Mestrado em Matemática e Aplicações

Especialização em Matemática Financeira

2017/2018, 1º semester

# Computational Methods

Test #1 – 5 January 2018

Duration: 2 hours

*Close Book (no consulting materials are allowed)*

Student nº ______ Name: ______________________________

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

```
i = +1;
p = -1;
while i < 5
   p = -p * i;
   i =  i + 1;
end
```

Answer: **c = -24**

**2.** **(1 pt)** What integer value should **k** take so that, at the end of the program below, variable **s** takes value **14**.

```
M = [1 3 5; 2 4 k]
s = 0;
for i = 1:2
   for j = 1:2:3
      s = s + M(i,j);
   end
end
```

Answer: **k = 6**

**3.** **(1 pt)** Given two vectors, A and B, with the same number of elements, assign an expression to variable **c** that computes the number of elements of **A** that are greater than the corresponding elements of **B**. For example, for `A = [4,9,-3,1,8]` and `B = [5,2,9,-2,3]` the expression should assign **3** to **c**, since the second, fourth and fifth elements of **A** are greater than those of **B** (i.e. `9>2, 1 > -2` and `8 > 3`).

Answer: `c = sum(A > B)`

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

```
A = [1 3 2 4];
B = [6 8 9 3];
s = A(1)*B(1)
for i = 2:length(A)
   s = s + A(i)*B(i);
end
s = s / length(B);
```

Answer: **s = 15**

**5.** **(1.5 pt)** Given the text file “test.txt” containing the text below

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

what is the value returned by the call `p = line_ch(2,”s”)`?

```
function c = line_ch(n, ch);
   fid = fopen(“test.txt”, “r”)
   for i = 1:n
     s = fgetl(fid);
   end
   for i = 1:length(s)
      if s(i) == ch
         c = c + 1;
      end
   end
   fclose(fid);
end
```

Answer: `p = 4`

**6.** **(1.5 pt)** What is the approximate value that you expect from the execution of the function below when the call `test(1000)` is made.

```
function t = test(n);
  t = 0;
  for i = 1:n
    if rand() <= 0.8 && rand() <= 0.4
      t = t + 1;
    end;
  end
end
```

Answer: `320`

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

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

Answer: **M** =
```
[ 0  0  0  0 ;
  0  1  2  3 ;
  0  2  4  6 ;
  0  3  6  9 ]
```

8. **(2 pt)** Complete the specification of the function below so that it returns the minimum difference (in absolute value) between any two elements of vector **`V`**. For example, the call

```
x =min_diff([26 19 50 48 10])
```

should assign **`x = 2`** (since **`abs(M(4)-M(5) == 2`**).

```
function m = min_diff(V);
```

```
m = inf;
for i = 1:length(V)-1
   for j = i+1:length(V)
      d = abs(V(i) - V(j));
      if d < m
         m = d;
      end
   end
end
```

```
end
```

9. **(2 pt)** Complete the specification of the function below so that it returns the index, **`r`**, of the row of a matrix **`M`** with largest sum, as well as the sum, **`s`**, itself. For example, the call of the function **`[r,s] = greatest_row( [1 6 5;4 8 7;3 7 2])`** should return values **`r=2`** and **`s=19`**, since the sum of row **`2`** (**`4+8+7=19`**) is the largest of the rows of **`M`**.

```
function [r,s] = greatest_row(M);
```

```
s = 0;
for i = 1:size(M,1);
   x = 0;
   for j = 1:size(M,2);
      x = x + M(i,j);
   end
   if x > s
      r = i;
      s = x;
   end
end
```

```
end
```

**10. (2 pt)** As you will recognise, the function below implements the recursive version of the bipartite search for a value **x** in a sorted vector **S**, between indices **lo** and **up.** As you know, this algorithm has worst case complexity of **O(ln n)**, since in the worst case the algorithm requires $\log_2(n)$ recursive calls of function **find_between/4**, but the exact number of calls depends on the actual vector **S** and the value **x**.

Adapt the function below, so that it not only returns **p**, the position of the value **x**, but also the number **n** of recursive calls to function **find_between/4.**

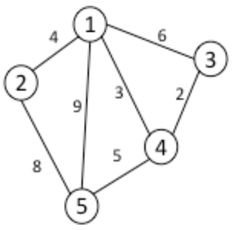
```
function [p,n] = find_between (x,S,lo,up);
% returns the index p of a vector S, sorted in
% increasing order, if x is to be found between
% indices lo and up. Otherwise, it returns p = 0
% it counts the number n of recursive calls

   n = 0;
   p = 0;

   if lo <= up

      m = round((lo+up))/2;

      if S(m) == x

         p = m;

         return;

      elseif x < S(m)

         [p,n1] = find_between(x,S,lo, m-1)
         n = n1 + 1;
      else

         [p,n2] = find_between(x,S,m+1, up)
         n = n2 + 1;
      end

   end

end
```

In the following questions, 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 =
  Inf  4    6    3    9
   4  Inf  Inf  Inf   8
   6  Inf  Inf   2   Inf
   3  Inf   2   Inf   5
   9   8   Inf   5   Inf
```

- **`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 =
  Inf  4   Inf   3   Inf
   4  Inf  Inf  Inf  Inf
  Inf Inf  Inf   2   Inf
   3  Inf   2   Inf   5
  Inf Ing  Inf   5   Inf
```

- **`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
```

**11. (2.5 pt)** Specify the function below that, for a graph specified in file ***GraphFile***, that is guaranteed to be connected, removes all the arcs belonging to a minimum spanning tree from that graph. Then it checks whether the reduced graph 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 c = reduced_spanning_tree(graphFile, treeFile);
```

```
G1 = read_graph(graphFile);
T1 = prim(G);
n = size(G,1);
G = G1;
for i = 1:n
   for j = 1:n
      if T1(i,j) < inf
         G(i,j) = inf;
      end
   end
end
c = connected(G)
if c
  T = prim(G);
  write_graph(T, treeFile)
end
```

```
end
```

**12. (2.5 pt)** Specify the function below that, for a graph specified by its adjacency matrix `G` (assumed to be connected) returns:

- the nodes `p` and `q` that are more far apart; and
- `d`, the distance between these nodes.

```
function [p,q,d] = farthest(G);
```

```
S = floyd(G)
n = size(G,1);
d = -inf;
for i = 1:n
   for j = 1:n
     if S(i,j) > d
        d = S(i,j);
        p = i;
        q = j;
     end
  end
endfor
```

```
endfunction
```