Computational Foundations of Cognitive Science

Lecture 14: Inverses and Eigenvectors in Matlab; Plotting and Graphics

Frank Keller

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

February 23, 2010

Reading: McMahon, Ch. 3

Inverse

The command `inv(A)` computes the inverse of A. Matlab complains if the matrix is singular:

```
> A = [1 2 3; 2 5 3; 1 0 8]; B = [1 6 4; 2 4 -1; -1 2 5];
> disp(inv(A));
  -40   16    9
   13   -5   -3
    5   -2   -1
> disp(inv(B));
warning: inverse: matrix singular to machine precision
```

We can test the property $AA^{-1} = I$:

```
> disp(inv(A) * A);
   1   0   0
   0   1   0
   0   0   1
```

Inverse

We can test a few more properties of the inverse, such as $(AB)^{-1} = B^{-1}A^{-1}$ and $(A^T)^{-1} = (A^{-1})^T$:

```
> format rat;
> B = [1 6 4; 2 4 1; 1 2 5];
> disp(inv(A * B));
      88/3          -209/18          -119/18
     -34/3           163/36            91/36
      -1/3             1/9              1/9
> disp(inv(B) * inv(A));
      88/3          -209/18          -119/18
     -34/3           163/36            91/36
      -1/3             1/9              1/9
> disp(inv(A'));
     -40              13                5
      16              -5               -2
       9              -3               -1
```

Determinant

The command `det(A)` computes the determinant of A:

```
> A = [1 2 3; 2 5 3; 1 0 8]; B = [1 6 4; 2 4 -1; -1 2 5];
> disp(det(A));
-1
> disp(det(B));
0
```

Recall that $\det(B) = 0$ indicates that B is singular (not invertible).
To compute the inverse based on the determinant:

```
> A = [1 2; 2 5]; disp(inv(A));
   5  -2
  -2   1
> Ai = 1/det(A) * [A(2, 2) -A(2, 1); -A(1, 2) A(1, 1)];
> disp(Ai);
   5  -2
  -2   1
```

Eigenvalues

Use `eig(A)` to obtain the eigenvalues of A:

```
> A = [1 3; 4 2];
> disp(eig(A));
  -2
   5
```

Let's check this against the characteristic equation of A:

```
> disp(-2 * eye(2) - A);
  -3  -3
  -4  -4
> disp(det(-2 * eye(2) - A));
0
```

Recall that the determinant of the characteristic equation of A has to be zero.

Eigenvectors

`[X, L] = eig(A)` returns a matrix X that contains the eigenvectors, and a matrix L that contains the eigenvalues of A:

```
> [X, L] = eig(A);
> disp(X);
   -0.7071   -0.6000
    0.7071   -0.8000
> disp(L);
   -2      0
    0      5
```

Note that Matlab scales the eigenvectors so that the norm of each vector is one. To avoid that, use the `nobalance` option:

```
> [X, L] = eig(A, 'nobalance'); disp(X);
   -1.0000   -0.7500
    1.0000   -1.0000
```

Eigenvectors

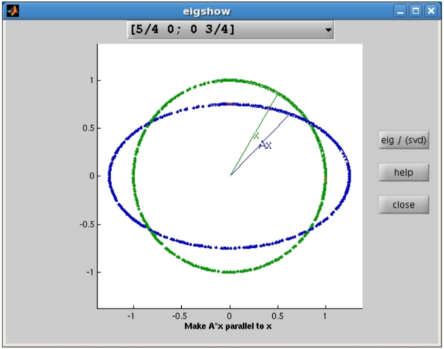
Let's check if these vectors are really eigenvectors. They have to have the property $A\mathbf{x} = \lambda\mathbf{x}$:

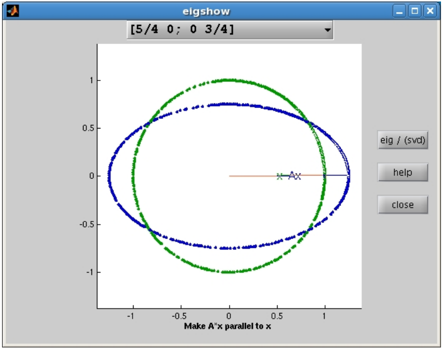
```
> disp(A * X(:,1));
    2
   -2
> disp(L(1,1) * X(:,1));
    2
   -2
```

Note that the eigenvectors for A actually involve a scaling factor: $\begin{bmatrix} -t \\ t \end{bmatrix}$ and $\begin{bmatrix} \frac{3}{4}t \\ t \end{bmatrix}$, but Matlab instantiates t.

Eigenvectors

Matlab's `eigshow` is a good way of getting an intuition for how eigenvectors work:





Mid-lecture Problem

For a matrix A, assume that X is a matrix that contains the eigenvectors of A, and Λ is a matrix containing the eigenvalues of A on the diagonal.

Use Matlab to show that:

$$A = X \Lambda X^{-1}$$

What is this decomposition useful for?

Plotting Functions

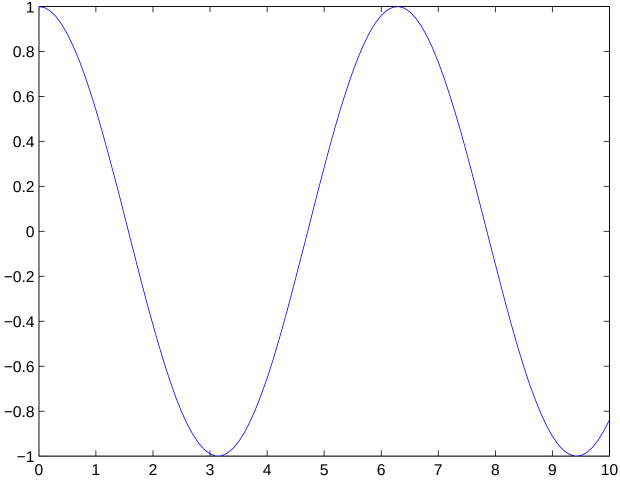
The `plot(x, y)` command in Matlab plots two vectors **x** and **y** against each other, with **x** representing the values on the x-axis and **y** representing the corresponding values on the y-axis.

The x-values can be generated with `x = [start:interval:end]`, which generates a vector with values ranging from `start` to `end`, spaced using `interval`.

We can then apply a function to **x** and call `plot`:

```
> x = [0:0.1:10];
> disp(x);
  0.00  0.10  0.20  0.30  ...  10.00
> y = cos(x);
> disp(y);
  1.00  0.99  0.98  0.95  ...  -0.83
> plot(x, y);
```

Plotting Functions



Plotting Functions

The command `plot(x, y)` plots the content of arbitrary vectors. Functions can also be plotted using `fplot(function_string, [start end])`. This automatically chooses an optimal interval.

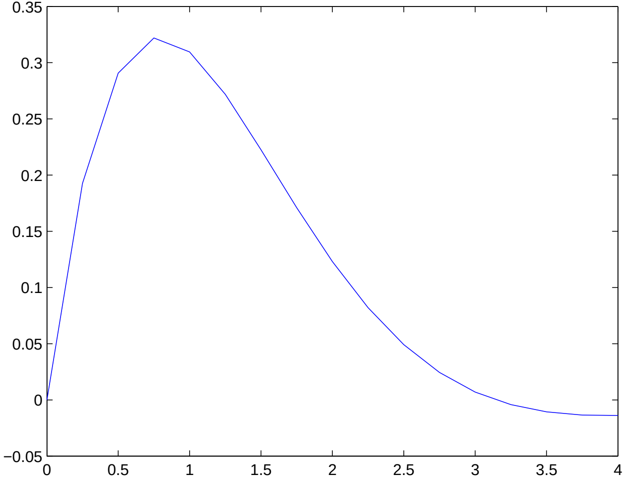
Compare:

```
> x = [0:0.25:4];
> y = exp(-x) .* sin(x);
> plot(x, y);
```

with:

```
> fplot('exp(-x) * sin(x)', [0, 4]);
```

Plotting Functions



Plotting Options

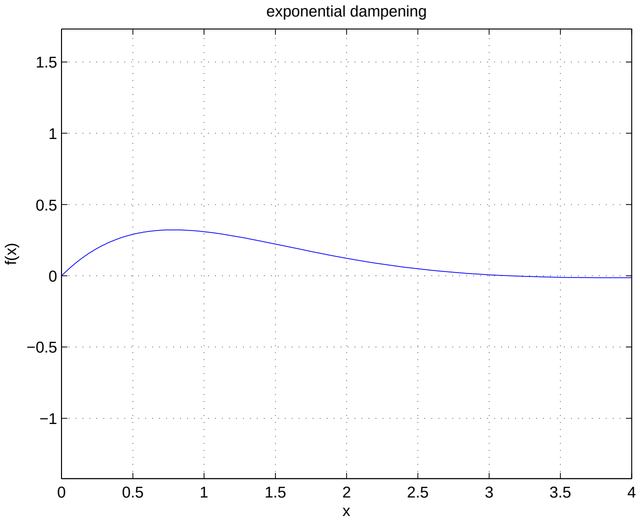
A range of options can be appended to `plot` or `fplot`:

`xlabel`	label of x-axis
`ylabel`	label of y-axis
`title`	graph title string
`legend`	graph legend string
`grid`	switch grid `on` or `off`
`axis`	axis spacing can be `square` or `equal` or `[xmin xmax ymin ymax]`

Example:

```
> fplot('exp(-x) * sin(x)', [0, 4]), xlabel('x'),
  ylabel('f(x)'), title('exponential dampening'),
  grid on, axis equal;
```

Plotting Functions



Plotting Options

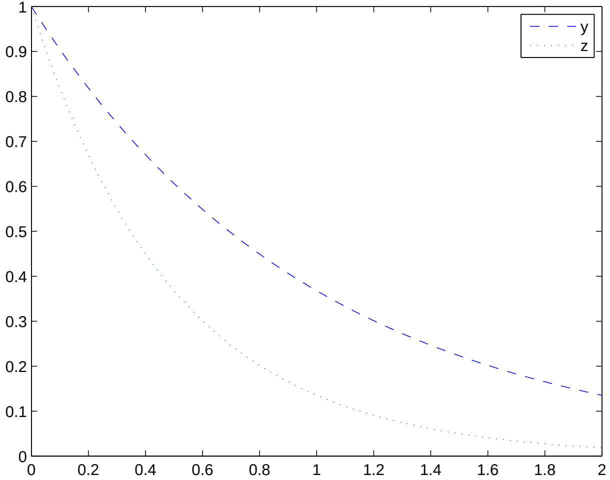
We can plot more than one function in the same graph, simply by giving `plot` multiple arguments:

```
> x = [0 : 0.01 : 5];
> y = exp(-x);
> z = exp(-2*x);
> plot(x, y, '--', x, z, ':'), legend ('y', 'z'),
  axis([0 2 0 1]);
```

Here, the third argument specifies the line type: `'-'` for straight line, `'--'` for dashed line line, `':'` for dotted line.

Note also the use of the `legend` option to introduce a legend, and the `axis` option to specify axis spacing.

Plotting Functions



Plotting Discrete Data

The `plot` command can be used to plot discrete data as well, but Matlab also offers a number of special graph types for this.

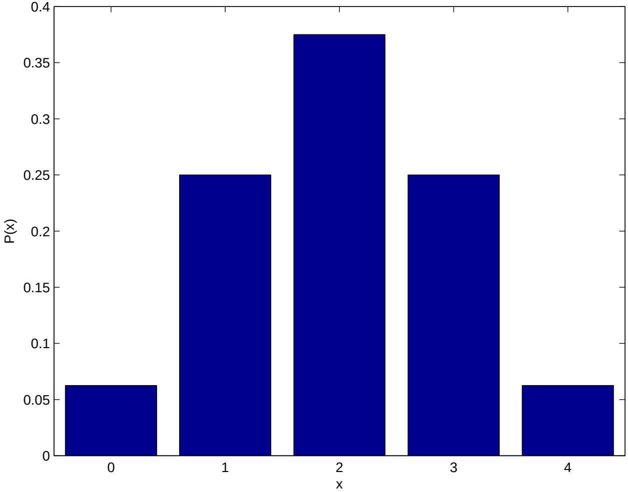
Assume we have the following probability distribution (probability $P(x)$ of obtaining x head when tossing a coin four times):

x	0	1	2	3	5
$P(x)$	$\frac{1}{16}$	$\frac{4}{16}$	$\frac{6}{16}$	$\frac{4}{16}$	$\frac{1}{16}$

Plot this distribution as a bar chart:

```
> x = [0 : 4];
> y = [1/16 4/16 6/16 4/16 1/16];
> bar(x, y), xlabel('x'), ylabel('P(x)');
```

Plotting Functions



Plotting Discrete Data

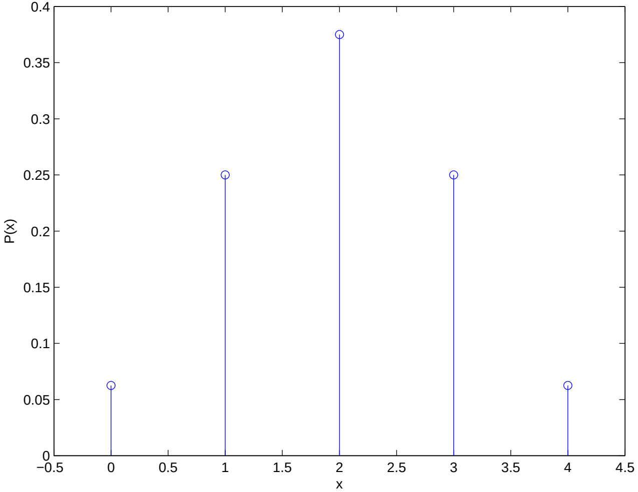
Plot the same data as a stem plot or as a scatter plot:

```
> stem(x, y), xlabel('x'), ylabel('P(x)'),
  axis([-0.5 4.5 0 0.4]);
> plot(x, y, 'o'), xlabel('x'), ylabel('P(x)'),
  axis([-0.5 4.5 0 0.4]);
```

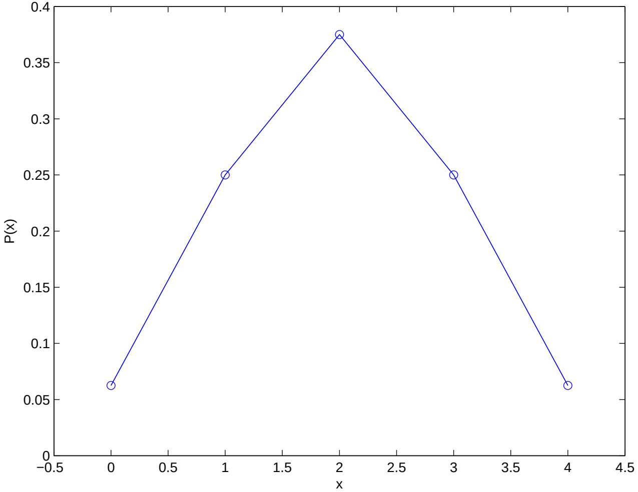
We can superimpose multiple graphs by saying `hold`. For example, we can use this connect the dots in the scatter plot:

```
> plot(x, y, 'o'), xlabel('x'), ylabel('P(x)');
> hold;
> plot(x, y);
```

Plotting Functions



Plotting Functions



Processing Images

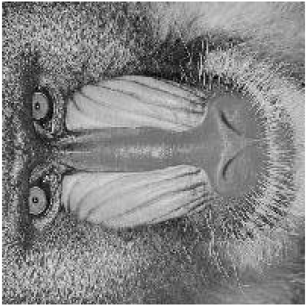
In Matlab, images can be processed as matrices. For example, a greyscale image of size 200×200 pixels is a matrix of integers ranging from 0 (black) to 255 (white).

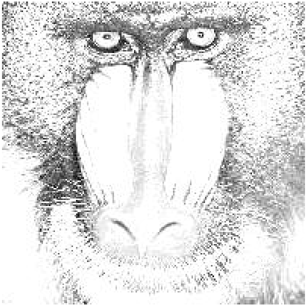
Images can be read from a file using `imread`, saved to a file using `imwrite`, and displayed using `imshow`.

```
> A = imread('baboon_grey.jpg');
> A = double(A);
> imshow(uint8(2 * A));
> imshow(uint8(A'));
> imwrite('baboon_rotated.jpg', uint8(A'));
```

Note that we need to convert the image matrix to format `double` for matrix operations (such as transpose). For input and output, the matrix need to be in format `uint8`.

Images Processing

`A' =` 

`2*A =` 

Processing Images

We can also multiply an image matrix with a matrix we have generated using Matlab:

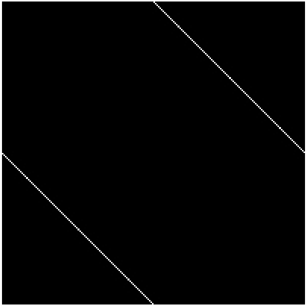
```
> B = [eye(100) eye(100); eye(100) eye(100)];
> B = B - eye(200);
> imshow(uint8(255 * B));
> imshow(uint8(A * B));
```

We can convolute an image with a kernel using the conv2 command (see next lecture for details):

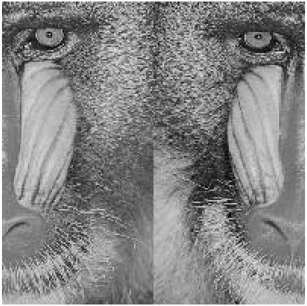
```
> K = [1/9 1/9 1/9; 1/9 1/9 1/9; 1/9 1/9 1/9];
> C = conv2(K, A);
> imshow(uint8(C));
> K = [1 0 -1; 2 0 -2; 1 0 -1];
> D = conv2(K, A);
> imshow(uint8(D));
```

Images Processing

B =



A*B =



Example: Image Processing

C = 

D = 

Summary

- Inverse: `inv(A)`;
- determinant: `det(A)`;
- eigenvalues and eigenvectors: `eig(A)`;
- plotting functions: `plot`, `fplot`;
- plotting discrete data: `bar`, `stem`;
- processing images: `imread`, `imwrite`, `imshow`;
- convolution: `conv2`.