Matrix Calculus

Matrix Calculus

Matrix calculus is the field dealing with multivariate calculus over spaces of matrices and vectors

Matlab® is optimised for those computations at the numerical level: whenever possible one should use *vectorized computations*

- Eases the lecture and understanding of the code's
- Increase in the speed performance
- Consistency with Matlab®'s spirit

Arrays

Arrays are the core of Matlab®. As Matlab® states it itself,

> While other programming languages mostly work with numbers one at a time, Matlab® is designed to operate primarily on whole matrices and arrays.

All variables are multidimensional arrays, regardless the type of data. There are two main categories

- Numerical arrays

  Scalar (1×1)

  Vector ($n \times 1$ or $1 \times n$)

  Matrix ($n \times m$)
- Cell array of objects

Vector arrays

Simple creation of a vector array

1. Try the following in the command line:

```
>> a=[8 2 23 5]
>> b=[8, 2, 23, 5]
>> c=[8; 2; 23; 5]
```

2. Check the workspace and look at the information on a, b, c.
3. What do `space` or `,` or `;` do?

Vector arrays

Simple creation of a vector array

1. Try the following in the command line:

```
>> a=[8 2 23 5]
>> b=[8, 2, 23, 5]
>> c=[8; 2; 23; 5]
```

2. Check the workspace and look at the information on *a*, *b*, *c*.
3. What do `space` or `,` or `;` do?

→ The separators `space` and `,` are building a row vector

→ The separator `;` is building a column vector

→ The storage's amount of a vector corresponds to its number of elements times the amount required by one element

Vector arrays

Automatic creation of a vector array

Matlab® has built-in constructors and functions that *initialises* automatically a vector array defined on a predefined sequence

```
>> a = 1:10
>> a = 1:2:10
>> a = 100:-2:-1
>> a = 1.2:0.2:10.5
```

```
>> b = zeros(1,10)
>> b = ones(10,1)
>> b = linspace(1,10,10)
>> b = linspace(1.2,10.4,47)
```

The *indexing* starts at 1 and finishes at `end`. The elements of a vector array are accessible through their index

```
>> a(1)
```

```
>> a(end)
```

```
>> a(2:end-1)
```

Note: Be careful to the bounds!

```
>> b(45:end-5)
```

```
>> a(-1)
```

Getting the properties of a vector array

To perform the desired algebra operations it is crucial to know the orientation (row or column) of a vector array and its length

```
>> size(a)
```

```
>> length(a)
```

Operations over a vector array: addition

```
>> a = 1:5;
>> b = 5:3:17;
>> c = a + b
```

The addition and subtraction are done element-wise


```
>> a = 1:5; b = 5:3:20;
>> c = a - b
```

→ The two vectors should have the same length

Operations over a vector array: transpose

The symbol `'` transposes a given vector: a row vector array will become a column array and *vis-versa*

```
>> a = 1:5;
>> b = a';
>> size(a)
>> size(b)
```

Operations over a vector array: broadcasting

In the new versions of Matlab® (from 2016b on), the operations are *broadcast*: adding two vectors that do not have the same orientation creates a matrix

```
>> b = [3, 2];
>> a = [1; 4];
>> a+b
```

```
ans =

     4     3
     7     6
```

Operations over a vector array: multiplication

The operator `*` is the matrix multiplication. Between row and column vectors, it is the scalar product. The operator `.*` is the element-wise multiplication

```
>> a = 1:5; b = 5:3:17;
>> c = a*b                  % Error
>> c = a.*b                 % Element-wise multiplication
>> c = a*b’                 % Scalar product
>> c = b’*a                 % Broadcast multiplication
```

Note: Be careful to the orientation of the vectors. Because of broadcasting, you may not see the errors

Operations over a vector array: right division

The operator `/` is the matrix right division: A/B determines the matrix C such that $C * B = A$. It applies to vectors by considering that a vector is a $n \times 1$ or $1 \times n$ matrix. The operator `./` is the element-wise usual division

```
>> a = 1:5; b = 5:3:17;
>> c = a/b          % Least-squares solution of x*b = a
>> c = b'/a'        % Least-squares solution of x*a = b
>> c = a./b         % Element-wise usual division
>> c = a/b'         % Error
>> c = a'/b         % Error
```

Matrices

Simple creation of a matrix

1. Try the following in the command line:

```
>> clear all; clc;
>> a=[1 2 3; 4 5 6]
>> b=[1,2,3; 4,5,6]
```

2. Check the workspace and look at the information on a, b.

3. What do `space` or `,` or `;` do?

Matrices

Simple creation of a matrix

1. Try the following in the command line:

```
>> clear all; clc;
>> a=[1 2 3; 4 5 6]
>> b=[1,2,3; 4,5,6]
```

2. Check the workspace and look at the information on a, b.
3. What do `space` or `,` or `;` do?

→ The separators `space` and `,` are separating the columns

→ The separator `;` is separating the rows

→ The storage's amount of a matrix corresponds to its number of elements times the amount required by one element

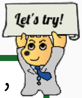
Automatic creation of a matrix

Matlab® has built-in functions and assembling techniques that *initialises* automatically a matrix defined on a predefined structure

```
>> % Using constructors
>> A = []            % Empty
>> A = eye(5)        % Identity
>> A = zeros(4,3)    % Zeros
>> A = ones(4,3)     % Ones
>> A = rand(5) (0,1) % Uniform
>> A = randn(3,4)    % Normal
>> A = magic(5)
```

$1 \cdots n^2$ random

```
>> % Assembling
>> A = eye(2)
>> B = [1:5]+[1:3]'
>> C = ones(2,3)
>> D = [A C; B]'
```




```
>> a = 2*1:5+[1:3]'
>> a = 2*[1:5]+[1:3]'
```

Matrices

Knowing the size of a matrix

As for vectors, the (multidimensional) size of a matrix is given by the command `size`. For a given matrix A, the command `length(A)` returns `max(size(A))`

```
>> size(D)
>> size(D,1) % Number of rows
>> size(D,2) % Number of columns
```

```
>> length(D)
>> max(size(D))
```

Reshaping matrices

One can construct a matrix from a single vector or stack a matrix into a vector. **The total number of elements must be preserved**

1 x 25

```
>> D2 = reshape(D,1,(size(D,1)*size(D,2)))
>> a = 1:16
>> A = reshape(a,4,4)  % Fills the columns successively
```

Accessing elements

As vector arrays, the *indexing* starts at 1 and finishes at `end`. The matrix's elements are accessible giving one index per dimension

```
>> D(2,3)          % Second line, third column
>> D(1,end)        % Last element of the first line
```

Extracting ang glueing submatrices

Extract a submatrix by giving a list of indices for each dimension

```
>> D(1:3, 1:2:6)
>> D(1:3, [1 4 3])
>> D(1:5)
>> D(1, 1:end)
```

```
>> D(:)
>> D(:,1:3)
>> D(1:3,:)
>> D(:,end)
```

Paste multiple copies of a same matrix with `repmat(D,2,2)`

Operations over matrices

Element-wise sum/substraction	+ -
Element-wise multiplication/division	.* .\ ./
Matrix multiplication (ex. `B*A, 3*B`)	*
Element-wise power	.^
Power for square matrix	^
Conjugate transposed (ex. `A'`)	'
Transposed (ex. `A.'`)	.'
Inverse of matrix	`inv`
Linear system solver :$A \backslash b$ solves $A * x = b$	\
Element-wise logic operations	| & ∼
Element-wise comparison operators	== ∼= < > <= >=

$$\begin{bmatrix} 3i & -2i \\ 1 & 5+6i \end{bmatrix} \downarrow \begin{bmatrix} -3i & 1 \\ 2i & 5-6i \end{bmatrix}$$

Matrices

Operations over a matrix's diagonal

To retrieve the values of a matrix's diagonal in a vector shape, use the function `diag`. The same function can also create a diagonal matrix from a single vector

```
>> A = diag([1,2,3])
>> a = diag(A)
```

Retrieving the k^{th} diagonal is done in a similar way

```
>> k = 1
>> A = diag([1,2,3],k)
>> A = diag([1,2,3],-k)
```

```
>> A = magic(5)
>> a = diag(A,2)
```

Broadcast operations

In the new versions of Matlab® (from 2016b on), the operations are *broadcast*: one can add scalars or vectors to matrices

```
>> A = [1, 2; 3, 4];
>> A + 1
   ans =
         2     3
         4     5
```

```
>> A = [1, 2; 3, 4];
>> b = [3, 2];
>> A+b
   ans =
         4     4
         6     6
```

Matrices

1. Define in Matlab® the following matrix and call it A.

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 5 & 4 & 3 & 2 & 1 \\ 5 & 8 & 2 & 0 & 1 & 4 \end{pmatrix}$$

2. What happens when using a single index?

```
>> A(5)
```

3. What happens here?

```
>> A=zeros(4,5);
>> A(:)=1:numel(A)
```

1. Define in Matlab® the following matrix and call it A.

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 5 & 4 & 3 & 2 & 1 \\ 5 & 8 & 2 & 0 & 1 & 4 \end{pmatrix}$$

2. What happens when using a single index?

```
>> A3(5)
```

3. What happens here?

```
>> A=zeros(4,5);
>> A(:)=1:numel(A)
```

column-wise

elements of A

→ The first element of the 5^{th} row is returned

→ An empty matrix is created. Then, the elements are filled by the integers 1 to the number of elements of A

Multidimensional arrays

In Matlab®, vector and matrices arrays behave alike. And more generally, any array has the same Matlab® structure

Initialisation

```
>> A=rand(4, 5, 7, 4)
>> size(A)
```

Access to values

```
>> A(1,2,3,4)
>> A(1,:,:,4)
```

The syntax introduced above apply in a similar way to multidimensional arrays. The operators also act alike provided that there exists an algebraic meaning to the given instruction

Operations on arrays

Modifying array's elements

The elements of any array can be changed by overwriting the values stored at locations pointed by given indices

```
>> D = [1,2,3;4,5,6;7,8,9]
>> D(2,3)=7           % Changes this specific element
>> D(:,2)= [12,13,14] % Changes the full row
```

Extending an array

An array extension is done by storing a value at a new location

```
>> D = [1,2,3;4,5,6;7,8,9]
>> D(4:5,:) = [7,8,9;10,11,12]  % Two rows are added
>> size(D)                      % The dimension changed
```

Note: Mind the dimension compatibility when changing arrays

Operations on arrays

Testing an array element-wise

Testing the value of the array's elements against some condition is done through logical operators. It returns a *logical array*

```
>> A = [1 2 3 4; 5 4 3 2]
>> A<4
```

Finding elements in an array

The command `find` returns the index of the array's elements that satisfy a wished condition. The array's structure is not preserved

```
>> find([1,2,3,4]>6)      % Returns empty vector
>> find([2,3,4,5]>3)      % [3,4]
>> find(A<4)              % [1 3 5 6 8]'
>> A(find(A<4))           % [1 2 3 3 2]'
```

column-wise

Operations on arrays

Clearing elements in an array

Removing a row or a column is done by assigning `[]` as a new value to the column or row one wishes to delete

```
>> A(:,[2,4]) = []   % Remove the 2nd and 4th columns
>> A(1,:) = []       % Remove the first row
>> A(1:3,1:2) = []   % Error: cannot remove block
>> A(1,1) = []       % Error : portions of the matrix
```

Removing spurious dimensions

The function `squeeze` removes the dimensions that are not necessary for representing the array

column-wise

```
>> D2=reshape(D,[3,3,1,1,2,1,1])
>> D3=squeeze(D2)
```

```
>> size(D2)   3,3,1,1,2
>> size(D3)   3,3,2.
```

Shifting dimensions of multidimensional array

The dimensions of any array can be shifted, rolling the matrix to the left (right) when the shift index is positive (negative)

```
>> C=shiftdim(D3,1);
>> size(C)
```

D:3,3,2

C:3 2 3

```
>> D3(3,2,1)
>> C(2,1,3)
```

Generic functions on multidimensional arrays

Most of the functions operate on multidimensional arrays with the same syntax as for vector and matrix arrays

```
>> A=rand(3);
>> B=repmat(A,[1,1,2])
>> size(B)
```

```
>> A=rand(3);
>> sin(A)
>> log(A)
```

Operations on arrays

Functions designed for arrays

Some functions used to extract and infer information from a given array are already implemented. By default, they act column-wise

max(A, [], dim)
[max of every row]
$= \begin{bmatrix} \text{maxRow1} \\ \text{maxRow2} \\ \vdots \end{bmatrix}$

`max`	Find the maximum
`min`	Find the minimum
`sum`, `mean`, `median`	Statistical quantities
`std`	Standard deviation

all element

```
>> max(b(:))
>> max(b)
```

max of each column

```
>> max(A(:))
>> max(A)
```

sum for each column

```
>> s = sum(D3)
>> size(s)
```

To specify the dimension along which the function acts:

```
>> max(A,2)
```

compare A & 2.

```
>> sum(D3,1)
>> sum(D3,[1,2])
```

sum (A, dim)

Functions that act on two multidimensional arrays

`isequal` Checks if two matrices are equal

`kron` Computes the outer tensor product of two matrices:

$$kron(A,B) = \begin{pmatrix} A_{11}B & A_{12}B & ... & A_{1n}B \\ ... & ... & ... & ... \\ A_{m1B} & ... & ... & A_{mn}B \end{pmatrix}$$

```
>> isequal(D2, D3)
```

```
>> kron(A, A')
```

Vectorisation spirit

1. Initialise the three quantities
   `A = magic(200); b = ones(200,1)` and `c=0;`
2. Create two scripts containing the following instructions

```
% Intuitive code
tic
c = 0;
for k=1:200
  for l=1:200
    c =c +...
       A(k,l)*b(l);
  end
end
speed1 = toc;
```

```
% Vectorised code
tic
c = sum(A*b);
speed2 = toc
```

3. Check that the two results for `c` match and compare the values of `speed1` and `speed2`.

Exercises

Exercises: Matrix calculus

Exercise

1. Construct in Matlab the following matrices in an efficient way (use `help diag`):

blkdiag

horzcat

vertcat

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \end{pmatrix}$$

$$B = \begin{pmatrix} 1 & 1 & 1 & 1 & 3 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 4 & 0 \\ 2 & 2 & 2 & 2 & 0 & 0 & 5 \\ 2 & 2 & 2 & 2 & 6 & 6 & 7 \\ 2 & 2 & 2 & 2 & 6 & 6 & 8 \end{pmatrix} \quad C = \begin{pmatrix} 5 & 1 & 0 & 0 & 0 & 5 & 5 & 6 & 7 & 8 \\ 6 & 0 & 2 & 0 & 0 & 6 & 1 & 0 & 3 & 0 \\ 7 & 0 & 0 & 3 & 0 & 7 & 0 & 2 & 0 & 4 \\ 8 & 0 & 0 & 0 & 4 & 8 & 5 & 6 & 7 & 8 \end{pmatrix}$$

Exercise

2. Generate a 10×10 random matrix. Then, find the values and indexes of the biggest element of each row.

   Possible hints: `sort, max, find`

3. Compute the inverse of a random vector (entry–wise).

4. Write a function, which shifts cyclically a vector or matrix by a certain number of rows down and columns to the right (**do not use** `circshift`).

Exercises: Matrix calculus

Exercise

5. Checkboard:
   a) Generate a random (`rand`) matrix A with size $n \times n$ and a second matrix with the same size as A and which takes some elements of A in the black fields, while zeros everywhere else. The structure of the second matrix should be similar to a chessboard.

   b) Make sure that your code works for the cases when n is even and odd. To check if you succeeded try `spy` on your matrix.

Exercises: Matrix calculus

Exercise

6. Give the in-built functions of MatLab that can you use to compute, given a matrix of type $m \times n$,
   a) the average,
   b) the median,
   c) the mode.

7. Check what the following functions do.
   a) `rank` 秩
   b) `null` A的零空间 $x = null(A) \Rightarrow Ax = 0$
   c) `rref` 行阶梯矩阵 reduced row echelon form
   d) `orth` 标准正交基 orthonormal basis

Exercises: Matrix calculus

Exercise

8. Guess and verify on the computer what is happening.

```
>> hist(rand(1,100000),100)
>> hist(randn(1,100000),100)
```

9. Let us consider the square matrix A defined as

```
>> A=[1+i, 1+2i; 2, 4i]
```

Check the following expressions.

```
>> A'
>> A.'
>> A.^A
```

Exercise

10. Consider `A=rand(10)` and check the result of

```
>> diag(diag(A)) + diag(diag(A,1),1)+...
   diag(diag(A,2),2)
```

11. Transform the matrix

```
>> A=reshape(1:20, 4, 5)';
```

through elementar row changes in a reduced row-echelon form (germ.: Zeilenstufenform). You should set to 0 any number smaller than `tol=10^(-13)`. You can compare your results with `rref`.

Exercise

12. Guess the type and value of the following expressions, where `A=[1 2; 3 4*i]`. Check it then in the prompt.

    a)
    ```
    >> isequal(A, A');
    ```

    b)
    ```
    >> sum(sum(A - A')') == 10;
    >> sum(diag(A == A')) & abs(sum(sum(A)));
    ```

Exercises: Matrix calculus

Exercise

13. Generate a table with N values of the function $f(x)$ on the interval $[a, b]$, s.t. you get an $N \times 2$ matrix having on each row x and $f(x)$. Consider the following functions:
    a) $f(x) = x^5 - 4x + 1$
    b) $f(x) = exp(i\frac{x}{10})$
14. Find and cure the error in the script on the webpage `search_error.m`. The script should give an ordered list of numbers generated by a random vector.