CS 1112 Spring 2020 Project 4 Part A **due Monday, April 13, 11pm EDT**

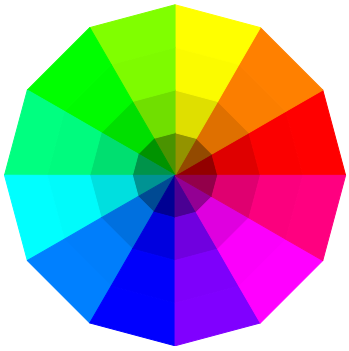
You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, "you" below refers to "your group." You may discuss background issues and general strategies with others and seek help from the course staff, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not OK for you to see or hear another student's code and it is certainly not OK to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on you own, seek help from the course staff.

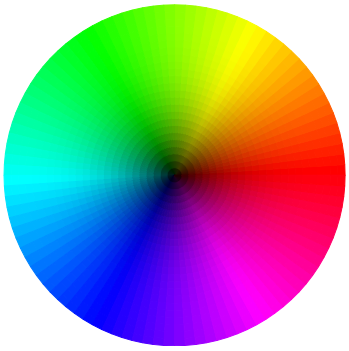
Objectives

Completing Part A of this project will help you learn about 2-dimensional arrays (matrices) and how they typically interact with 1-dimensional arrays (vectors). Other themes include RGB computation and the use of subfunctions.

1 Color circles

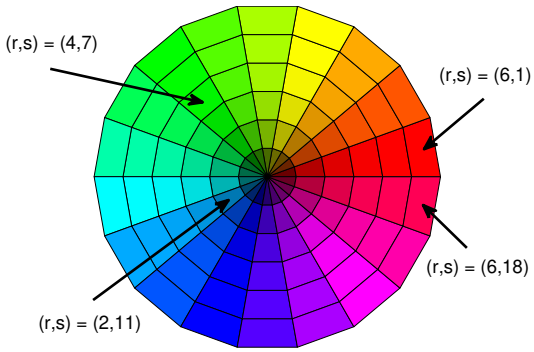
The display of color spectra is discussed in §4.2 in *Insight*. Color circles are another approach for communicating the range of RGB (red-green-blue) possibilities:





This project is about computing and displaying color circles.

We can approximate a color circle as a collection of trapezoidal tiles, each of which is colored according to a rule. Tiles can be organized into *sectors* and *rings*; here is an example in which $n_{\text{rings}} = 6$ and $n_{\text{sectors}} = 18$:



The notation $(2, 11)$ refers to the tile that is in ring 2 and sector 11. To specify the location of tile (r, s) it helps to define two key parameters: the sector angle $\Delta_{\text{sectors}} = 2\pi/n_{\text{sectors}}$ and the ring width $\Delta_{\text{rings}} = 1/n_{\text{rings}}$. Tile (r, s) then has its two "inner" vertices on a circle with radius $(r-1)\Delta_{\text{rings}}$ and its two "outer" vertices on a circle with radius $r\Delta_{\text{rings}}$ (we are centering the color circle at $(0, 0)$). Notice that the tiles in ring 1 are degenerate trapezoids (a.k.a. triangles) because the inner radius is zero. The two polar angles associated with tile (r, s) are $(s-1)\Delta_{\text{sectors}}$ and $s\Delta_{\text{sectors}}$.

A handy way to encode the RGB values of the colors that show up in the circle is to have a triplet of matrices R, G, and B, each of which is n_{rings}-by-n_{sectors} in dimension. Our convention throughout is to house the red, green, and blue values for radial tile (r, s) in `R(r,s)`, `G(r,s)`, and `B(r,s)` respectively.

1.1 Displaying a color circle

Complete the following function so that it performs as specified:

```
function drawColorCircle(R,G,B)
% Draw a color circle in the current figure window given matrices R, G,
% and B.  Assumes that the hold toggle is on.
% R, G, and B are matrices of the same size storing the red, green, and
% blue values, respectively, of the tiles in the color circle.  The
% number of rows is the number of rings in the color circle; the number
% of columns is the number of sectors in the color circle.
```

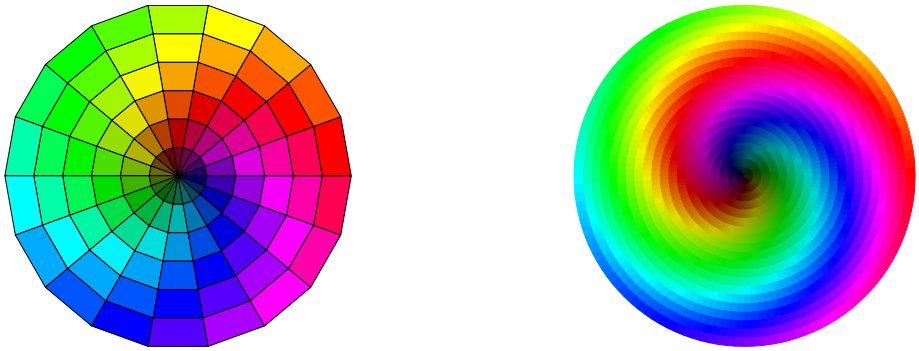
For full credit, your implementation of `drawColorCircle()` must include and make effective use of a subfunction `drawTile()` that draws a single tile. You are free to design this subfunction any way you want. Be sure to document your subfunction appropriately.

To actually draw a tile, you should use the `fill()` function. The command `fill(x,y,c)` colors the polygon defined by vectors `x` and `y` in a color specified by the RGB 3-vector `c`. The polygon will have vertices at `(x(1), y(1))`, `(x(2), y(2))`, ..., `(x(n), y(n))` where `n` is the length of the vector `x` (and `y`). Unlike `plot()`, the last point will automatically be connected to the first. To "turn off" the black border that is drawn by default, use the additional arguments `'LineStyle', 'none'`, like this:
`fill(x, y, c, 'LineStyle', 'none')`

An example script `showDrawColorCircle` that calls your function to draw a "color circle" with random colors is available on the website for your convenience.

1.2 Shifting a color circle

We can produce spirals by systematically rotating the rings of a color circle:



For the spiral on the left, we start with the "aligned" color circle displayed on page 1 in which $n_{\text{rings}} = 6$ and $n_{\text{sectors}} = 18$. We then rotate its rings as follows:

Ring	Action
6	rotated 0 sectors counter-clockwise
5	rotated 1 sectors counter-clockwise
4	rotated 2 sectors counter-clockwise
3	rotated 3 sectors counter-clockwise
2	rotated 4 sectors counter-clockwise
1	rotated 5 sectors counter-clockwise

Instead of these single-sector "offsets," we could have rotated the rings by other amounts. Sticking with the same example, if p is a nonnegative integer, then here is how we would generate a color circle with offset p:

Ring	Action
6	rotated 0*p sectors counter-clockwise
5	rotated 1*p sectors counter-clockwise
4	rotated 2*p sectors counter-clockwise
3	rotated 3*p sectors counter-clockwise
2	rotated 4*p sectors counter-clockwise
1	rotated 5*p sectors counter-clockwise

For your information, the above spiral on the right has $n_{\text{rings}} = 20$, $n_{\text{sectors}} = 120$, and offset $p = 5$.

If R, G, and B are matrices that define the red, green, and blue values of an aligned color circle, then by shifting their rows we can obtain a new triplet of matrices $\tilde{R}$, $\tilde{G}$, and $\tilde{B}$ that define a spiral. Here is an illustration of how get $\tilde{R}$ from R in the case $n_{rings} = 5$, $n_{sectors} = 8$, and $p = 3$:

$$R = \begin{array}{|cccccccc|l} \hline R(1,1) & R(1,2) & R(1,3) & R(1,4) & R(1,5) & R(1,6) & R(1,7) & R(1,8) & \leftarrow \text{ ring 1} \\ \hline R(2,1) & R(2,2) & R(2,3) & R(2,4) & R(2,5) & R(2,6) & R(2,7) & R(2,8) & \leftarrow \text{ ring 2} \\ \hline R(3,1) & R(3,2) & R(3,3) & R(3,4) & R(3,5) & R(3,6) & R(3,7) & R(3,8) & \leftarrow \text{ ring 3} \\ \hline R(4,1) & R(4,2) & R(4,3) & R(4,4) & R(4,5) & R(4,6) & R(4,7) & R(4,8) & \leftarrow \text{ ring 4} \\ \hline R(5,1) & R(5,2) & R(5,3) & R(5,4) & R(5,5) & R(5,6) & R(5,7) & R(5,8) & \leftarrow \text{ ring 5} \\ \hline \end{array}$$

$$\tilde{R} = \begin{array}{|cccccccc|l} \hline R(1,5) & R(1,6) & R(1,7) & R(1,8) & R(1,1) & R(1,2) & R(1,3) & R(1,4) & \leftarrow \text{ ring 1} \\ \hline R(2,8) & R(2,1) & R(2,2) & R(2,3) & R(2,4) & R(2,5) & R(2,6) & R(2,7) & \leftarrow \text{ ring 2} \\ \hline R(3,3) & R(3,4) & R(3,5) & R(3,6) & R(3,7) & R(3,8) & R(3,1) & R(3,2) & \leftarrow \text{ ring 3} \\ \hline R(4,6) & R(4,7) & R(4,8) & R(4,1) & R(4,2) & R(4,3) & R(4,4) & R(4,5) & \leftarrow \text{ ring 4} \\ \hline R(5,1) & R(5,2) & R(5,3) & R(5,4) & R(5,5) & R(5,6) & R(5,7) & R(5,8) & \leftarrow \text{ ring 5} \\ \hline \end{array}$$

In this example,

You get $\tilde{R}(1,:)$ by right-shifting $R(1,:)$ 12 components.
You get $\tilde{R}(2,:)$ by right-shifting $R(2,:)$ 9 components.
You get $\tilde{R}(3,:)$ by right-shifting $R(3,:)$ 6 components.
You get $\tilde{R}(4,:)$ by right-shifting $R(4,:)$ 3 components.
You get $\tilde{R}(5,:)$ by right-shifting $R(5,:)$ 0 components.

The matrices $\tilde{G}$ and $\tilde{B}$ are computed similarly. Complete the following function so that it performs as specified.

```
function [Rtilde,Gtilde,Btilde] = makeSpiral(R,G,B,p)
% R, G, and B are matrices of the same size that define a color circle.
% p is a nonnegative integer.
% Rtilde, Gtilde, and Btilde are matrices of the same size that define a
% new color circle that is the given color circle with offset p.
```

Your implementation must include and make effective use of a subfunction `v = shift(u,q)` that takes a row vector `u` and shifts its component values right by the amount specified by `q` (a nonnegative integer). Thus,

```
v = shift([10 20 30 40 50],3)
```

should give `v = [30 40 50 10 20]`. Below we introduce "modular arithmetic" and the built-in function `mod`, which will be useful for "shifting" the components in a vector.

An example script `showMakeSpiral` is available on the course website for calling your function `makeSpiral()`, again with random colors, to help you check your functionality so far.

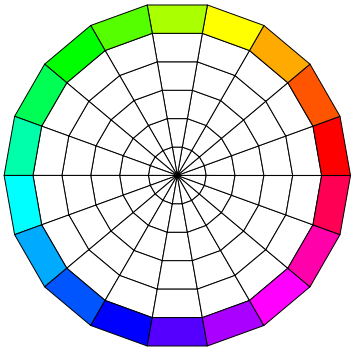
1.2.1 Modular arithmetic

Modular Arithmetic describes a system of arithmetic for integers that "wrap around," the most familiar of which is probably the 12-hour clock. In the 12-hour clock system, there's no 13 o'clock or higher because the value "wraps around" at 12 and starts over at 1. 10 o'clock + 5 hours gives 3 o'clock, not 15. Arithmetic on the 12-hour clock is therefore *modulo 12*, commonly read as *mod 12*. Notice that 12 is congruent not only to 12 but also to 0.

To compute $10 + 5$ *modulo* 12, we can use the built-in function `mod()`: `mod(10+5, 12)` returns 3. Given positive integers `a` and `b`, `mod(a,b)` always returns an integer in [0..b-1] .[1] In your computation, beware of 0 if you're dealing with vector indices! Although modular arithmetic is defined for integers, `mod()` does work with non-integer arguments.

1.3 Making a color circle

We now turn our attention to the production of the matrices R, G, and B that collectively define the RGB values assigned to each radial tile. We illustrate the main ideas with the example $n_{\text{rings}} = 6$, $n_{\text{sectors}} = 18$. The first step is to determine the RGB values of the tiles that are on the "rim" of the circle:

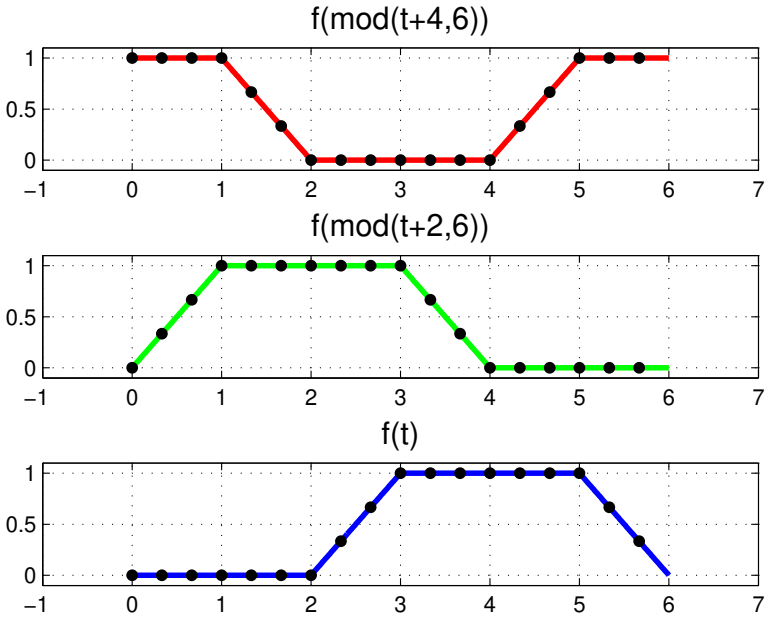


Notice how the colors transition from red ($s = 1$) to yellow ($s = 4$) to green ($s = 7$) to cyan ($s = 10$) to blue ($s = 13$) to magenta ($s = 16$) and on back to red ($s = 1$) as we travel counterclockwise around the

[1] Notice any similarity between `rem()` and `mod()`? In fact, when the arguments are positive values there's no difference between `rem()` and `mod()`. Read MATLAB documentation on those two functions if you want to learn more.

rim. The actual RGB values are determined by carefully sampling a function f that we shall refer to as the *rim function*. The rim function f is defined on the interval $[0, 6]$ as follows:

$$f(t) \;=\; \begin{cases} 0 & \text{if} \quad 0 \le t \le 2 \\ t-2 & \text{if} \quad 2 \le t \le 3 \\ 1 & \text{if} \quad 3 \le t \le 5 \\ 6-t & \text{if} \quad 5 \le t \le 6 \end{cases} .$$



Sector	red	green	blue
1	1.00	0.00	0.00
2	1.00	0.33	0.00
3	1.00	0.67	0.00
4	1.00	1.00	0.00
5	0.67	1.00	0.00
6	0.33	1.00	0.00
7	0.00	1.00	0.00
8	0.00	1.00	0.33
9	0.00	1.00	0.67
10	0.00	1.00	1.00
11	0.00	0.67	1.00
12	0.00	0.33	1.00
13	0.00	0.00	1.00
14	0.33	0.00	1.00
15	0.67	0.00	1.00
16	1.00	0.00	1.00
17	1.00	0.00	0.67
18	1.00	0.00	0.33

From the above plots and table we see that the blue values for the 18 rim tiles are obtained by evaluating f at 18 equally spaced points, including 0 but not 6. Similarly, the green values are obtained by sampling $f(\text{mod}(t+2, 6))$ while the red values are computed by sampling $f(\text{mod}(t+4, 6))$.

Once the color of a rim tile is known, then we can compute the color of the other tiles in its sector by using the following "darkening" rule:

> If `c` is the RGB vector that defines the color of the rim tile in sector s, then the color of tile (r, s) is given by `rho*c` where the value of `rho` is $\sin((r/n_{\text{rings}})(\pi/2))$

Notice that the value of $\sin((r/n_{\text{rings}})(\pi/2))$ decreases as the ring index r decreases forcing the tiles to get darker as we move towards the center of the circle.

Now that we have described how the rim tiles and interior tiles are colored, it is possible for you to complete the following function so that it performs as specified:

```
function [R,G,B] = makeColorCircle(nRings,nSectors)
% nRings is a positive integer
% nSectors is a positive integer >= 3.
% R, G, and B are matrices with nRings rows and nSectors columns
% that define a color circle that has nRings rings and nSectors
% sectors. The rgb color of radial tile (r,s) is given by
% R(r,s), G(r,s), and B(r,s).
```

Include a subfuction to return the value of the *rim function* described above given t.

An example script `showMakeColorCircle` is available on the course website for calling your function `makeColorCircle()`. Play with the parameters in `showMakeColorCircle` to create different spirals!

Submission

Submit your files `drawColorCircle.m`, `makeSpiral.m`, and `makeColorCircle.m` on CMS.

Part B will appear in a separate document. Parts A and B have the same due date.