Matrices

Peter Hertel

University of Osnabrück, Germany

Lecture presented at APS, Nankai University, China

`http://www.home.uni-osnabrueck.de/phertel`

Spring 2012

Overview

- In MATLAB practically everything is a matrix
- numbers, row and column vectors, proper matrices, strings, etc.
- more methods to construct a matrix
- operations on matrices
- solving systems of linear equations
- various methods to access the elements of a matrix
- example: reorganize a random matrix
- remarks on rounding errors
- more on matrices

Assembling matrices

- assemble horizontally by a comma
- `FRA=[50.0439,8.5492];`
- `PEK=[39.9937,116.4992];`
- `PVG=[31.1433,121.8053];`
- assemble vertically by a semicolon
- `flight=[FRA;PEK;PVG]`
- this 3×2 (3 rows, 2 columns) matrix describes a flight from Frankfurt via Beijing to Shanghai/Pudong.

Assembling matrices cntd.

- the same can be acchieved by
- `LAT=[50.0439;39.9937;31.1433];`
- `LON=[8.5492;116.4992;121.8053];`
- `flight=[LAT,LON];`
- when composing a matrix: all columns must have the same length, and all rows must have the same length
- try `A=[1,2,3;4,5]`
- MATLAB does not tolerate errors, e.g. by filling in zeros for missing entries.
- MATLAB is type-safe.

More methods

- construct a $r \times c$ matrix of zeros
  ```
  X=zeros(r,c);
  ```
- construct a $r \times c$ matrix of ones
  ```
  Y=eye(r,c);
  ```
- row vector of equally spaced numbers
  ```
  x=linspace(0,2*pi,1024)
  ```
- $r \times c$ matrix of random numbers in $[0, 1]$
  ```
  R=rand(r,c);
  ```
- $r \times c$ matrix of normally distributed random numbers
  ```
  N=randn(r,c);
  ```
- diagonal square matrix
  ```
  D=diag(v);
  ```
  where `v` is a vector of diagonal elements
- more on strings

Strings

- strings are row vectors of characters
- they are entered by `'...'`
- such as `name='phertel';`
- or `server='uos.de';`
- strings may be concatenated
  `email=[name,'@',server];`
- yielding `phertel@uos.de`

Multiplication by a number and addition

- the shape of a matrix `A` is given by `[r,c]=size(A)`
- `[r,c]=size([1,2,3;4,5,6]` gives `r=2` and `c=3`
- `B=lambda*A` multiplies each element of matrix `A` with number `lambda`
- same for division by a number: element- or pointwise
- two matrices `A` and `B` of the same shape are added pointwise, such as in `C=A+B`
- same for `C=A-B`

Multiplication of matrices

- matrices of the same shape may be multiplied pointwise
- $C_{ij} = A_{ij}B_{ij}$ is achieved by `C=A.*B`
- if `A` has as many columns as `B` has rows, the matrix product

  $$C_{ik} = \sum_j A_{ij}B_{jk}$$

  is well defined
- in MATLAB: `C=A*B`
- the code for this is extremely efficient

```
% demonstrate efficiency of matrix multiplication
RA=100;
CA=200;
RB=CA;
CB=300;
A=randn(RA,CA);
B=randn(RB,CB);
C=zeros(RA,CB);
tic;
for i=1:RA
    for k=1:CB
        sum=0;
        for j=1:CA
            sum=sum+A(i,j)*B(j,k);
        end;
        C(i,k)=sum;
    end;
end;
toc
tic;
C=A*B;
toc
```

Block operations are efficient!

- the `for` loop version requires 17 seconds.
- the block operation command `C=B*A` was done in less than 4 milliseconds.
- If possible, avoid `for` loops
- However, sometimes a `for` loop construction is easier to program and to understand

Linear equations

- let us discuss the normal case
- there are N linear equations for N unknown variables x_j
- i.e.
$$\sum_{j=1}^{N} M_{ij}x_j = y_i \text{ for } i = 1, 2, \ldots, N$$
- the solution must fulfill `M*x=y`
- this is solved in MATLAB by `x=M\y`
- `M\` should be read as M^{-1}
- there is more to say on systems of linear equations
- overdetermined, underdetermined, singular, many right hand sides, etc.

Extracting sub-matrices

- `x=A(j,k);` extracts a matrix element, a number
- `C=A(:,k);` extracts k-th column
- `r=R(j,:);` extracts j-th row
- most general `B=A(r,c);`
- where `r` and `c` are vectors of indexes
- e.g. the upper left corner `B=A([1,2],[1,2]);`
- `ndx=[1:5];` gives `[1,2,3,4,5]`
- generalization `from:step:to`
- if `[M,N]=size(A);`
- `B=A([1:2:M],[1:2:N]);` is a sub-matrix with odd indexes
- when indexing, the `[...]` brackets may be omitted

```
% dissect a 2N times 2N matrix into four N times N blocks
% displace them counter-clockwise
% the sum over all entries should be invariant
% up to rounding errors
% (a+b)+c = a+(b+c) for real numbers is not guaranteed
% eps is the smallest number such that 1 and 1+eps differ

N=4;
A=randn(2*N,2*N);
lo=1:N;
hi=N+1:2*N;
B=zeros(2*N,2*N);
B(hi,lo)=A(lo,lo);
B(hi,hi)=A(hi,lo);
B(lo,hi)=A(hi,hi);
B(lo,lo)=A(lo,hi);
check=sum(sum(B))-sum(sum(A));
fprintf(1,'error is %i bits\n',round(check/eps));
```

More on matrices

- `d=det(A);` gives the determinant of a square matrix `A`
- `B=A';` produces the conjugate matrix $B = A^\dagger$ defined by $B_{jk} = (A_{kj})^*$
- `eig` is one of the most powerful commands in MATLAB. It comprises tons of software, old and modern, optimized to the last bit and checked like no other piece of software. It comes also in a version for sparse matrices.
- `eval=eig(A);` returns the eigenvalues of a square matrix `A`. In general, they are complex numbers.
- `[V,D]=eig(A)` delivers a matrix `V` of eigenvectors (columns) and a diagonal matrix `D`. The entries on the diagonal are eigenvalues. In fact, the problem $AV = VD$ is solved, or $A = VDV^{-1}$, or $D = V^{-1}AV$.
- hermitian, unitary, real and symmetric, orthogonal ...

Summary

- in MATLAB, practically everything is a matrix
- numbers, row and column vectors, matrices, strings
- `A=[1,2,3;4,5,6]`
- `size, zeros, ones, eye, linspace, rand, randn`
- dotwise addition, dotwise multiplication and genuine matrix multiplication
- matrix division or systems of linear equations
- sub-matrix extraction by `A(j,k)`
- `[from:step:to]` notation for indexes
- `[from:to]` assumes step 1
- `:` assumes all allowed indexes
- eigenvalues and diagonalization
- and much more...