Computational Foundations of Cognitive Science

Lecture 11: Matrices in Matlab

Frank Keller

School of Informatics
University of Edinburgh
keller@inf.ed.ac.uk

February 23, 2010

Reading: McMahon, Ch. 2

Sum and Difference

In Matlab, matrices are input as lists of numbers; columns are separated by spaces or commas, rows by semicolons or newlines:

```
> A = [2, 1, 0, 3; -1, 0, 2, 4; 4, -2, 7, 0];
> B = [-4, 3,  5,  1
        2, 2,  0, -1
        3, 2, -4,  5];
> C = [1 1; 2 2];
```

The sum and difference of two matrices can be computed using the operators + and −:

```
> disp(A + B);
 -2   4   5   4
  1   2   2   3
  7   0   3   5
```

Sum and Difference

For sum and difference, matrices have to have the same dimensions:

```
> disp(A - B);
    6   -2   -5    2
   -3   -2    2    5
    1   -4   11   -5
> disp(A + C);
error: operator +: nonconformant arguments
(op1 is 3x4, op2 is 2x2)
```

Size; Product with Scalar

Matlab uses the functions `columns(A)`, `rows(A)`, and `size(A)` for determining the size of a matrix:

```
> disp(columns(A));
  4
> disp(rows(A));
  3
> disp(size(A));
  3   4
```

A matrix can be multiplied with a scalar using the operator `*`:

```
> disp(A * 2);
    4    2    0    6
   -2    0    4    8
    8   -4   14    0
```

Zero and Identity Matrix

The command `zeros(n)` generates a zero matrix of size n. Use `zeros(n, m)` if the matrix isn't square:

```
> disp(zeros(2));
   0   0
   0   0
> disp(zeros(2, 4));
   0   0   0   0
   0   0   0   0
```

The command `ones(n)` and `ones(n, m)` construct a matrix of ones in the same way. To generate the identity matrix, use `eye(n)`:

```
> disp(eye(3));
   1   0   0
   0   1   0
   0   0   1
```

Diagonal Matrices

To extract the main diagonal of a matrix A use `diag(A)`:

```
> A = [3 1 -7; 2 4 11; 3 3 9];
> disp(diag(A));
   3
   4
   9
```

To create a matrix based on a diagonal use:

```
> A = diag([1 2 3]);
> disp(A);
   1    0    0
   0    2    0
   0    0    3
```

Triangular Matrices

Use `triu(A)` to get the upper triangular part of A, and `tril(A)` to get the lower triangular part.

```
> A = [3 1 -7; 2 4 11; 3 3 9];
> disp(triu(A));
    3     1    -7
    0     4    11
    0     0     9
> disp(tril(A));
   3    0    0
   2    4    0
   3    3    9
```

You can also use `triu(A, k)` to get the elements above the main diagonal ($k > 0$) or below the main diagonal ($k < 0$).

Block Matrices

A block matrix is a matrix that can be partitioned into smaller matrices called blocks. We can generate this in Matlab by concatenating the blocks:

```
> A = [1, 1; 1 1];
> B = [2, 2; 2 2];
> disp([A B A]);
   1   1   2   2   1   1
   1   1   2   2   1   1
> disp([A B; A]);
error: number of columns must match (2 != 4)
> disp([A B; B A]);
   1   1   2   2
   1   1   2   2
   2   2   1   1
   2   2   1   1
```

Block Matrices

Alternatively, we can generate a block matrix by repeating the same block multiple times using `repmat(A)` or `repmat(A, k)`:

```
> A = [1, 2; 3 4];
> disp(repmat(A, 2));
   1   2   1   2
   3   4   3   4
   1   2   1   2
   3   4   3   4
> disp(repmat(A, 2, 3));
   1   2   1   2   1   2
   3   4   3   4   3   4
   1   2   1   2   1   2
   3   4   3   4   3   4
```

Row and Column Vectors

To extract the element $(A)_{ij}$ of matrix A, use `A(i, j)` in Matlab:

```
> A = [2, 1, 0, 3; -1, 0, 2, 4; 4, -2, 7, 0];
> disp(A(1, 4));
  3
> disp(A(2, 3));
  2
```

To extract the row vector $\mathbf{r}_i(A)$, use `A(i, :)`, for the column vector $\mathbf{c}_j(A)$, use `A(:, j)`:

```
> disp(A(1, :));
   2    1    0    3
> disp(A(:, 4));
   3
   4
   0
```

Row and Column Vectors

Vectors can be concatenated to form a matrix:

```
> v1 = [8; 2; 1; 4]; v2 = [3; 9; 11; 6];
> v3 = [0; 2; 2; 4];
> A = [v1, v2, v3];  disp(A);
    8    3    0
    2    9    2
    1   11    2
    4    6    4
```

We can also change entries using `A(i, j) = n` or delete rows or columns using `A(i, :) = []` and `A(:, j) = []`:

```
> A(1, :) = []; disp(A);
    2    9    2
    1   11    2
    4    6    4
```

Mid-lecture Problem

Suppose you have the matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$.

How do you use Matlab to turn it into $B = \begin{bmatrix} 7 & 8 & 9 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \end{bmatrix}$?

Matrix Product

The operator * can also be used to multiply two matrices. Again, the dimensions have to agree:

```
> A = [ 2  1  0;
       -1  0  2;
        4 -2  0];
> B = [1 2;
       2 1;
       0 6];
> disp(A * B);
    4    5
   -1   10
    0    6
> disp(B * A);
error: operator *: nonconformant arguments
(op1 is 3x2, op2 is 3x3)
```

Matrix Product

There is also the operator `.*`, which multiplies matrices element by element:

```
> C = [0 0 1; 2 1/2 1; 1 1 5];
> disp(A .* C);
   0    0    0
  -2    0    2
   4   -2    0
```

This has no equivalent in mathematics, but is useful for programming (other elementwise operators exist, e.g., `./` and `.^` for elementwise division and exponentiation).

Product with Vector

The matrix multiplication operator * can be used to multiply a matrix with a vector:

```
> u = [1; 2;  1];
> v = [0; 1; -2];
> disp(A * v);
  1
 -4
 -2
```

And the array multiplication operator .* can also be applied to vectors:

```
> disp(u .* v);
  0
  2
 -2
```

Product with Vector

To compute $A\mathbf{v}$, we can also extract the column vectors of A and multiply them with the components of $\mathbf{v}$:

```
> disp(v(1) * A(:, 1) + v(2) * A(:, 2) + v(3) * A(:, 3));
   1
  -4
  -2
```

We can check the linearity properties of the product with a vector:

```
> disp(A * (1/2 * v)); disp(1/2 * (A * v));
   0.5                    0.5
  -2                     -2
  -2                     -2
> disp(A * (u + v));    disp(A * u + A * v);
   5                      5
  -3                     -3
  -2                     -2
```

Mid-lecture Problem

Suppose you have the matrix $A = \begin{bmatrix} 3.5 & 7.4 & 3.2 \\ 1.5 & 3.9 & 4.0 \\ 9.2 & 4.8 & 4.2 \\ 1.0 & 3.1 & 0.3 \end{bmatrix}$.

Assume that each of the rows in the matrix represent a series of measurement for a given experiment. Use Matlab to compute the mean for each experiment, and assign the result to a vector.

Transpose

The transpose of a matrix can be computed using `'`. To compute the trace, use the function `trace`:

```
> A = [3 1 -7; 2 4 11; 3 3 9];
> disp(A');
    3     2     3
    1     4     3
   -7    11     9
> disp(trace(A'));
 16
```

With `'` we turn column vectors into row vectors and vice versa:

```
> disp(u');
   1   2   1
> disp(v');
   0   1  -2
```

Symmetric Matrices

We get a symmetric matrix by multiplying it with its transpose:

```
> disp(A * A');
   59   -67   -51
  -67   141   117
  -51   117    99
> disp(A' * A);
   22    20    28
   20    26    64
   28    64   251
```

To check whether a matrix is symmetric use `issymmetric(A)`:

```
> disp(issymmetric(A));
 0
> disp(issymmetric(A * A'));
 3
```

Inner and Outer Product

The inner product $\mathbf{u}^T\mathbf{v}$ and the outer product $\mathbf{u}\mathbf{v}^T$ can be computed using matrix multiplication and the transpose operator:

```
> disp(u' * v);
  0
> disp(u * v');
  0   1  -2
  0   2  -4
  0   1  -2
```

For the inner product, the function `dot` can be used, which computes the dot product:

```
> disp(dot(u, v));
  0
```

Summary

- Matrix sum and difference: `A + B`, `A - B`;
- zero and identity matrix: `zero(n)` and `eye(n)`;
- product of two matrices: `A * B`;
- product of the elements of a matrix: `A .* B`;
- product of a matrix and a scalar, of a matrix and a vector: `A * c`, `A * v`;
- extracting matrix elements and row and column vectors: `A(i, j)`, `A(i, :)`, `A(:, j)`;
- transpose and trace: `A'`, `trace(A)`;
- inner product and outer product: `u' * v`, `u * v'`.