Задание

Сформировать 50 файлов, в каждый из которых записать матрицу 100x100 из случайных целых чисел в диапазоне от -100 до 100. Для матриц сравнить время работы алгоритмов (согласно номеру варианта), реализованных в MATLAB и Python (Numpay). Результаты записать во вновь созданные 50 файлов. Оформить отчет в Microsoft Word.

Варианты заданий

№	**Алгоритм** *(номер в скобках – см. таблицу ниже на стр. 3* ***Linear algebra MATLAB и NumPy****)*
1	Ранг матрицы (63)
2	Транспонирование матрицы (18)
3	Вектор уникальных значений в массиве (80)
4	Сортировка по первому столбцу (77)
5	Выделение главной диагонали (41)
6	Сортировка по столбцам (75)
7	Найти собственные значения и собственные значения матрицы (68)
8	Решение СЛАУ ax=b (64) (*нужен еще вектор свободных членов*)
9	QR-разложение (71)
10	Факторизация Холецкого (67)
11	Максимальный элемент каждого столбца массива (53)
12	Максимальный элемент каждой строки массива (54)
13	Нечетные строки матрицы (15)
14	Найти индексы, для которых элементы матрицы больше 50 (25)
15	Получить матрицу со строками в обратном порядке (16)
16	Максимальный элемент матрицы (52)
17	Решение xa=b (65) (*нужен еще вектор свободных членов*)
18	Обнулить элементы матрицы, которые больше 50 (28)
19	Поэлементно возвести элементы матрицы в 3 степень (23)
20	Создать новую матрицу 2*2 копированием старой матрицы (49)
21	Выделить последние n строк двумерного массива (11)
22	Умножить матрицы (20)
23	Поэлементно умножить матрицы (21)
24	Поэлементно разделить матрицы (22)
25	Поэлементный оператор ИЛИ (58)
26	Поэлементный оператор И (57)
27	Побитовый оператор И (59)
28	Побитовый оператор ИЛИ (60)
29	Формирование новой матрицы из указанных строк и столбцов исходной (13)
30	Добавить первую строку в конец матрицы (17)

Документация и учебники

1. https://docs.scipy.org/doc/numpy/user/index.html - Numpy (+tutorial)
2. https://docs.scipy.org/doc/numpy/reference/index.html#reference - Numpy документация
3. http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html - Numpy tutorial by Nicolas P. Rougier
4. http://www.labri.fr/perso/nrougier/teaching/numpy.100/index.html - 100 упражнений по numpy
5. Numpy_Python_Cheat_Sheet.pdf: шпаргалка на 1 лист А4

Пример. Производительность numpy. Сумма первых 108 чисел

Цикл

```
from datetime import datetime
import time

start_time = datetime.now()

sum_value = 0
for i in range(10 ** 8):
    sum_value += i
print(sum_value)

print(datetime.now() - start_time)
```

range, sum

```
from datetime import datetime
import time

start_time = datetime.now()

sum_value = sum(range(10 ** 8))
print(sum_value)

print(datetime.now() - start_time)
```

Библиотека numpy

```
import numpy as np
from datetime import datetime
import time

start_time = datetime.now()

sum_value = np.arange(10 ** 8).sum()
print(sum_value)

print(datetime.now() - start_time)
```

Linear algebra MATLAB и NumPy

https://numpy.org/doc/stable/user/numpy-for-matlab-users.html

№	MATLAB	NumPy	Notes
1	`ndims(a)`	`np.ndim(a)` or `a.ndim`	number of dimensions of array `a`
2	`numel(a)`	`np.size(a)` or `a.size`	number of elements of array `a`
3	`size(a)`	`np.shape(a)` or `a.shape`	“size” of array `a`
4	`size(a,n)`	`a.shape[n-1]`	get the number of elements of the n-th dimension of array `a`. (Note that MATLAB uses 1 based indexing while Python uses 0 based indexing, See note INDEXING)
5	`[ 1 2 3; 4 5 6 ]`	`np.array([[1. ,2. ,3.], [4. ,5. ,6.]])`	define a 2x3 2D array
6	`[ a b; c d ]`	`np.block([[a, b], [c, d]])`	construct a matrix from blocks `a`, `b`, `c`, and `d`
7	`a(end)`	`a[-1]`	access last element in MATLAB vector (1xn or nx1) or 1D NumPy array `a` (length n)
8	`a(2,5)`	`a[1, 4]`	access element in second row, fifth column in 2D array `a`
9	`a(2,:)`	`a[1]` or `a[1, :]`	entire second row of 2D array `a`
10	`a(1:5,:)`	`a[0:5]` or `a[:5]` or `a[0:5, :]`	first 5 rows of 2D array `a`
11	`a(end-4:end,:)`	`a[-5:]`	last 5 rows of 2D array `a`
12	`a(1:3,5:9)`	`a[0:3, 4:9]`	The first through third rows and fifth through ninth columns of a 2D array, `a`.
13	`a([2,4,5],[1,3])`	`a[np.ix_([1, 3, 4], [0, 2])]`	rows 2,4 and 5 and columns 1 and 3. This allows the matrix to be modified, and doesn’t require a regular slice.
14	`a(3:2:21,:)`	`a[2:21:2,:]`	every other row of `a`, starting with the third and going to the twenty-first
15	`a(1:2:end,:)`	`a[ ::2,:]`	every other row of `a`, starting with the first
16	`a(end:-1:1,:)` or `flipud(a)`	`a[::-1,:]`	`a` with rows in reverse order
17	`a([1:end 1],:)`	`a[np.r_[:len(a),0]]`	`a` with copy of the first row appended to the end
18	`a.'`	`a.transpose()` or `a.T`	transpose of `a`
19	`a'`	`a.conj().transpose()` or `a.conj().T`	conjugate transpose of `a`
20	`a * b`	`a @ b`	matrix multiply
21	`a .* b`	`a * b`	element-wise multiply
22	`a./b`	`a/b`	element-wise divide

№	MATLAB	NumPy	Notes
23	`a.^3`	`a**3`	element-wise exponentiation
24	`(a > 0.5)`	`(a > 0.5)`	matrix whose i,jth element is (a_ij > 0.5). The MATLAB result is an array of logical values 0 and 1. The NumPy result is an array of the boolean values `False` and `True`.
25	`find(a > 0.5)`	`np.nonzero(a > 0.5)`	find the indices where (`a` > 0.5)
26	`a(:,find(v > 0.5))`	`a[:,np.nonzero(v > 0.5)[0]]`	extract the columns of `a` where vector v > 0.5
27	`a(:,find(v>0.5))`	`a[:, v.T > 0.5]`	extract the columns of `a` where column vector v > 0.5
28	`a(a<0.5)=0`	`a[a < 0.5]=0`	`a` with elements less than 0.5 zeroed out
29	`a .* (a>0.5)`	`a * (a > 0.5)`	`a` with elements less than 0.5 zeroed out
30	`a(:) = 3`	`a[:] = 3`	set all values to the same scalar value
31	`y=x`	`y = x.copy()`	NumPy assigns by reference
32	`y=x(2,:)`	`y = x[1, :].copy()`	NumPy slices are by reference
33	`y=x(:)`	`y = x.flatten()`	turn array into vector (note that this forces a copy). To obtain the same data ordering as in MATLAB, use `x.flatten('F')`.
34	`1:10`	`np.arange(1., 11.)` or `np.r_[1.:11.]` or `np.r_[1:10:10j]`	create an increasing vector (see note RANGES)
35	`0:9`	`np.arange(10.)` or `np.r_[:10.]` or `np.r_[:9:10j]`	create an increasing vector (see note RANGES)
36	`[1:10]'`	`np.arange(1.,11.)[:, np.newaxis]`	create a column vector
37	`zeros(3,4)`	`np.zeros((3, 4))`	3x4 two-dimensional array full of 64-bit floating point zeros
38	`zeros(3,4,5)`	`np.zeros((3, 4, 5))`	3x4x5 three-dimensional array full of 64-bit floating point zeros
39	`ones(3,4)`	`np.ones((3, 4))`	3x4 two-dimensional array full of 64-bit floating point ones
40	`eye(3)`	`np.eye(3)`	3x3 identity matrix
41	`diag(a)`	`np.diag(a)`	returns a vector of the diagonal elements of 2D array, `a`
42	`diag(v,0)`	`np.diag(v, 0)`	returns a square diagonal matrix whose nonzero values are the elements of vector, `v`

№	MATLAB	NumPy	Notes

43	rng(42,'twister')
	rand(3,4)
		from numpy.random import default_rng
		rng = default_rng(42)
		rng.random(3, 4)
		or older version: random.rand((3, 4))
			generate a random 3x4 array with default random number generator and seed = 42

44	linspace(1,3,4)	np.linspace(1,3,4)	4 equally spaced samples between 1 and 3, inclusive
45	[x,y]=meshgrid(0:8,0:5)	np.mgrid[0:9.,0:6.] or np.meshgrid(r_[0:9.],r_[0:6.]	two 2D arrays: one of x values, the other of y values
46		ogrid[0:9.,0:6.] or np.ix_(np.r_[0:9.],np.r_[0:6.]	the best way to eval functions on a grid
47	[x,y]=meshgrid([1,2,4],[2,4,5])	np.meshgrid([1,2,4],[2,4,5])	
48		ix_([1,2,4],[2,4,5])	the best way to eval functions on a grid
49	repmat(a, m, n)	np.tile(a, (m, n))	create m by n copies of a
50	[a b]	np.concatenate((a,b),1) or np.hstack((a,b)) or np.column_stack((a,b)) or np.c_[a,b]	concatenate columns of a and b
51	[a; b]	np.concatenate((a,b)) or np.vstack((a,b)) or np.r_[a,b]	concatenate rows of a and b
52	max(max(a))	a.max() or np.nanmax(a)	maximum element of a (with ndims(a)<=2 for MATLAB, if there are NaN's, nanmax will ignore these and return largest value)
53	max(a)	a.max(0)	maximum element of each column of array a
54	max(a,[],2)	a.max(1)	maximum element of each row of array a
55	max(a,b)	np.maximum(a, b)	compares a and b element-wise, and returns the maximum value from each pair
56	norm(v)	np.sqrt(v @ v) or np.linalg.norm(v)	L2 norm of vector v
57	a & b	logical_and(a,b)	element-by-element AND operator (NumPy ufunc) See note LOGICOPS
58	a | b	np.logical_or(a,b)	element-by-element OR operator (NumPy ufunc) See note LOGICOPS
59	bitand(a,b)	a & b	bitwise AND operator (Python native and NumPy ufunc)

№	MATLAB	NumPy	Notes
60	`bitor(a,b)`	`a | b`	bitwise OR operator (Python native and NumPy ufunc)
61	`inv(a)`	`linalg.inv(a)`	inverse of square 2D array `a`
62	`pinv(a)`	`linalg.pinv(a)`	pseudo-inverse of 2D array `a`
63	`rank(a)`	`linalg.matrix_rank(a)`	matrix rank of a 2D array `a`
64	`a\b`	`linalg.solve(a, b)` if `a` is square; `linalg.lstsq(a, b)` otherwise	solution of a x = b for x
65	`b/a`	Solve `a.T x.T = b.T` instead	solution of x a = b for x
66	`[U,S,V]=svd(a)`	`U, S, Vh = linalg.svd(a), V = Vh.T`	singular value decomposition of `a`
67	`c=chol(a)` where `a==c'*c`	`c = linalg.cholesky(a)` where `a == c@c.T`	Cholesky factorization of a 2D array (`chol(a)` in MATLAB returns an upper triangular 2D array, but **`cholesky`** returns a lower triangular 2D array)
68	`[V,D]=eig(a)`	`D,V = linalg.eig(a)`	eigenvalues λ and eigenvectors v¯ of `a`, where λv¯=av¯
69	`[V,D]=eig(a,b)`	`D,V = linalg.eig(a, b)`	eigenvalues λ and eigenvectors v¯ of `a`, `b` where λbv¯=av¯
70	`[V,D]=eigs(a,3)`	`D,V = eigs(a, k = 3)`	find the `k=3` largest eigenvalues and eigenvectors of 2D array, `a`
71	`[Q,R,P]=qr(a,0)`	`Q,R = linalg.qr(a)`	QR decomposition
	`[L,U,P]=lu(a)` where `a==P'*L*U`	`P,L,U = linalg.lu(a)` where `a == P@L@U`	LU decomposition (note: P(MATLAB) == transpose(P(NumPy)))
72	`conjgrad`	`cg`	Conjugate gradients solver
73	`fft(a)`	`np.fft(a)`	Fourier transform of `a`
74	`ifft(a)`	`np.ifft(a)`	inverse Fourier transform of `a`
75	`sort(a)`	`np.sort(a)` or `a.sort(axis=0)`	sort each column of a 2D array, `a`
76	`sort(a, 2)`	`np.sort(a, axis = 1)` or `a.sort(axis = 1)`	sort the each row of 2D array, `a`
77	`[b,I]=sortrows(a,1)`	`I = np.argsort(a[:, 0]); b = a[I,:]`	save the array `a` as array `b` with rows sorted by the first column
78	`x = Z\y`	`x = linalg.lstsq(Z, y)`	perform a linear regression of the form Zx=y
79	`decimate(x, q)`	`signal.resample(x, np.ceil(len(x)/q))`	downsample with low-pass filtering
80	`unique(a)`	`np.unique(a)`	a vector of unique values in array `a`
81	`squeeze(a)`	`a.squeeze()`	remove singleton dimensions of array `a`. Note that MATLAB will always return arrays of 2D or higher while NumPy will return arrays of 0D or higher