Strings and Arrays in Matlab

1 Strings

Although the primary strengths of Matlab are vector and matrix computations, Matlab supports character strings, and provides helpful facilities to work with them. A string in Matlab is treated as a vector of characters, and it can be indexed and operated upon as such. The following two assignments are identical:

```
>> a = ['a', 'b', 'c']

a =

abc

>> b = 'abc'

b =

abc

>> a(2:3)

ans =

bc
```

Furthermore, characters in Matlab are actually represented by their ASCII values. An ASCII value is a unique integer value associated associated with each character in the ASCII character set. The ASCII character set consists of 128 commonly used characters, including all letters, digits, and punctuation marks. Since Matlab characters are really numbers, we can perform various arithmetic operations on them.

```
>> 'x'+0     % Retrieve the ASCII value of 'x'

ans =
      120

>> 'X'+0     % Retrieve the ASCII value of 'X' (note the difference!)

ans =
      88
```

```
>> a + b      % Add numeric vector representations of strings a and b

ans =
      194    196    198
```

How do we compare two strings? One way to do it is with the by using the == operator. This will certainly work for one-character strings, since these are simply numbers. However, if a string has more than one characters, == will not produce the desired results:

```
>> s = 'abc-def';
>> s(4) == '-'

ans =

     1

>> s(1:3) == 'abd'

ans =

     1     1     0
```

As we can see, the == operator performs character-by-character comparisons at each position i in the string, and returns a vector v s.t. $v(i) = 1$ if the characters at position i match, and $v(i) = 0$ otherwise. This could be useful, but more often we will just want something that will compare two strings of arbitrary length, and return 1 if they are equal, and 0 if they are not. Fortunately, Matlab provides exactly such functionality, in the form of the strcmp() function.

```
>> strcmp(s(1:3),'abd')

ans =

     0

>> strcmp(s(5:7),'def')

ans =

     1

>> strcmp(s(5:7),'dEf')
```

```
ans =

     0
```

Note that strings 'dEf' and 'def' are not the same: strcmp() distinguishes upper-case characters from the lower-case. In case we want to do case-insensitive comparisons, we should convert both strings to the same case (either upper or lower), and then call strcmp(). The upper() and lower() Matlab functions do just that.

```
>> upper('abc')

ans =

ABC

>> strcmp(s(5:7), lower('DEf'))

ans =

     1
```

Matlab provides a number of useful functions that operate on strings. Arguably, one of the most useful of these is the function findstr(), which locates a sub-string within a string. findstr(s1, s2) returns the indices of all occurrences of s2 within s1. For example:

```
>> s = 'Once upon a time';
>> findstr(s, 'c')

ans =

     3

>> findstr(s, 'n')

ans =

     2     9

>> findstr(s, 'up')

ans =

     6
```

Now that we have become familiar with the basic string concepts and functions, let us view an illustrative example. The following function computes the sequence alignment score given two aligned sequences, a scoring matrix, and a gap score:

```
function score = getAlignmentScore(s1, s2, SM, GS)

%GETALIGNMENTSCORE  Compute score of a sequence alignment
%   score = getAlignmentScore(s1, s2, SM, GS), where
%     s1 and s2 are aligned sequences (must have the same length)
%     SM is the scoring matrix
%     GS is the gap score

if (length(s1) ~= length(s2))
   error('Aligned sequences must have the same length');
end

score = 0;
for i=1:length(s1)
   if (or(s1(i) == '-', s2(i) == '-'))
      score = score + GS;
   else
      score = score + lookupScore(s1(i), s2(i), SM);
   end
end

function score = lookupScore(a1, a2, SM)
%% Returns the score matrix entry for an amino acid pair
%% Assumes that the scoring matrix SM is indexed in the standard

aa_abbrev = 'ARNDCQEGHILKMFPSTWYV'; i =
findstr(aa_abbrev,upper(a1)); j = findstr(aa_abbrev, upper(a2));
score = SM(i,j);
```

2 Cell Arrays

One of Matlab's more powerful features is the ability to have arrays that can contain any type of data. Whereas Matlab's vectors and matrices are restricted to only holding floating-point values, *cell arrays* and *cell matrices* can store any data type, from strings, vectors, and matrices, to other cell arrays and complicated data structures. A good way to conceptualize a cell array is literally as a row (or a column, or – in the case of cell matrices – a grid) of cells, where each separate cell can store whatever we wish.

Despite the fact that cell arrays and matrices are considerably more powerful than regular arrays and matrices in storage capacity, there is not much difference in the way that Matlab treats them. Cell arrays, just like regular arrays, are dynamically allocated, automatically grow and shrink as needed, and they support the same set of subscript operations. The key difference is that we use curly braces – {} – to subscript cell arrays, whereas we use parentheses – () – to subscript regular arrays and matrices.

Another difference is that if a variable of a certain name used to refer to something other than a cell array, then using cell array subscripting with this variable will generate an error. For that reason, it is usually a good idea to use the cell() function to declare, and if possible, pre-allocate the cell array. cell(N) creates an $N \times N$ cell array of empty matrices; cell(N,M) creates an $N \times M$ cell array. If you don't know the array's dimensions in advance, use A=cell(0,0) or A={}; this will let Matlab know that A is a cell array, and Matlab will then grow and shrink the cell array as necessary. For example,

```
>> A = cell(0,0)        % A is an empty cell array; al


A =
     {}

>> A{1} = ([1,2;3,4])  % A has 1 entry, which is a 2x2 matrix of doubles


A =
    [2x2 double]

>> A{2} = 'abcdefg'     % A is now a 1x2 cell array, containing
                        % a matrix and a string
A =
    [2x2 double]    'abcdefg'

>> A{2,2} = A          % A is a 2x2 cell matrix, containing, among other
                       % things, an previous copy of itself!
A =
    [2x2 double]    'abcdefg'
             [ ]    {1x2 cell}
```

3 Deleting Array Rows and Columns

Matlab allows you to yank elements out of an array, or entire rows and columns out of a matrix. It is not immediately obvious how to do this, but it is very simple nonetheless.

To delete the i-th element from an array a, simply set a(i) to the empty list; that is, a(i) = [ ]. Similarly, to delete a row or column of a matrix, set that entire or column to the empty list; if M is a matrix, and you wish to delete the j-th column, set M(:,j) = [ ]. Finally, deleting elements from a cell array or matrix is absolutely identical to deleting elements from regular matrices. Note that for deletion purposes, you would use parentheses to subscript the cell array, rather than the usual curly braces.

```
>> a = 1:7          % a is vector of intergers 1 .. 7

a =
     1     2     3     4     5     6     7

>> a(3:5) = [ ]     % Delete elements 3 through 5

a =
     1     2     6     7

>> M = floor(rand(3)*10)    % M is a 3x3 matrix of random integers
                            % between 1 and 10
M =
     3     5     8
     8     4     6
     8     8     8

>> M(:,2) = [ ]           % Delete the 2nd column of M

M =
     3     8
     8     6
     8     8

>> M(2,:) = [ ]          % Delete the 2nd row of M

M =
     3     8
     8     8

>> T = {'abc','def','ghi','jkl'}    % T is a cell array of 4 strings

T =
    'abc'    'def'    'ghi'    'jkl'

>> T(2:3) = [ ]       % Delete the 2nd and 3rd elements of the cell array

T =
    'abc'    'jkl'
```