Mestrado em Matemática e Aplicações

Especialização em Matemática Financeira

2016/2017, 1º semester

Computational Methods

Test #1 – 16 December 2016

Duration: 2 hours

Close Book (no consulting materials are allowed)

Student nº _____ Name: _____

1. (1 pt) What is the value of variable `c` at the end of the following program

```
k = 0;
s = 1;
c = 0;
while c > -5
   k = k + 2;
   s = -s;
   c = c + s*k;
endwhile
```

Answer: `c = -6`

2. (1 pt) What integer value, between `0` and `5`, should `k` take so that, at the end of the program below, variable `x` takes value `27`.

```
x = 1;
for i = 3:k:12
   x = 3*x;
endfor
```

Answer: `k = 4`

3. (1 pt) Assign to variable **b** an expression evaluates to the number of occurrences of number `x` in vector `V`. For example, for `x = 4` and `V = [4,2,0,4,3]` the expression should assign `2` to **b**, since `4` occurs twice in `V`, but for `V = [3,5,0,7,3]` the value of the expression shoul be 0, since `4` does not occur in `V`.

Resposta: `b = sum(x == V)`

4. **(1 pt)** What value of variables **a** and **b** ensure that **X == A \ B** (i.e. **A*X == B**)?

```
A = [a -4 ; 1 3];
X = [ 5 ; 1];
B = [ 6 ; b];
```

Answer: **a = 2 ; b = 8**

5. **(1 pt)** If the call **pnf("file",5)** is made to the function below, what string will be written? Where?

```
function pnf(pref, n);
   fname = strcat(pref,num2str(n),".txt");
   fid = fopen(fname, "w")
   fprintf(fid, "Value = %i\n.", n+1);
   fclose(fid);
endfunction
```

String: **value = 6.**
Where: **file with name "file5.txt"**

6. **(1 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
      t = t + 1;
    endif;
  endfor
endfunction
```

Answer: **200**

7. **(1 pt)** What is the final value of matrix M computed by the program below?

```
m = 3; n = 4;
M = ones(m,n);
for i = 2:m
  for j = 2:n
    M(i,j) = M(i-1,j) + M(i,j-1);
  endfor
endfor
```

Answer: **M =**

```
[ 1  1  1  1 ;
  1  2  3  4 ;
  1  3  6 10 ]
```

8. (2 pt) Complete the specification of the function below so that it returns `m`, the maximum element of matrix `M` that is less than `v`. Assume that `M` is a matrix of positive integer values, and if none is less than `v` the value returned should be `0`.

```
function m = max_less(M,v);
```

```
m = 0;
for i = 1:size(M,1)
  for j = 1:size(M,2)
    if M(i,j) < v && M(i,j) > m
      m = M(i,j);
    endif
  endfor
endfor
```

```
endfunction
```

9. (2 pt) Complete the specification of the function below so that it returns the pair `[a,b]` where `a` and `b` (where `a` $\geq$ `b`) are the two largest elements of vector `V`. For example, if `V = [5,82,48,75,65,58,78,16]` then the returned values should be `a = 82` and `b = 78`.

```
function [a,b] = two_largest(V);
```

```
a = -Inf;
b = -Inf;
length(V)
for i = 1:length(V)
   if V(i) > a
      b = a; a = V(i);
   elseif V(i) > b
      b = V(i);
   endif
endfor
```

```
endfunction
```

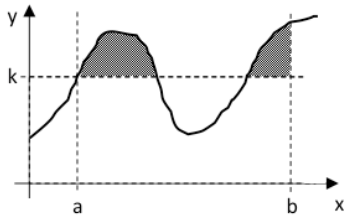
10. (2 pt) As you will recognise, the function below implements the bubble method for sorting a vector **v** in increasing order, returning its sorted version **s**. As you know, in the worst case the algorithm performs $O(n^2)$ "swaps" of the bubbles (i.e. pairs of contiguous elements of the vector) of elements of the vector, but the exact number of swaps depend on the actual vector **v**. Adapt the function below, so that it not only returns the sorted vector **s**, but also the number of swaps, **sw**, made during the sorted process.

```
function [S,sw] = bubble_sort(V);
```

```
    S = V;
    n = length(V);
    sw = 0;
    for k = n:-1:2

        for i = 1:k-1

            if S(i) > S(i+1)
                sw = sw + 1;
                x = S(i);
                S(i) = S(i+1);
                S(i+1) = x;

            endif;

        endfor;

     endfor;
```

```
endfunction
```

11. (2 pt) Assume that function **f(x)** is specified in a file "**f.m**" with signature function **y = f(x)**. Complete the specification of the function area that returns the area of the function above some threshold **k**, between **x = a** and **x = b**, as shown in the figure.




```
function s = area(a,b,k);
```

```
s = 0;
delta = (b-a)/1000;
for x = a:delta:b
   if y > k
      s = s + (y-k)*delta;
   endif
endfor
```

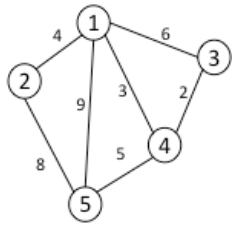
```
endfunction;
```

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 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 a matrix S with the shortest distances between any two vertices of the graph graph 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
```

12. (2.5 pt) Specify the function below that, for a graph specified in file ***GraphFile***, returns Boolean **c**, indicating whether the graph is connected, and if so, writes in file ***TreeFile*** the minimum spanning tree of the graph, in the same format of the input file.

```
function c = spanning_tree(graphFile, treeFile);
```

```
G = read_graph(graphFile);
c = connected(G);
if c == 1
  T = prim(G)
  fid = fopen(treeFile,"w");
  n = size(G,1);
  fprintf(fid, "%i; %i\n", n, n-1);
  for i = 1:n
     for j = i+1:n
        if G(i,j) < Inf
           fprintf(fid, "%i; %i; %i \n", i, j, G(i,j));
        endif
     endfor
  endfor;
  fclose(fid);
endif;
```

```
endfunction
```

13. (2.5 pt) Specify the function below that, for a graph specified by its adjacency matrix G (assumed to be connected) returns,

- `k`, the index of its "central" node (i.e. that with minimum average distance to all the other nodes of the graph); and
- `r`, the value of the average distance between node k and all the others.

```
function [k,r] = central_node(G);
```

```
S = floyd(G)
n = size(G,1);
r = Inf;
for i = 1:n
   av = 0;
   for j = 1:n
      av = av + S(i,j);
   endfor
   av = av / n;
   if av < r
      r = av;
      k = i;
   endif
endfor;
```

```
endfunction
```