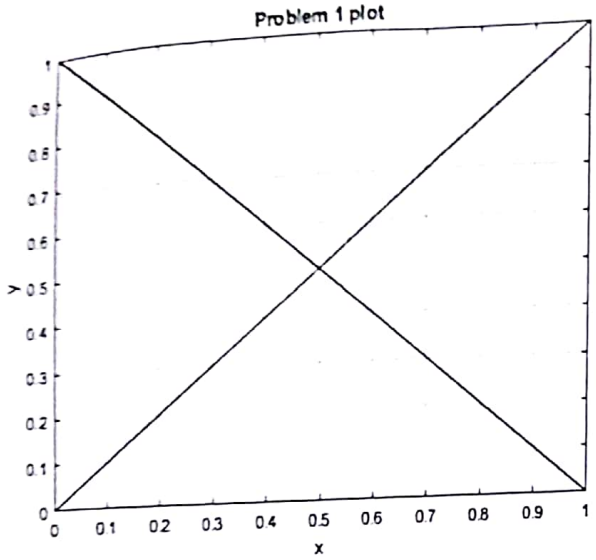
EECE 231 – Introduction to Programming using C++ and MATLAB, Sections 7,8,and 9
Quiz II

Nov 25, 2017

- The duration of this exam is 2 hours and 10 minutes. Keep in mind that you need around 10 minutes at the end of the duration of the exam to submit your answers.
- The exam consists of 3 problems for 190 points.
- You can use all the material in the exam zip file on moodle (lecture slides, associated source code, programming assignments, and solutions). Once you download the zip file, moodle will be disconnected.
- At the end of the exam, moodle will reopen for exam submission. If you would like to submit your work before the end of the exam, please talk to the proctors for instructions.
- You are asked to submit a single zip file containing your *MATLAB* files (ending with .m extension) and *C++* files (ending with the .cpp extension). Failure to do so may lead to a failing grade on the exam. It is your responsibility to make sure your files are correctly submitted.
- You are **NOT** allowed to use the **web**. You are not allowed to use **USB's** or files previously stored on your machine.
- Cell phones and any other unauthorized electronic devices are absolutely not allowed in the exam rooms. They should be turned off and put away.
- If you get caught violating the above rules or if you communicate with a person other than the exam proctors during the exam, you will immediately get zero and you will be referred to the appropriate disciplinary committee.
- The problems are of varying difficulty. Below is rough ordering estimate of the problems in order of increasing difficulty.
  - Level 1 (125 points):
    - Level 1.1 (80 points): Problem 1, Part 2.a, and Part 3.a.non-efficient
    - Level 1.2 (20 points): Part 2.b.non-efficient
    - Level 1.3 (25 points): Part 2.b.quadratic-time and Part 2.c.non-efficient
  - Level 2 (50 points): Part 3.a.efficient, Part 2.b.linear-time, and Part 3.b
  - Level 3 (15 points): Part 2.c.efficient
- Detailed comments are worth partial credit.
- Plan your time wisely. Do not spend too much time on any one problem. Read through all of them first and attack them in the order that allows you to make the most progress.
- Submit your solutions for each part in a separate file as indicated in the booklet. Include your name and ID number in each file.
- Good luck!



Problem 1 (30 points). Graphics

Write a MATLAB script which reproduces the above figure. Show the title and the labels of the axes.

Any correct solution is worth full grade.

Submit your solution in a file called `Prob1.m` including your name and ID number in the comments.

Problem 2 (100 points). Max-Sum

You are asked to solve this problem in MATLAB.

a) **Vector cumulative sums (30 points).** Write a function `cumulative(V)`, which given a $1 \times n$ vector V, returns the $1 \times n$ vector W given by: $W(i) = V(1) + \ldots + V(i)$, for $i = 1, \ldots, n$.

Examples:

```
>> cumulative([ 1 2 3])
ans =
     1     3     6
>> cumulative([ 1 1 -30 2 6 7])
ans =

     1     2   -28   -26   -20   -13
```

```
>> cumulative([ 0 0 0])
ans =
     0     0     0

>> cumulative([3 ])
ans =
     3
```

Any correct solution is worth 20/30 points. To get full grade, do it in linear time.

Submit your solution in a file called `cumulative.m` including your name and ID number in the comments.

b) **Vector Max-Sum (35 points).** Write a function `maxSumVec(V)`, which given a vector V, returns the the maximum sum of the elements of a subvector of V. That is, $maxSumVec(V)$ should return the maximum of $V(i) + \ldots + V(j)$, over the choice of all pairs of indices i and j such that $1 \leq i \leq j \leq n$, where n is the length of V.

For instance assume that $V = [-1, 2, 1, -3]$. The subvectors of V are:

$$[-1]\ [-1,2]\ [-1,2,1]\ [-1,2,1,-3]\ [2]\ [\mathbf{2,1}]\ [2,1,-3]\ [1]\ [1,-3]\ [-3].$$

Their sums are -1, $-1+2=1$, $-1+2+1=2$, $-1+2+1-3=-1$, 2, $\mathbf{2+1=3}$, $2+1-3=0$, 1, $1-3=-2$, and -3, respectively. Thus $maxSumVec(V) = \mathbf{3}$.

We also account for the empty subvector of V whose sum is zero by definition. Thus, if all the entries of V are negative, $maxSumVec(V)$ should return zero.

Examples: In the following, the subvector which achieves the max-sum is highlighted in bold.

```
>> maxSumVec([-5  1 2 -1 30 0 -5 10 11 -5 2])
ans =
    48
>> maxSumVec([-5 1 -20 -1  30.5  -5 -10  -5 2])
ans =
  30.5000
>> maxSumVec([-5  1])
ans =
     1
```

```
>> maxSumVec([ 1 -5])
ans =
     1
>> maxSumVec([-5 -5])
ans =
     0
>> maxSumVec([ 1 4 1 1])
ans =
    7
```

Any correct solution is worth 20/35 points. If you do it in quadratic time, you will get 25/35 points. To get full grade, do it in linear time.

(*Hints:* i) For the quadratic time solution, use Part (a); ii) Part (a) is not useful for the the linear time solution.)

Submit your solution in a file called `maxSumVec.m` including your name and ID number in the comments.

c) **Matrix Max-Sum (35 points).** Write a function `maxSumMat(A)`, which given a matrix A, returns the maximum sum of a submatrix of A. See the examples below. As in Part (b), if all the entries of A are negative, $maxSumMat(A)$ should return zero.

Any correct solution is worth 20/35 points points. To get full grade, aim for a number of steps in the order of m^2n if $m \leq n$, where $[m\ n] = size(A)$.

(*Hint :* For the efficient solution, use the previous parts).

Submit your solution in a file called `maxSumMat.m` including your name and ID number in the comments.

Examples: In the following examples, the submatrix which achieves the max-sum is highlighted in bold.

```
>> A=[-1 -5 0 -1 3; -20 -5 1 20 -1 ; 10 30  5 -10  -5 ; 2 1 -1 -3 2]
A =
   -1   -5   0   -1    3
  -20   -5   1   20   -1
   10   30   5  -10   -5
    2    1  -1   -3    2
>> maxSumMat(A)
ans =
    47

>> A = [1 2 3 ; -10  20 -3]
A =
     1    2    3
   -10   20   -3
>> maxSumMat(A)
ans =
   22

>> A = [-10; 3 ; 5]
A =
   -10
     3
     5
>> maxSumMat(A)
ans =
     8
```

```
>> A = [-1 -1 ; -1 -1]
A =
    -1    -1
    -1    -1
>> maxSumMat(A)
ans =
0

>> A = [3]
A =
     3
>> maxSumMat(A)
ans =
     3
```

For i:m
for j:n
if sumA(i:j):m
sum=sum(A(i:j))

Problem 3 (60 points). Recursion

a) **Recursive multiplication (30 points).** In this part, we are interested in multiplying numbers without using the multiplication operator * and without using for or while loop. Write a recursive C++ function

```
int recMult(int x,int y)
```

that takes two non-negative integers x and y and returns their product xy. Note that you don't have to handle the cases when $x < 0$ or $y < 0$; assume that they are both nonnegative.

In this part, the use of **for/while loops, the multiplication operator *, or functions defined in the `cmath` library are strictly forbidden**; use recursion.

Any correct solution is worth 20/30 points. To get full grade, do it efficiently.

(*Hint:* the efficient solution should make around 10 recursive calls for x and y in the order of 1000).

Use the following test program.

```
int main()  {
  cout<<recMult(7,0)<<endl;
  cout<<recMult(7,1)<<endl;
  cout<<recMult(7,3)<<endl;
  cout<<recMult(2017,2018)<<endl;
  system("PAUSE");
  return 0;}
```

```
Output:

0
7
21
4070306
```

Submit your solution in a file called `Prob3a.cpp` including your name and ID number in the comments.

b) **Print all permutations (30 points).** Write a MATLAB function

```
function [P] = findAllPerms(V)
```

which given a $1 \times n$ vector V, finds the $n! \times n$ matrix P consisting of all permutations of V. See the examples below.

Any correct solution is worth 20/30 points. Faster algorithms are worth more points. Note that you are not allowed to use the MATLAB function *perms*, which does exactly what you are asked to do.

(*Hint:* Use recursion).

Submit your solution in a file called `findAllPerms.m` including your name and ID number in the comments.

```
>> findAllPerms([1 2])
ans =
     1     2
     2     1

>> findAllPerms([3 1])
ans =
     3     1
     1     3

>> findAllPerms([5 1 2])
ans =
     5     1     2
     5     2     1
     1     5     2
     1     2     5
     2     5     1
     2     1     5

>> findAllPerms([2])
ans =
     2
```

```
>> findAllPerms([5 1 2 7])
ans =
     5     1     2     7
     5     1     7     2
     5     2     1     7
     5     2     7     1
     5     7     1     2
     5     7     2     1
     1     5     2     7
     1     5     7     2
     1     2     5     7
     1     2     7     5
     1     7     5     2
     1     7     2     5
     2     5     1     7
     2     5     7     1
     2     1     5     7
     2     1     7     5
     2     7     5     1
     2     7     1     5
     7     5     1     2
     7     5     2     1
     7     1     5     2
     7     1     2     5
     7     2     5     1
     7     2     1     5
```