Homework 4

Math 3607, Autumn 2021

Marco LoPiccolo

Table of Contents

Problem 1.

a.)

1. (Sliders moving along grooves; adapted from **LM** 2.1–12 and Sample HW01) The mechanical device shown in Figure 1 consists of two grooves in which sliders slide. These sliders are connected to a straight rod.

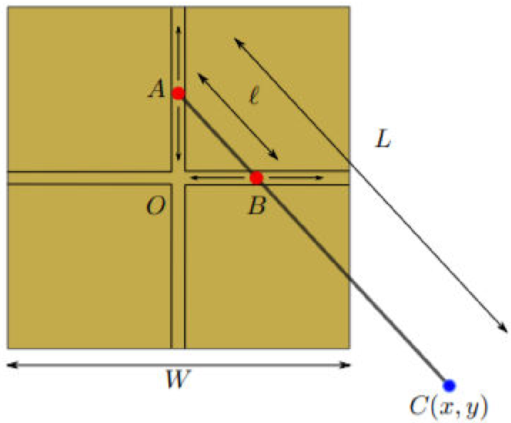
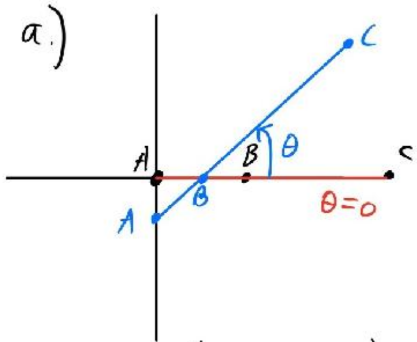


Figure 1: The bronze square is a piece of metal with two grooves cut out of it as shown. There are sliders at the points A and B which slide in these grooves. The slider at A can only slide vertically, and the one at B can only slide horizontally. There is a straight rod attached at A and B, which extends to C. As the point C moves around the block, it traces out a closed curve.

(a) ✏ Analytically, determine the curve which is traced out by C in one rotation.

Suggestion. Let (x, y) be the coordinates of the point C. Express the variables x, y in terms of L, ℓ, and θ, where $\theta \in [0, 2\pi)$ is the angle from the part of the horizontal groove which is to the right of B to the rod BC.

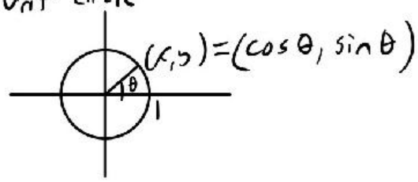
a.)

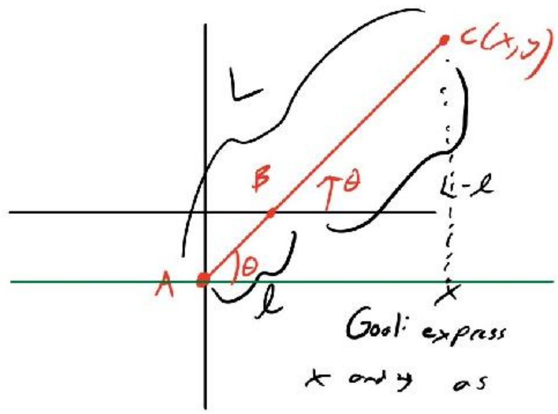


$X = x(\theta; L, \ell)$

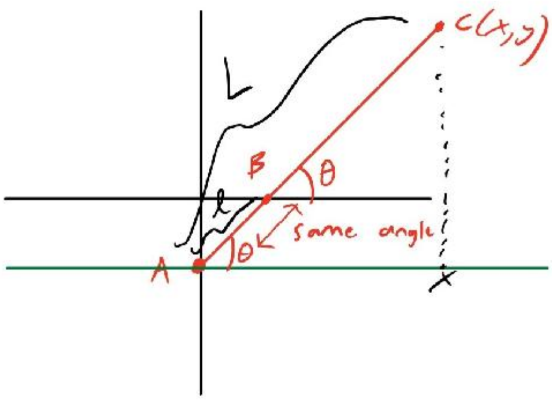
$y = y(\theta; L, \ell)$

unit circle





Goal: express x and y as functions of $\theta; L, \ell$



$$\begin{cases} x = L\cos(\theta) \\ y = L - \ell\sin(\theta) \end{cases}$$

axis equal

parametric equations

$$\cos\theta = \frac{x}{L} \qquad \sin\theta = \frac{y}{L-\ell}$$

$$\Big\downarrow \text{ Pythagoras}$$

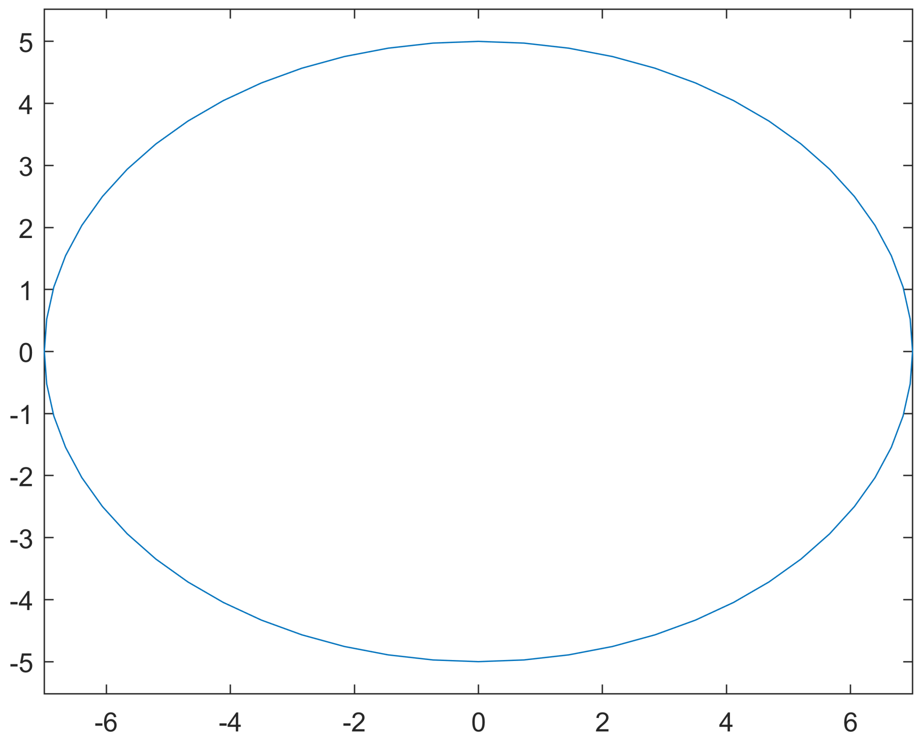
$$\cos^2\theta + \sin^2\theta = \boxed{\frac{x^2}{L^2} + \frac{y^2}{(L-\ell)^2} = 1}$$

ellipse

b.)

(b) Using the previous result, plot the trajectory of C in one rotation for $\ell = 2$ and $L = 7$.

```
theta = linspace(0, 2*pi, 61);
L = 7;
l = 2;
x = L * cos(theta);
y = (L - l) * sin(theta);
clf
plot(x,y)
axis equal
```



Problem 2.

a.)

(a) Write a function named `spiralgon` by modifying the script so that it generates spirals using m regular n-gons for any $n \geqslant 3$. Your function must be written at the end of your homework live script (.mlx) file. Begin the function with the following header and comments.

```
function V = spiralgon(n, m, d_angle, d_rot)
% SPIRALGON plots spiraling regular n-gons
% input:    n = the number of vertices
%           m = the number of regular n-gons
%           d_angle = the degree angle between successive n-gons
%           (can be positive or negative)
%           d_rot = the degree angle by which the innermost n-gon
%                   is rotated
% output:   V = the vertices of the outermost n-gon
....
```

```
clf
% Test Code, actual function found at the bottom of the live script
% m = 21; d_angle = 4.5; d_rot = 90; n = 10;
% th = linspace(0, 360, n + 1) + d_rot;
% V = [cosd(th);
% sind(th)];
```

```
% C = colormap(hsv(m));
% sPre1 = (n - 2) * 180;
% sPre2 = (sPre1 / n) / 2;
% s = sind((sPre1 - sPre2) - abs(d_angle))/sind(sPre2);
% R = [cosd(d_angle) -sind(d_angle);
% sind(d_angle) cosd(d_angle)];
% hold off
% for i = 1:m
% if i > 1
% V = s*R*V;
% end
% plot(V(1,:), V(2,:), 'Color', C(i,:))
% hold on
% end
% set(gcf, 'Color', 'w')
% axis equal, axis off
```

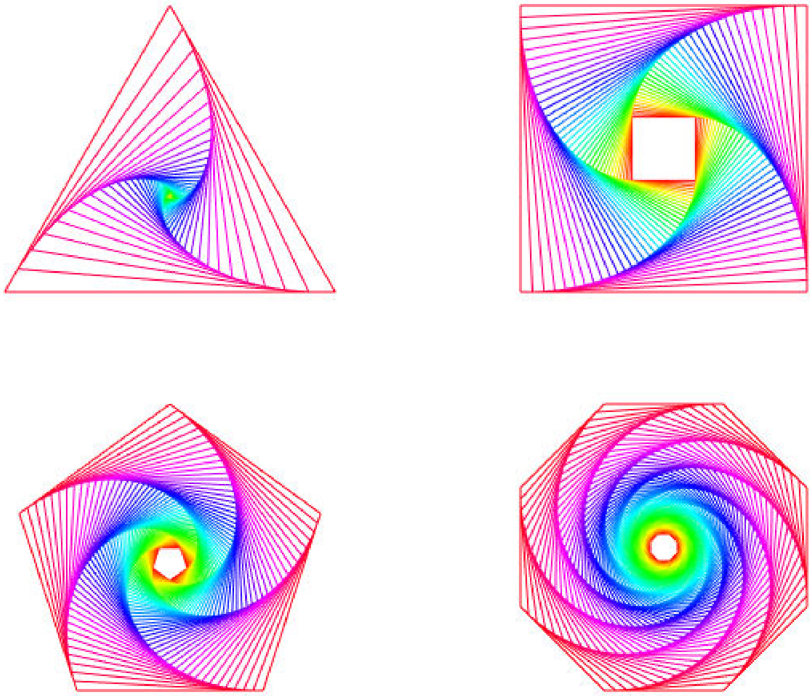
b.)

(b) Run the statements below to generate some aesthetic shapes.

```
clf
subplot(2, 2, 1), spiralgon(3, 41, 4.5, -90);
subplot(2, 2, 2), spiralgon(4, 37, -2.5, 45);
subplot(2, 2, 3), spiralgon(5, 61, 3, -90);
subplot(2, 2, 4), spiralgon(8, 91, -4, 22.5);
```

Note. Copy the five lines, paste them inside a single code block, and run it. This code block must *precede* your function(s).

```
clf
subplot(2, 2, 1), spiralgon(3, 41, 4.5, -90);
subplot(2, 2, 2), spiralgon(4, 37, -2.5, 45);
subplot(2, 2, 3), spiralgon(5, 61, 3, -90);
subplot(2, 2, 4), spiralgon(8, 91, -4, 22.5);
```



Problem 3.

a.)

3. (Machine epsilon; adapted from **LM** 9.3–3(a)) Recall that the number in the computer which follows 1 is $1 + \boxed{\text{eps}}$, which can be verified in MATLAB by

```
>> format long
```

```
>> (1 + 0.5*eps) - 1
ans =
     0
>> (1 + 0.51*eps) - 1
ans =
     2.220446049250313e-16
```

(a) Verify that the number in the computer which follows 8 is $8 + 8\boxed{\text{eps}}$ by numerically calculating $8 + 4\boxed{\text{eps}}$ and $8 + 4.01\boxed{\text{eps}}$.

```
format long
(1 + 0.51*eps) - 1
```

```
ans =
     2.220446049250313e-16
```

```
eps
```

```
ans =
     2.220446049250313e-16
```

```
(8 + 8*eps) - 8
```

```
ans =
     1.776356839400250e-15
```

```
(8 + 4*eps) - 8
```

```
ans =
     0
```

```
(8 + 4.01*eps) - 8
```

```
ans =
     1.776356839400250e-15
```

b.)

(b) Verify that the number in the computer which precedes 16 is $16 - 8\boxed{\text{eps}}$ by numerically calculating $16 - 4.01\boxed{\text{eps}}$ and $16 - 4\boxed{\text{eps}}$.

```
(16 - 8*eps) - 16
```

```
ans =
    -1.776356839400250e-15
```

```
(16 - 4.01*eps) - 16
```

```
ans =
    -1.776356839400250e-15
```

```
(16 - 4*eps) - 16
```

```
ans =
     0
```

c.)

(c) What are the numbers in the computer that precedes and follows $2^{10} = 1024$, respectively? Verify your claims in MATLAB by carrying out appropriate calculations.

Note. Begin with `format long` as shown in the example above. This is needed only once before the beginning of part (a).

Note. Answer each part of the problem in a single code block. No external script needs to be written.

```
(1024 + 1024*eps) - 1024
```

```
ans =
     2.273736754432321e-13
```

```
(1024 + 512*eps) - 1024
```

```
ans =
     0
```

```
(1024 + 512.01*eps) - 1024
```

```
ans =
     2.273736754432321e-13
```

```
(1024 - 512*eps) - 1024
```

```
ans =
    -1.136868377216160e-13
```

```
(1024 - 256*eps) - 1024
```

```
ans =
     0
```

```
(1024 - 256.01*eps) - 1024
```

```
ans =
    -1.136868377216160e-13
```

Problem 4.

a.)

4. (Catastrophic cancellation; **LM** 9.3–10) We revisit the function from Problem 3 of Homework 3. Consider the function

$$f(x) = \begin{cases} \dfrac{e^x - 1}{x} & \text{if } x \neq 0 \\ 1 & \text{if } x = 0, \end{cases}$$

where we are interested in exploring the catastrophic cancellation which occurs as $x \to 0$ since $e^x \to 1$ as $x \to 0$.

(a) ✏ Use the Taylor series expansion of e^x to prove that f is continuous at 0.

Def: f is continuous at a if

$$\lim_{x \to a} f(x) = f(a)$$

We need to show that:

$$\lim_{x \to 0} \frac{e^x - 1}{x} = 1 = f(0)$$

Using Taylor series of e^x:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots\ldots$$

$$\lim_{x \to 0} \frac{e^x - 1}{x} = \lim_{x \to 0} \frac{\left(1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots\ldots\right) - 1}{x}$$

$$= \lim_{x \to 0} \frac{x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots}{x}$$

$$= \lim_{x \to 0} 1 + \frac{x}{2!} + \frac{x^2}{3!} + \frac{x^3}{4!} + \ldots$$

$$= \boxed{1}$$

b.i.)

(b) Now calculate $f(x)$ numerically for $x = 10^{-k}$ where $k \in \mathbb{N}[1, 20]$ in three slightly different ways:

i. Calculate $f(x)$ as written.

ii. Calculate it as

$$f_1(x) = \frac{e^x - 1}{\log e^x}, \quad \text{for } x \neq 0.$$

(You and I know that analytically $f_1(x) \equiv f(x)$ for all nonzero x – but MATLAB doesn't.)

iii. MATLAB has a function which analytically subtracts 1 from the exponential to avoid catastrophic cancellation before the result is calculated numerically. So define the function $f_2(x)$ to be the same as $f(x)$ except that $e^x - 1$ is replaced by `expm1(x)`.

Tabulate the results using `disp` or `fprintf`. The table should have four columns with the first being x, the second using $f(x)$, the third using $f_1(x)$, and the fourth using $f_2(x)$, with all shown to full accuracy. Do it as efficiently as you can, without using a loop.

Note. Write your code for this part in a single code block. No external script needs to be written.

```
k = [1:20]';
x = 10 .^ -k;
fx = (exp(x) - 1) ./ x;
```

b.ii.)

```
f1x = (exp(x) - 1) ./ (log(exp(x)));
```

b.iii.)

```
f2x = expm1(x) ./ x;
format long
fprintf('    x                    fx                    f1x                    f2x')
```

```
    x                    fx                    f1x                    f2x
```

```
fprintf('    -----------------------------------------------------------------')
```

```
    -----------------------------------------------------------------
```

```
fprintf('%20.16f %20.16f %20.16f %20.16f\n', [x, fx, f1x, f2x]')
```

```
  0.1000000000000000   1.0517091807564771   1.0517091807564762   1.0517091807564762
  0.0100000000000000   1.0050167084167949   1.0050167084168058   1.0050167084168058
  0.0010000000000000   1.0005001667083846   1.0005001667083415   1.0005001667083417
  0.0001000000000000   1.0000500016671410   1.0000500016667082   1.0000500016667084
  0.0000100000000000   1.0000050000069649   1.0000050000166667   1.0000050000166667
  0.0000010000000000   1.0000004999621837   1.0000005000001666   1.0000005000001666
  0.0000001000000000   1.0000000494336803   1.0000000500000017   1.0000000500000017
  0.0000000100000000   0.9999999939225290   1.0000000050000000   1.0000000050000002
```

0.0000000010000000	1.0000000827403710	1.0000000005000000	1.0000000005000000
0.0000000001000000	1.0000000827403710	1.0000000000500000	1.0000000000500000
0.0000000000100000	1.0000000827403710	1.0000000000050000	1.0000000000050000
0.0000000000010000	1.0000889005823410	1.0000000000005000	1.0000000000005000
0.0000000000001000	0.9992007221626409	1.0000000000000500	1.0000000000000500
0.0000000000000100	0.9992007221626409	1.0000000000000051	1.0000000000000051
0.0000000000000010	1.1102230246251565	1.0000000000000004	1.0000000000000007
0.0000000000000001	0.0000000000000000	NaN	1.0000000000000000
0.0000000000000000	0.0000000000000000	NaN	1.0000000000000000
0.0000000000000000	0.0000000000000000	NaN	1.0000000000000000
0.0000000000000000	0.0000000000000000	NaN	1.0000000000000000
0.0000000000000000	0.0000000000000000	NaN	1.0000000000000000

c.)

(c) Comment on the results obtained in the previous part. Explain why certain methods work well while others do not.

The values in all three columns differ quite a lot as x gets smaller, but even from the start $f(x)$ differs from both $f_1(x)$ and $f_2(x)$ slightly. This can stem from the fact that $exp(x)$ can have roundoff error when directly computed and as we see as x gets smaller $f(x)$ starts producing nonsense numbers which is caused by catastrophic cancellation. $f_1(x)$ does not suffer from as many problems because it allows for a better approximation of the taylor series when compared to $f(x)$ which is subtracting two small numbers which causes that catastrophic cancellation. $f_1(x)$ allows us to not be subtracting and dividing numbers as close to each other but once x is small enough the value is indistinguishable for $f_1(x)$ causing the values $k \geq 16$ to just be 0 for both the numerator and denominator causing that NaN. $f_2(x)$ was designed with expm1 function which was designed to avoid catastrophic cancellation in the value $exp(x)-1$ for small x by using a different taylor series expansion instead of the normal one used in e^x which means that the values $K \geq 16$ actually get to and approximate to 1 instead of just displaying 0 or NaN.

Problem 5.

a.)

5. (Inverting hyperbolic cosine; **FNC** 1.3.6) The function

$$x = \cosh(t) = \frac{e^t + e^{-t}}{2}$$

can be inverted to yield a formula for acosh(x):

$$t = \log\left(x - \sqrt{x^2 - 1}\right). \qquad (\star)$$

In MATLAB, let `t=-4:-4:-16` and `x=cosh(t)`.

(a) Find the condition number of the problem $f(x) = \operatorname{acosh}(x)$ by hand. (You may use Equation ($\star$), or look up a formula for f' in a calculus book.) Then evaluate κ_f at the elements of x in MATLAB.

Condition Number formula:

a.)

$$\left| \frac{x \cdot f'(x)}{f(x)} \right|$$

$$t = \log\left(x - \sqrt{x^2-1}\right)$$

$$t' = \frac{1}{x-\sqrt{x^2-1}} \cdot \left(1 - \left(x \cdot \left(x^2-1\right)^{-\frac{1}{2}}\right)\right)$$

$$= \frac{1 - \frac{x}{\sqrt{x^2-1}}}{x - \sqrt{x^2-1}}$$

= Simplified

$$t' = -\frac{1}{\sqrt{x^2-1}}$$

Condition Number: $\dfrac{x \cdot -\left(\frac{1}{\sqrt{x^2-1}}\right)}{\log\left(x-\sqrt{x^2-1}\right)}$

$$\boxed{\left| \frac{-x}{\left(\sqrt{x^2-1}\right) \cdot \log\left(x-\sqrt{x^2-1}\right)} \right|}$$

```
clear
t = (-4:-4:-16)'
```

```
t = 4×1
    -4
    -8
   -12
   -16
```

```
x = cosh(t)
```

```
x = 4×1
10^6 ×
   0.000027308232836
   0.001490479161252
   0.081377395712574
   4.443055260253992
```

```
Kf = abs(-x ./ ((sqrt((x .^ 2)-1)) .* log(x-(sqrt((x .^ 2)-1)))))
```

```
Kf = 4×1
   0.250167787600418
   0.125000028131124
   0.083333332387735
   0.062505372058575
```

b.)

(b) Evaluate the right-hand side of Equation ($\star$) using `x` to approximate `t`. Record the accuracy of the answers (by displaying absolute and/or relative errors), and explain. (Warning: Use `format long` to get enough digits or use `fprintf` with a suitable format.)

```
format long
star = log(x-(sqrt((x.^2)-1)));
absErr = star - t;
relErr = absErr ./ t;
fprintf('    t                    star                 absErr               relErr')
```

```
    t                    star                 absErr               relErr
```

```
fprintf('    ------------------------------------------------------------------')
```

```
    ------------------------------------------------------------------
```

```
disp([t, star, absErr, relErr])
```

```
  -4.000000000000000  -4.000000000000046  -0.000000000000046   0.000000000000012
  -8.000000000000000  -8.000000000171090  -0.000000000171090   0.000000000021386
 -12.000000000000000 -12.000000137072186  -0.000000137072186   0.000000011422682
 -16.000000000000000 -15.998624871201619   0.001375128798381  -0.000085945549899
```

Based on looking at both t and equation star we see that there are some inconsistencies. While for the first three values of equation star we see that the absolute and relative errors are relatively small but as we go further from -4 it does get larger until at -16 we see that there is a stark difference between the two values and much more significant change in absolute error and relative error. This means that the equation star used for the inversion of the hyperbolic cosine function in matlab struggles to produce t back directly versus acosh which would produce t back directly.

c.)

(c) An alternate formula for $\operatorname{acosh}(x)$ is

$$t = -2\log\left(\sqrt{\frac{x+1}{2}} + \sqrt{\frac{x-1}{2}}\right). \qquad (\dagger)$$

Apply Equation ($\dagger$) to `x` and record the accuracy as in part (b). Comment on your observation.

```
cross = -2 * log((sqrt((x+1)/2))+(sqrt((x-1)/2)));
```

```
absErr = cross - t;
relErr = absErr ./ t;
fprintf('    t     star    absErr   relErr')
```

```
    t     star    absErr   relErr
```

```
fprintf('   -----------------------------------------------------------------')
```

```
   -----------------------------------------------------------------
```

```
disp([t, cross, absErr, relErr])
```

```
    -4    -4     0     0
    -8    -8     0     0
   -12   -12     0     0
   -16   -16     0     0
```

With this alternative formula we see that it produces the array of t back without any issues and as we see with the absolute error and relative error we see no problems which means that this equation produced a more effective result in matlab compared to the previous equation even though from a theoretical perspective they should produce the same result.

d.)

(d) ✎ Based on your experiments, which of the formulas ($\star$) and ($\dagger$) is unstable? What is the problem with that formula?

Note. Write your code for each of parts (a), (b), and (c) in a single code block. No external script needs to be written.

Of the two formulas (*) and (†) we see that equation (*) is unstable. As we see it does not produce t back as it should and therefore not giving us the true solution that we expect. This is encouraged by the fact that the absolute and relative errors increased as the numbers in the array increased. The possible problem with that formula is that it experiences catastrophic cancelation from that comes from the subtraction of $x - \sqrt{x^2-1}$ when $x>0$ grows extremely large and $x \approx \sqrt{x^2-1}$. This means that cosh(16) is extremely large which is why we see that problem. While the alternative mitigates the issue through avoiding any direct subtraction that could cause catastrophic cancellation.

Functions Used

```
function V = spiralgon(n, m, d_angle, d_rot)
% SPIRALGON plots spiraling regular n-gons
% input: n = the number of vertices
% m = the number of regular n-gons
% d_angle = the degree angle between successive n-gons
% (can be positive or negative)
```

```
% d_rot = the degree angle by which the innermost n-gon
% is rotated
% output: V = the vertices of the outermost n-gon

th = linspace(0, 360, n + 1) + d_rot;
V = [cosd(th);
    sind(th)];
C = colormap(hsv(m));
sPre1 = (n - 2) * 180;
sPre2 = (sPre1 / n) / 2;
s = sind((sPre1 - sPre2) - abs(d_angle))/sind(sPre2);
R = [cosd(d_angle) -sind(d_angle);
    sind(d_angle) cosd(d_angle)];
hold off
for i = 1:m
    if i > 1
        V = s*R*V;
    end
    plot(V(1,:), V(2,:), 'Color', C(i,:))
    hold on
end
set(gcf, 'Color', 'w')
axis equal, axis off
end
```