Basics: vectors, matrices, arrays

October 30, 2016

Vectors, matrices, arrays

Basic data structure: Matrix (array), elements: complex numbers. Let's limit ourselves at first to two-dimensional arrays.

```
>> rowvect=[1 2 3 4]   % List of elements
>> 1:4                 % Same with colon(:)-operator
 ans = 1   2   3   4
                      % ans:previous non-assigned result
>> colvect=[1;2;3;4]
>> rowvect'== colvect    % Transpose of row-vector
                    % NOTE lhs == rhs -> true/false
>> length(rowvect)        % Nr. of elements
     ans = 4
```

Vectors, matrices,arrays (continued)


```
>> A=[1 2 3 4 ;5 6 7 8; 9 10 11 12]
>> [m,n]=size(A)  % --> m=3, n=4
>> [size(A,1) size(A,2)]
ans =
   3     4
>> v=-[1 2 3 4 ]
>> length(v)
ans =
   4
>> who, whos      % show workspace variables
```

- Column vector: `(m,1)`-matrix
- Row vector: `(1,n)`-matrix
- Scalar: `(1,1)`-matrix
- Empty: `(m,0)` or `(0,n)`-matrix

Creating arrays from smaller parts

- Square brackets `[...]` to define arrays
- Spaces (and/or commas) to separate columns (elemnts of row vector).
- Semi-colons (;) to separate rows (elements of column vector)
- `>> [ 3 4 5 ; 6 7 8 ]` is a 2-by-3 matrix
- If A and B are arrays with the same number of rows, then `>> C = [ A B ]` is the array formed by stacking A and B side by side

```
 >> A=ones(2,2);B=2*ones(2,3);[A B]
ans =
     1     1     2     2     2
     1     1     2     2     2
```

Creating arrays..., continued

- If A and B are arrays with the same number of columns, then `>> [ A ; B ]` is the array formed by stacking A on top of B.
- So, `[[ 3 ; 6 ] [ 4 5 ; 7 8 ] ]` is equal to `[ 3 4 5;6 7 8 ]`

Calculus with vectors

- The numbers $0, 0.1, 0.2, \ldots, 10$ can be assigned to the variable `u` by typing `u = 0:0.1:10;`
- `length(u)` reveals us that there are 101 elements in `u`.
- To compute $w = 5 \sin u$ for $u = 0, 0.1, 0.2, \ldots, 10$, the session is;

```
>>u = 0:0.1:10;                  % By 'misuse' of Matlab:
>>w = 5*sin(u);                  >>for k=1:length(u)
%'Implied for-loop',              w(k)=5*sin(u(k));
% vectorization                    end;
```

- This was our first acquaintance with "vectorization".

"Scalar functions" support vectorization

The previous example leads us to the following general idea: Functions which applied to a scalar produce a scalar result are called *scalar functions*. When such functions are applied to an array, they operate on every element of the array. Mathematical functions `help elfun, specfun` among others are of this type.

```
>> t = [-1 0 1];
>> y = exp(t)
y =
    0.3679    1.0000    2.7183
>> [exp(-1) exp(0) exp(1)]
ans =
   0.3679    1.0000    2.7183
```

"Scalar functions" support vectorization (contnd.)

Assume we want to compute values of

$$y = e^{-x} \sin x$$

at a vector x. We need the vector
$y = (e^{-x(1)} \sin(x(1)), e^{-x(2)} \sin(x(2)), \ldots, e^{-x(n)} \sin(x(n)))$
Here we need the pointwise product `(.*)` of two vectors:

```
>> x=-pi:.1:pi;
>> y=exp(-x).*sin(x);
```

This is just the data we need for plotting. `>> plot(x,y)`

"Vector functions" (also support vectorization)

- Let vector function f mean that `f(vector)=scalar`. Such function operate on a matrix columnwise returnig a vector whose length is the length of rows (= nr. of columns). Examples are many that are classified under *datafunction*, `help datafun`.
- Example: `max,min,mean,sum,...`. Also many functions of type `f(vector)=vector` behave similarly, except they applied to a matrix return a matrix "columnwise". Examples: `sort,cumsum,cumprod, ...`
- These functions can be called to operate rowwise: for example `min(A,2)` forms a column vector of row-minimums. Simlarly for higher-dimensional arrays as well.

Functions for building vectors colon(:),linspace,logspace

- `v=a:b, w=a:h:b;` default: h=1
- `v=linspace(a,b,N);` default: N=100
- `v=logspace(a,b,N);` $10^a, \ldots, 10^b$, N points

```
>> 0:10; 0:.1:1;
>> 10:-2:0
  ans =
    10     8     6     4     2     0
>> logspace(0,1,4)
   ans =
    1.0000    2.1544    4.6416   10.0000
>> 10.^linspace(0,1,4)
   ans =
    1.0000    2.1544    4.6416   10.0000
```

Note: Remember semicolon (;) for large N or small h.

Some functions for building matrices

`eye,vander,hilb,zeros,ones,diag,rand,reshape,magic`

Complete list: `help elmat`

```
>> A = zeros(2,5)
>> B = ones(3)      % or ones(3,3)
>> R = rand(3,2)
>> N = randn(3,2)
>> D = diag(-2:2)
```

Compare `rand` and `randn` Try repeatedly

`>> R = rand(3,2)` Use (↑) in command window

Repeat : `>>rand('twister',0); R = rand(3,2)`

Matrices: building, parts, decomposing

reshape

- Forms a matrix of given size for given data.
- Data will be placed in "frame" of given size in **column order**. (Matlab is column oriented.)
- Nr. of datapoints (`numel(data)`) has to match product of dimensions.

```
>> A=reshape(1:6,2,3)   % 2x3 matrix from data ...
    1:6 in column order
>> B=reshape(1:6,3,2)' % Row-order
>> C=reshape(A,1,6)     % Back to vector 1:6
```

Matrices, building blocks


```
>> A=reshape(1:6,2,3); B=ones(2,2),C=diag(1:3)
>> [A B]  % Side by side.
%ans =
     1     3     5     1     1
     2     4     6     1     1
>> [A;C]  On top of each other.
ans =
     1     3     5
     2     4     6
     1     0     0
     0     2     0
     0     0     3
>> cumsum(ones(3,4)) % What happens here?
          % One of many ways to duplicate a column.
```

Matrix- and array algebra

`A`, `B` matrices, matching size, `c` scalar.

Matrix algebra

- `A + B`, `A+c`
- `A*B` matrix product
- `A'` (conjugate) transpose
- `A.'` transpose without conjugation
- `A^p` (`A*A*...A`) Matrix power (A square matrix.)
- `A\b`
  $Ax = b \iff x = A\backslash b$ (if A is invertible)

Array algebra

"scalar extension"

- `A + B`, `A+c`
- `A.*B` Pointwise product
- `A.^p`, `A.^B` Pointwise power, p scalar, A and B of same size.
- `A./B`, `c./A` Pointwise divide. Subtle `1.0/A`, `1.0./A`,`1./A`
- **Note:** `c/A` usually leads to an error.