Programmer's Toolbox

Hidden Toolkit, Where to Find It?

-a computer engineering student
during speech processing lecture

CS100 @ Özyeğin University

Gonca Gürsun & Cevat Balek

By the end of this lecture, we will learn…

- We will learn many frequently used built-in functions
- We will learn about polymorphism and how MATLAB exploits it to change a functions behavior on the basis of the number and type of its inputs
- Because random numbers play an important role in computer programming, we will learn how to use the MATLAB random number generator

Matrix Building Functions

FUNCTION	RETURNS N by M matrix
zeros(N, M)	zeros
ones(N, M)	ones
eye(N, M)	zeros except the diagonal elemets that are ones
rand(N, M)	Random numbers uniformly distributed between 0 and 1

eye(n,m)

eye(n,m) returns an n-by-m matrix that has all zeros, except for those elements on the diagonal, which are equal to one. The diagonal of a matrix is the set of elements whose indices are equal to each other

```
>> I = eye(3, 3)
I =

Diagonal Matrix

   1   0   0
   0   1   0
   0   0   1
```

eye(n,m)

eye(n,m) returns an n-by-m matrix that has all zeros, except for those elements on the diagonal, which are equal to one. The diagonal of a matrix is the set of elements whose indices are equal to each other

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =

   1   2   3
   4   5   6
   7   8   9
```

eye(n,m)

eye(n,m) returns an n-by-m matrix that has all zeros, except for those elements on the diagonal, which are equal to one. The diagonal of a matrix is the set of elements whose indices are equal to each other

```
>> A * I
ans =

   1   2   3
   4   5   6
   7   8   9
```

Polymorphism

In the study of programming languages, when the type of an argument used in a function can vary (as for example, from a scalar to a vector to a matrix) from one call of the function to the next, the function is said to be polymorphic

```
>> sqrt(9)
ans =  3
>> A = [1 4 9]
A =
   1   4   9
>> sqrt(A)
ans =
   1   2   3
```

Polymorphism - sum

The function sum, when given a row or column vector, returns a scalar–not a row or column vector–that is equal to the sum of the elements of the vector:

```
>> v = [1, -3, 5, 10]
v =

   1   -3    5   10

>> sum(v)
ans =  13
```

Polymorphism - sum

When **sum** is given a two-dimensional matrix, it calculates the sum for each column of the matrix and returns a row vector–not a two-dimensional matrix– of those elements

```
>> M = [1, 10, 100; 2, 20 200; 3 30 300]
M =

     1    10   100
     2    20   200
     3    30   300
```

Polymorphism - sum

The second argument tells **sum** which dimension it is to sum over and whether to return a row vector (second argument equal to 1) or column vector (second argument equal to 2)

The call **sum(M,2)** means to sum across the rows

```
>> sum(M)
ans =
     6    60   600
>> sum(M,2)
ans =
   111
   222
   333
```

Returning more than 1 object

In the second example, max returned two objects. The first one is the maximum value; the second one is the index of the first element that contains the maximum value

```
>> a = max([1 4 -5 0])

a =  4

>> [a b] = max([1 4 -5 0])

a =  4
b =  2
```

Trigonometric functions

FUNCTION	RETURNS
`acos(x)`	Angle in radians whose cosine equals x
`acot(x)`	Angle in radians whose cotangent equals x
`asin(x)`	Angle in radians whose sine equals x
`atan(x)`	Angle in radians whose tangent equals x
`atan2(x)`	Four-quadrant inverse tangent of x
`cos(x)`	Cosine of x (x in radians)
`cot(x)`	Cotangent of x (x in radians)
`sin(x)`	Sine of x (x in radians)
`tan(x)`	Tangent of x (x in radians)

Exponential and complex number functions

FUNCTION	RETURNS
`exp(x)`	*e* raised to the x power
`log(x)`	Natural logarithm x
`log2(x)`	Base-2 logarithm of x
`log10(x)`	Base-10 logarithm of x
`sqrt(x)`	Square root of x

FUNCTION	RETURNS
`abs(z)`	Absolute value of z
`angle(z)`	Phase angle of z
`conj(z)`	Complex conjugate of z
`imag(z)`	Imaginary part of z
`real(z)`	Real part of z

Rounding and remainder functions

FUNCTION	RETURNS
`fix(x)`	Round x towards zero
`floor(x)`	Round x towards minus infinity
`ceil(x)`	Round x towards plus infinity
`round(x)`	Round x towards nearest integer
`rem(x,n)`	Remainder of **`x/n`** (see help for case of noninteger n)
`sign(x)`	`1` if **`x`**`>0`; `0` if **`x`** equals `0`; `-1` if **`x`**`<0`

Descriptive functions applied to a vector

FUNCTION	RETURNS
`length(v)`	Number of elements of v
`max(v)`	Largest element of v
`min(v)`	Smallest element of v
`mean(v)`	Mean of v
`median(v)`	Median element of v
`sort(v)`	Sorted version of v in ascending order
`std(v)`	Standard deviation of v
`sum(v)`	Sum of the elements of v

Descriptive functions applied to a two-dimensional matrix

FUNCTION	RETURNS A ROW VECTOR CONSISTING OF
`max(M)`	Largest element of each column
`min(M)`	Smallest element of each column
`mean(M)`	Mean of each column
`median(M)`	Median of each column
`size(M)`	Number of rows, number of columns
`sort(M)`	Sorted version, in ascending order, of each column
`std(M)`	Standard deviation of each column
`sum(M)`	Sum of the elements of each column

Random Number Generation

In many scientific and engineering problems random numbers play an important role. MATLAB, like most programming languages, has built-in support for generating -pseudo- random numbers

```
>> rand
ans =  0.15627

>> rand(3)
ans =
   0.64214   0.13089   0.63022
   0.61530   0.12722   0.26077
   0.46292   0.43453   0.65314
```

Random Number Generation

The rand function returns numbers strictly larger than 0 and smaller than 1 that are uniformly distributed. What this means is that any number between 0 and 1 has the exact same probability to appear as an output of rand

```
>> rand(3,5)
ans =

   0.141370   0.027092   0.097699   0.130504   0.110343
   0.982535   0.539317   0.197634   0.021707   0.384430
   0.684847   0.371502   0.591600   0.713910   0.049125

>> hist(rand(1,1000))
```

Random Number Generation

How can one get pseudo numbers that fall into an interval other than (0, 1)?

If you need a 5-by-5 matrix of random numbers between 2 and 8 instead of 0 and 1, simply do this:

```
>> rand(5) * 6 + 2
ans =

   6.7967   3.0838   2.0432   4.0875   6.8272
   7.7237   5.6133   4.2631   4.3693   2.3549
   3.1218   2.8307   7.7447   6.1020   3.6698
   4.9294   5.7707   7.9961   6.1238   2.6142
   7.8434   6.8232   4.2218   5.0492   6.6025
```

Random Number Generation

How can one get pseudo numbers that fall into an interval other than (0, 1)?

If you need a 5-by-5 matrix of random numbers between 2 and 8 instead of 0 and 1, simply do this:

>> rand(5) * 6 + 2 **randi([2, 8], 5)?**

ans =

6.7967	**3.0838**	**2.0432**	**4.0875**	**6.8272**
7.7237	**5.6133**	**4.2631**	**4.3693**	**2.3549**
3.1218	**2.8307**	**7.7447**	**6.1020**	**3.6698**
4.9294	**5.7707**	**7.9961**	**6.1238**	**2.6142**
7.8434	**6.8232**	**4.2218**	**5.0492**	**6.6025**

Random Number Generation

The function fix rounds toward zero. Multiplying the output of rand by 10 will give us numbers greater than 0 and smaller than 10, so fix will return integers between 0 and 9 inclusive. Hence, adding 1 will generate integers in the range from 1 to 10.

```
>> fix(rand(5) * 10) + 1        randi(10, 5)
ans =

    8    7    7    6    2
    4    9   10    5    9
    1    2    1    6    1
   10    7   10    3    3
    9    4    4    2    7
```

Random Number Generator

•If we start MATLAB and call rand and then restart MATLAB and call rand again, we get the exact same “random” number. Repeatability is an important concept. The function rng is provided to initialize the pseudo random number generator of MATLAB.

```
>> rng(1)
>> rng(“shuffle”)

>> help rng
error: help:
the 'rng' function is not yet implemented in Octave
Please read `http://www.octave.org/missing.html'
to learn how you can contribute missing functionality
```