vektorizatsiya orqali samaradorlik sezilarli darajada oshiriladi. Parallel bajarish imkoniyati mavjud bo‘lib, vaqt murakkabligi O(n/p) bo‘ladi, bu yerda p - parallel oqimlar soni.

Maxsus funksiyalar va optimallashtirish usullari

NumPy kutubxonasi universal funksiyalar (ufunc) tushunchasini joriy qilgan bo‘lib, bu elementma-element ishlash uchun maxsus optimallashtirilgan funksiyalardir.

Universal funksiyalar bir nechta toifalarga bo‘linadi. Arifmetik ufunc’lar np.add, np.subtract, np.multiply, np.divide operatsiyalarini o‘z ichiga oladi. Trigonometrik ufunc’lar np.sin, np.cos, np.tan, np.arcsin, np.arccos, np.arctan funksiyalarini qamrab oladi. Eksponensial ufunc’lar np.exp, np.log, np.log10, np.power, np.sqrt operatsiyalarini bajaradi. Taqqoslash ufunc’lari np.greater, np.less, np.equal, np.not_equal funksiyalarini o‘z ichiga oladi.

(cc) BY

Ufunc xususiyatlari ularni oddiy funksiyalardan ajratib turadi. Broadcasting qobiliyati turli o‘lchamdagi massivlar bilan ishlash imkonini beradi. Type casting avtomatik tur konvertatsiyasini ta’minlaydi. Vektorizatsiya SIMD ko‘rsatmalaridan foydalanadi va bir nechta elementni bir vaqtning o‘zida qayta ishlaydi. Output parametr natijani mavjud massivga yozish imkonini beradi va qo‘shimcha xotira ajratishni oldini oladi.

Kodni optimallashtirish bir nechta strategiyalarni o‘z ichiga oladi. Vektorizatsiya loop’larni NumPy funksiyalari bilan almashtirish orqali samaradorlikni oshiradi. Xotira joylashuvi muhim bo‘lib, C-tartibli massivlarda qatorlar ketma-ket, Fortran-tartibli massivlarda ustunlar ketma-ket joylashadi. Kesh xotiridan samarali foydalanish uchun xotirada ketma-ket joylashgan elementlarga murojaat qilish kerak.

In-place operatsiyalar qo‘shimcha xotira ajratishni oldini oladi. Masalan, A += B operatsiyasi A = A + B operatsiyasidan samaraliroq, chunki yangi massiv yaratilmaydi. View va copy o‘rtasidagi farqni tushunish muhim bo‘lib, view xotira tejaydi, copy esa mustaqil massiv yaratadi. Parallel hisoblash katta massivlar uchun sezilarli tezlashtirish beradi va NumPy ba’zi funksiyalarda avtomatik parallellashtirish qo‘llaydi.

Xotira boshqaruvi va samaradorlik

Xotira boshqaruvi ko‘p o‘lchovli massivlar bilan samarali ishlashning muhim jihati hisoblanadi.

NumPy massivlari xotirada uzluksiz blok sifatida saqlanadi. Bu kesh xotiridan samarali foydalanish imkonini beradi. Stride tushunchasi har bir o‘lcham bo‘yicha keyingi elementga o‘tish uchun zarur baytlar sonini bildiradi. C-tartibli saqlaShda oxirgi o‘lcham eng tez o‘zgaradi va stride’lar chapdan o‘ngga ortadi. Fortran-tartibli saqlashda birinchi o‘lcham eng tez o‘zgaradi va stride’lar o‘ngdan chapga ortadi.

View va copy mexanizmlari xotira samaradorligiga ta’sir qiladi. View asl massiv bilan xotirani bo‘lishadi va yangi xotira ajratmaydi. Copy mustaqil massiv yaratadi va asl massivdan ajralgan xotiraga ega. View yaratish operatsiyalari: slicing, transpose, reshape (ko‘pincha). Copy yaratish operatsiyalari: fancy indexing, copy() metodi, ba’zi arifmetik operatsiyalar.

Xotira tozalash Python da garbage collector tomonidan avtomatik amalga oshiriladi. Massiv ob’ektiga hech qanday havola qolmaganda, xotira avtomatik bo‘shatiladi. Katta massivlar bilan ishlashda del operatoridan foydalanish xotirani tezroq bo‘shatishga yordam beradi. C++ da esa qo‘lda xotirani tozalash delete operatori bilan amalga oshirilishi kerak.

Xotira optimallashtirish strategiyalari bir nechta yo‘nalishlarni o‘z ichiga oladi. Ma’lumot turini to‘g‘ri tanlash xotirani tejaydi, masalan, float32 o‘rniga float16 ishlatish xotirani ikki baravarga kamaytiradi. Siyrak matritsalar uchun maxsus tuzilmalar (CSR, CSC formatlar) xotirani sezilarli darajada tejaydi. Xotira mapping

(cc) BY

katta fayllar bilan ishlashda foydalaniladigan texnika bo‘lib, butun faylni xotiraga yuklamasdan ishlash imkonini beradi.

Didaktik yondashuv va o‘qitish metodikasi

Ko‘p o‘lchovli massiv funksiyalarini o‘qitishda samarali didaktik yondashuv qo‘llash zarur.

Bosqichma-bosqich o‘qitish prinsipi muhim bo‘lib, oddiy tushunchalardan murakkab tushunchalarga o‘tish kerak. Dastlab bir o‘lchovli massivlar va ularning asosiy funksiyalari o‘rganiladi. Keyin ikki o‘lchovli matritsalar va ularga xos funksiyalar o‘rgatiladi. Nihoyat, yuqori o‘lchovli tensorlar va murakkab operatsiyalar o‘zlashtiriladi. Har bir bosqichda talabalar oldingi bilimlarni mustahkamlashi va yangi tushunchalarni organik ravishda qabul qilishi ta’minlanadi.

Vizualizatsiya talabalar uchun tushunishni osonlashtiradi. Ikki o‘lchovli massivlarni jadval ko‘rinishida ko‘rsatish intuitsiyani rivojlantiradi. Grafiklar orqali operatsiyalarni tasvirlash abstrakt tushunchalarni konkret qiladi. Uch o‘lchovli massivlarni bo‘laklash (slicing) orqali ko‘rsatish murakkab strukturalarni tushunishga yordam beradi. Zamonaviy vizualizatsiya vositalari (matplotlib, seaborn) dan foydalanish tavsiya etiladi.

Amaliy mashqlar nazariy bilimlarni mustahkamlaydi. Oddiy masalalardan boshlash va asta-sekin murakkablashtirish kerak. Hayotiy misollar bilan ishlash (rasmlar, jadvallar, vaqt qatorlari) motivatsiyani oshiradi. Dasturlash tili va kutubxonalaridan amaliy foydalanish ko‘nikmalarni rivojlantiradi. Talabalar o‘z loyihalarini bajarishlari orqali mustaqil ishlash qobiliyati shakllanadi.

Xatolarni tahlil qilish va oldini olish muhim o‘quv jarayoni hisoblanadi. Indeksdan chiqib ketish (Index out of bounds) eng tez-tez uchraydigan xato bo‘lib, massiv chegaralarini doimo tekshirish kerakligi ta’kidlanadi. Noto‘g‘ri o‘lchamlar bilan ishlash operatsiyalar bajarilishidan oldin o‘lchamlarni tasdiqlashni talab qiladi. Xotira to‘lib ketishi katta massivlar bilan ishlashda kuzatiladi va xotirani tejash usullari o‘rgatiladi. View va copy farqini tushunmaslik asl ma’lumotlarni kutilmaganda o‘zgartirish xatolariga olib keladi.

Turli dasturlash tillari va kutubxonalarda amalga oshirish

Har bir dasturlash tili va kutubxonasi massiv funksiyalarini o‘ziga xos tarzda amalga oshiradi.

Python va NumPy yuqori darajadagi abstraktsiya va qulaylikni taqdim etadi. Ko‘plab tayyor funksiyalar mavjud bo‘lib, ular yaxshi hujjatlashtirilgan. C va Fortran bilan integratsiya tufayli yuqori samaradorlik ta’minlanadi. Interaktiv muhit (Jupyter) tezkor prototiplash imkonini beradi. Kamchiliklari: interpretirlangan til bo‘lgani uchun ba’zi operatsiyalar sekinroq bo‘lishi mumkin, GIL (Global Interpreter Lock) parallel hisoblashni cheklaydi.

(cc) BY

MATLAB matematik hisoblashlar uchun maxsus yaratilgan muhit hisoblanadi. Matritsalar bilan ishlash juda qulay va tabiiy. Kuchli vizualizatsiya vositalari o‘rnatilgan. Simulink bilan integratsiya tizimlarni modellashtirish imkonini beradi. Kamchiliklari: litsenziya to‘lovi talab qilinadi, boshqa tillarga integratsiya murakkab, umumiy maqsadli dasturlash uchun kamroq mos.

Java ob’ektga yo‘naltirilgan paradigmani qo‘llaydi. Ko‘p o‘lchovli massivlar massivlar massivi sifatida amalga oshiriladi. Qat’iy tip tekshiruvi xatoliklarni erta aniqlashga yordam beradi. Platform mustaqilligi har qanday operatsion tizimda ishlash imkonini beradi. Apache Commons Math va ND4J kutubxonalari qo‘shimcha imkoniyatlar beradi. Kamchiliklari: sintaksis ko‘proq verboz, NumPy ga nisbatan kamroq matematik funksiyalar.

C va C++ past darajali nazorat va maksimal samaradorlikni ta’minlaydi. Xotirani qo‘lda boshqarish to‘liq nazoratni beradi. SIMD ko‘rsatmalaridan bevosita foydalanish mumkin. Eigen, Armadillo kutubxonalari yuqori darajadagi interfeys taqdim etadi. Kamchiliklari: murakkablik va rivojlantirish vaqti ko‘proq, xotirani qo‘lda boshqarish xatolarga olib kelishi mumkin, portativlik muammolari bo‘lishi mumkin.

Amaliy qo‘llanmalar va dasturlash namunalari

Ko‘p o‘lchovli massiv funksiyalari turli sohalarda qo‘llaniladi va har bir sohada o‘ziga xos talablar mavjud.

Raqamli tasvirlarni qayta ishlashda tasvirlar uch o‘lchovli massiv [tinglik×kenglik×3] sifatida saqlanadi, bu yerda 3 - RGB kanallari. Filtrlash operatsiyalari konvolyutsiya funksiyalari yordamida amalga oshiriladi. Tasvir transformatsiyalari (aylanish, масштаблаш) affin transformatsiya matritsalari orqali bajariladi. Rangli modellar o‘rtasida konvertatsiya (RGB, HSV, YUV) matematik formulalar yordamida amalga oshiriladi.

Ma’lumotlar tahlilida jadval ko‘rinishidagi ma’lumotlar ikki o‘lchovli massivda saqlanadi. Agregatsiya funksiyalari statistik ko‘rsatkichlarni hisoblaydi. Filtrlash va saralash ma’lumotlarni tashkil etishga yordam beradi. Correlatsiya va regressiya tahlili matritsali operatsiyalar orqali amalga oshiriladi. Pandas kutubxonasi NumPy ustiga qurilgan bo‘lib, yuqori darajali ma’lumotlar bilan ishlash vositalarini taqdim etadi.

Sun’iy neyron tarmoqlarda og‘irliklar matritsalari va aktivatsiya tensorlari asosiy tuzilmalar hisoblanadi. Forward propagation matritsalarni ko‘paytirish va aktivatsiya funksiyalarini qo‘llashdan iborat. Backpropagation gradientlarni hisoblash uchun zanjir qoidasini qo‘llaydi. Batch processing bir nechta namunalarni bir vaqtda qayta ishlash orqali samaradorlikni oshiradi. TensorFlow va PyTorch kutubxonalari bu operatsiyalarni avtomatlashtiradi va GPU da tezlashtiradi.

Ilmiy hisoblashlarda chiziqli tenglamalar tizimini yechish matritsali usullar orqali amalga oshiriladi. Diferensial tenglamalarni raqamli yechish diskretizatsiya va iterativ usullarni talab qiladi. Monte-Karlo simulatsiyalari tasodifiy massivlar bilan ishlashni

(cc) BY

o‘z ichiga oladi. Optimallashtirish masalalari gradientli usullar va matritsali operatsiyalar yordamida yechiladi.

Kelajak yo‘nalishlari va rivojlanish istiqbollari

Ko‘p o‘lchovli massiv funksiyalari sohasida doimiy rivojlanish kuzatilmoqda va bir nechta istiqbolli yo‘nalishlar mavjud.

GPU hisoblashlari massiv operatsiyalarini sezilarli tezlashtiradi. CUDA va OpenCL platformalari parallel hisoblash imkoniyatlarini kengaytiradi. CuPy kutubxonasi NumPy interfeysi bilan GPU da ishlaydi. Tensor o‘zaklar maxsus tezlashtirish uchun mo‘ljallangan apparatlar. Kelajakda GPU hisoblashlari yanada kengroq tarqalishi kutilmoqda.

Avtomatik differentsiatsiya zamonaviy chuqur o‘rganishning asosi hisoblanadi. PyTorch va TensorFlow avtomatik gradientlarni hisoblaydi. Statik va dinamik hisoblash graflari turli optimallash imkoniyatlarini beradi. Kelajakda avtomatik differentsiatsiya yangi algoritmlar va ilovalar uchun standart vosita bo‘lishi mumkin.

Kvant hisoblash yangi paradigma bo‘lib, kvant massivlari va kvant algoritmlar rivojlanmoqda. Kvant superpozitsiyasi parallel hisoblashni ta’minlaydi. Ba’zi masalalar uchun eksponensial tezlashtirish mumkin. Hozircha dastlabki bosqichda bo‘lsa-da, kelajakda katta ahamiyat kasb etishi kutilmoqda.

Distributed hisoblash katta hajmdagi ma’lumotlar bilan ishlash uchun zarur. Dask kutubxonasi NumPy interfeysi bilan parallel va taqsimlangan hisoblashni ta’minlaydi. Apache Spark katta ma’lumotlar uchun taqsimlangan hisoblash platformasi. Kelajakda taqsimlangan tizimlar yanada samarali va foydalanish uchun qulayroq bo‘lishi kutilmoqda.

Xulosa. Ko‘p o‘lchovli massivlarda belgilangan funksiyalarning xususiyatlari va tavsiflari, ularning tuzilishi va tahlili tadqiqoti quyidagi muhim xulosalarga olib keldi. Massiv funksiyalarining tizimli klassifikatsiyasi ularni o‘rganish va qo‘llashni osonlashtiradi. Yaratish va initsializatsiya, kirish va o‘zgartirish, matematik operatsiyalar, qidiruv va filtratsiya, agregatsiya, transformatsiya, saralash, mantiqiy operatsiyalar va boshqa toifalar aniq belgilandi. Har bir toifaning o‘ziga xos xususiyatlari, qo‘llanish sohalari va samaradorlik xarakteristikalari aniqlandi.

Funksiyalarning ichki mexanizmlari va algoritmik tuzilmasi chuqur o‘rganildi. Xotira bilan ishlash prinsipi, stride va offset hisoblash, view va copy o‘rtasidagi farq, broadcasting mexanizmi kabi muhim tushunchalar tahlil qilindi. Bu bilimlar samarali va optimallashtirilgan kod yozish uchun zarur.

Vaqt va xotira murakkabligining baholash metodikasi ishlab chiqildi. Har bir funksiya uchun Big-O notatsiyasida murakkablik ko‘rsatkichlari berildi. Oddiy operatsiyalar $O(1)$, chiziqli operatsiyalar $O(n)$, matritsali operatsiyalar $O(n^2)$ yoki $O(n^3)$ murakkablikka ega ekanligi aniqlandi. Bu bilimlar algoritmlarni tanlash va optimallashtirish uchun muhim asos yaratadi.

(cc) BY

Turli dasturlash tillari va kutubxonalarida funksiyalarning qiyosiy tahlili amalga oshirildi. Python/NumPy qulaylik va keng imkoniyatlar, MATLAB matematik hisoblashlar, Java mustahkamlik va platformalar mustaqilligi, C/C++ maksimal samaradorlik va nazorat tomonlari bilan ajralib turadi. Har bir muhit o‘z afzalliklari va kamchiliklariga ega.

Optimallash strategiyalari va samaradorlikni oshirish usullari ishlab chiqildi. Vektorizatsiya, parallellashtirish, kesh optimizatsiyasi, in-place operatsiyalar, to‘g‘ri ma’lumot turini tanlash va xotirani tejash texnikalari tavsiya etildi. Bu usullar dastur ishlashini sezilarli darajada yaxshilaydi.

Didaktik yondashuv va o‘qitish metodikasi ishlab chiqildi. Bosqichma-bosqich o‘rganish, vizualizatsiya, amaliy mashqlar, xatolarni tahlil qilish va mustaqil loyihalar bajarish o‘quv jarayonining samaradorligini oshiradi. Bu yondashuv talabalar uchun murakkab mavzuni tushunarli va qiziqarli qiladi.

Amaliy qo‘llanmalar va real hayotdagi misollar ko‘rsatildi. Tasvirlarni qayta ishlash, ma’lumotlar tahlili, sun’iy intellekt, ilmiy hisoblashlar va boshqa sohalarda massiv funksiyalari asosiy rol o‘ynaydi. Konkret dasturlash namunalari nazariy bilimlarni amaliyotga tatbiq etish yo‘llarini ko‘rsatdi.

Kelajak istiqbollari va rivojlanish yo‘nalishlari belgilandi. GPU hisoblashlari, avtomatik differentsiatsiya, kvant hisoblash va taqsimlangan tizimlar sohasida katta o‘zgarishlar kutilmoqda. Bu texnologiyalar massiv funksiyalarining qo‘llanilish doirasini kengaytiradi va yangi imkoniyatlar ochadi.

Tadqiqot ko‘rsatdiki, ko‘p o‘lchovli massiv funksiyalarini to‘liq tushunish nazariy bilim, algoritmik fikrlash va amaliy ko‘nikmalarni birlashtiradi. Bu bilimlar asosida samarali, optimallashtirilgan va tushunarli kod yozish mumkin. Ta’lim jarayonida ushbu tushunchalarni to‘g‘ri yetkazish kelajak dasturchilarni tayyorlashda muhim ahamiyatga ega.

Foydalanilgan adabiyotlar

1. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition. MIT Press, 2009. 1312 p.

2. Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P. Numerical Recipes: The Art of Scientific Computing, 3rd Edition. Cambridge University Press, 2007. 1235 p.

3. Strang G. Introduction to Linear Algebra, 5th Edition. Wellesley-Cambridge Press, 2016. 584 p.

4. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 775 p.

5. Van Rossum G., Drake F.L. The Python Language Reference Manual. Network Theory Ltd, 2011. 151 p.

(cc) BY

6. Harris C.R., Millman K.J., van der Walt S.J. et al. Array programming with NumPy // Nature. 2020. Vol. 585. P. 357-362.

7. Golub G.H., Van Loan C.F. Matrix Computations, 4th Edition. Johns Hopkins University Press, 2013. 756 p.

8. Trefethen L.N., Bau D. Numerical Linear Algebra. SIAM, 1997. 361 p.

9. McKinney W. Python for Data Analysis, 2nd Edition. O‘Reilly Media, 2017. 544 p.

10. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. М.: Вильямс, 2000. 384 с.

11. Кнут Д. Искусство программирования, том 1. Основные алгоритмы. М.: Вильямс, 2006. 720 с.

12. Вирт Н. Алгоритмы и структуры данных. СПб.: Невский Диалект, 2001. 352 с.

13. Sedgewick R., Wayne K. Algorithms, 4th Edition. Addison-Wesley Professional, 2011. 976 p.

14. Skiena S.S. The Algorithm Design Manual, 3rd Edition. Springer, 2020. 793 p.

15. Kolter J.Z., Johnson M.J. Convex Optimization for Machine Learning. Cambridge University Press, 2020. 234 p.