Mestrado em Matemática e Aplicações

Especialização em Matemática Financeira

2018/2019, 1º semester

Computational Methods

Test #1 – 21 December 2018

Duration: 2 hours

Close Book (no consulting materials are allowed)

Student nº _____ Name: _____

1. (1 pt) What is the value of variable **x** at the end of the following program

```
i = 1;
s = 1;
x = 0;
while x < 20
   x = x + s * i;
   s = 1 - s;
   i = i + 1;
end
```

Answer: **x = 25**

2. (1 pt) What integer value should **k** take so that, at the end of the program below, variable **x** takes value **8**.

```
M = [1 4 k; 2 4 6]
x = -inf;
for i = 1:2
   for j = 1:2:3
      if M(i,j) - M(1,1) > x
         x = M(i,j)-M(1,1)
      end
   end
end
```

Answer: **k = 9**

3. (1 pt) Given two vectors, A and B, with the same number of elements, assign an expression to variable **c** that computes whether there are at least 3 elements of **A** that are different from the corresponding elements of **B**. For example, for **A = [4,9,7,1,4]** and **B = [4,8,7,1,3]** the expression should assign **0** to **c**, since there are only 2 elements of vectors **A** and **B** (the 2nd and the 5th) that are different.

Answer: **c = sum(A == B) >= 3**

4. (1 pt) After running the sequence of instructions below, what is the value of the variable **s**?

```
A = [1 8 2 6];
B = [6 3 9 3];
s = 0
for i = 1:length(A)
   s = s + abs(A(i)-B(i));
end
s = s / length(A);
```

Answer: **s = 5**

5. **(1.5 pt)** Given the text file "testing.txt" containing the text below

This is a file with 3 lines
This line is the second of these lines.
And this is the last line.

what is the value returned by the call `p = count_lines("i",3)`?

```
function n = count_lines(ch, k);
   fid = fopen("testing.txt", "r")
   n = 0;
   while !feof(fid)
     line = fgetl(fid);
     c = 0;
     for i = 1:length(line)
        c = c + (line(i) == ch);
     end
     n = n + (c > k)
   end
   fclose(fid);
end
```

Answer: `p = 2`

6. **(1.5 pt)** What is the approximate value that you expect from the execution of the function below when the call `equal_dice_pair(600)` is made.

```
function t = equal_dice_pair(a);
   c = 0;
   for k = 1:n
      dice_1 = ceiling(rand()*6);
      dice_2 = ceiling(rand()*6);
      c = c + (dice_1 == dice_2);
   end;
end
```

Answer: 100

7. **(2 pt)** What is the final value of matrix `M` computed by the program below?

```
m = 3;
n = 4;
M = zeros(m,n);
for i = 1:m
  for j = 1:n
    M(i,j) = (i-j)^2;
  end
end
```

Answer:

```
M =  [ 0  1  4  9;
       1  0  1  4;
       4  1  0  1]
```

8. **(2 pt)** Complete the specification of the function below so that it returns the average absolute difference between the elements of vector `V` wrt their mean value. For example, the call

```
V = [8 1 4 2 5]; x = abs_dev(V)
```

should assign `a = 2` (since the average value of the elements of `V` is `4`, the sum of the absolute differences is `4+3+0+2+1 = 10`), and hence its mean is `10/5 = 2`.

```
function a = abs_dev(V);
```

```
   s = 0;
   n = length(V);
   for i = 1:n
      s = s + V(i);
   end
   m = s / n;
   d = 0;
   for i = 1:n
      d = d + abs(V(i)-m);
   end
   a = d / n;
```

```
end
```

9. **(2 pt)** Complete the specification of the function below so that it returns a column vector where each element is the maximum value of the elements of the corresponding rows of input matrix `M`. For example, for `M = [1 4 5; 4 8 7; 7 4 2]` the call `V = largest_in_rows(M)` should return vector `V = [5; 8; 7]`.

```
function V = largest_in_row(M);
```

```
  [m,n] = size(M);
  V = zeros(n,1);
  for i = 1:m
     lgt = -inf;
     for j = 1:n;
        if M(i,j) > lgt
           lgt = M(i,j);
        end
     V(i) = lgt;
  end
```

```
end
```

10. (2 pt) As you will recognise, the function below implements the sorting of a vector **V** by means of the **insert_sort** algorithm. As you know, this algorithm has worst case complexity of $O(n^2)$, (where **n** is the length of vector **V**), since in the worst case the algorithm requires $n \cdot (n+1)/2$ insertions of values in the different positions of vector **S**.

Adapt the function below, so that it not only returns **S**, the sorted version of vector **V**, but also the actual number **c** of times there is an insertion of a value in any position of vector **S.** For example, if **V** is already sorted there are only **n** insertions.

```
function [S,c] = insert_sort(V)
% returns a vector S, which is the sorted version
% of a vector V, whose elements are all non-negative
% numbers. It also returns
% c: the number of comparisons that the function performs
%    between any two values.

   n = length(V);

   S = zeros(1,n);
   c = 0;
   for i = 1:n

      j = i;

      while j > 1 && S(j-1) > V(i)
         c = c + 1;
         S(j) = S(j-1);

         j = j - 1;

      end

      S(j) = V(i);
      c = c + 1;
   end

end
```

11. (2.5 pt) Specify the function below that, for a graph specified in file ***GraphFile***, removes a vertex ***v***, and all the arcs connecting it to the other vertices of the graph. Then it checks whether the reduced graph ***G*** is connected and, if so, writes into a file ***TreeFile*** the minimum spanning tree of the reduced graph with the same format of the input file. The returned value ***c*** indicates whether the reduced graph is connected.

```
function [G, c] = reduced_spanning_tree(graphFile, v, treeFile);
```

```
 G = read_graph(graphFile);
 n = size(G,1);
 G = [G(1:v-1,:); G(v+1:n,:)];  % eliminates row v
 G = [G(:,1:v-1), G(:,v+1:n)];  % eliminates col v
 c = connected(G)
 if c
   T = prim(G);
   write_graph(T, treeFile)
 end
```

```
NOTE: Alternative to lines 3 and 4

 for i = n:-1:v+1              % eliminates row v
    for j = 1:n
       G(i-1,j) = G(i,j)
    end
 end
 for j = n:-1:v+1              % eliminates col v
    for i = 1:n-1
       G(i,j-1) = G(i,j)
    end
 end
```

```
end
```

12. **(2.5 pt)** Assume a network of cities represented in a graph represented by its adjacency matrix `G` (where each city corresponds to a vertex of the graph and `G(i,j)` represents the length of an arc between vertices i and j; `G(i,j)=inf` if there is no direct connection). You want to locate a warehouse in one of the cities, so that it is closest in average to all the other cities.
Specify the function below that, for a graph specified by its adjacency matrix `G` (assumed to be connected) returns:

- `v`: the vertex where you should locate the warehouse; and
- `d`: the average distance from this vertex to all the other vertices;

```
function [v,d] = warehouse_location(G);
```

```
S = floyd(G);
n = size(S,1);
amin = inf;
for i = 1:n
   a = 0;
   for j = 1:n
      a = a + S(i,j);
   end
   a
 if a < amin
    amin = a;
      v = i;
   end
end
d = amin / (n - 1);
```

```
End
```

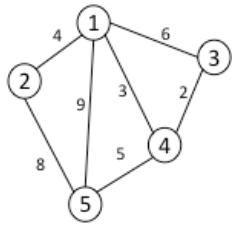
Annex

In questions 11 and 12 you should consider the functions that were studied in the classes regarding weighted undirected graphs where the weights may be interpreted as distances between vertices of the graph. If no edge exists between two vertices of a graph G = <V, E>, a virtual edge with value Inf is assumed in the adjacency matrix G of the graph.

- **`function G = read_graph(filename)`**
  - returns the adjacency matrix, **`G`**, of a weighted undirected graph specified in file with <**`filename`**>. The first line of the graph contains the number of vertices and arcs, and the subsequent lines the triple <**`n1, n2, w`**>, where **`w`** is the weight of the edge connecting the vertices **`n1`** and **`n2`** (**`n1 < n2`**). The integers **`n1`**, **`n2`** and **`w`** are separated by semicolons ("**`;`**").

graph_x.txt

```
5; 7
1; 2; 4
1; 3; 6
1; 4; 3
1; 5; 9
2; 5; 8
3; 4; 2
4; 5; 5
```




```
G =
   0    4    6    3    9
   4    0  Inf  Inf    8
   6  Inf    0    2  Inf
   3  Inf    2    0    5
   9    8  Inf    5    0
```

- **`function G = write_graph(G, filename)`**
  - writes graph G, given by its adjacency matrix, in file <filename>, with the format specified above.

- **`function b = connected(G)`**
  - Boolean b indicates whether the weighted undirected graph specified by adjacency matrix, G, is connected.

- **`function T = prim(G)`**
  - Returns a minimum spanning tree, T, of the weighted undirected graph G. Both T and G are represented by the corresponding adjacency matrices.

```
T =
   0    4  Inf    3  Inf
   4    0  Inf  Inf  Inf
 Inf  Inf    0    2  Inf
   3  Inf    2    0    5
 Inf  Inf  Inf    5    0
```

- **`function S = floyd(G)`**
  - returns the matrix S with the shortest distances between any two vertices of the graph G specified by adjacency matrix, G.

```
S =
   0    4    5    3    8
   4    0    9    7    8
   5    9    0    2    7
   3    7    2    0    5
   8    8    7    5    0
```