Programming Assignments

Introduction to Programming with MATLAB

Lesson 7

- Unless otherwise indicated, you may assume that each function will be given the correct number of inputs and that those inputs have the correct dimensions. For example, if the input is stated to be three row vectors of four elements each, your function is not required to determine whether the input consists of three two-dimensional arrays, each with one row and four columns.
- Unless otherwise indicated, your function should not print anything to the Command Window, but your function will not be counted incorrect if it does.
- Note that you are not required to use the suggested names of input variables and output variables, but you must use the specified function names.
- Also, read the instructions on the web page on how to test your functions with the auto-grader program provided, and what to submit to Coursera to get credit.
- Note that starred problems, marked by ***, are harder than usual, so do not get discouraged if you have difficulty solving them.
- You need MATLAB r2012a or newer or MATLAB Online to run the grader! Older versions are not supported.

1. Write a function called **`integerize`** that takes as its input a matrix **`A`** of integers of type **`double`**, and returns the name of the "smallest" signed integer class to which **`A`** can be converted without loss of information. If no such class exists, the text 'NONE' is returned. For example, if the smallest element of **`A`** is -100 and the largest is +100, then the function would return **`'int8'`**. As another example, if there is an element of **`A`** equal to -1e20, then the function would return 'NONE'.

2. Write a function called **`year2016`** that returns a row-vector of **`struct`**-s whose elements correspond to the days of a month in 2016 as specified by the input argument. If the input is not an integer between 1 and 12, the function returns the empty array. Each **`struct`** should contain three fields with these (exact) field names: "month", "date", and "day" (all lower case).
   - The month field must contain a string with the name of the month (first letter capitalized).
   - The date field must contain a scalar specifying the day of the month.
   - The day field must contain the three-letter abbreviation of the day chosen from this list: 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'.

   For example, here is a call of the function followed by a command that shows the seventh element of the **`struct`** array that is returned by the function:

```
>> m = year2016(2);
>> m(7)
ans =
    month: 'February'
     date: 7
      day: 'Sun'
```

3. *** A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is 9009 = 91 × 99. Write a function that is called this way:

```
>> n = palin_product(dig,lim);
```

The function returns the largest palindrome smaller than **lim** that is the product of two **dig** digit numbers. If no such number exists, the function returns 0. (Inspired by Project Euler.)

4. Each number on telephone keypads, except 0 and 1, corresponds to a set of uppercase letters as shown in this list:

2 ABC, 3 DEF, 4 GHI, 5 JKL, 6 MNO, 7 PQRS, 8 TUV, 9 WXYZ

Hence, a phone-number specification can include uppercase letters and digits. Write a function called **dial** that takes as its input argument a **char** vector of length 16 or less that includes only these characters and returns as its output argument the telephone number as a **uint64**. Here is the input and output for one example of a call of the function:

Input: **'1FUNDOG4YOU'**

Output: **13863644968**

You can assume that a phone number never starts with 0. If the input contains any illegal characters, the function returns 0. You are not allowed to use the built-in function **strrep**.

5. An n-by-n square logical matrix can be represented by a cell vector of n elements where the k^{th} element corresponds to the k^{th} row of the matrix. Each element of the cell vector is a row vector of positive integers in increasing order representing the column indexes of the logical **true** values in the given row of the matrix. All other elements in the given row of the logical matrix are **false**. Write a function called **logiunpack** that takes such a cell vector as its only input and returns the corresponding square logical matrix. For example, such a cell vector representation of a 100-by-100 logical matrix with the only true elements at indexes (10,20) and (10,75) would have only one non-empty element of the cell vector at index 10. That element is the vector [20 75].

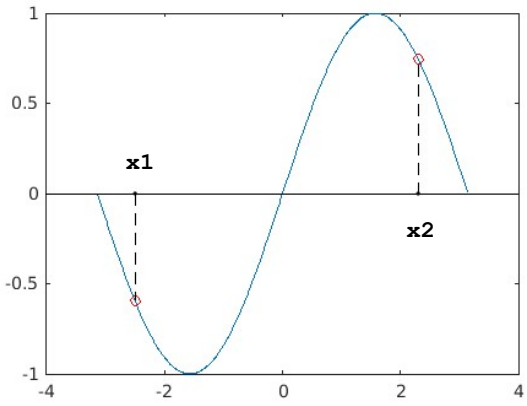
6. Solve the inverse of the previous problem. That is, write a function called **logipack** that takes a square logical matrix as its only input argument and returns its cell vector representation as specified in the previous assignment. Note that empty array elements of the cell vector corresponding to rows with all **false** values must have a size of 0x0.

7. Write a function called **centuries** that takes a positive integer smaller than or equal to 3000 representing a year as its input and returns a **char** vector with the century the given year falls into. If the input is invalid, the function returns the empty char vector **''** (there is no space between the apostrophes). Centuries are specified using roman numerals. Note that we require the shortest legal roman number. For a complete list, refer to: http://www.romannumerals.co/roman-numerals-1-to-30. Note that a century goes from year 1 to 100, so for example, the XX^{th} century ended on December 31^{st}, 2000. As an example, the call

```
>> cent = centuries(1864);
```

will make **cent** equal to **'XIX'**.

8. Write the function **find_zero** that is defined like this **function x = find_zero(f,x1,x2)**. The first input argument is special. It is a "function handle". A function handle is gotten by typing **@** and the name of any function. For example, **x = find_zero(@sin,-1,1)** will give **f** the function handle for MATLAB's built-in **sin** function. Then, inside **find_zero**, the statement **y = f(-1)** would set **y = sin(-1)**. Note that the **@** sign is not used inside the function. Only the caller uses it. All other arguments to **find_zero** are scalar numbers, and **x1** is less than **x2**. The goal of the function is to find an **x** that lies in the range from **x1** to **x2** such that after the command, **y = f(x)**, is executed inside the function **find_zero**, **y** is approximately zero as defined by **abs(y) < 1e-10**. All you know about the function **f** is that it has one scalar input and one scalar output, and a plot of its values crosses the x-axis exactly once between **x1** and **x2**, as, for example, in the figure. It is the responsibility of the caller to call the function with arguments that obey these rules. Here are two sample runs:




```
>> find_zero(@sin,-2.5,2.3)  % as shown in the figure
ans =
    -6.4000e-11
>> format long
>> find_zero(@cos,-2,1.3)
ans =
  -1.570796326871000
```

Note that you are not allowed to use the built-in function **fzero**. Hint: you may want to check the value of the function halfway between **x1** and **x2** and decide what to do next based on that.