SOLUTIONS

Problem 1 [10 points] *magenta-font items in DIS's notes and other general concepts*

Answer the following questions. Be concise and clear.

1a [1 point] Is it likely that the fate of the universe depends on your performance on this exam?

saves lives

1b [1 point] What is the social impact of solving the sheet pile equation in terms of helping people?

(hopefully) no

1c [1 point] What is code vectorization in MATLAB?

avoid using control structures by using arrays and functions of arrays

1d [2 points] Although arrays and matrices are typically interchangeable in MATLAB, they are different. Please complete this sentence:

Whereas arrays represent a general collection of values, matrices represent a

linear transformation . (Hint: 2 words.)

1e [2 points] Why is binary search more efficient than linear search?

divides data in half at each step of the search, so amount to search becomes much less
linear search looks at all the data

1f [3 points] For binary search to work, how should your data be organized? (Hint: one word gets you 3 points!)

sorted

Problem 2 [20 points] *MATLAB arrays*

Fill in the boxes for the following MATLAB sessions. Consider each problem as a fresh MATLAB session. If you want partial credit, you *may* show your work.

2a [2 points]

```
>> a = [1:3;4:6]; a .* 2
ans =
```

```
     2    4    6
     8   10   12
```

1: attempted to multiply
1: result (right shape)

2b [4 points]

```
>> a = [1:3;4:6]; a(1:end,end:-1:1)
ans =
```

1: rows kept unchanged
3: cols switched
1: result (right shape)

```
   3 2 1
   6 5 4
```

Problem 2 continues on the next page.

2c [7 points]

```
>> a = [1:3;4:6]; a([2 1],[1 2]) = a([1 2],[3 2]) + a([2 1],[1 2])
a =
```

```
7 7 3
7 7 6
```

2: a([1 2],[3 2])
2: a([2 1],[1 2])
2: prints entire matrix a
1: if has only 7 7
7 7

2d [7 points]

```
>> a = [1:3;4:6]; a(1,mod(a(2,:),2)~=0)
ans =
```

```
2
```

1: insert 2nd row
2: mod
3: logical indicies
1: value of 2

Problem 3 [35 points] *selection, repetition (for-loops), character graphics*

Problem: Complete the following program that prints a rectangular grid using the backslash, forward slash, and blank characters. The grid is composed of two kinds of *alternating rows*, each with a different pattern of slashes:

- ***uppyrow***: a row starting with `\` and ending with `/`, e.g., `\/\/\/\/`. This row has four ***uppies*** (`\/`).
- ***downyrow***: a row starting with `/` and ending with `\`, e.g., `/\/\/\/\`. This row has four ***downies*** (`/\`).

The user supplies a non-negative legal **`size`** parameter to function **`drawGrid`**, which determines the amount of rows in the grid and amount of pairs of characters in each row. As an added twist, *about* 10% of the time the program will randomly print a blank space instead of either slash (or both) in an uppy or downy.

Specifications: The two types of rows must alternate, starting with an uppyrow. You must use the code that we have given, which is a collection of functions **`drawGrid`**, **`drawRow`**, **`drawPair`**, and **`chooseChar`**. Note that MATLAB (and many other languages) require **`'\\'`** to represent a single backslash.

Example session:

```
>> drawGrid(4)
\/\ \/\/
/\/\/\/\
\ \/\/\/
 \/\/\/
```

```
% Draw entire grid:
function drawGrid(size)
   FS = '/';                    % forward slash char
   BS = '\\';                   % backslash char
1  UPPY  = strcat( BS , FS );   % \/
1  DOWNY = strcat( FS , BS );   % /\
   % draw alternating rows of uppies and downies (uppyrow, downyrow, uppyrow, ...):
2    for count = 1:size
        % alternate drawing of odd and even rows:
5         if mod(count,2)~=0
3             drawRow(size,UPPY);
          else
3             drawRow(size,DOWNY);
          end
     end

% Draw just one row:
function drawRow(size,pair)
2    for count = 1:size
2       drawPair(pair);
     end
     fprintf('\n');
```

Problem 3 continues on the next page.

```
% Draw a pair:
function drawPair(pair)

3    pair(1) = chooseChar(pair(1)); % choose between blank or first char in pair

3    pair(2) = chooseChar(pair(2)); % choose between blank or second char in pair

     fprintf(pair);


% Randomly choose between a character and a blank (blank: about 10% of the time) and
% return the chosen character:
function char = chooseChar(slash)
```

```
char = slash;                3: if
if floor(rand*10) < 1        3: rand*10
  char = ' ';                2: ' ' (blanky)
end
```

Problem 4 [35 points] *nested control structures, searching*

Background: Recall that in the binary search algorithm for discrete data you can use two "fingers" that start at the left and right side of a given search interval. The algorithm moves the "fingers" until the target is found or the fingers cross. For continuous data, there is a related approach for linear search.

Problem: On the next page, you will complete the code for function **`doubleLHSRHS`** that tries to find one root of subfunction **`f`**. Refer to the figure below, which shows an initial search interval [**`L`**, **`R`**] for **`f`**. The key idea is that both **`L`** and **`R`** "move" towards each other by **`inc`** (both directions). Each time *before* **`L`** and **`R`** are updated, **`inc`** is checked to ensure that **`L`** and **`R`** do not cross. If the current **`inc`** would cause **`L`** and **`R`** to cross, **`inc`** is reduced. **`L`** and **`R`** are updated *after* checking **`inc`**.

Detailed version of algorithm (what would have been problem "Problem 4a"):

- Check if **`decrs`** does not exceed **`MAXDECRS`** and that neither **`f(L)`** nor **`f(R)`** (using the current values of **`L`** and **`R`**) produce values within a tolerance **`eps`**. If so, do the following:
  - Check **`inc`** and reduce it, if necessary, by doing the following steps:
    * Check if the current value of **`inc`** would cause the next values of **`R`** and **`L`** to cross.
    * If so, divide **`inc`** by **`10`**, increment **`decrs`**, and repeat. (Do not modify **`L`** and **`R`** when checking **`inc`**.)
  - Using the current value of **`inc`** (which may have been reduced), update the current values of **`L`** and **`R`**.
  - Repeat the check of **`MAXDECRS`** and **`f`**.
- Report either **`root`** (answer found!) or **`-1`** (algorithm failed!).

Note that this algorithm will work well if the root is near the middle of the initial interval. Otherwise, the algorithm might perform very poorly and possibly even "hang" if **`inc`** becomes too small. For up to 7 bonus points, explain the weaknesses of the given algorithm and list some improvements on the back of the next page.

Additional Specifications:

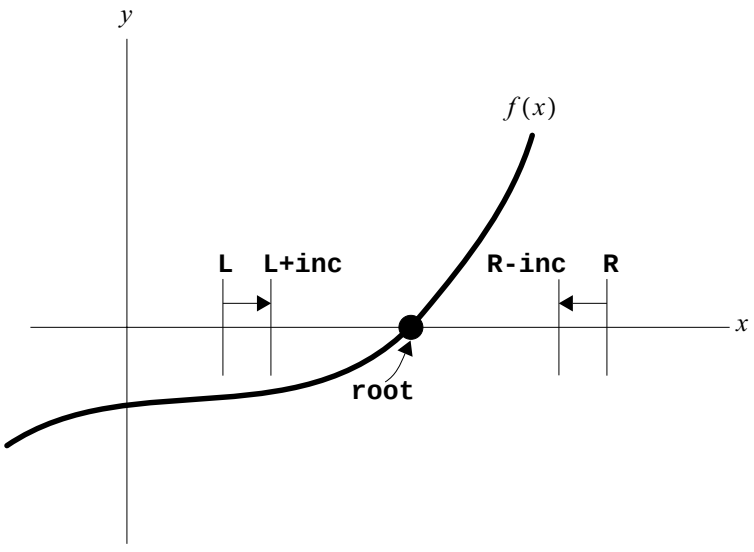
- All inputs to **`doubleLHSRHS`** are legal.
- Do not check for a maximum number of iterations, change of sign, or "hitting" the *y*-axis, as you did in the Assignment 3.
- For full credit, do not use **`break`**. You should also avoid using additional variables.

Example session (using $f(x) = x^2 - 4$):

```
>> doubleLHSRHS(0, 4, 0.0001, 1, 100 )
ans =

     2
```

Depiction of LHSRHS with search in two directions:



```
function root = doubleLHSRHS(L,R,eps,inc,MAXDECRS)
% Returns the ROOT of F by incrementing L and decrementing R
% If MAXDECRS is exceeded during search, return an out-of-bounds value

  root  = ___-1_____ ;   % default value of ROOT before search          1

  decrs = ___0_____ ;   % number of times INC has been reduced so far   1

% Search for root:

  while decr <= MAXDECRS && abs(f(L)) > eps && abs(f(R)) > eps          8

     % Check INC and reduce it if necessary:

       while L+inc > R-inc                                  4 (2 if just L < R)

            inc = inc/10;                                              3
            decr = decr + 1;                                           3

       end

     L = L + inc ; % update L                                          2

     R = R - inc ; % update R                                          2

  end

% Check if root is found; assign appropriate return value:

 if decr <= MAXDECRS
    if abs(f(L) <= eps)
      root=L;
    elseif abs(f(R) <= eps)
      root=R;
    end
 end
                                3: checks MAXDECRS or detect failure correctly
                                6: correctly returns L or R (2 if only display)
                                -4: function calls itself
                                -1: syntax (each kind of mistake)
                                -1: unnecessary code


function y = f(x)
% code not shown for this function (could be many different functions)
% Y is the value of the function F at X. For example, y = x.^2-4
```