E7: Introduction to Computer Programming for Scientists and Engineers
University of California at Berkeley, Spring 2017
Instructor: Lucas A. J. Bastien

Diary for lecture 24: Least-Squares Linear Regression
Version: release

```
% This document presents and illustrates concepts related to:
%
% - Using the functions find and arrayfun
%
% Note: although this diary is part of lecture L24 (Least-Squares Linear
% Regression), the topics covered in this diary are not related to
% least-squares linear regression. The topics presented in this diary are
% general topics that can be useful for any application.

%%%%%%%%%%%%%%%%%%%%%
% The function find %
%%%%%%%%%%%%%%%%%%%%%

% The function "find" is used to find the indices of all non-zero elements
% of an array. To illustrate the use of the function "find", let us start
% with using it on a vector
>> v = [2, 4, 0, 1, 0, 0, 10]

v =
     2     4     0     1     0     0    10

% In the vector "v", the first, second, fourth, and seventh elements are
% non-zero:
>> indices = find(v)

indices =
     1     2     4     7

% The function "find" returns linear indices if only one output argument is
% requested, and returns row and column indices if two output arguments are
% requested
>> a = [0, 0, 0, 1; 9, 0, 0, 0; 8, 7, 1, 0]

a =
     0     0     0     1
     9     0     0     0
     8     7     1     0

% ---> Linear indices
>> linear_indices = find(a)
```

```
linear_indices =
     2
     3
     6
     9
    10

% ---> Row and column indices
>> [row_indices, column_indices] = find(a)

row_indices =
     2
     3
     3
     3
     1
column_indices =
     1
     1
     2
     3
     4

% The function "find" is often used in combination with logical operations
% on arrays. For example, find the indices of the elements of the array "a"
% that are greater than 2:
>> indices = find(a > 2)

indices =
     2
     3
     6

%%%%%%%%%%%%%%%%%%%%%%%%
% The function arrayfun %
%%%%%%%%%%%%%%%%%%%%%%%%

% The function "arrayfun" applies a function to each element of an array.
% The use of this function might be best explained with examples:
>> a = [1, 6, 5, 11; 5, 1, 9, 2; 4, 1, 10, 1]

a =
     1     6     5    11
     5     1     9     2
     4     1    10     1

% Define a function handle that calculates the square of a number
>> f = @(x) x.^2;

% Apply this function to each element of the array "a"
>> result = arrayfun(f, a)
```

```
result =
     1    36    25   121
    25     1    81     4
    16     1   100     1

% For the previous example, we could obtain the same array "result" without
% using the function "arrayfun":
>> result = a .^ 2

result =
     1    36    25   121
    25     1    81     4
    16     1   100     1

% There are cases, however, where using the function "arrayfun" avoids
% having to explicitely write loops. For example, consider that, for each
% element x = a(i,j) of the array "a", you want to calculate the sum:
%
% 1 + 2 + ... + x
%
% (assuming that x is a positive integer). One approach consists of writing
% a for loop that looks at each element of the array "a" and calculates the
% corresponding sum:
>> result = zeros(size(a));
>> for i = 1:numel(a)
>>     result(i) = sum(1:a(i));
>> end
>> result

result =
     1    21    15    66
    15     1    45     3
    10     1    55     1

% Alternatively, one can use the function "arrayfun":
>> f = @(x) sum(1:x);
>> result = arrayfun(f, a)

result =
     1    21    15    66
    15     1    45     3
    10     1    55     1

% The class of the array returned by the function "arrayfun" depends on the
% class of the data returned by the function handle that is to be applied
% to the input array. For example, in the previous example, f returns
% scalars of class double, so result is of class double:
>> class(result)

ans =
double
```

```
% If we define a function handle that returns scalars of class logical, and
% use it with arrayfun, then the array returned by "arrayfun" will be of
% class logical
>> is_greater_than_five = @(x) x > 5;
>> result = arrayfun(is_greater_than_five, a)

result =
  3x4 logical array
   0   1   0   1
   0   0   1   0
   0   0   1   0

>> class(result)

ans =
logical

% The function "arrayfun" also works on cell arrays. For example:
>> my_names = {'constant', 'affine', 'quadratic', 'cubic'}

my_names =
  1x4 cell array
    'constant'    'affine'    'quadratic'    'cubic'

>> is_quadratic = @(name) strcmp(name, 'quadratic');
>> result = arrayfun(is_quadratic, my_names)

result =
  1x4 logical array
   0   0   1   0
```