7 Line

Consider two points in a grid, such as pixels in a digital image. We want to determine all the lattice points that connect the two points to approximate a straight (linear) line. We will explain how you can determine these points later.

We can specify a point by indicating a pair (or vector with length 2) [i, j], whereby i and j represent whole numbers that describe the *x* and *y* coordinates on the lattice respectively.

Create a function in the file line.m that starts with

```
function list = line (i1, j1, i2, j 2)
```

that returns a list of lattice points that approximate a straight line between [i1, j1] and [i2, j2]; the start and end points must also be included in the list. The result will thus be in the form of:

```
[ [i1, j1]; [i2, j 2]; [i3, j 3] ;   . . .   ; [in, jn] ]
```

The order in which the points are placed is irrelevant in the first instance, but each point may only occur once. You can use the `unique` matlab function to this end. This command will order the points for you sequentially; first for the *x* coordinates and then for the y coordinates. An example is given below.

```
>> A = [[1,2];  [1, 3]; [- 5, 6]; [- 5, 7]; [1, 3 ] ]
A =
   1      2
   1      3
  -5      6
  -5      7
   1      3

>> unique (A, 'rows')
ans    =
 -5      6
 -5      7
  1      2
  1      3
```

You also need to have the function produce an error message if the user fails to enter whole numbers for the two points. You can program the error message as follows:

```
error ('bad parameter in line’)
```

Make sure that you reproduce this exactly, otherwise justitia will mark it as incorrect. You will have to ensure that 'line' is a wrapper function that monitors incorrect inputs and ensures that the points in the list are sequentially ordered and unique. To actually calculate the points, you need to make a second help function called helperline in line.m. This help function does not need to return only unique points. The help function is recursive and works as follows:

- (stop condition) If the two lattice points `([i1,j1]` and `[i2,j2]`) are neighbours, then we add both to the list. Two lattice points are neighbours if
  `abs(i1 - i2) <= 1` and `abs(j1 - j2) <= 1`.
- (recursive step) If the two lattice points are not neighbours, you will need to determine a third point between them. You can do this as follows:
  `i3 = floor((i1+i2)/2)` and `j3 = floor((j1+j2)/2)`.
  You then return the list with all the elements from
  `helperline (i1,j1,i3,j3)` and `helperline (i3,j3,i2,j2)`.

If you include several functions in one file, then you must clearly indicate where one function ends and the other begins. For this reason you must complete both functions with 'end'. You can only start the second function once you have completed the first. This will look something like this:

```
function a = f 1 ( )
% code   . . .
end

function b = f 2 ( )
% code   . . .
end
```

You can use the following useful commands:

- `list = [list; [a, b]]` adds a lattice point `[a,b]` to `list`
- `list = unique (list, 'rows')` deletes duplicate rows and orders them.

If you want to see all the points in a lattice then you can plot them. You should not include this in the program, but it can help you to trace problems. You can plot all the points found in a graph using the following commands:

```
list = line ( . . .)
plot (list ( : , 1) ,   list ( : , 2) , 'ks')
```

This visualization should not be included in the program that you submit in justitia!

Submit:	line.m