Programming Assignment 4

Complex Numbers and Matrices

Submission Deadline

11:55PM, Thursday, 6th October 2014

Submissions only via Moodle:

Total Marks: 60

Complex numbers are of the form $a + ib$, where $a, b \in \mathbb{R}$. Thus they can be modelled as a pair of floats. The following code provides some basic functions for complex numbers that you will need for this assignment. You can cut and paste it and use it.

```
exception ZeroDiv;;

type complex = float * float;;

let eps = 1E-12;;

let re ((a,b): complex) = a;;
let im ((a,b): complex) = b;;

(* magcx finds the magnitude of a complex number *)
let magcx ((a,b):complex) = sqrt(a*.a +. b*.b);;

(* conjugcx finds the complex conjugate of a complex number *)
let conjugcx ((a,b): complex) : complex = (a, -.b);;

(* real2complex takes a float and returns a complex number with
  input float as its real part *)
let real2complex a : complex = (a, 0.);;

(* addinvcx finds the additive inverse of a complex number i.e.
  the number that when added to a complex number returns 0 *)
let addinvcx ((a,b): complex) : complex = (-.a,-.b);;

(* addcx adds two complex numbers *)
let addcx c1 c2 : complex =
    ((re c1)+.(re c2), (im c1)+.(im c2));;

(* multcx multiplies two complex numbers *)
let multcx c1 c2 : complex =
   let a1 = re c1 and b1 = im c1
   and a2 = re c2 and b2 = im c2
   in  (a1*.a2 -.  b1*.b2,  a1*.b2 +. a2*.b1);;

(* recipcx takes a complex number c and returns a complex
 number e such that multcx c e returns 1 *)
let recipcx c :  complex  =
                  let a = re c and b = im c
                  in let d = (magcx c)*.(magcx c)
                        in if d <= eps
                        then raise ZeroDiv
                        else (a/.d, -. b/.d);;
```

We will now modify the programs for vectors and matrices discussed in class to work on complex numbers, and write the remaining programs for matrix operations. Begin by using the following definitions.

```
exception Dimension;;

type  vectorcx = complex list;;

type matrixcx = complex list list;;
```

For the rest of this assignment you will need to write the following five functions. These will not carry any marks.

- Write a program `dimvcx: vectorcx -> int` that computes dimension of a given vector.
- Write a program `isMatrix: matrixcx -> bool` that checks whether a given list of complex lists is indeed a well-formed matrix.
- Write a program `dims: matrixcx -> (int * int)` that returns the dimensions of a given matrix, raising `Dimension` if the given input is not a valid matrix.
- Write programs `addvcx: vectorcx -> vectorcx -> vectorcx` and `addmcx: matrixcx -> matrixcx -> matrixcx` that add (respectively) two given vectors, and two given matrices, raising an exception if necessary.
- Write programs `scalarmultvcx: complex -> vectorcx -> vectorcx` and `scalarmultmcx: complex -> matrixcx -> matrixcx` that multiply (respectively) a given vector, or given matrix by a given complex number.

The following functions carry marks and will be tested in the demo.

1. (8 marks) Write a program `transpose: matrixcx -> matrixcx`, that given an $m \times n$ matrix M, returns the $n \times m$ matrix M^T that is its *transpose*.

2. (12 marks) Write a program `matmult: matrixcx -> matrixcx -> matrixcx`, that given an $m \times n$ matrix M and an $n \times p$ matrix N both of complex numbers, returns the $n \times p$ matrix MN obtained by multiplying them.

3. (10 marks) Write the program `delrow: int -> matrixcx -> matrixcx`, that given an $m \times n$ matrix M of complex numbers and a row number i, returns the $(m-1) \times n$ submatrix of M obtained by deleting the i^{th} row of M, raising an exception if the i^{th} row does not exist.

4. (10 marks) Write the program `delcol: int -> matrixcx -> matrixcx`, that given an $m \times n$ matrix M of complex numbers and a column number j, returns the $m \times (n-1)$ submatrix of M obtained by deleting the j^{th} column of M, raising an exception if the j^{th} column does not exist.

5. (20 marks) Write a program `det: matrixcx -> complex` that calculates the *determinant* of a $m \times m$ matrix, raising exception `Dimension` if the matrix is not square. [Hint: Recall that to find the determinant of a matrix we consider any row of elements and multiply each element of the row with the determinant (recursively computed) of its cofactor with the appropriate sign. You can use `delrow` and `delcol` to find the cofactors. Use a 1×1 matrix as the base case to avoid the confusion of having to decide what the cofactor of an empty matrix is.]

You must document your programs adequately, and provide enough test inputs on which your programs run.