Additional MATLAB data types

This is Chapter 6 of the textbook

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Fall 2014 ; distribution outside this group of students is NOT allowed.*

1.1 Complex field. The field of complex numbers C extends the field of real numbers R by including one more element denoted i or j and sometimes called the `imaginary unit`. This is the solution of $x^2 = -1$, and also denoted as $i = j = \sqrt{-1}$, i.e. $i^2 = j^2 = -1$.

1.2 Complex number. Any complex number $z = (a, b)$ is represented either as a pair of real numbers (a, b) or in the form $z = a + ib$, i.e as a linear combination of 1 and i. In fact $z = a + ib$ is to mean $z = 1 \cdot a + i \cdot b$. With complex numbers we interchangeably use either representation. In any such representation a is called the **real part** and b the **imaginary part** of the complex number z. Since a complex number has two components that are real it can be plotted in the two-dimnensional space as a point with x coordinate a and y coordinate b. We also write $a = Re(z)$ and $b = Im(z)$. A complex number can also be represented in **polar form**

$$z = a + ib = |z|\,(\theta = re^{i\theta},$$

where $r = |z| = \sqrt{a^2 + b^2}$, $a = r\cos(\theta)$ and $b = r\sin(\theta)$. One can also conclude that $\theta = \tan^{-1}(b/a)$. The **magnitude** of complex number z is thus denoted by r (r is always real).

1.3 Addition/Subtraction. Two complex numbers are added/subtracted by adding/subcracting their corresponding real/imaginary parts. That is, for $z_1 = (a_1, b_1)$ and $z_2 = (a_2, b_2)$, then $z_3 = z_1 + z_2$ is defined as $z_3 = (a_1 + a_2, b_1 + b_2)$, the sum of z_1, z_2. Similarly, $z_4 = z_1 - z_2$, is the difference of z_1 and z_2. Another way to view this is to consider the form $z_1 + z_2 = (a_1 + ib_1) + (a_2 + ib_2)$ as a polynomial addition, where the polynomial unknown (indeterminate) is i not x. Then $z_4 = (a_1 - a_2) + i(b_1 - b_2) = (a_1 - a_2, b_1 - b_2)$.

1.4 Multiplication. Two complex numbers can also be multiplied to each other. For the multiplication operation to be understood, we use the alternate representation of the numbers as linear combinations of $1, i$. Consider again, $z_1 = (a_1, b_1)$ and $z_2 = (a_2, b_2)$ and let $z_5 = z_1 \cdot z_2$ be the product of z_1, z_2. How can we express it in terms of a_1, a_2, b_1, b_2? The answer is somewhat obvious if one considers $z_5 = (a_1 + ib_1) \cdot (a_2 + ib_2) = a_1a_2 + i^2b_1b_2 + i(a_1b_2 + a_2b_1)$. Given that $i^2 = -1$ the quadratic on i term gets eliminated and thus $z_5 = (a_1a_2 - b_1b_2) + i(a_1b_2 + a_2b_1)$. Therefore $z_5 = (a_5, b_5) = (a_1a_2 - b_1b_2, a_1b_2 + a_2b_1)$.

(c) Copyright A. Gerbessiotis. All rights reserved.

1.4 Division. Two complex numbers can also be divided by each other. Consider again, $z_1 = (a_1, b_1)$ and $z_2 = (a_2, b_2)$ and let $z_6 = z_1/z_2$ be the quotient of z_1 by z_2.

$$z_6 = \frac{z_1}{z_2} = \frac{a_1 + b_1 i}{a_2 + b_2 i} = \frac{(a_1 + b_1 i)(a_2 - i b_2)}{(a_2 + b_2 i)(a_2 - i b_2)} = \frac{(a_1 + b_1 i)(a_2 - i b_2)}{a_2^2 + b_2^2} = \frac{a_1 a_2 + b_1 b_2}{a_2^2 + b_2^2} + i \frac{a_2 b_1 - a_1 b_2}{a_2^2 + b_2^2}$$

1.5 Conjugates. $z = a + ib$ and $\bar{z} = a - ib$ are called conjugate of each other, i.e. $\bar{\bar{z}} = z$. Conjugate pairs has some interesting properties: $z + \bar{z}$ and $z\bar{z}$ are both real (for $z + \bar{z} = 2s$, and $z\bar{z} = r^2 = a^2 + b^2$). In fact $z\bar{z} = (a^2 + b^2)$ was quite helpful in division, since it formed the denominator of z_1/z_2 . (We should have said in that context that $z_2\bar{z_2}$ was helpful in division since the denominator involved a z_2 not just a z.)

1.6 Interesting and relevant MATLAB functions.

- `conj(z)`. For a complex number z, it returns the conjugate $\bar{z}$ of z.
- `real(z)`. For a complex number z, it returns $a = Re(z)$ as a double.
- `imag(z)`. For a complex number z, it returns $b = Im(z)$ as a double.
- `angle(z)`. It returns `atan2(imag(z),real(z))`, thus producing a θ such that $-\pi \leq \theta \leq \pi$.
- `isreal(z)`. For a complex number z, it returns `true` if z is real and `false` otherwise.
- `abs(z)`. For a complex number z, it returns its magnitude $\sqrt{(a^2 + b^2)}$ of z.
- $==, \sim=$ operators. Testing for equality or non-equality works fine for complex numbers. All other relational operators work but do not produce the expected result(s); only the real-part of the complex number is used to establish the relationship between the two sides.

(c) Copyright A. Gerbessiotis. All rights reserved.

1.7 Mathematical functions. Several elementary functions work with complex numbers. These include `sin, cos, log, sqrt,` etc.

```
>> a=-1;
>> sqrt(a)
   ans = 0 + 1.0000i
>> angle(a)
   ans = 3.1416
>> conj(a)
   ans = -1
>> imag(sqrt(a))
   ans=1
>> real(sqrt(a))
   ans=0
>> abs(sqrt(a))
   ans=1
```

(c) Copyright A. Gerbessiotis. All rights reserved.

2. String conversion

2.1 String conversion. For a string `a` one can convert the characters of the string into an array of numbers by performing a corresponding conversion: thus `b=double(a)` will convert the ASCII/Unicode values of the characters into `double` values. This can also be effected by a statement `a+0.0`. On the other hand if `d` is an array of ASCII-range values, `char(d)` will make a string out of these values.

```
>> a='Computer'
a = 'Computer';
>> b=double(a)
b =   67   111   109   112   117   116   101   114
>> a+0.0
ans = 67   111   109   112   117   116   101   114
>> d= [ 67 111 109 112 ];
>> char(d)
ans= Comp
>>
```

2.2 Two-dimensional character arrays. If one decides to create a two-dimensional character array, then each row must have the same length, i.e. the same number of characters. Unless of course, one uses `char` to create the array; in such a case `char` will pad the strings to the length of the longest string.

```
>> monthlist= ['January  '; 'February ';'March    '; 'April    '; 'May      '; 'June     ';
               'July     ';'August   '; 'September'; 'October  '; 'November '; 'December ' ];
>> mont2list= char('January', 'February','March', 'April', 'May', 'June', 'July',
                   'August', 'September', 'October', 'November', 'December' );
>> monthlist(2,:);
ans= February
>> mont2list(3,:);
ans= March
```

(c) Copyright A. Gerbessiotis. All rights reserved.

2.3 Concatenating strings. One can use function `strcat` to concatenate two strings. If one uses `strvcat` the strings are concatenated vertically to create a two dimensional array properly padded (see 2.2).

2.4 Comparing strings. The functions `strcmp, strcmpi, strncmp, strncmpi` can be used to determine whether two strings, (a) are the same, (b) are the same ignoring case, (c) whether the first n characters are the same, (d) whether the first n characters are the same ignoring case.

2.5 Comparing strings. A relational operator applied to two strings generates an array of `true, false` values (i.e. `1,0` values) indicating whether the corresponding characters are the same or not.

```
>> a='Computer'; b='Science';
>> x=strcat(a,b)
x= ComputerScience
>> y=strvcat(a,b)
y=
Computer
Science
>> y(2,:)
ans =Science
>> c='computer';
>> strcmp(a,c)
ans = 0
>> strcmpi(a,c)
ans = 1
>> strncmp(a,'C',1);
ans = 1
>> f= a==c
f= 0 1 1 1 1 1 1 1
```

(c) Copyright A. Gerbessiotis. All rights reserved.

3. Character Predicate functions

3.1 Predicate character functions.. A predicate character function can be used to determine whether a character is a letter (`isletter`), or a whitespace that is, space, tab, new line (`isspace`), or into a more general category (`isstrprop('str','category')`). For the `isstrprop` function, a `category` can be one of

`'alpha'`] for alphabetic characters.

`'alphanum'`] for alphabetic or numeric (alphanumeric) characters.

`'cntrl'`] for control characters.

`'digit'`] for digits (not numeric).

`'lower'`] for lower-case characters.

`'upper'`] for upper-case characters.

`'wspace'`] for whitespace.

`'xdigit'`] for hexadecimal digit.

```
>> a='CS 101-104';
>> isspace(a)
ans = 0 0 1 0 0 0 0 0 0 0
>> isletter(a)
ans = 1 1 0 0 0 0 0 0 0 0
>>isstrprop(a,'upper')
ans = 1 1 0 0 0 0 0 0 0 0
>>isstrprop(a,'digit')
ans = 0 0 0 1 1 1 0 1 1 1
>>isstrprop(a,'xdigit')
ans = 1 0 0 1 1 1 0 1 1 1
```

(c) Copyright A. Gerbessiotis. All rights reserved.

4. String substitution functions

4.1 Searching characters within a string. One can use the function `findstr(text,pattern)` to return the starting positions of `pattern` within a potentially longer string `text`. If the `text` is a two dimensional array of strings, one should use `strmatch(pattern,text)` instead; in such a case the rows that start (i.e. have `pattern` as a prefix) will be returned.

4.2 Searching and replacing characters within a string. Function `strrep(text,pattern, new)` will search for `pattern` in `text` and will replace it by `new`.

4.3 Tokenization of a string. The `strtok (text,delimiter)` function returns the characters before the first occurrence of a delimiting character in the input string plus the remaining characters of the string. The default delimiting character is whitespace.

4.4 Other functions. One can use the functions `upper, lower` to convert case (note we write `upper` not `isupper`). Function `length` returns the length of a string, and `deblank` removes TRAILING whitespace, whereas `strtrim` removes both LEADING and TRAILING whitespace.

```
>> monthlist= ['January  '; 'February ';'March    '; 'April    '; 'May      '; 'June     ';
               'July     ';'August   '; 'September'; 'October  '; 'November '; 'December ' ];
>> strmatch('Ju',monthlist);
ans =
   6
   7
>> for jj=1:12
     b = [b monthlist(jj,:) ];
   end
>> findstr(b,'Ju');
ans =  46 55
>> a='123456789123456789';
>> strrep(a,'234','0000');
ans= 10000567891000056789
>> [token remainder] = strtok(a,'5');
>> token =1234
>> remainder =56789123456789
```

(c) Copyright A. Gerbessiotis. All rights reserved.

5. String conversion functions

5.1 Numeric to string conversion. Function `int2str` can convert an integer into a string. For floating-point numbers `num2str` must be used. Function `num2str` can accept a second argument which can be either a numeric value that indicates the number of digits (including decimal) that will be converted or a generic argument similar to that used in an `fprintf` function. Thus a `'%10.5f'` can be a second argument to `num2str`.

5.2 Matrix to string conversion. Function `mat2str` can convert a matrix into a string.

5.3 sprintf. Function `sprintf` is similar to `fprintf` except that the output goes to a string instead of the MATLAB command window.

5.4 String-to-Numeric conversion. Function `eval` accepts a string as an argument and evaluates it, as long as it is a MATLAB expression.

5.5 str2double. Function `str2double` accepts a string as an argument and converts it into a `double` number.

5.6 sscanf . Function `sscanf(string,format)` can convert a string into numeric form according to a format conversion character which can be `'%g'` for floating-point numbers, or `'%d'` for integers. Thus `sscanf(num2str(pi,7),'%g');` returns `3.1416` and `sscanf(num2str(pi,7),'%d');` returns `3`.

(c) Copyright A. Gerbessiotis. All rights reserved.

6. Numeric data types in MATLAB

6.1 Default numeric data type: `double` and `single`. In MATLAB, a variable is initialized to a `double` data type that is 8 byte long. Thus if if write $2*3$ in MATLAB we obtain the result in `ans`. If we check using `whos` the type of `ans` we realize it is defined as a `1x1 double array`. We can overwrite this default behavior for example by typing in MATLAB `single(2*3)`. In this latter case the result will be stored in a floating number that is a `float` i.e. requires only 4 bytes. So if we type `whos` we realize that `ans` is a `single array`

6.2 Integer data types. MATLAB supports 8-, 16-, 32-, and 64-bit `signed` and `unsigned` integers. These data types are `int8, int16, int32, int64, uint8, uint16, uint32, uint64`. If one uses integer data types, one needs to be careful not to cross boundaries related to the representation of integers. For example if one defines `a=1024` and then writes `b=uint8(a)`, to convert a into an 8-bit unsigned integer, the 1024 value can not be represented with 8 bits of an unsigned integer that can only represent integers from 0 through 255. In that case the value stored in `b` will be `255` since a's value is larger than the larger possible value represented by `uint8`.

6.3 Mix and matching data types. Be careful when you perform operations in which the operands are of different types (integer or floating-point). Certain operations might not be supported. For example `single(10)+int32(100)` is not possible.

(c) Copyright A. Gerbessiotis. All rights reserved.